Bilkent University

Department of Computer Engineering

CS 319 Term Project

Icy Tower

Design Report

Section 1

Group 1C

Project Group Members:

Cansu Yıldırım

Berke Soysal

Ozan Kerem Devamlı

Supervisor: Eray Tüzün

Contents

1.Introduction

1.1 Purpose of the system

Icy Tower is a well-known action and platform game. This version of the Icy Tower is different from the original version in that there are bonuses, various character options, slightly different features of bars, and level options to play in our version. The goal of the player is to go up in the icy tower by jumping on to the bars as high as possible while collecting points. The design of the game is implemented to make the game user-friendly and make it run at high performance in order to make the players satisfied and entertained from the game.

1.2 Design Goals

In this section, the design goals of the Icy Tower System will be introduced. The design goals are derived from the nonfunctional requirements that are described in Analysis Report. Design goals guide the decisions we made when trade-offs are needed [1].

1.2.1 Usability

The game will be easily understandable for every person who is greater than age 6. The game will be easily playable even without looking How to Play section, and the GUI of the menus will be clear for most people.

1.2.2 Performance

The game will run at 30 frame per second, without any flickering or freezing. The game will run without any performance problem with any computer which at least have Windows XP, Intel Pentium III as CPU, and any graphics card which have 32MB memory.

1.2.3 Extendibility

The game will be implemented in an object oriented way that it will be extendable. Extra features and extensions will be easy to implement. These implementations will be done without making dramatic changes on the core code of the game. We will obey to the Object Oriented Approach to maintain the extendable design.

1.2.4 Robustness

There will be not any bugs will be in the game that leads to termination of the game, or user is being stuck on a place or a game freeze. We care about robustness for the best gaming experience to users.

Trade-Offs:

• Memory vs Performance

Performance is more valuable for the game Icy Tower. Since the drop in performance would result in the fps drop, which makes a game unplayable, we chose performance over memory. We used more memory, in order to decrease the number of instructions to call each object (such as providing memory for each imageview everytime), therefore increased the performance.

- Functionality vs Usability

Since what we are want to do is just a entertaining game, we try to make it a user friendly software. The user will not get confused by a lot of options, he may easily understand the game and enjoy it. Thus, we choose usability over functionality.

- Efficiency vs Portability

Icy tower game will not be a portable game so that it cannot be played in different platforms. However, since we optimize the game for one platform, it will provide the efficiency requirements.

2. High-Level Software Architecture

2.1 Subsystem Decomposition

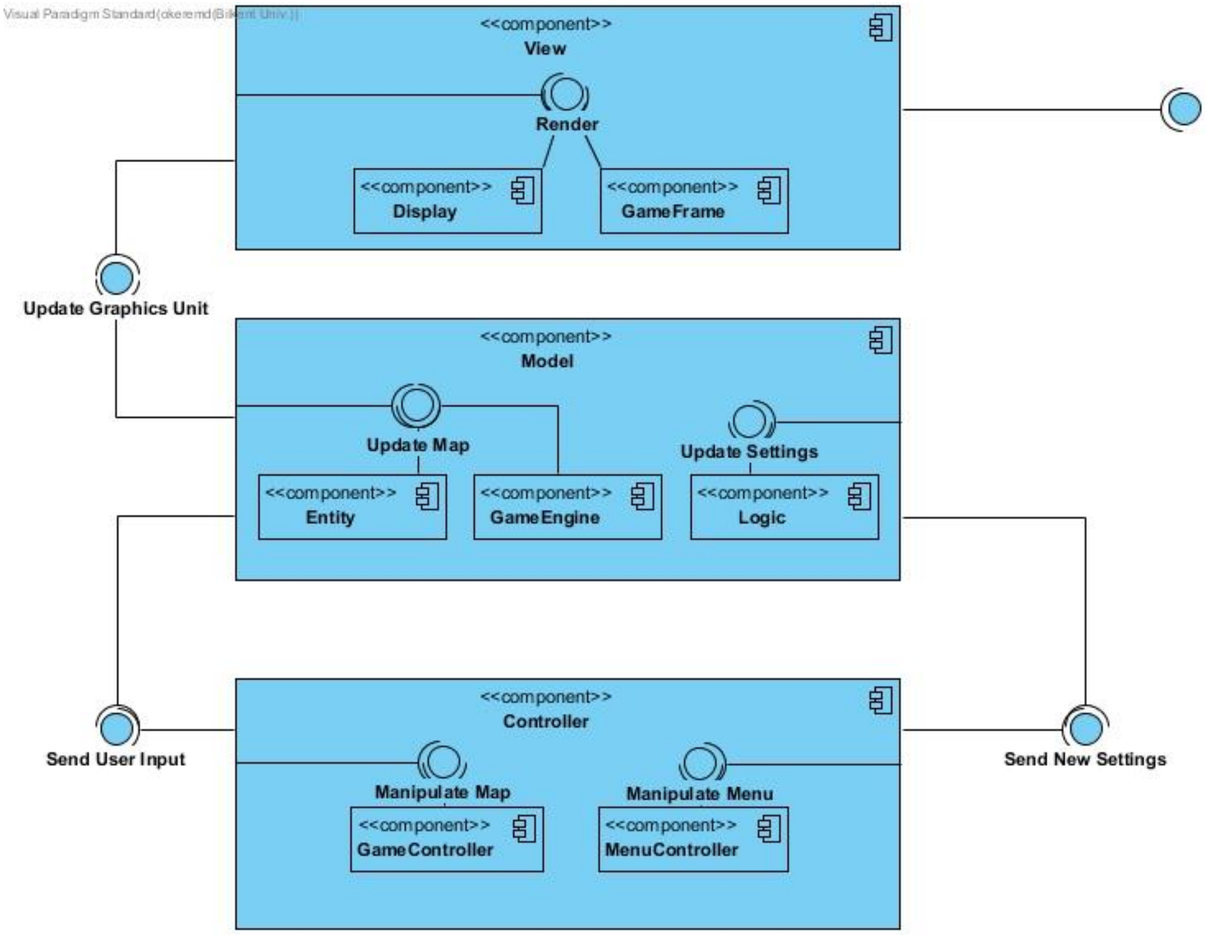


Figure 2.1.1: System Decomposition Diagram

The Icy Tower system is decomposed into three components. These are, *Model* (*Entity & GameLogic)*, *View*, and *Controller*. The system adopts a MVC (Model-View-Controller) architecture. View is completely independent from model, and the controller. Controller listens the user actions and interacts with model. Model is divided into two subpackage, Entity and Logic.

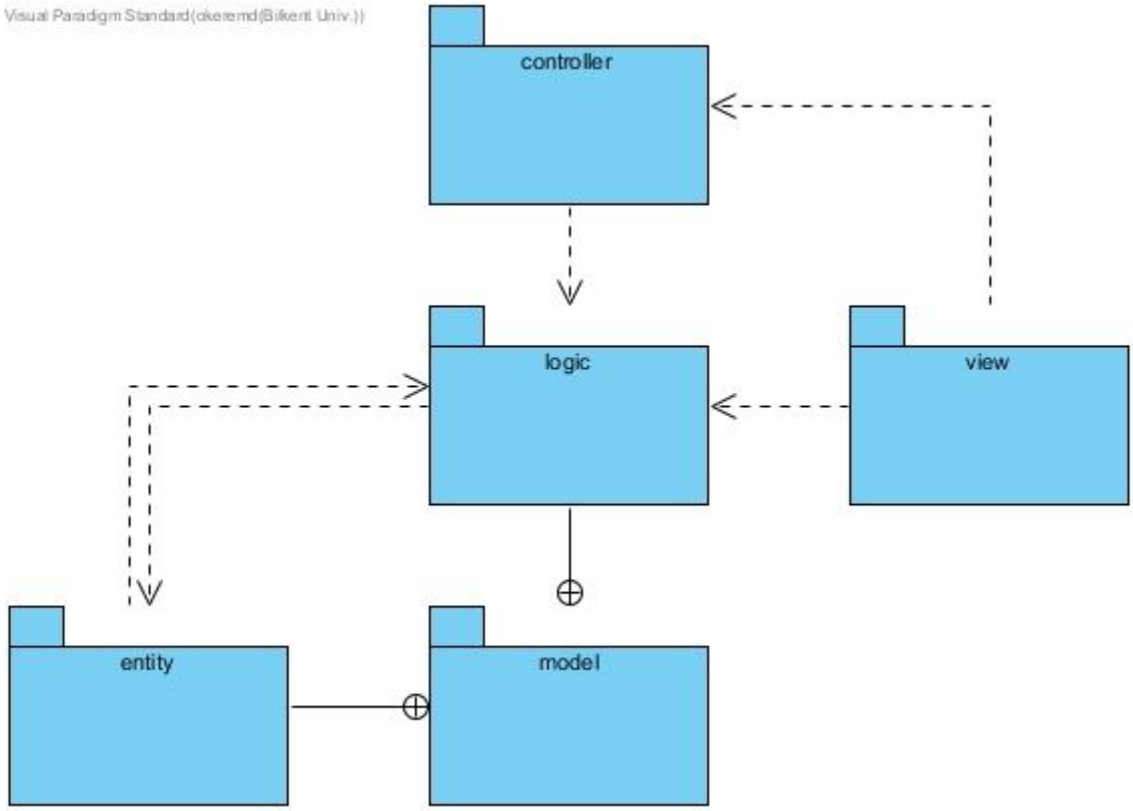


Figure 2.1.2: Packages

View: The view part is consisting of fxml[1] files that are being created for menu interfaces. These interfaces are designed with Scene Builder, which is a layout tool that let us create JavaFX application with less hard-code.

[1] **FXML** is an XML-based user interface markup language created by Oracle Corporation for defining the user interface of a JavaFX application.[2]

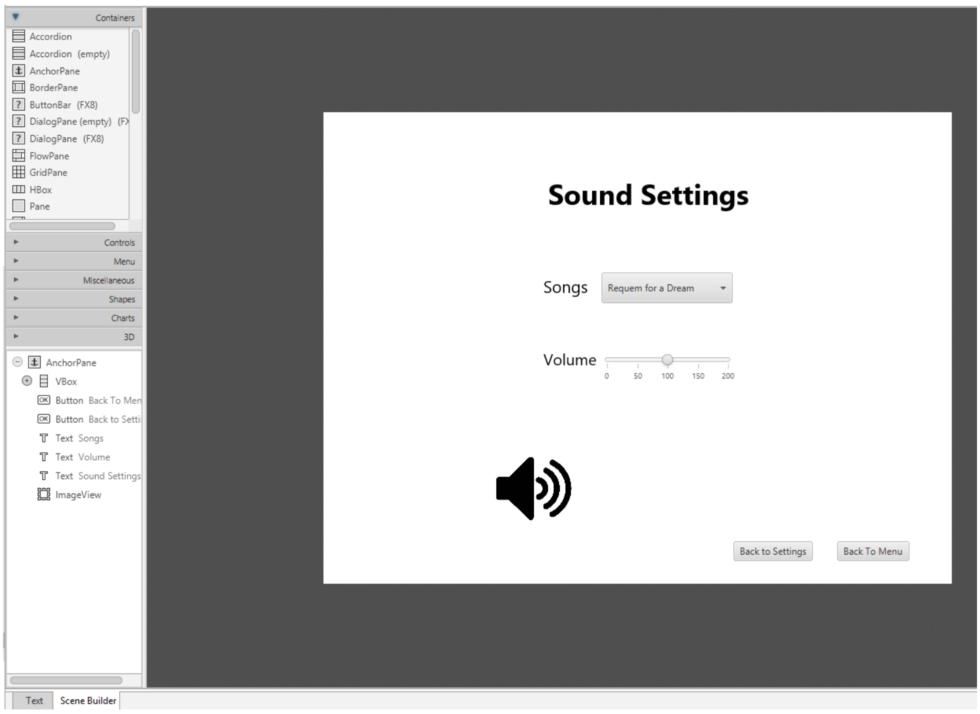


Figure 2.1.3: Screenshot of Scene Builder to Create Fxml files

Controller: The controller part is about handling user actions, each menu panel has a controller to listen user inputs and manipulating model. Controller has no other functionality than taking pressed buttons and sending it to model.

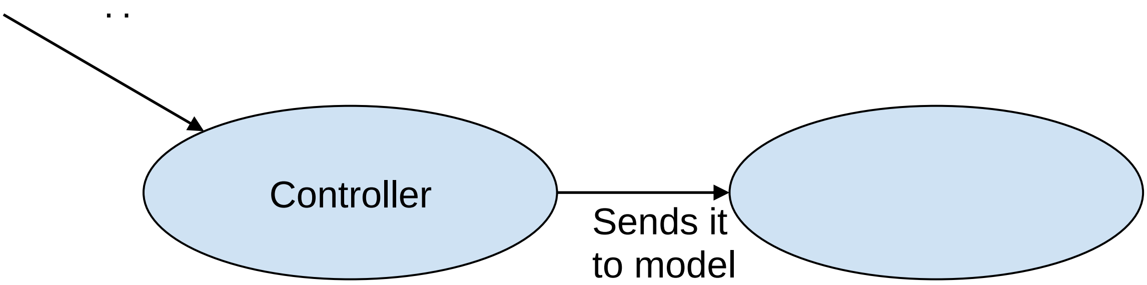


Figure 2.1.4: Behavior of controller

Model: The model consists of game logic and entity objects. User actions are interpreted as functions that manipulate entity objects according to game logic. Then model notifies the view **directly** to change the current display according to changed entity objects.

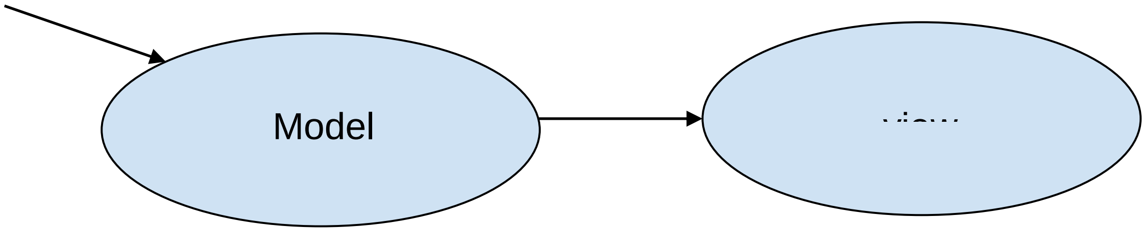


Figure 2.1.5: Relationship between model and view

Why we are choosing MVC?

