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

Table of 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, traps, 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

Our main aim is to provide a user-friendly and entertaining game. Thus, we discoursed on some non-functional requirements such that usability, maintainability, extensibility, understandability, performance, reliability and portability.

1.2.1. Usability

Icy Tower will be easily understandable and usable game for all people with different ages and different educational levels. Menu is being in the first place, all pages in the game will be simple and easy to understand. By looking the “How to Play” screen, the logic of the game, directions, hints etc. can be easily understand by the users.

1.2.2. Performance

The game performance is one of our main concerns about the game. The game is expected to run at 30 frame per second, without any flickering. The game will run without

any freeze 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 a 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

We are promising that no major bugs will be in the game unexpectedly closes, or user is being stuck on a place or a game freeze. We care about robustness for a better gaming experience to users.

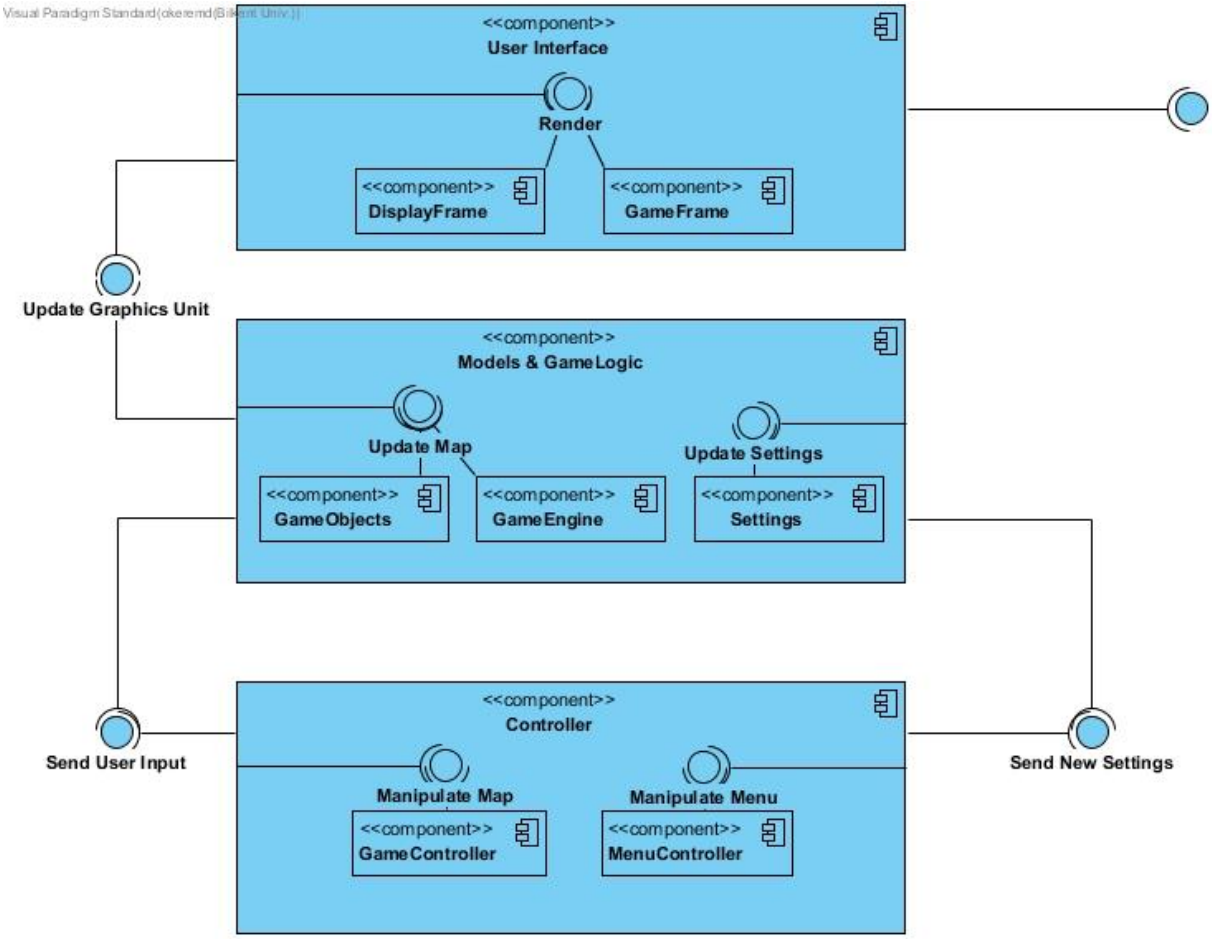
1.2.5. Portability

The game is currently being developed for Desktop. However, since our game will be implemented in Java, Icy Tower will be a portable game and can be played in different devices, it can easily be converted to be executable on Android/IOS Platforms.