- We are using MVC because we want to separate the user interactions with the general logic of the game.
- We want to separate game logic and rendering components. We want to keep the code as maintainable and reusable as possible.
- We want to write unit tests to test rendering, general game logic, MVC eases it because of separation of parts.
- MVC allows us to modify a subsystem without encountering failures in other subsystems.

2.2 Hardware/Software Mapping

- Icy Tower will be developed in Java Environment, and it can run on Windows, OSX or Linux distributions.
- The sounds of the game will be in .wav format. We are not using .mp3 because we don’t need compression for sounds as we have enough storage for them.
- The images of the game including the game and the menu will be in .png format because it’s a lossless compression unlike .jpg and we care about the quality of the game graphics.
- To play the game, a keyboard and a monitor is needed.
- To implement the game we are using JavaFX API. JavaFX is a graphics library allows us to create applications with GUI. We do not have enough time or experience to create our own graphics library

Why JavaFX? Why not Swing?

- JavaFX allows us to create FXML files that creates GUI with Scene Builder, without any Java Code. This makes the view part more maintainable.
- JavaFX fits for MVC architecture (FXML files for display, Java code for model and controller.)
- JavaFX has support for CSS, that helps us to designing GUI more conveniently.

2.3 Persistent Data Management

The game will keep data for the user preferences and high scores. The user preferences that being stored are the controller settings, audio volume and selected game character. Additionally, best 10 high score will be keeping by the game. Both these will be kept in separate text files and will be no larger than 200 byte.

Why we are not using a Database Management System?

We don't need to use a database for the data we hold, because we only need to hold a few bytes of storage, for high score and settings. These can be easily handled with file system.

2.4 Access Control and Security

The game needs no Internet connection, it only uses some local storage to keep the images, and the settings file. Thus, security is not our main concern. There is no authentication needed to play the game. Anybody who has access to that computer can play the game without any authentication.

2.5 Boundary conditions

In this section, the boundary conditions of the system will be explained. These are initialization, termination and failure.

2.5.1. Initialization

After selecting play at main menu, player can select a difficulty level and then get into the game. At first initialization of the game the high score will be empty, and default settings will be loaded for the game. These settings will use arrow keys for button, set sound level at maximum, and the default game character for the appearance.

2.5.2. Termination

User can quit any time from the game, by pausing the game and then selecting the quit game. He can also go to main menu from there. The user can also close the application window to quit the game.

2.5.3. Failure

If an unexpected failure occurs, the game software may have a runtime error and quit. But we will try to make this possibility as less as possible. If the program can’t load game images from the file system, the game may crash.

3. Subsystem services

3.1 View Subsystem

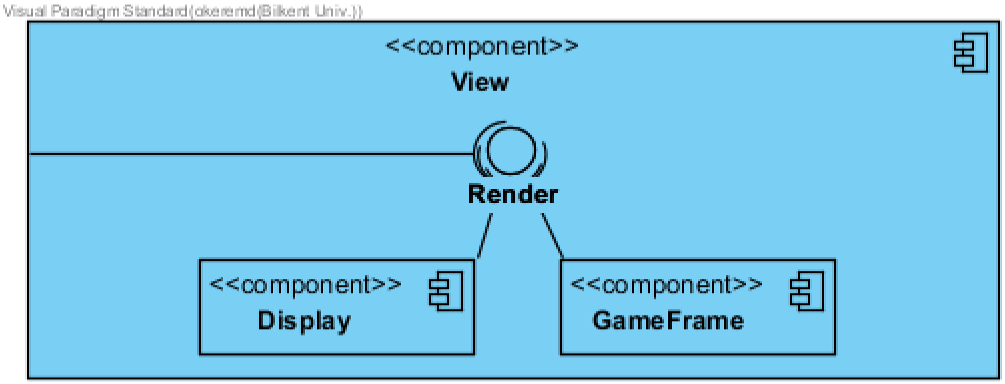


Figure 3.1: View Subsystem Service

View:

This part is responsible for rendering what it gets. The DisplayFrame is responsible for rendering the parts about menu. The GameFrame is assigned to draw game graphics. These two subcomponents listen the model subsystem and get notified, and being updated by the model subsystem. It also sends to controller the user inputs.

3.2 Controller Subsystem

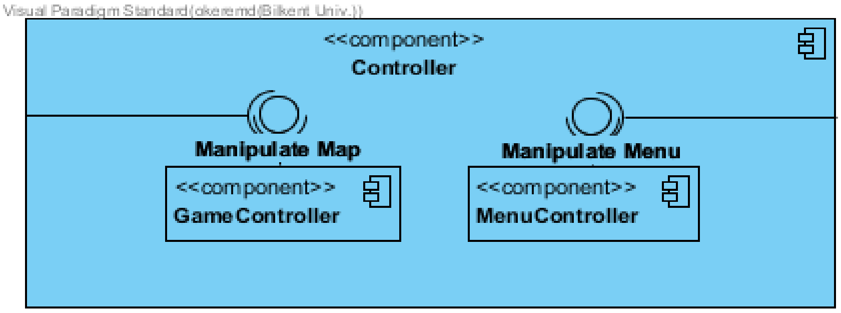


Figure 3.2: Controller Subsystem Service

Controller:

This part is a link between the model and the view. Basically what it does is to turn user inputs into meaningful commands for the system. It has no logic mechanism in it. It tries to manipulate the Model Subsystem according to commands.The Model Subsystem is responsible for determining what actions will be done by the system according to game logic.

3.3 Model (Entity & Game Logic) Subsystem

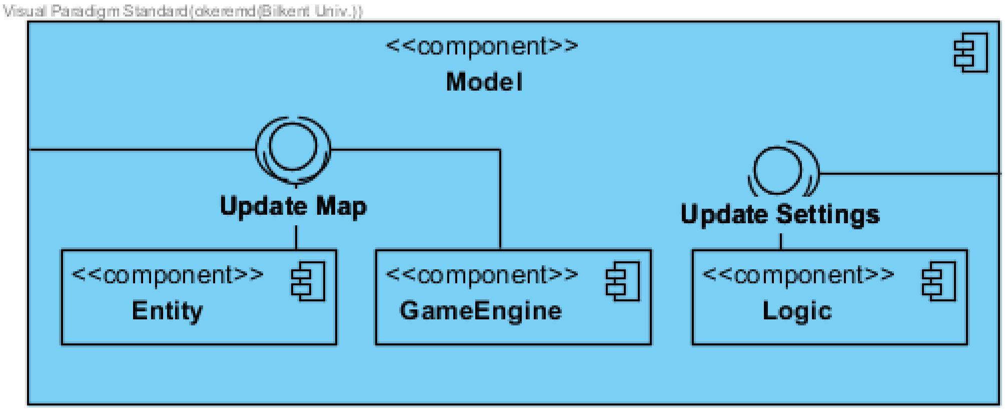


Figure 3.3: Model Subsystem Service

Model (Entity & Gamelogic):

This is the most crucial part of the system. It holds all entity objects of the game, and the game logic. The Controller subsystem tries to modify it. The Model Subsystem responds the modify requests by it’s logic subsystem. If it allows, then entity objects can be modified, and the modified entities notifies the View Subsystem to make them changed.

4. Low-level design

4.1 Object design trade-offs

4.1.1 Patterns and Their Trade-Offs

We have used some design patterns which have some trade-offs in themselves.

Singleton Design Pattern:

We used **singleton pattern** for the manager objects on the model and controller objects. The reason of using singleton is that these classes need only one instance to get all the functionality we need. A possible trade-off this might be the lack of flexibility as we do not allow creating multiple instances for those classes, but the code gets more reliable as we are ensuring that we always get the same instance.

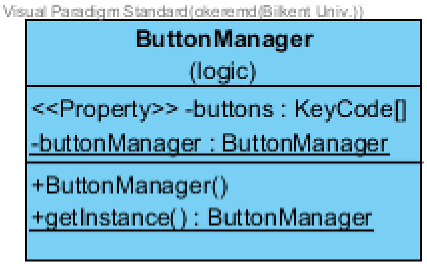

Visual Paradigm Standard(okeremd(Bilkent Univ.))
ButtonManager
(logic)
<<Property>> -buttons : KeyCode[]
-buttonManager : ButtonManager
+ButtonManager()
+getInstance( ) : ButtonManager

Figure 4.1.1: Singleton Pattern Example

Facade Design Pattern:

We used **facade pattern** with *gameController* class. *gameController* class is like an interface for user that has all functionalities that a user can perform during the game, moving, jumping and pausing the game, and it communicates with model objects to perform the desired behaviour of user. A negative effect of this pattern might be the restriction of the user from the system, and allow him only a few commands, but this is for preventing unintended system behaviours.

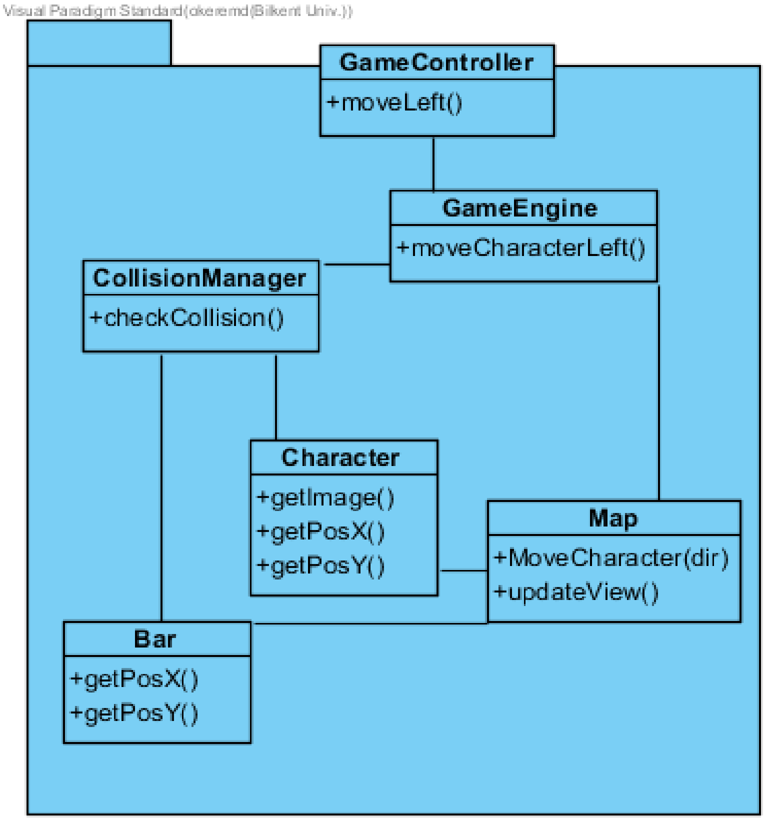


Figure 4.1.2: Façade Pattern

Strategy Pattern:

Strategy pattern is used for the game object **bars**. Some of the bars have being broken functionality as time goes and some of them are not. We are realizing this functionality using an interface called **BreakingBehavior**, and it implemented by two classes **Breakable** and **Unbreakable**. Each bar uses one of these classes to implement one of these behaviours. This pattern may give to much overhead, as if we need too many extra functionalities for different bars, but it increases reusability of the code.

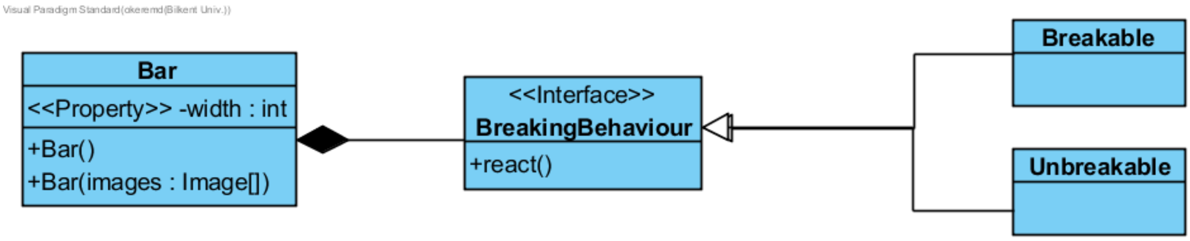


Figure 4.1.3: Strategy Pattern

4.1.2 System Architecture Style and its’ Trade-off’s

We used MVC as general system architecture as we have described in section 2. This approach has several pros/cons that we will discuss in this section.

Pros:

- Separation of Concerns: We have model, view and controller that each part works on distinct sections, eases our job for implementation and testing.
- Fast Development: MVC allowed each one of us to work on different aspects of the game, as the subsystems model, view and controller are has little effect on each other.
- Independent Parts: As we have independent model and view, one modification we made on model, does not entirely affect the whole structure.

Cons:

- More Complex System: MVC made the system more complex since it required the division between elements which would be in same classes. The division causes by MVC created a set of relationships between classes which might not occur if we did not used MVC.
- Inefficiency: When view package access the images in model package, it uses additional instructions, since the different packages checks for different privacy settings. This decreases performance.
- Requirement for Multiple Developers: As all of us agreed on, we would not use MVC if we would code this project individually. Since every package has its own expert and own rules, one developer could feel overwhelmed while trying to master each subsystem.