2. Software Architecture

In this section, design choices and the general architecture of the system will be explained. There will be explanations about subsystems of the software, hardware/software mapping data management, access controls and boundary conditions.

2.1. Subsystem Decomposition



The Icy Tower system is decomposed into three parts. These are, *User Interface*, *Controller* and *Models & GameLogic*. The system adopts a MVC(Model-View-Controller) architecture. The *view* part will be handled by the *User Interface* subsystem. This subsystem consists of frames that renders the display . It has two subcomponents, *DisplayFrame* and *GameFrame*. *DisplayFrame* is for menu screens of the game and the *GameFrame* is for

rendering game graphics. This frames are takes inputs from the user, and sends it to the *Controller* subsystem, they are updated by the *Model* part of the system. We use the *Controller* as a connection between view and the model. It takes input from the view, and manipulates the model according to the command the user entered. The model on our subsystem is our *Models & GameLogic* subsystem. It contains all entity objects and also the game logic. During gametime, user inputs are evaluated by the *GameEngine* instance, and *GameEngine* is responsible for changing the GameObject instances. After the entity objects are changed, they are notifies the related view class, *GameFrame* or *DisplayFrame*.

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.

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 keep by the game. Both these will be kept in separate text files and will be no larger than 4KB.

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

2.5.1. Initialization

Game does not need any initialization requirements. 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 will be selected.

2.5.2. Termination

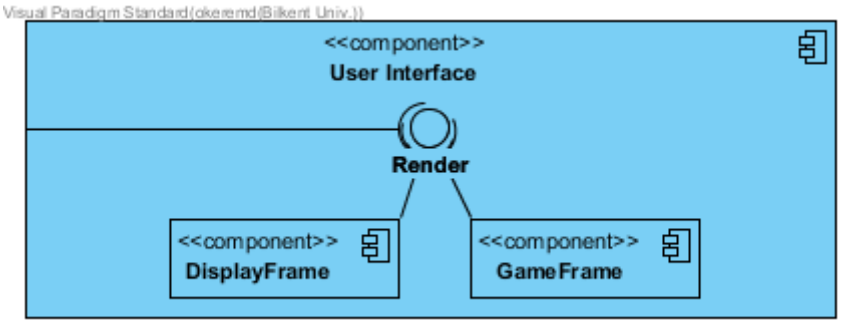
User can quit any time from the game, by pausing the game and then selecting the quit game. He can also go to main menu from there. The user can also close the application window to quit the game.

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

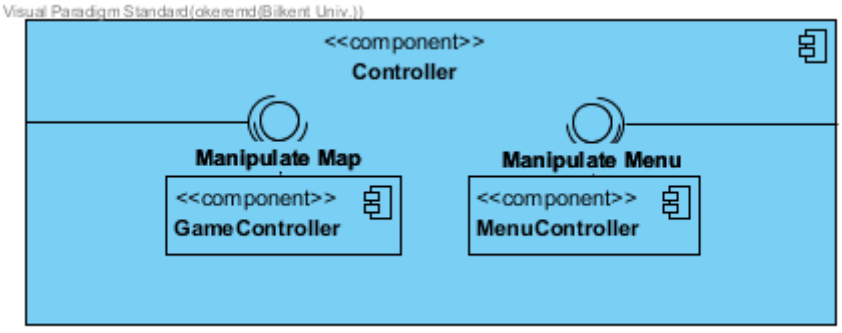
3.1. User Interface Subsystem



User Interface:

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.

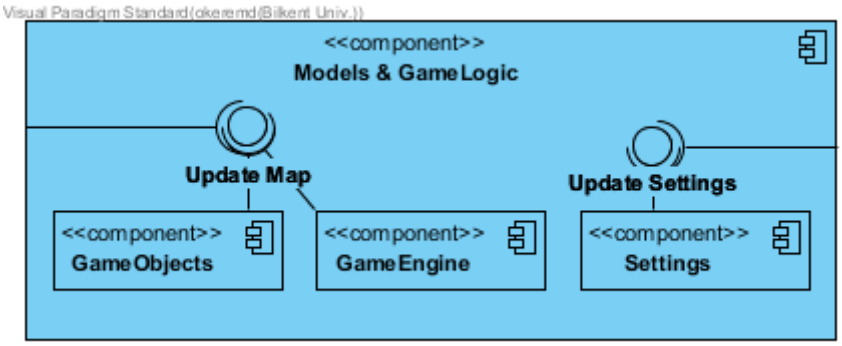
3.2 Controller Subsystem



Controller:

This part is 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 *Models & GameLogic* subsystem according to commands.The *Models & GameLogic* subsystem is responsible for determining what actions will be done by the system according to game logic.