4.2 Final object design

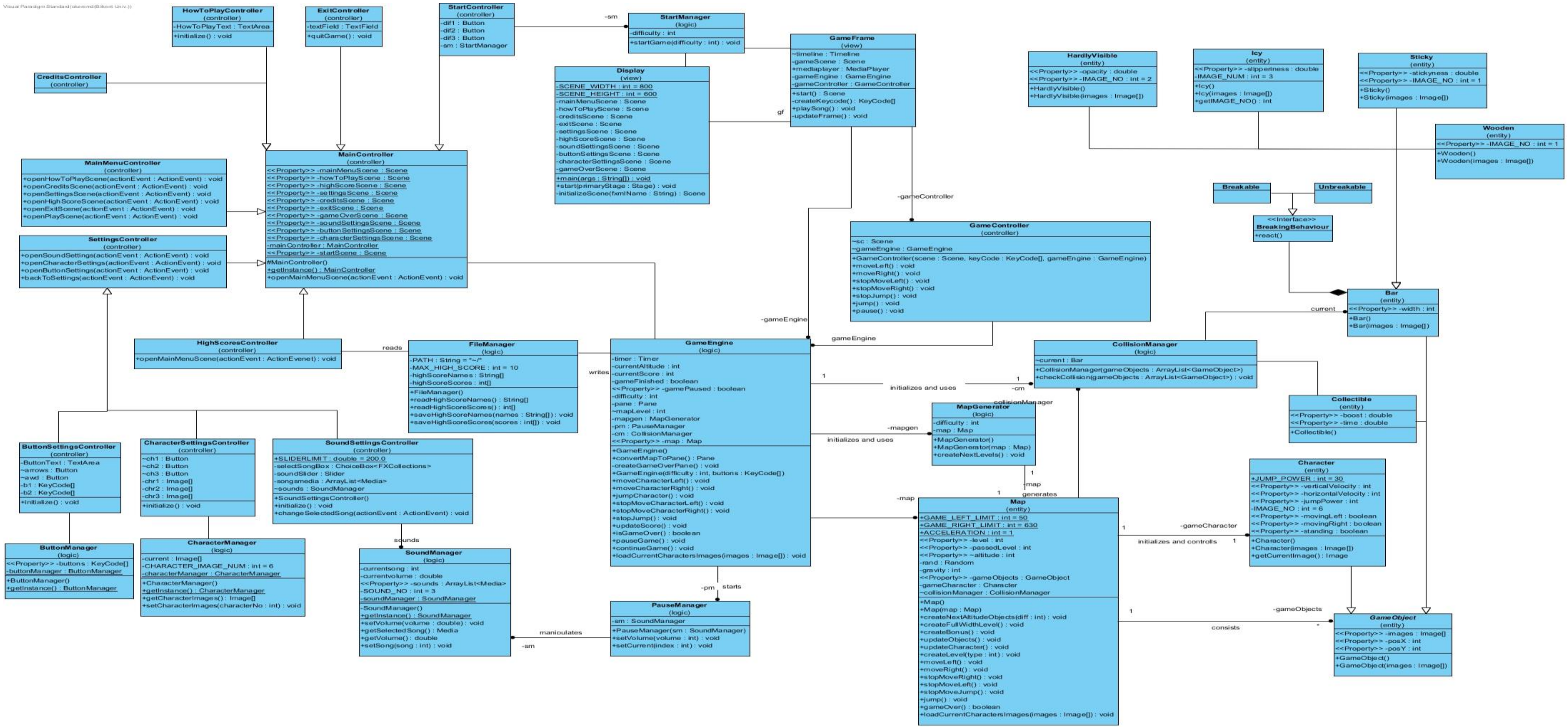

Visual Paradigm Standard(okeremd(Bilkent Univ.))
HowToPlayController
(controller)
-HowToPlayText : TextArea
+initialize() : void
ExitController
(controller)
-textField : TextField
+quitGame() : void
StartController
(controller)
-dif1 : Button
-dif2 : Button
-dif3 : Button
-sm : StartManager
-sm
StartManager
(logic)
-difficulty : int
+startGame(difficulty : int) : void
CreditsController
(controller)
GameFrame
(view)
~timeline : Timeline
-gameScene : Scene
+mediaplayer : MediaPlayer
-gameEngine : GameEngine
-gameController : GameController
+start() : Scene
-createKeycode() : KeyCode[]
+playSong() : void
-updateFrame() : void
gf
Display
(view)
-SCENE_WIDTH : int = 800
-SCENE_HEIGHT : int = 600
-mainMenuScene : Scene
-howToPlayScene : Scene
-creditsScene : Scene
-exitScene : Scene
-settingsScene : Scene
-highScoreScene : Scene
-soundSettingsScene : Scene
-buttonSettingsScene : Scene
-characterSettingsScene : Scene
-gameOverScene : Scene
+main(args : String[]) : void
+start(primaryStage : Stage) : void
-initializeScene(fxmlName : String) : Scene
MainMenuController
(controller)
+openHowToPlayScene(actionEvent : ActionEvent) : void
+openCreditsScene(actionEvent : ActionEvent) : void
+openSettingsScene(actionEvent : ActionEvent) : void
+openHighScoreScene(actionEvent : ActionEvent) : void
+openExitScene(actionEvent : ActionEvent) : void
+openPlayScene(actionEvent : ActionEvent) : void
MainController
(controller)
<<Property>> -mainMenuScene : Scene
<<Property>> -howToPlayScene : Scene
<<Property>> -highScoreScene : Scene
<<Property>> -settingsScene : Scene
<<Property>> -creditsScene : Scene
<<Property>> -exitScene : Scene
<<Property>> -gameOverScene : Scene
<<Property>> -soundSettingsScene : Scene
<<Property>> -buttonSettingsScene : Scene
<<Property>> -characterSettingsScene : Scene
-mainController : MainController
<<Property>> -startScene : Scene
#MainController()
+getInstance() : MainController
+openMainMenuScene(actionEvent : ActionEvent) : void
SettingsController
(controller)
+openSoundSettings(actionEvent : ActionEvent) : void
+openCharacterSettings(actionEvent : ActionEvent) : void
+openButtonSettings(actionEvent : ActionEvent) : void
+backToSettings(actionEvent : ActionEvent) : void
-gameController
GameController
(controller)
~sc : Scene
~gameEngine : GameEngine
+GameController(scene : Scene, keyCode : KeyCode[], gameEngine : GameEngine)
+moveLeft() : void
+moveRight() : void
+stopMoveLeft() : void
+stopMoveRight() : void
+stopJump() : void
+jump() : void
+pause() : void
-gameEngine
gameEngine
HardlyVisible
(entity)
<<Property>> -opacity : double
<<Property>> -IMAGE_NO : int = 2
+HardlyVisible()
+HardlyVisible(images : Image[])
Icy
(entity)
<<Property>> -slipperiness : double
-IMAGE_NUM : int = 3
+Icy()
+Icy(images : Image[])
+getIMAGE_NO() : int
Sticky
(entity)
<<Property>> -stickyness : double
<<Property>> -IMAGE_NO : int = 1
+Sticky()
+Sticky(images : Image[])
Wooden
(entity)
<<Property>> -IMAGE_NO : int = 1
+Wooden()
+Wooden(images : Image[])
Breakable
Unbreakable
<<Interface>>
BreakingBehaviour
+react()
Bar
(entity)
<<Property>> -width : int
+Bar()
+Bar(images : Image[])
current
HighScoresController
(controller)
+openMainMenuScene(actionEvent : ActionEvenet) : void
reads
FileManager
(logic)
-PATH : String = "~/"
-MAX_HIGH_SCORE : int = 10
-highScoreNames : String[]
-highScoreScores : int[]
+FileManager()
+readHighScoreNames() : String[]
+readHighScoreScores() : int[]
+saveHighScoreNames(names : String[]) : void
+saveHighScoreScores(scores : int[]) : void
writes
GameEngine
(logic)
-timer : Timer
-currentAltitude : int
-currentScore : int
-gameFinished : boolean
<<Property>> -gamePaused : boolean
-difficulty : int
-pane : Pane
~mapLevel : int
-mapgen : MapGenerator
-pm : PauseManager
-cm : CollisionManager
<<Property>> -map : Map
+GameEngine()
+convertMapToPane() : Pane
-createGameOverPane() : void
+GameEngine(difficulty : int, buttons : KeyCode[])
+moveCharacterLeft() : void
+moveCharacterRight() : void
+jumpCharacter() : void
+stopMoveCharacterLeft() : void
+stopMoveCharacterRight() : void
+stopJump() : void
+updateScore() : void
+isGameOver() : boolean
+pauseGame() : void
+continueGame() : void
+loadCurrentCharactersImages(images : Image[]) : void
1
initializes and uses
1
-cm
CollisionManager
(logic)
~current : Bar
+CollisionManager(gameObjects : ArrayList<GameObject>)
+checkCollision(gameObjects : ArrayList<GameObject>) : void
collisionManager
Collectible
(entity)
<<Property>> -boost : double
<<Property>> -time : double
+Collectible()
-mapgen
initializes and uses
MapGenerator
(logic)
-difficulty : int
-map : Map
+MapGenerator()
+MapGenerator(map : Map)
+createNextLevels() : void
1
-map
1
generates
-map
ButtonSettingsController
(controller)
-ButtonText : TextArea
~arrows : Button
~awd : Button
-b1 : KeyCode[]
-b2 : KeyCode[]
+initialize() : void
CharacterSettingsController
(controller)
~ch1 : Button
~ch2 : Button
~ch3 : Button
-chr1 : Image[]
-chr2 : Image[]
-chr3 : Image[]
+initialize() : void
SoundSettingsController
(controller)
+SLIDERLIMIT : double = 200.0
-selectSongBox : ChoiceBox<FXCollections>
-soundSlider : Slider
-songsmedia : ArrayList<Media>
~sounds : SoundManager
+SoundSettingsController()
+initialize() : void
+changeSelectedSong(actionEvent : ActionEvent) : void
sounds
ButtonManager
(logic)
<<Property>> -buttons : KeyCode[]
-buttonManager : ButtonManager
+ButtonManager()
+getInstance() : ButtonManager
CharacterManager
(logic)
-current : Image[]
-CHARACTER_IMAGE_NUM : int = 6
-characterManager : CharacterManager
+CharacterManager()
+getInstance() : CharacterManager
+getCharacterImages() : Image[]
+setCharacterImages(characterNo : int) : void
SoundManager
(logic)
-currentsong : int
-currentvolume : double
<<Property>> -sounds : ArrayList<Media>
-SOUND_NO : int = 3
-soundManager : SoundManager
-SoundManager()
+getInstance() : SoundManager
+setVolume(volume : double) : void
+getSelectedSong() : Media
+getVolume() : double
+setSong(song : int) : void
manipulates
-sm
-pm
starts
PauseManager
(logic)
-sm : SoundManager
+PauseManager(sm : SoundManager)
+setVolume(volume : int) : void
+setCurrent(index : int) : void
Map
(entity)
+GAME_LEFT_LIMIT : int = 50
+GAME_RIGHT_LIMIT : int = 630
+ACCELERATION : int = 1
<<Property>> -level : int
<<Property>> -passedLevel : int
<<Property>> ~altitude : int
-rand : Random
-gravity : int
<<Property>> -gameObjects : GameObject
-gameCharacter : Character
~collisionManager : CollisionManager
+Map()
+Map(map : Map)
+createNextAltitudeObjects(diff : int) : void
+createFullWidthLevel() : void
+createBonus() : void
+updateObjects() : void
+updateCharacter() : void
+createLevel(type : int) : void
+moveLeft() : void
+moveRight() : void
+stopMoveRight() : void
+stopMoveLeft() : void
+stopMoveJump() : void
+jump() : void
+gameOver() : boolean
+loadCurrentCharactersImages(images : Image[]) : void
1
-gameCharacter
initializes and controlls
1
Character
(entity)
+JUMP_POWER : int = 30
<<Property>> -verticalVelocity : int
<<Property>> -horizontalVelocity : int
<<Property>> -jumpPower : int
-IMAGE_NO : int = 6
<<Property>> -movingLeft : boolean
<<Property>> -movingRight : boolean
<<Property>> -standing : boolean
+Character()
+Character(images : Image[])
+getCurrentImage() : Image
1
-gameObjects
consists
*
GameObject
(entity)
<<Property>> -images : Image[]
<<Property>> -posX : int
<<Property>> -posY : int
+GameObject()
+GameObject(images : Image[])

Figure 4.2.1: Class Diagram

4.3 Packages

4.3.1 Internal Packages

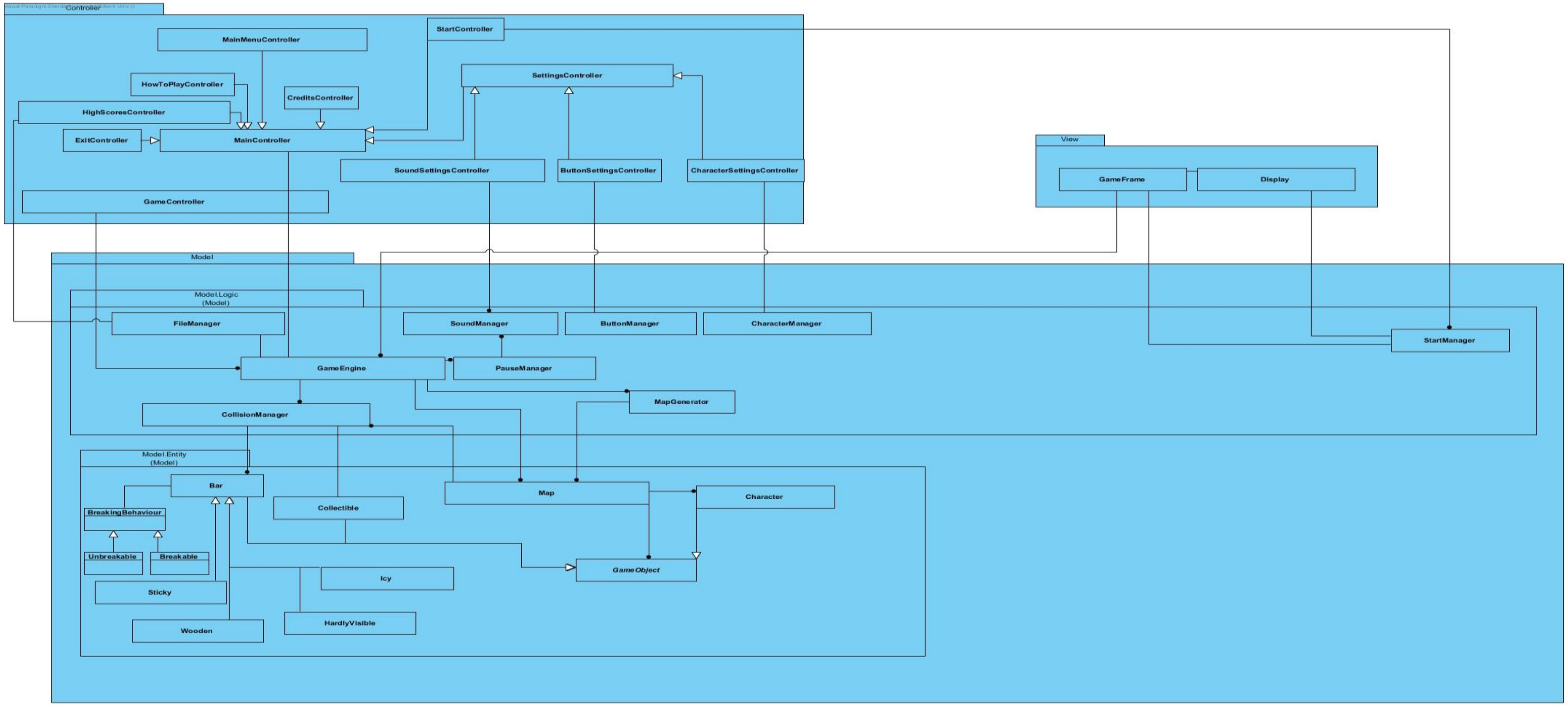


Figure 4.3.1: Packages with classes

4.3.1.1 model.entity

This package has game objects like character, bar etc.

4.3.1.2 model.logic

This package has elements about the management of the settings and the gameplay.

4.3.1.3.view

This package has fxml files for rendering the interfaces and game.

4.3.1.4 controller

This package has classes for each fxml file on view, to get user actions.

4.3.2. External Packages

4.3.2.1 java.util

This package provides lists to hold and manipulate data better, scanners to interact with files, random generators to add game luck factor.

4.3.2.2 javafx.scene.layout

This package provides User Interface Layouts.

4.3.2.3 javafx.scene.image

This package is for providing the set of classes to display images. To use the original images of characters, walls and backgrounds, we will include this package.

4.3.2.4 javafx.scene.scene

This package is for creating scenes and stages. It also contains methods that handles user events. Scenes and stages are the main source of handling the frames, which will be an important job in a game.

3.2.5 javafx.scene.input

This package is for handling the inputs from keyboard and mouse.

4.3.2.6 javafx.animations

Because of Icy Tower is being an animated game, we will use this package for moving images.

4.3.2.7 java.io

This package provides the interaction with operating system's file system. Also, this package will provide us assertions before encountering with file exceptions.

4.3.2.8 java.nio.file

This package defines interfaces and classes to access files, file attributes, and file systems.

4.4 Class Interfaces

4.4.1 Model.Entity

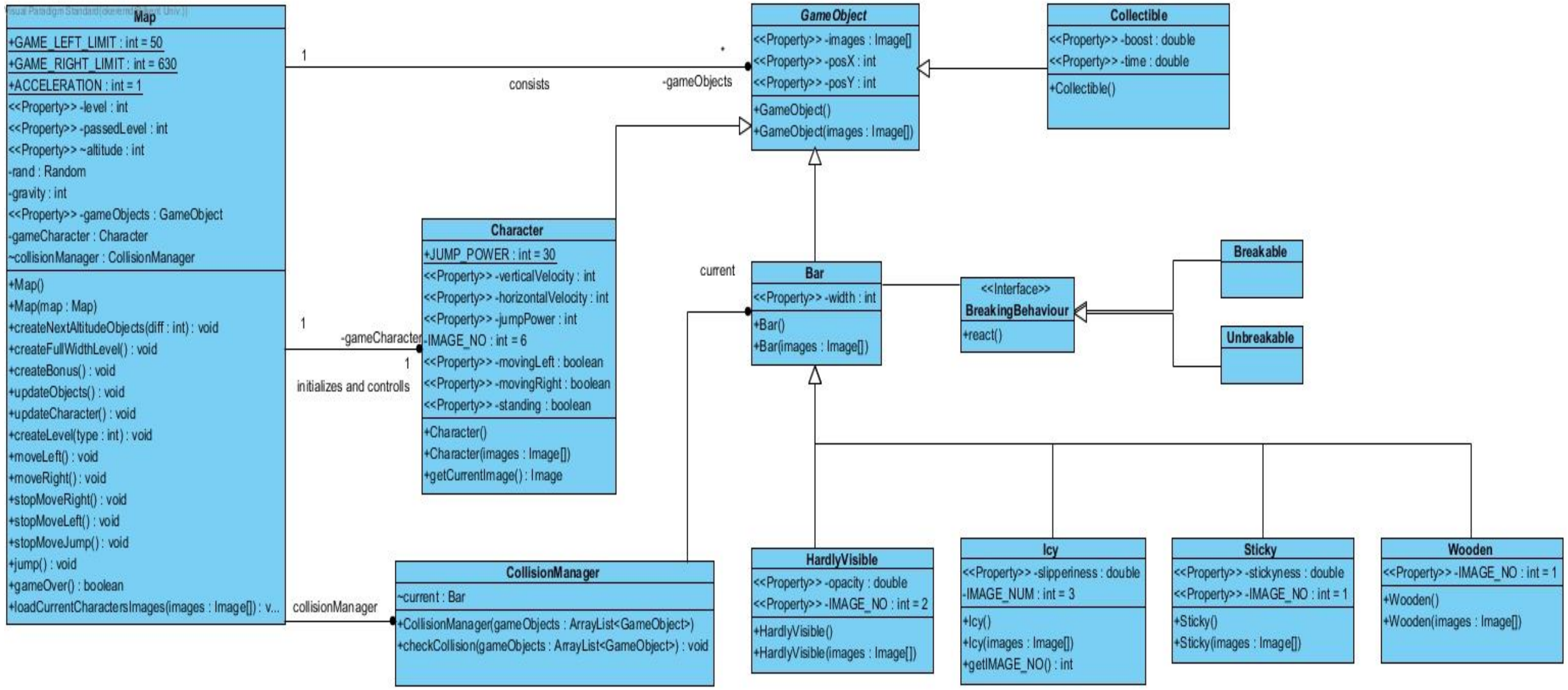

Visual Paradigm Standard(okeremd(Bilkent Univ.))
Map
+GAME_LEFT_LIMIT : int = 50
+GAME_RIGHT_LIMIT : int = 630
+ACCELERATION : int = 1
<<Property>> -level : int
<<Property>> -passedLevel : int
<<Property>> ~altitude : int
-rand : Random
-gravity : int
<<Property>> -gameObjects : GameObject
-gameCharacter : Character
~collisionManager : CollisionManager
+Map()
+Map(map : Map)
+createNextAltitudeObjects(diff : int) : void
+createFullWidthLevel() : void
+createBonus() : void
+updateObjects() : void
+updateCharacter() : void
+createLevel(type : int) : void
+moveLeft() : void
+moveRight() : void
+stopMoveRight() : void
+stopMoveLeft() : void
+stopMoveJump() : void
+jump() : void
+gameOver() : boolean
+loadCurrentCharactersImages(images : Image[]) : v...
1
consists
*
-gameObjects
1
-gameCharacter
1
initializes and controlls
collisionManager
GameObject
<<Property>> -images : Image[]
<<Property>> -posX : int
<<Property>> -posY : int
+GameObject()
+GameObject(images : Image[])
Collectible
<<Property>> -boost : double
<<Property>> -time : double
+Collectible()
Character
+JUMP_POWER : int = 30
<<Property>> -verticalVelocity : int
<<Property>> -horizontalVelocity : int
<<Property>> -jumpPower : int
-IMAGE_NO : int = 6
<<Property>> -movingLeft : boolean
<<Property>> -movingRight : boolean
<<Property>> -standing : boolean
+Character()
+Character(images : Image[])
+getCurrentImage() : Image
current
Bar
<<Property>> -width : int
+Bar()
+Bar(images : Image[])
<<Interface>>
BreakingBehaviour
+react()
Breakable
Unbreakable
CollisionManager
~current : Bar
+CollisionManager(gameObjects : ArrayList<GameObject>)
+checkCollision(gameObjects : ArrayList<GameObject>) : void
HardlyVisible
<<Property>> -opacity : double
<<Property>> -IMAGE_NO : int = 2
+HardlyVisible()
+HardlyVisible(images : Image[])
Icy
<<Property>> -slipperiness : double
-IMAGE_NUM : int = 3
+Icy()
+Icy(images : Image[])
+getIMAGE_NO() : int
Sticky
<<Property>> -stickyness : double
<<Property>> -IMAGE_NO : int = 1
+Sticky()
+Sticky(images : Image[])
Wooden
<<Property>> -IMAGE_NO : int = 1
+Wooden()
+Wooden(images : Image[])

Figure 4.4.1: Model.Entity Subsystem

4.4.1.1 Game Object

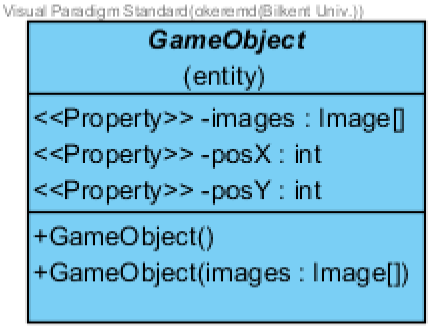

Visual Paradigm Standard(okeremd(Bilkent Univ.))
GameObject
(entity)
<<Property>> -images : Image[]
<<Property>> -posX : int
<<Property>> -posY : int
+GameObject()
+GameObject(images : Image[])

Figure 4.4.1.1: GameObject class

GameObject is an abstract object that contains properties that each GameObject should contain in order to be converted to a gameScene properly.

- **private Image[] images:** Image array that holds the images of the objects.
- **private int posX**: Holds the width value
- **private int posY:** Holds the height value

4.4.1.2 Character

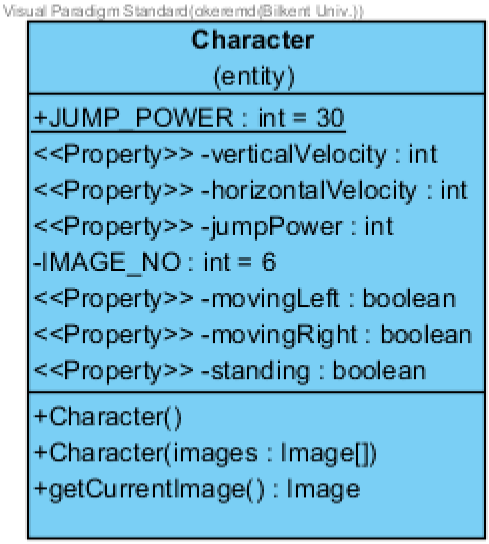

Visual Paradigm Standard(okeremd(Bilkent Univ.))
Character
(entity)
+JUMP_POWER : int = 30
<<Property>> -verticalVelocity : int
<<Property>> -horizontalVelocity : int
<<Property>> -jumpPower : int
-IMAGE_NO : int = 6
<<Property>> -movingLeft : boolean
<<Property>> -movingRight : boolean
<<Property>> -standing : boolean
+Character()
+Character(images : Image[])
+getCurrentImage() : Image

Figure 4.2.1.2: Character class

Character is the object that player controlls. It can go left, right or jump.

- **private int verticalVelocity:** Vertical velocity of the character
- **private int horizontalVelocity:** Horizontal velocity of the character
- **private int jumpPower:** The value of how much the character can jump

- **private final int IMAGE_NO:** The attribute that will hold how many images are necessary to create the object in frame.
- **private boolean movingLeft:** True if the character is moving left
- **private boolean movingRight:** True if the character is moving right
- **private boolean standing:** True if the character is not jumping
- **public Image getCurrentImage():** Returns the proper image of the character according to the character’s situation

4.4.1.3 Bar

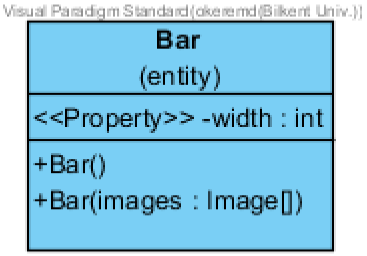

Visual Paradigm Standard(okeremd(Bilkent Univ.))
Bar
(entity)
<<Property>> -width : int
+Bar()
+Bar(images : Image[])

Figure 4.4.1.3: Bar class

Bar is the abstact class that has the common property of each bar which is width.

- **private int width:** Holds the width value of the bars

4.4.1.4 Wooden

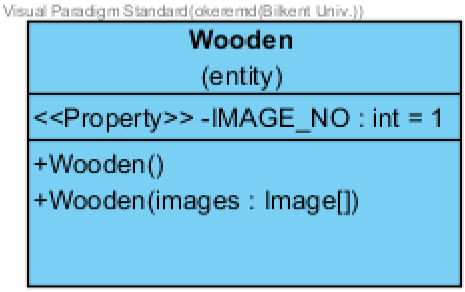

Visual Paradigm Standard(okeremd(Bilkent Univ.))
Wooden
(entity)
<<Property>> -IMAGE_NO : int = 1
+Wooden()
+Wooden(images : Image[])

Figure 4.4.1.4: Wooden class

Wooden bar does not raise any difficulties for the player. The game starts with this bar to make the players adapted the game without any hardships.

- **private final int IMAGE_NO:** The attribute that will hold how many images are necessary to create the object in frame.

4.4.1.5 Icy

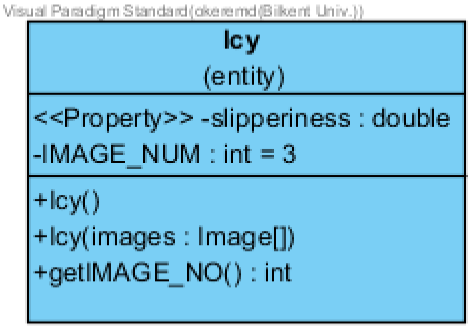

Visual Paradigm Standard(okeremd(Bilkent Univ.))
Icy
(entity)
<<Property>> -slipperiness : double
-IMAGE_NUM : int = 3
+Icy()
+Icy(images : Image[])
+getIMAGE_NO() : int

Figure 4.4.1.5: Icy class

Icy is type of bar, which has a slipperiness that causes character to accelerate.

- **private double slipperiness:** The rate of slipperiness of the bar. On the icy bar, the character can accelerate and it may be hard to stop.
- **private final int IMAGE_NO:** The attribute that will hold how many images are necessary to create the object in frame.

4.4.1.6 Sticky

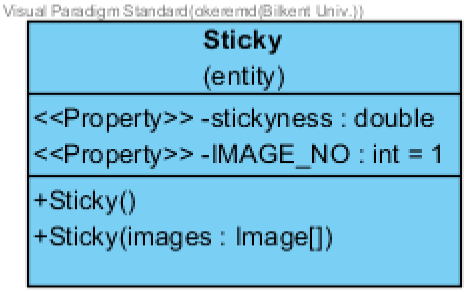
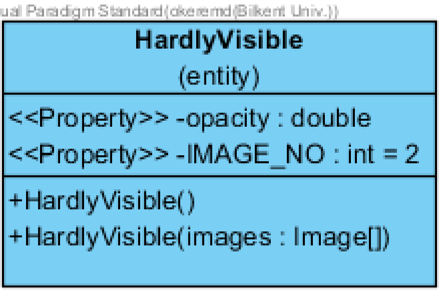

Visual Paradigm Standard(okeremd(Bilkent Univ.))
Sticky
(entity)
<<Property>> -stickyness : double
<<Property>> -IMAGE_NO : int = 1
+Sticky()
+Sticky(images : Image[])

Figure 4.4.1.6: Sticky class

Sticky is type of bar, which has stickiness that causes character to move hardly on it.