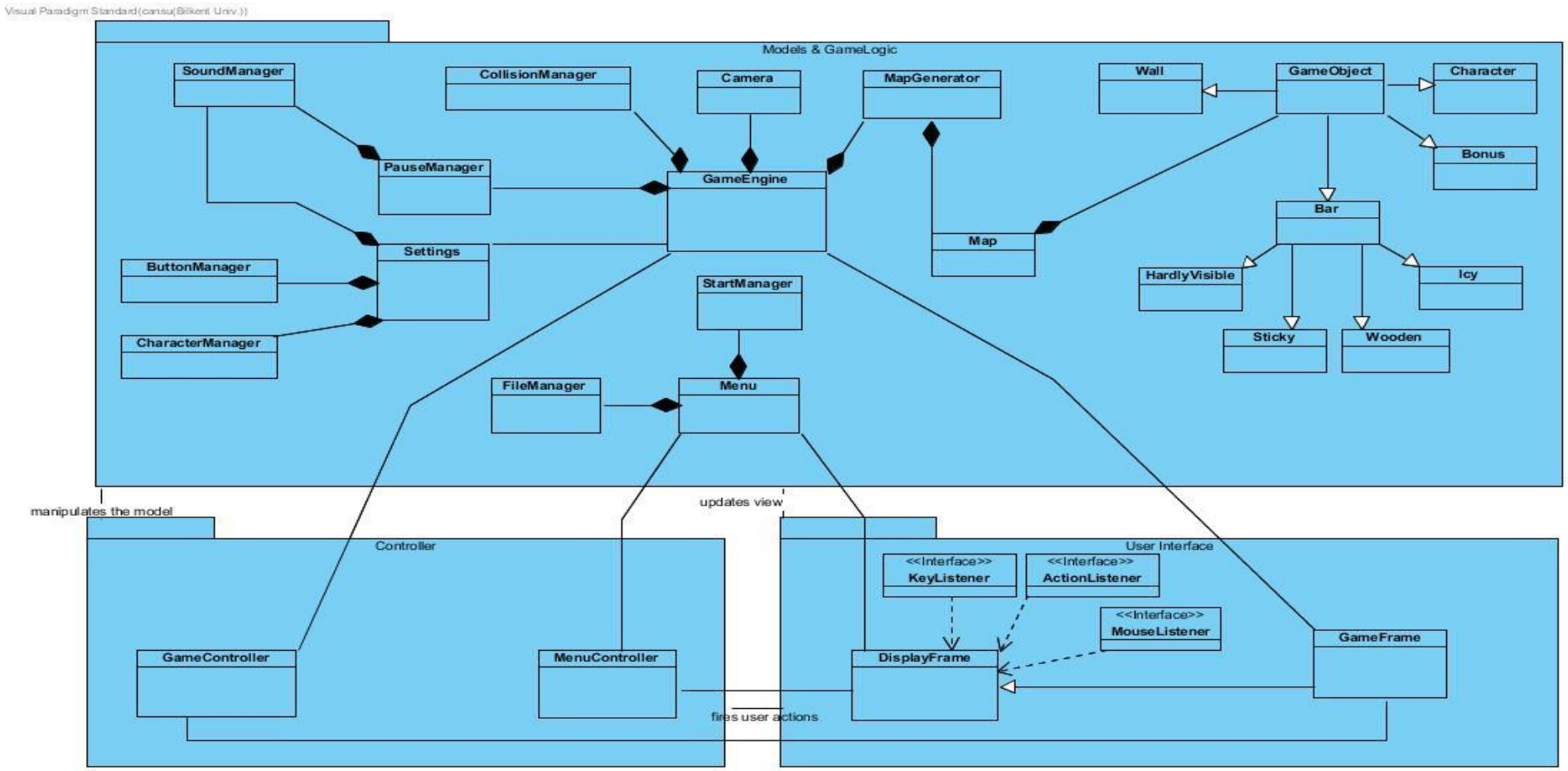
3.3 Models & Game Logic Subsystem



Models & 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 Models&GameLogic system corresponds 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.