- **private double stickiness:** The rate of stickiness of the bar. Sticky bar makes the character moves slow down.
- **private final int IMAGE_NO:** The attribute that will hold how many images are necessary to create the object in frame.

4.4.1.7 HardlyVisible



HardlyVisible
(entity)

<<Property>> -opacity : double
<<Property>> -IMAGE_NO : int = 2

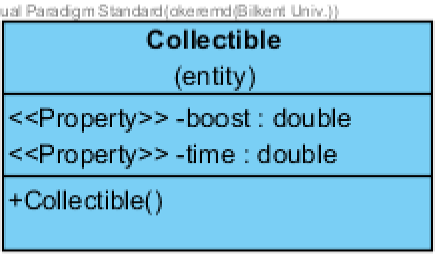
+HardlyVisible()
+HardlyVisible(images : Image[])

Figure 4.4.1.7: HardlyVisible class

HardlyVisible is type of bar, which has a value of transparancy.

- **private double opacity:** This bar will be transparant so that it will hard to see by the players.
- **private final int IMAGE_NO:** The attribute that will hold how many images are necessary to create the object in frame.

4.4.1.8 Collectible



Collectible
(entity)

<<Property>> -boost : double
<<Property>> -time : double

+Collectible()

Figure 4.4.1.8: Collectible class

This class is for collectible items which are bonuses.

4.4.1.9 Map

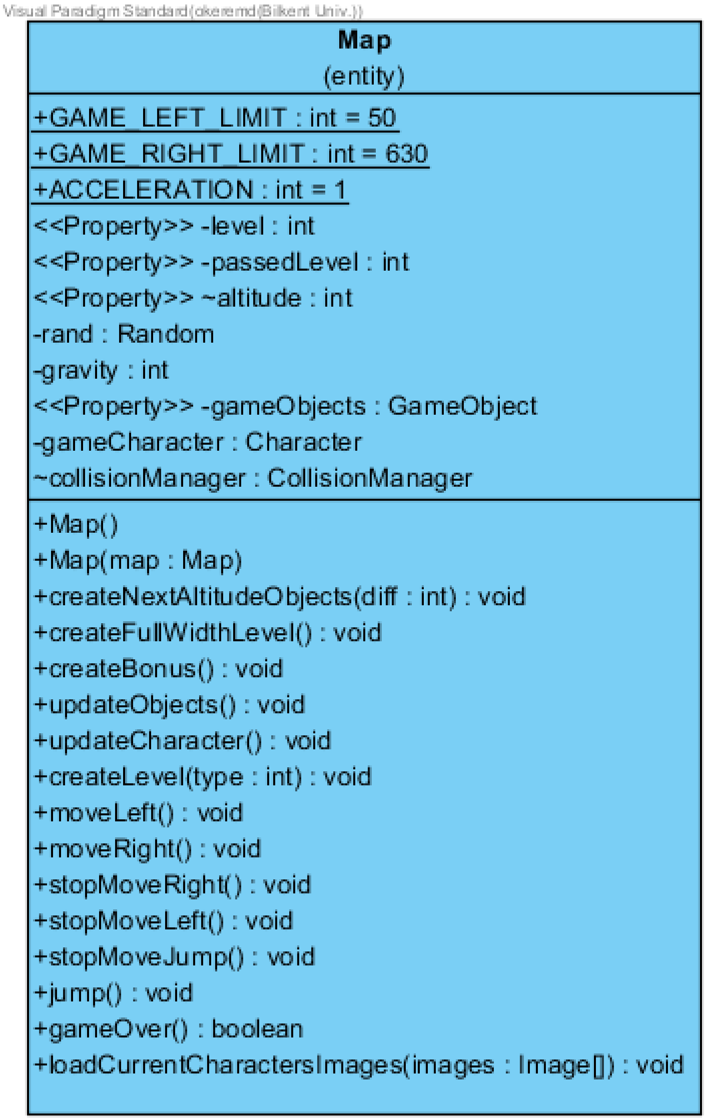

Visual Paradigm Standard(okeremd(Bilkent Univ.))
Map
(entity)
+GAME_LEFT_LIMIT : int = 50
+GAME_RIGHT_LIMIT : int = 630
+ACCELERATION : int = 1
<<Property>> -level : int
<<Property>> -passedLevel : int
<<Property>> ~altitude : int
-rand : Random
-gravity : int
<<Property>> -gameObjects : GameObject
-gameCharacter : Character
~collisionManager : CollisionManager
+Map()
+Map(map : Map)
+createNextAltitudeObjects(diff : int) : void
+createFullWidthLevel() : void
+createBonus() : void
+updateObjects() : void
+updateCharacter() : void
+createLevel(type : int) : void
+moveLeft() : void
+moveRight() : void
+stopMoveRight() : void
+stopMoveLeft() : void
+stopMoveJump() : void
+jump() : void
+gameOver() : boolean
+loadCurrentCharactersImages(images : Image[]) : void
[2]

Figure 4.4.1.9: Map class

Map is the class that contains GameObjects and Character in order to determine the interactions between them.

- **public static final int *GAME_LEFT_LIMIT*:** The left limit of the frame and game area.
- **public static final int *GAME_RIGHT_LIMIT*:** The right limit of the frame and game area.
- **public static final int *ACCELERATION*:** The change limit of the velocity of the character.
- **private ArrayList<GameObject> gameObjects:** Holds the GameObjects that created.
- **private Character gameCharacter:** Hold the Character that controlled by the player.
- **private int level:** Holds the last level created by MapGenerator.
- **private int passedLevel:** Holds the last level passed by the character.

[2] "~" means that the access modifier is not specified.

- **private int altitude:** Holds the pixelwise altitude of the character for high score calculation purposes.
- **private Random rand:** Random utility in order to create position of bars, creation of bonuses etc.
- **private int gravity:** Determines the change of player’s speed vertically.
- **CollisionManager collisionManager:** CollisionManager instance to determine whether character stands on a bar, takes a bonus etc.
- **public void createNextAltitudeObjects(int diff):** Creates the bars and bonuses that will be reached after passing current bar levels.
- **public void updateObjects():** Updates the gameObject’s positions.
- **public void updateCharacter():** Updates the character’s positions
- **public void createLevel(int type):** Creates the bar and its positions according to the progress of the player
- **public void moveLeft():** Sets the movingLeft attribute to true
- **public void moveRight():** Sets the movingRight attribute to true
- **public void stopMoveRight():** Sets the horizontal velocity to 0 and the movingRight attribute to false
- **public void stopMoveLeft():** Sets the horizontal velocity to 0 and the movingLeft attribute to false
- **public void jump():** Sets the vertical velocity to the jump power and the standing attribute to false
- **public boolean gameOver():** Returns true if the character goes out of the game frame
- **public void loadCurrentCharactersImages(images : Image[]):** Updates the image array of the character with user decided character’s images.

4.4.1.10 BreakingBehaviour

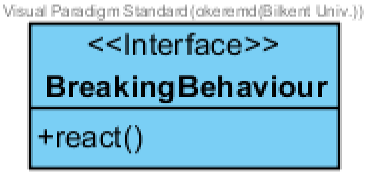


Figure 4.4.1.10: BreakingBehaviour Interface

public react(): the behavior of the bars, breakable or unbreakable is being implemented by this method, on Breakable and UnBreakable classes.

4.4.2 Model.Logic

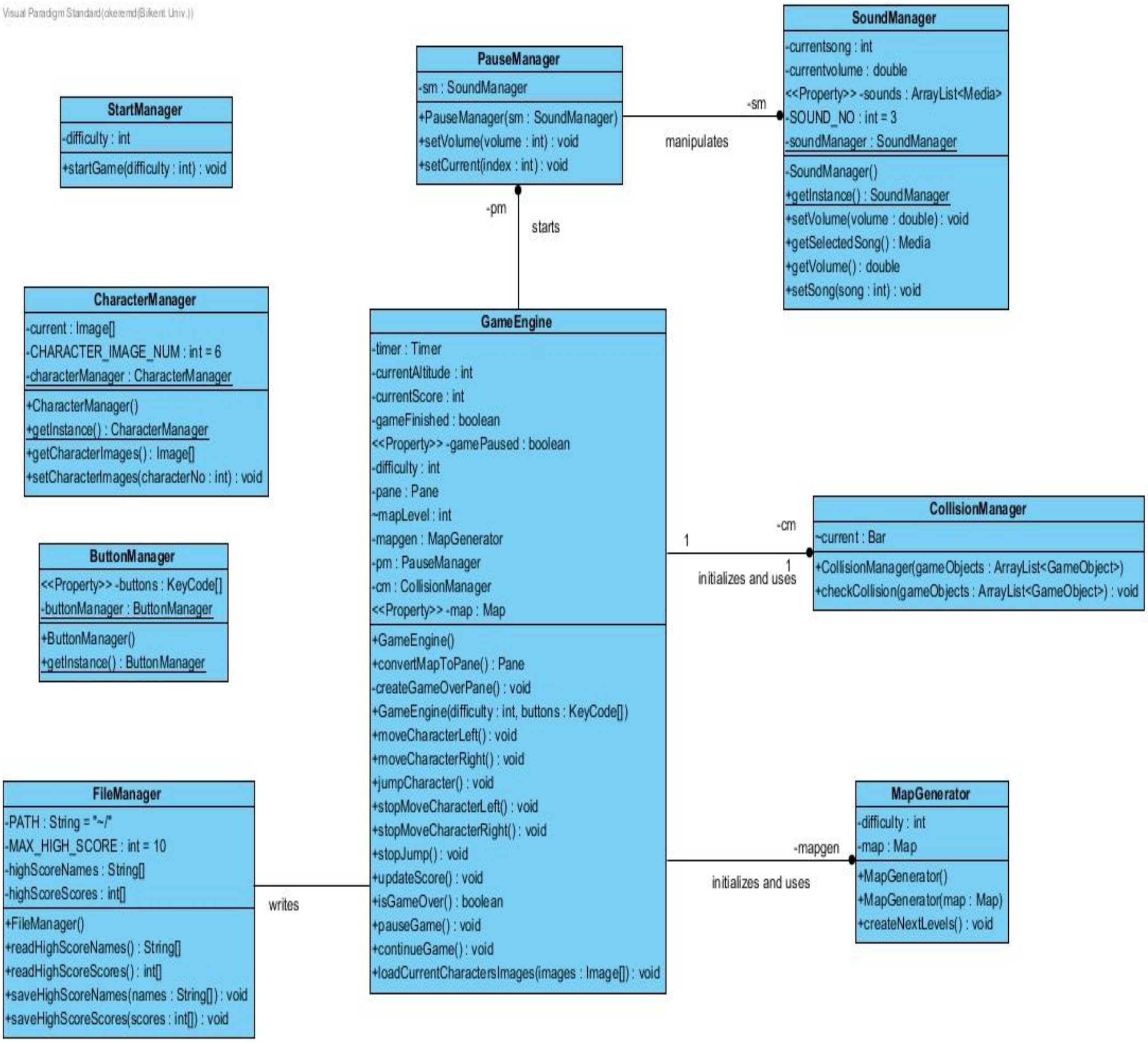

StartManager
-difficulty : int
+startGame(difficulty : int) : void
PauseManager
-sm : SoundManager
+PauseManager(sm : SoundManager)
+setVolume(volume : int) : void
+setCurrent(index : int) : void
SoundManager
-currentsong : int
-currentvolume : double
<<Property>> -sounds : ArrayList<Media>
-SOUND_NO : int = 3
-soundManager : SoundManager
-SoundManager()
+getInstance() : SoundManager
+setVolume(volume : double) : void
+getSelectedSong() : Media
+getVolume() : double
+setSong(song : int) : void
-sm
manipulates
-pm
starts
CharacterManager
-current : Image[]
-CHARACTER_IMAGE_NUM : int = 6
-characterManager : CharacterManager
+CharacterManager()
+getInstance() : CharacterManager
+getCharacterImages() : Image[]
+setCharacterImages(characterNo : int) : void
GameEngine
-timer : Timer
-currentAltitude : int
-currentScore : int
-gameFinished : boolean
<<Property>> -gamePaused : boolean
-difficulty : int
-pane : Pane
~mapLevel : int
-mapgen : MapGenerator
-pm : PauseManager
-cm : CollisionManager
<<Property>> -map : Map
+GameEngine()
+convertMapToPane() : Pane
-createGameOverPane() : void
+GameEngine(difficulty : int, buttons : KeyCode[])
+moveCharacterLeft() : void
+moveCharacterRight() : void
+jumpCharacter() : void
+stopMoveCharacterLeft() : void
+stopMoveCharacterRight() : void
+stopJump() : void
+updateScore() : void
+isGameOver() : boolean
+pauseGame() : void
+continueGame() : void
+loadCurrentCharactersImages(images : Image[]) : void
1
-cm
1
initializes and uses
CollisionManager
~current : Bar
+CollisionManager(gameObjects : ArrayList<GameObject>)
+checkCollision(gameObjects : ArrayList<GameObject>) : void
ButtonManager
<<Property>> -buttons : KeyCode[]
-buttonManager : ButtonManager
+ButtonManager()
+getInstance() : ButtonManager
FileManager
-PATH : String = "~/"
-MAX_HIGH_SCORE : int = 10
-highScoreNames : String[]
-highScoreScores : int[]
+FileManager()
+readHighScoreNames() : String[]
+readHighScoreScores() : int[]
+saveHighScoreNames(names : String[]) : void
+saveHighScoreScores(scores : int[]) : void
writes
-mapgen
initializes and uses
MapGenerator
-difficulty : int
-map : Map
+MapGenerator()
+MapGenerator(map : Map)
+createNextLevels() : void

Figure 4.4.2: Model.Logic Subsystem

4.4.2.1 GameEngine

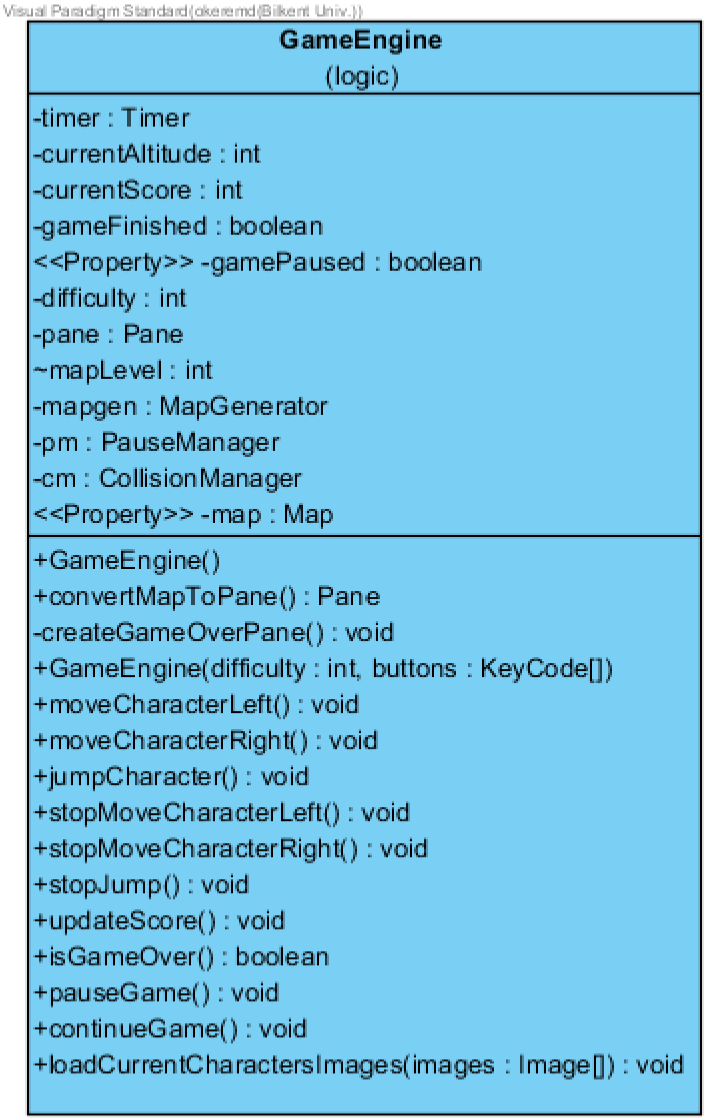


Figure 4.4.2.1: GameEngine class

GameEngine is the class that controlls the Map. It also provides PauseMenu and collects the score of the user.

- **private MapGenerator mapgen:** Holds MapGenerator instance.
- **private PauseManager pm:** Holds PauseManager instance.
- **private CollisionManager cm:** Holds CollisionManager instance.
- **private Timer timer:** Holds the time passed from the start of the game.
- **private int currentAltitude:** Holds the current altitude of the character as pixelwise.
- **private int currentScore:** Holds the current score of the character.
- **private boolean gameFinished:** Holds whether the game is finished.
- **private boolean gamePaused:** Holds whether the game is paused.
- **private int difficulty:** Holds the difficulty that decided by player before game.
- **private Map map:** Holds current Map's instance.

- **private Pane pane:** Holds the orientation of images of current Map.
- **int mapLevel:** Holds the number of created levels.
- **public GameEngine():** Default constructor.
- **public GameEngine(difficulty : int, keys : KeyCode[]):** Constructor that takes difficulty and play buttons that decided by player before game.
- **public Pane convertMapToPane():** Converts the map instance to pane which can be shown to player.
- **private void createGameOverPane():** In the end of the game, updates the pane with return to menu button.
- **public void loadCurrentCharactersImages(images : Image[]):** Updates the image array of the character with user decided character's images.
- **public void moveCharacterLeft():** Sets character's movingLeft to true.
- **public void moveCharacterRight():** Sets character's movingRight to true.
- **public void jumpCharacter():** Sets character's jump to true.
- **public void stopMoveCharacterLeft():** Sets character's movingLeft to false.
- **public void stopMoveCharacterRight():** Sets character's movingRight to false.
- **public void stopJump():** Sets character's jump to false.

4.4.2.2 MapGenerator

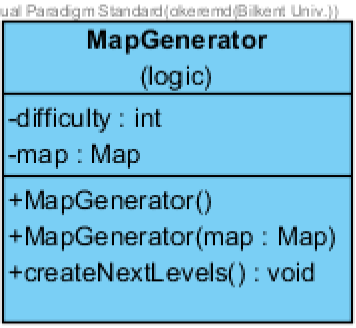
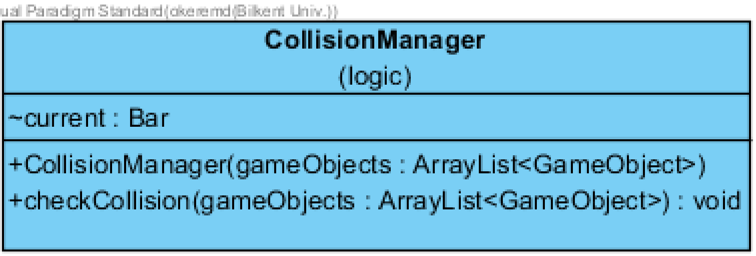


Figure 4.4.2.2: MapGenerator class

MapGenerator is responsible for creating a number of bars and bonuses when player reach the limit of the last created bars.

- **private Map map:** Holds the instance of the current Map.
- **public MapGenerator():** Default constructor.
- **public MapGenerator(Map map) :** Constructor with input current Map.
- **public void createNextLevels():** Created a number of levels that will be reached by character.

4.4.2.3 CollisionManager



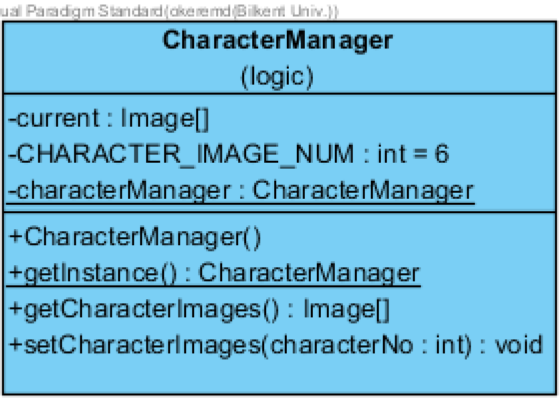
```
CollisionManager
(logic)
~current : Bar
+CollisionManager(gameObjects : ArrayList<GameObject>)
+checkCollision(gameObjects : ArrayList<GameObject>) : void
```

Figure 4.4.2.3: CollisionManager class

CollisionManager is responsible for determining the interactions between GameObjects and Character.

- **Bar current:** Holds the bar that player currently is on.
- **public CollisionManager(ArrayList<GameObject> gameObjects) :** Constructor that takes gameObjects as input.
- **public void checkCollision(ArrayList<GameObject> gameObjects) :** Checks each bar and bonus in order to determine whether they are in contact with character.

4.4.2.4 CharacterManager



```
CharacterManager
(logic)
-current : Image[]
-CHARACTER_IMAGE_NUM : int = 6
-characterManager : CharacterManager
+CharacterManager()
+getInstance() : CharacterManager
+getCharacterImages() : Image[]
+setCharacterImages(characterNo : int) : void
```

Figure 4.4.2.4: CharacterManager class

CharacterManager is responsible for the Character's images which is chosen by player.

- **private Image[] current:** Holds the images of the current selected character.
- **private final int CHARACTER_IMAGE_NUM:** Holds the number of images character contains.
- **private static CharacterManager *characterManager*:** Instance of characterManager.
- **public CharacterManager() :** Default constructor.
- **public static CharacterManager getInstance():** Singleton method that returns this.
- **public Image[] getCharacterImages() :** Returns the current selected character's images.
- **public void setCharacterImages(int characterNo):** Sets the current.

4.4.2.5 ButtonManager

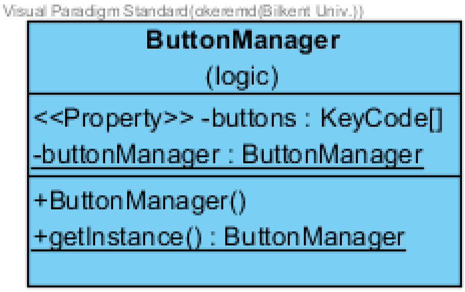

Visual Paradigm Standard(okeremd(Bilkent Univ.))
ButtonManager
(logic)
<<Property>> -buttons : KeyCode[]
-buttonManager : ButtonManager
+ButtonManager()
+getInstance() : ButtonManager

Figure 4.4.2.5: ButtonManager class

ButtonManager is responsible for the buttons that user decided.

- **private KeyCode[] buttons:** Holds the buttons that decided by player.
- **private static ButtonManager *buttonManager*:** Instance of buttonManager.
- **public ButtonManager():** Default constructor.
- **public static ButtonManager getInstance():** Singleton method that returns this.

4.4.2.6 FileManager

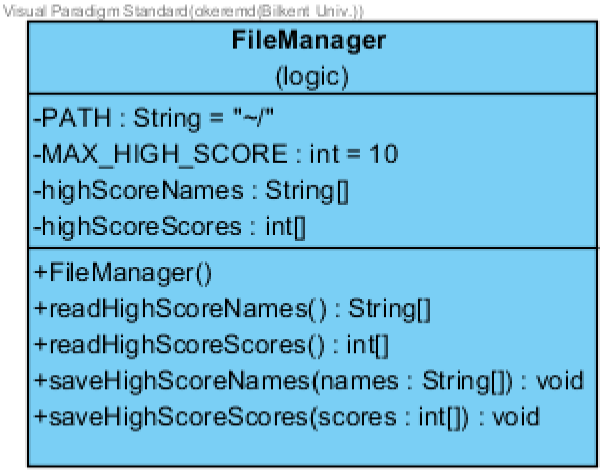

Visual Paradigm Standard(okeremd(Bilkent Univ.))
FileManager
(logic)
-PATH : String = "~/"
-MAX_HIGH_SCORE : int = 10
-highScoreNames : String[]
-highScoreScores : int[]
+FileManager()
+readHighScoreNames() : String[]
+readHighScoreScores() : int[]
+saveHighScoreNames(names : String[]) : void
+saveHighScoreScores(scores : int[]) : void

Figure 4.4.2.6: FileManager class

FileManager is responsible for loading current high score table at beginning and save updated high score table at the end of each game.

- **private final String PATH :** Holds the absolute path to the HighScores.txt file.
- **private final int MAX_HIGH_SCORE :** Holds how many high score can exist.
- **private String highScoreNames[]:** Holds the names of the each high score.
- **private int highScoreScores[]:** Holds the scores of the each high score.
- **public FileManager():** Default constructor.
- **public String[] readHighScoreNames():** Reads the name of the high scores from file.
- **public int[] readHighScoreScores():** Reads the scores of the high scores from file.

- **public void saveHighScores():** Writes the HighScores.txt file with current high scores.

4.4.2.7 PauseManager

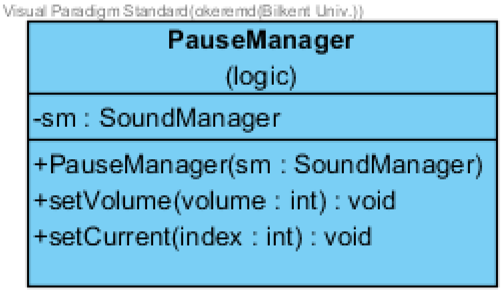

Visual Paradigm Standard(okeremd(Bilkent Univ.))
PauseManager
(logic)
-sm : SoundManager
+PauseManager(sm : SoundManager)
+setVolume(volume : int) : void
+setCurrent(index : int) : void

Figure 4.4.2.7: PauseManager class

PauseManager is responsible for the pause menu that will be shown when user presses pause button.

- **private SoundManager sm:** Holds SoundManager instance.
- **public PauseManager(SoundManager sm):** Constructor that takes SoundManager as input.
- **public void setVolume(int volume):** Changes the volume of the current music.
- **public void setCurrent(int index):** Changes the current music.

4.4.2.8 SoundManager

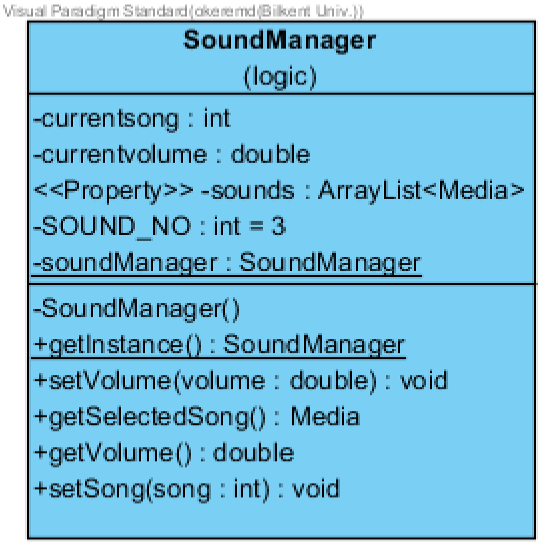

Visual Paradigm Standard(okeremd(Bilkent Univ.))
SoundManager
(logic)
-currentsong : int
-currentvolume : double
<<Property>> -sounds : ArrayList<Media>
-SOUND_NO : int = 3
-soundManager : SoundManager
-SoundManager()
+getInstance() : SoundManager
+setVolume(volume : double) : void
+getSelectedSong() : Media
+getVolume() : double
+setSong(song : int) : void

Figure 4.4.2.8: SoundManager class

SoundManager is responsible for the musics that played at game.

- **private int currentsong:** Holds the index of the currently playing song.
- **private double currentvolume:** Holds the current volume.
- **private ArrayList<Media> sounds:** Holds the songs that provided from game.
- **private final int SOUND_NO :** Holds the number of songs.

- **private static SoundManager *soundManager*:** Holds SoundManager instance.
- **private SoundManager():** Default constructor.
- **public static SoundManager getInstance() :** Singleton method that returns this.

4.4.2.9 StartManager

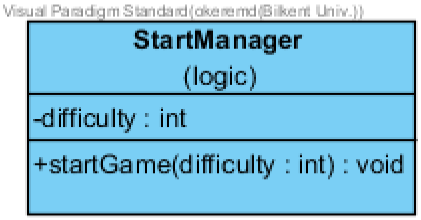

Visual Paradigm Standard(okeremd(Bilkent Univ.))
StartManager
(logic)
-difficulty : int
+startGame(difficulty : int) : void

Figure 4.4.2.9: StartManager class

StartManager is responsible for the difficulty of the game that will be selected by user.

- **private final int DIF_BUTTON_NO :** Holds the number of difficulties that game provides.
- **public void startGame(int difficulty) :** Starts the game with chosen difficulty.