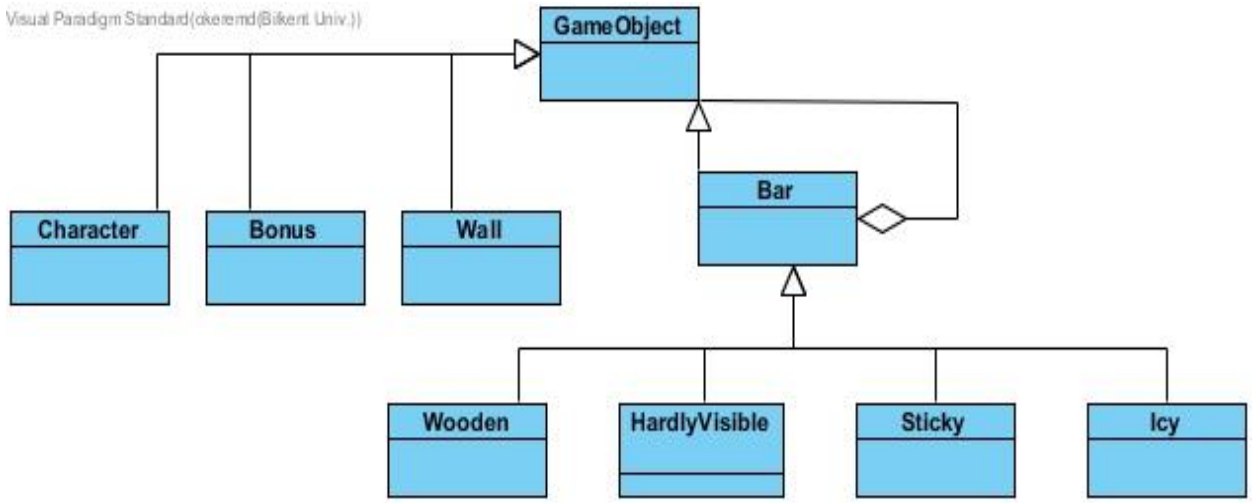
4. Low-level design



Design Patterns:

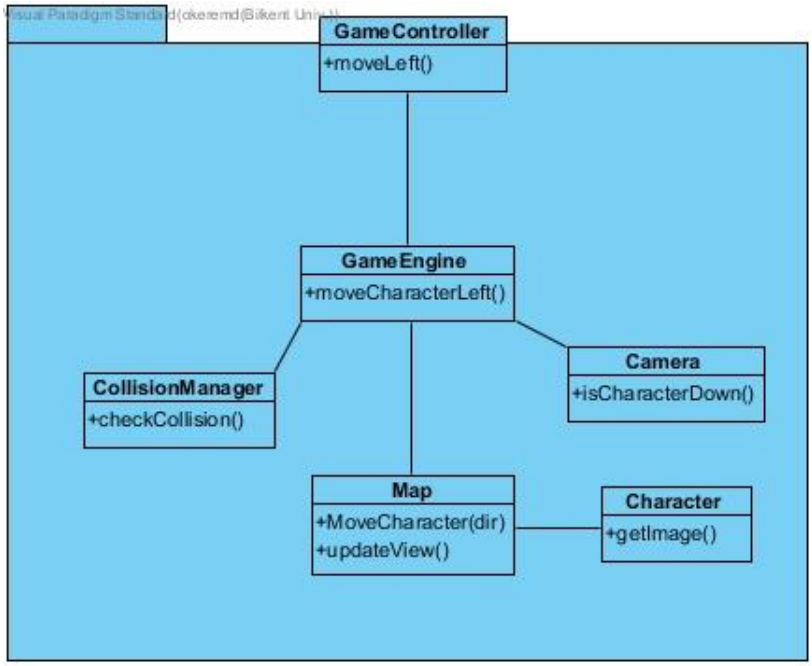
Composite Design Pattern:

Our GameObject class and it's hierarchical design is inspired by composite design pattern. Each class that is inherited by the GameObject can be treated uniformly. This eases to move the objects during the game, because each object can be considered separately. Here each GameObject instance has some common methods, but they get different meanings on that specific instance (e.g Bar).



Facade Design Pattern:

We use facade between our Controller and Model. Controller is giving high-level commands to Model, but the Logic part of the Model evaluates it by low-level components and generate a meaningful output.