4.4.3 View

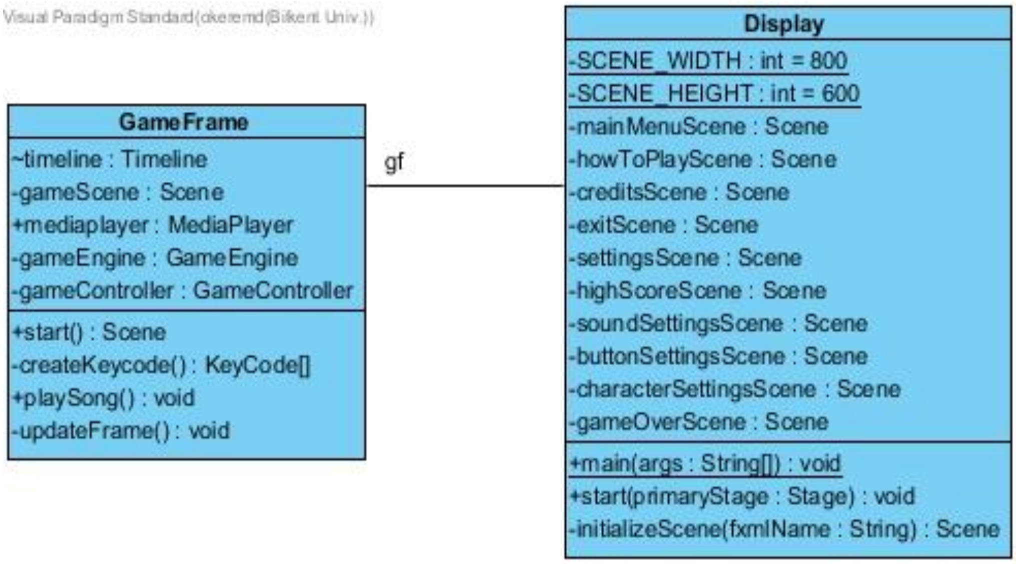

Visual Paradigm Standard(okeremd(Bilkent Univ.))
GameFrame
~timeline : Timeline
-gameScene : Scene
+mediaplayer : MediaPlayer
-gameEngine : GameEngine
-gameController : GameController
+start() : Scene
-createKeycode() : KeyCode[]
+playSong() : void
-updateFrame() : void
gf
Display
-SCENE_WIDTH : int = 800
-SCENE_HEIGHT : int = 600
-mainMenuScene : Scene
-howToPlayScene : Scene
-creditsScene : Scene
-exitScene : Scene
-settingsScene : Scene
-highScoreScene : Scene
-soundSettingsScene : Scene
-buttonSettingsScene : Scene
-characterSettingsScene : Scene
-gameOverScene : Scene
+main(args : String[]) : void
+start(primaryStage : Stage) : void
-initializeScene(fxmlName : String) : Scene

Figure 4.4.3: View Subsystem

4.4.3.1 Display

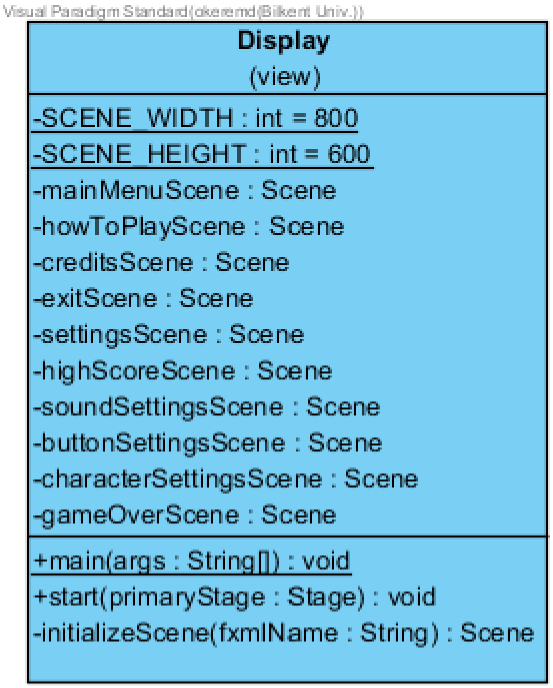

Visual Paradigm Standard(okeremd(Bilkent Univ.))
Display
(view)
-SCENE_WIDTH : int = 800
-SCENE_HEIGHT : int = 600
-mainMenuScene : Scene
-howToPlayScene : Scene
-creditsScene : Scene
-exitScene : Scene
-settingsScene : Scene
-highScoreScene : Scene
-soundSettingsScene : Scene
-buttonSettingsScene : Scene
-characterSettingsScene : Scene
-gameOverScene : Scene
+main(args : String[]) : void
+start(primaryStage : Stage) : void
-initializeScene(fxmlName : String) : Scene

Figure 4.4.3.1: Display class

- **private static final int *SCENE_WIDTH* :** Holds the width of the frame.
- **private static final int *SCENE_HEIGHT* :** Holds the height of the frame.
- **private Scene mainMenuScene:** Scene for main menu
- **private Scene howToPlayScene:** Scene for how to play
- **private Scene creditsScene:** Scene for credits
- **private Scene exitScene:** Scene for exit
- **private Scene settingsScene:** Scene for settings
- **private Scene highScoreScene:** Scene for high score
- **private Scene soundSettingsScene:** Scene for sound settings
- **private Scene buttonSettingsScene:** Scene for button settings
- **private Scene characterSettingsScene:** Scene for character settings
- **private Scene gameOverScene:** Scene for game over
- **public static void main(String[] args):** Main method
- **public void start(Stage primaryStage):** Starts JavaFX application with current stage.
- **private Scene initializeScene(String fxmlName):** Gets fxml files and puts it into Java objects.

4.4.3.2 GameFrame

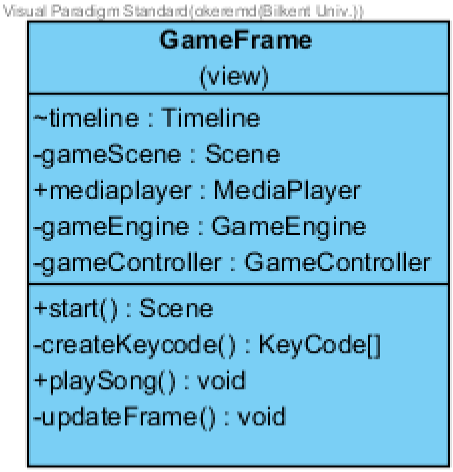

Visual Paradigm Standard(okeremd(Bilkent Univ.))
GameFrame
(view)
~timeline : Timeline
-gameScene : Scene
+mediaplayer : MediaPlayer
-gameEngine : GameEngine
-gameController : GameController
+start() : Scene
-createKeycode() : KeyCode[]
+playSong() : void
-updateFrame() : void

Figure 4.4.3.2: GameFrame class

- **private GameEngine gameEngine:** Holds the GameEngine instance.
- **private GameController gameController:** Holds the GameController instance.
- **Timeline timeline:** Holds the timeline that is used to calculate frame per second.
- **private Scene gameScene:** Holds the Scene[3] instance for game.
- **public MediaPlayer mediaplayer:** Holds the MediaPlayer instance to play songs.
- **public Scene start() :**
- **private KeyCode[] createKeycode():** Gets the chosen buttons from ButtonManager and arrange it to play
- **public void playSong():** Starts to play current song.
- **private void updateFrame():** Updates the current gameScene at each timeline call.

[3] Scene: The JavaFX Scene class is the container for all content in a scene graph, buttons texts etc.[3]

4.4.4 Controller

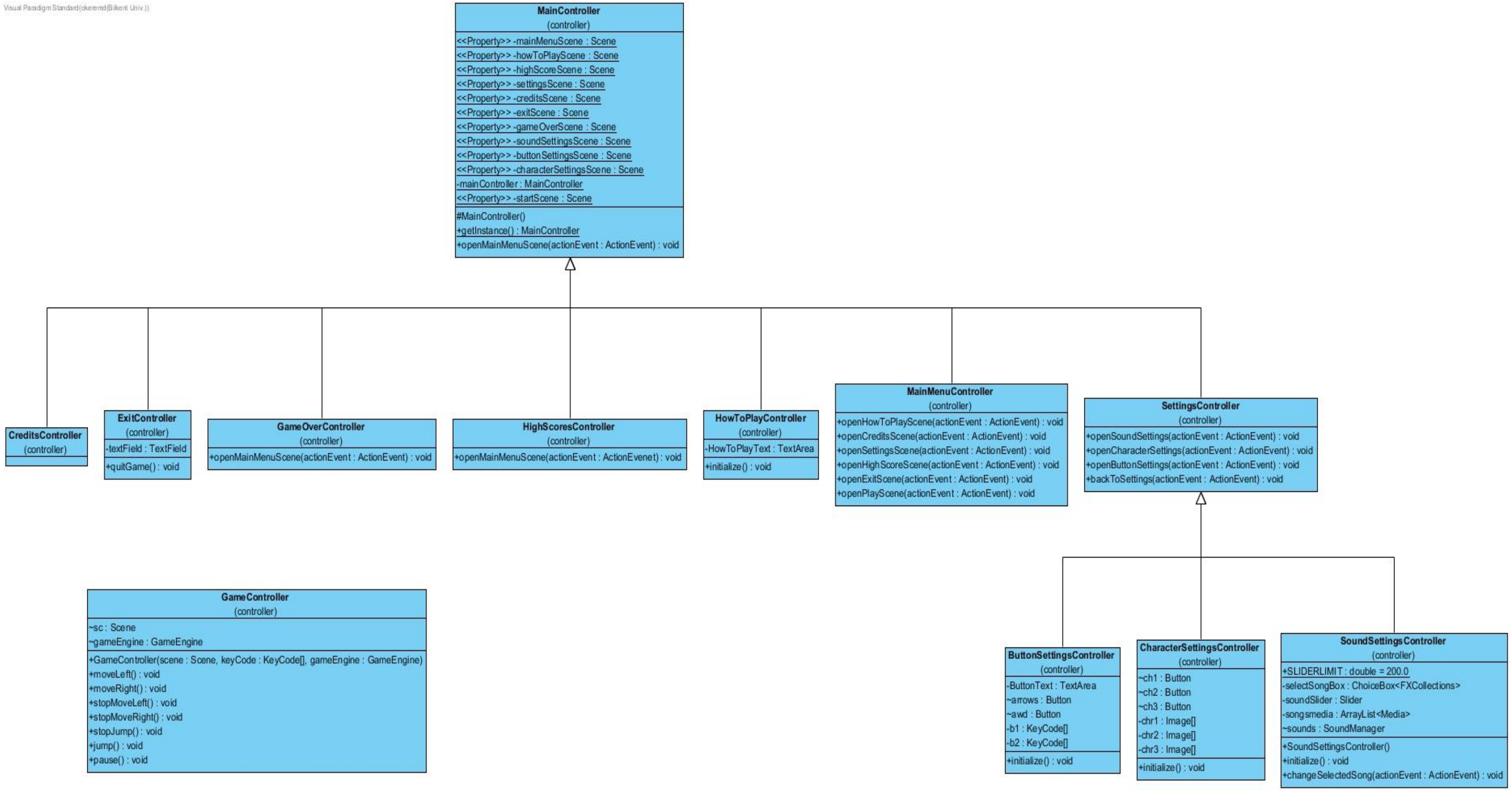

MainController
(controller)
<<Property>> -mainMenuScene : Scene
<<Property>> -howToPlayScene : Scene
<<Property>> -highScoreScene : Scene
<<Property>> -settingsScene : Scene
<<Property>> -creditsScene : Scene
<<Property>> -exitScene : Scene
<<Property>> -gameOverScene : Scene
<<Property>> -soundSettingsScene : Scene
<<Property>> -buttonSettingsScene : Scene
<<Property>> -characterSettingsScene : Scene
-mainController : MainController
<<Property>> -startScene : Scene
#MainController()
+getInstance() : MainController
+openMainMenuScene(actionEvent : ActionEvent) : void
CreditsController
(controller)
ExitController
(controller)
-textField : TextField
+quitGame() : void
GameOverController
(controller)
+openMainMenuScene(actionEvent : ActionEvent) : void
HighScoresController
(controller)
+openMainMenuScene(actionEvent : ActionEvenet) : void
HowToPlayController
(controller)
-HowToPlayText : TextArea
+initialize() : void
MainMenuController
(controller)
+openHowToPlayScene(actionEvent : ActionEvent) : void
+openCreditsScene(actionEvent : ActionEvent) : void
+openSettingsScene(actionEvent : ActionEvent) : void
+openHighScoreScene(actionEvent : ActionEvent) : void
+openExitScene(actionEvent : ActionEvent) : void
+openPlayScene(actionEvent : ActionEvent) : void
SettingsController
(controller)
+openSoundSettings(actionEvent : ActionEvent) : void
+openCharacterSettings(actionEvent : ActionEvent) : void
+openButtonSettings(actionEvent : ActionEvent) : void
+backToSettings(actionEvent : ActionEvent) : void
GameController
(controller)
~sc : Scene
~gameEngine : GameEngine
+GameController(scene : Scene, keyCode : KeyCode[], gameEngine : GameEngine)
+moveLeft() : void
+moveRight() : void
+stopMoveLeft() : void
+stopMoveRight() : void
+stopJump() : void
+jump() : void
+pause() : void
ButtonSettingsController
(controller)
-ButtonText : TextArea
~arrows : Button
~awd : Button
-b1 : KeyCode[]
-b2 : KeyCode[]
+initialize() : void
CharacterSettingsController
(controller)
~ch1 : Button
~ch2 : Button
~ch3 : Button
-chr1 : Image[]
-chr2 : Image[]
-chr3 : Image[]
+initialize() : void
SoundSettingsController
(controller)
+SLIDERLIMIT : double = 200.0
-selectSongBox : ChoiceBox<FXCollections>
-soundSlider : Slider
-songsmedia : ArrayList<Media>
~sounds : SoundManager
+SoundSettingsController()
+initialize() : void
+changeSelectedSong(actionEvent : ActionEvent) : void

Figure 4.4.4: Controller Subsystem

4.4.4.1 ButtonSettingsController

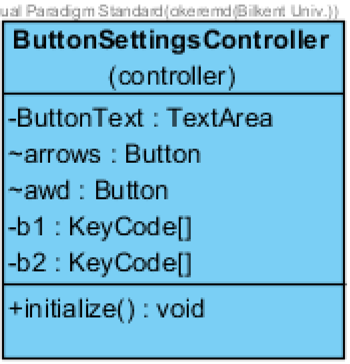


Figure 4.4.4.1: ButtonSettingsController class

- **private KeyCode[] arrowButtons:** Control buttons array for using arrows
- **private KeyCode[] awdButtons:** Control buttons array for using A-W-D
- **public void initialize():** Gets the user input and sends these to ButtonManager to arrange

4.4.4.2 CharacterSettingsController

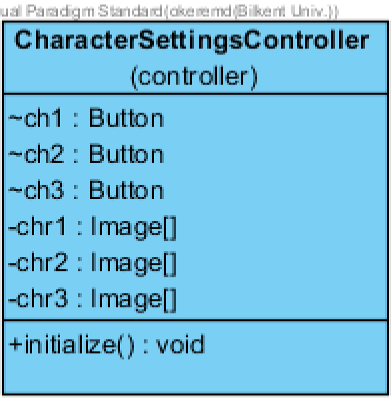


Figure 4.4.4.2: CharacterSettingsController class

- **private Image[] chr1:** First character’s image array
- **private Image[] chr2:** Second character’s image array
- **private Image[] chr3:** Third character’s image array
- **public void initialize():** Gets the user input and sends it to CharacterManager to arrange the character’s appearance

4.4.4.3 CreditsController

This class inherits the mainController() for return main menu button, and is currently empty.

4.4.4.4 ExitController

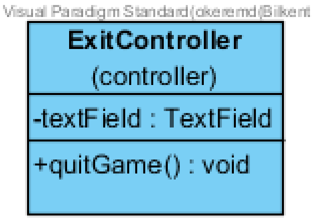


Figure 4.4.4.4: ExitController class

- **public void quitGame():** Ends the game

4.4.4.5 GameController

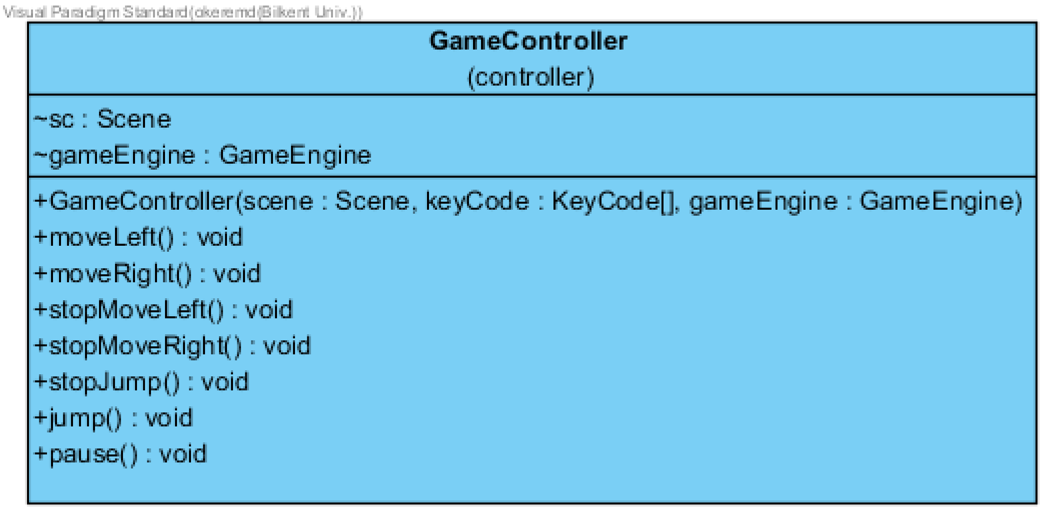


Figure 4.4.4.5: GameController class

GameController is responsible for taking the in-game input from user and pass it to GameEngine.

- **Scene sc:** Holds the current scene.
- **GameEngine gameEngine:** Holds the gameEngine that will be manipulated.
- **public GameController(Scene scene, KeyCode[] keyCode, GameEngine gameEngine):** Constructor with parameters Scene, KeyCode[] and GameEngine.
- **public void moveLeft() :** Signals the gameEngine when left button that decided by player is pressed.
- **public void moveRight():** Signals the gameEngine when right button that decided by player is pressed.
- **public void stopMoveLeft():** Signals the gameEngine when left button that decided by player is released.
- **public void stopMoveRight():** Signals the gameEngine when right button that decided by player is released.

- **public void stopJump():** Signals the gameEngine when jump button that decided by player is released.
- **public void jump() :** Signals the gameEngine when jump button that decided by player is pressed.
- **public void pause():** Signals the gameEngine when pause button pressed.

4.4.4.6 HighScoresController

This class inherits the mainController() for return main menu button, and is currently empty.

4.4.4.7 HowToPlayController

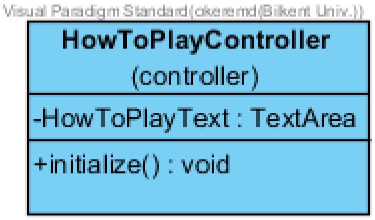

Visual Paradigm Standard(okeremd(Bilkent Univ.))
HowToPlayController
(controller)
-HowToPlayText : TextArea
+initialize() : void

Figure 4.4.4.7: HowToPlayController class

- **public void initialize() :** Sets the priorities of objects at the FXML file

4.4.4.8 MainController

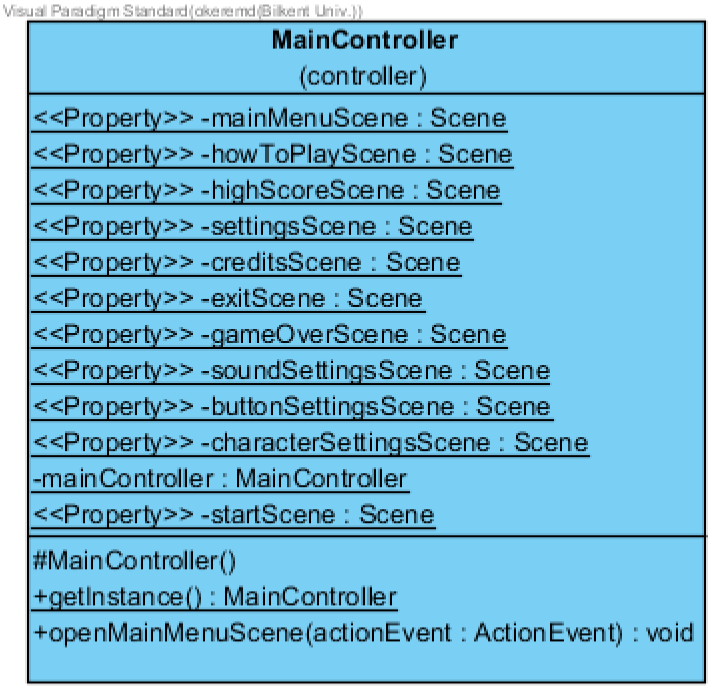

Visual Paradigm Standard(okeremd(Bilkent Univ.))
MainController
(controller)
<<Property>> -mainMenuScene : Scene
<<Property>> -howToPlayScene : Scene
<<Property>> -highScoreScene : Scene
<<Property>> -settingsScene : Scene
<<Property>> -creditsScene : Scene
<<Property>> -exitScene : Scene
<<Property>> -gameOverScene : Scene
<<Property>> -soundSettingsScene : Scene
<<Property>> -buttonSettingsScene : Scene
<<Property>> -characterSettingsScene : Scene
-mainController : MainController
<<Property>> -startScene : Scene
#MainController()
+getInstance() : MainController
+openMainMenuScene(actionEvent : ActionEvent) : void

Figure 4.4.4.8: MainController class

- **private static Scene *mainMenuScene*:** Scene for main menu
- **private static MainController *mainController*:** The single instance of mainController
- **protected MainController():** The singleton constructor for mainController
- **public static MainController getInstance():** Returns Singleton MainController object.

4.4.4.9 MainMenuController

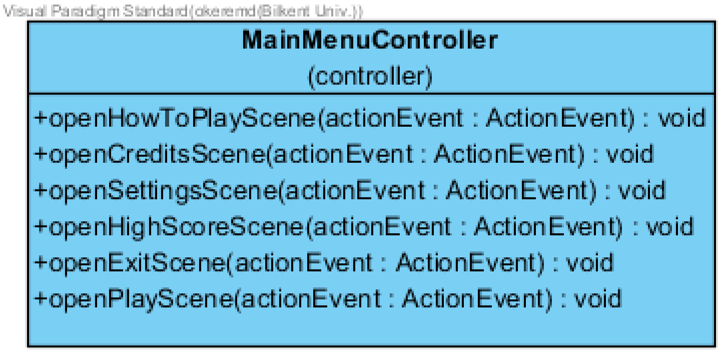

Visual Paradigm Standard(okeremd(Bilkent Univ.))
MainMenuController
(controller)
+openHowToPlayScene(actionEvent : ActionEvent) : void
+openCreditsScene(actionEvent : ActionEvent) : void
+openSettingsScene(actionEvent : ActionEvent) : void
+openHighScoreScene(actionEvent : ActionEvent) : void
+openExitScene(actionEvent : ActionEvent) : void
+openPlayScene(actionEvent : ActionEvent) : void

Figure 4.4.4.9: MainMenuController class

- **public void openHowToPlayScene(ActionEvent actionEvent) :** Opens how to play scene
- **public void openCreditsScene(ActionEvent actionEvent):** Opens credits scene
- **public void openSettingsScene(ActionEvent actionEvent):** Opens settings scene
- **public void openHighScoreScene(ActionEvent actionEvent):** Opens high score scene
- **public void openExitScene(ActionEvent actionEvent):** Opens exit scene
- **public void openPlayScene(ActionEvent actionEvent):** Opens game

4.4.4.10 SettingsController

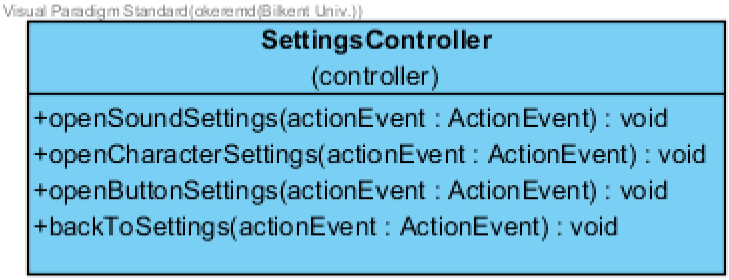

Visual Paradigm Standard(okeremd(Bilkent Univ.))
SettingsController
(controller)
+openSoundSettings(actionEvent : ActionEvent) : void
+openCharacterSettings(actionEvent : ActionEvent) : void
+openButtonSettings(actionEvent : ActionEvent) : void
+backToSettings(actionEvent : ActionEvent) : void

Figure 4.4.4.10: SettingsController class

- **public void openSoundSettings(ActionEvent actionEvent):** Opens sound setting screen
- **public void openCharacterSettings(ActionEvent actionEvent):** Opens character setting screen
- **public void openButtonSettings(ActionEvent actionEvent):** Opens button setting screen

- **public void backToSettings(ActionEvent actionEvent):** Opens setting screen back

4.4.4.11 SoundSettingsController

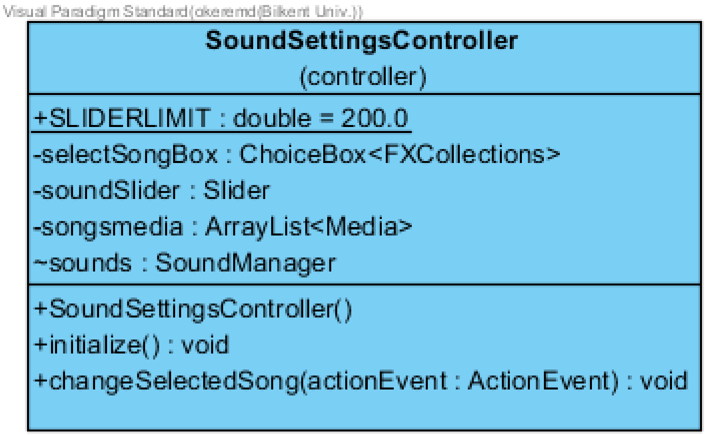

Visual Paradigm Standard(okeremd(Bilkent Univ.))
SoundSettingsController
(controller)
+SLIDERLIMIT : double = 200.0
-selectSongBox : ChoiceBox<FXCollections>
-soundSlider : Slider
-songsmedia : ArrayList<Media>
~sounds : SoundManager
+SoundSettingsController()
+initialize() : void
+changeSelectedSong(actionEvent : ActionEvent) : void

Figure 4.4.4.11: SoundSettingsController class

- **public static final double *SLIDERLIMIT:*** Max limit for volume slider
- **private ChoiceBox<FXCollections> selectSongBox:** ChoiceBox for user's selecting song
- **private Slider soundSlider:** Slider for volume
- **public SoundSettingsController():** Default constructor
- **public void initialize() :** Adjusts the initial value of volume slider
- **public void changeSelectedSong(ActionEvent actionEvent):** Sends the selected song which is from choice box to the SoundManager to arrange

4.4.4.12 StartController

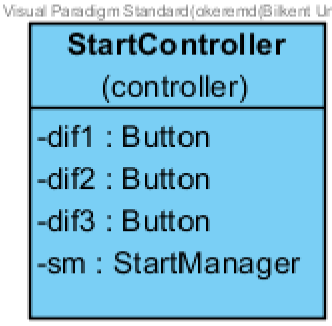


Figure 4.4.4.12: StartController class

- **Button dif1:** Easy Difficulty button.
- **Button dif2:** Medium Difficulty button.
- **Button dif3:** Hard Difficulty button.
- **StartManager sm:** StartManager instance that starts the game based on difficulty.

5. Improvement summary

Improvements of Design

Because of the changes on the Analysis Report, our object design is undergone major changes. Object Class Diagram is reconstructed to handle new additional functionalities.

We brought new design patterns to handle reusing issues. We decided not to use the Composite Design Pattern because it is not compatible with our implementation. We added two new design patterns which are Singleton Pattern and Strategy Pattern. We used Singleton to restrict the system to use only one instance of the manager classes. We used the Strategy Pattern to implement additional bar functionalities.

We looked through our prior design report and its feedback, and fixed previous problems about the document.

Improvements of Implementation

We made some arrangements for having much cleaner and readable code, we refactored some of the classes and methods. We also removed some unnecessary components in the code. We implemented new design patterns for more reusable code.

6. Glossary & references

[1] Bruegge. Object-Oriented Software Engineering.

[2]"Mastering FXML: About This Tutorial | JavaFX 2 Tutorials and Documentation". [Online].

Available: https://docs.oracle.com/javafx/2/fxml_get_started/jfxpub-fxml_get_started.htm.

[3] *Scene (JavaFX 8)*, 10 Feb. 2015,

docs.oracle.com/javase/8/javafx/api/javafx/scene/Scene.html.