4.1. Object design trade-offs

4.1.1 Memory vs Performance

We care more about the performance than the memory. The game does not need a high amount storage, it is expected to be at most 40 MB. The memory usage of the game on the other hand, may be a little high for such a minigame. But for a better performance we need it, so the game will be able to run at high speed without any freeze or flickering.

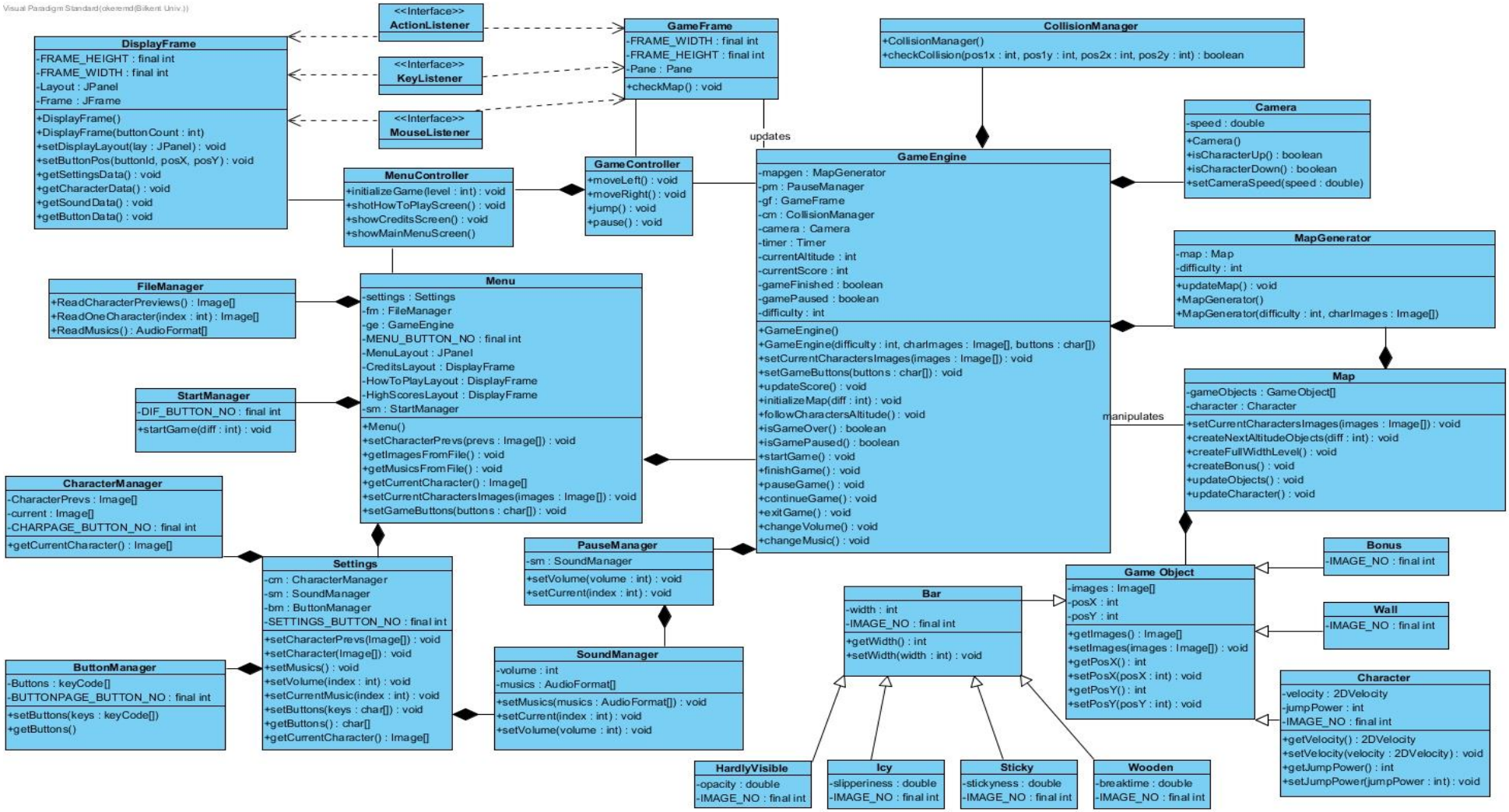
4.1.2 Functionality vs Usability

Since what we are want to do is just a entertaining game, we try to make it a user friendly software. The user will don't get confused by a lot of options, he may just click to the play game and enjoy it. Although the game in fact has a complex algorithm in it and hard to implement, what user will see it will be very understandable for her.

4.1.3 Efficiency vs Portability

We are currently developing our game only in Desktop, however it may be easily ported to smartphone devices. But we don't think it will create any efficiency problems for our game since we will try to make the game as optimizable as possible.

4.2. Final object design



4.3. Packages

4.3.1. 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. javafx.scene.layout

This package provides User Interface Layouts.

4.3.3. 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.

4.3.4. javafx.scene.paint

This package is for providing the set of classes for colors and gradients and handle visual outputs.

4.3.5. 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

4.3.6. javafx.scene.input

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

4.3.7. javafx.animations

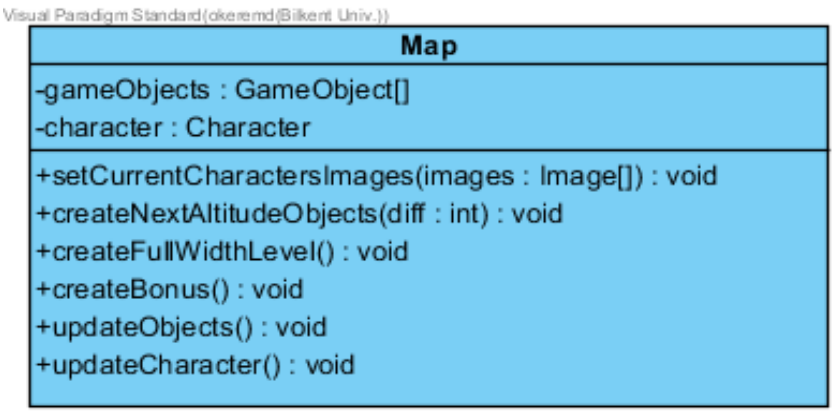
Because of Icy Tower is being an animated game, we will use this package for moving images.

4.3.8. 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.

4.4. Class Interfaces

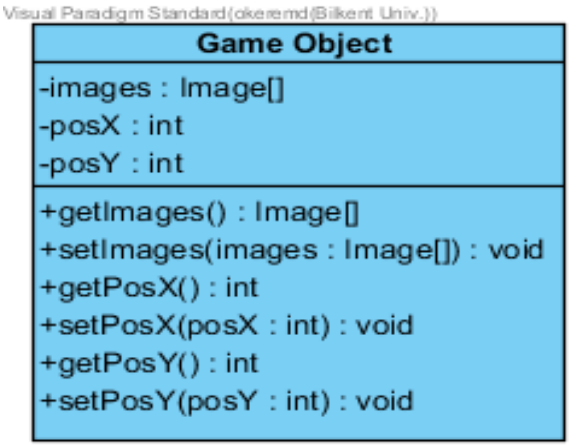
4.4.1 Map



- **private GameObject[] gameObjects:** Game object array that holds every object in the map
- **private Character character:** Character object that will be controlled by player in game.

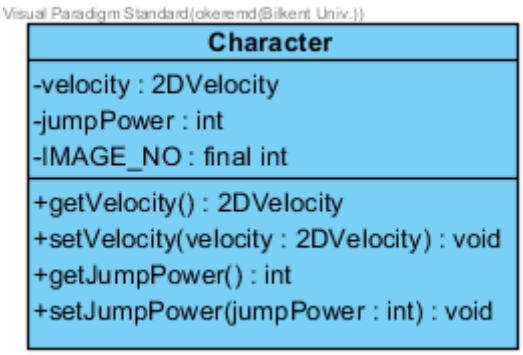
- **public void setCurrentCharactersImages(images: Image[]):** Due to the player's character choice, the appereance of the chosen character is set to the character object that will be played with
- **public void createNextAltitudeObjects(diff:int):** When user goes up, this method creates new objects for upper platforms.
- **public void createFullWidthLevel():** There will be levels that occurs periodically which are extends towards to walls. This method will decide when this levels will be created.
- **public void createBonus():** Creates bonuses and traps at certain intervals
- **public void updateObjects():** Updates all the objects in the map
- **public void updateCharacter():** Updates the character due to the player's choice

4.4.2 GameObject



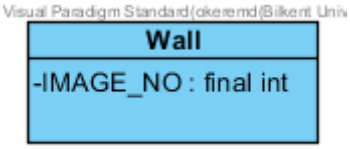
- **private Image[] images:** Image array that holds the images of the objects.
- **private int posX:** Holds the width value
- **private int posY:** Holds the height value

4.4.3 Character



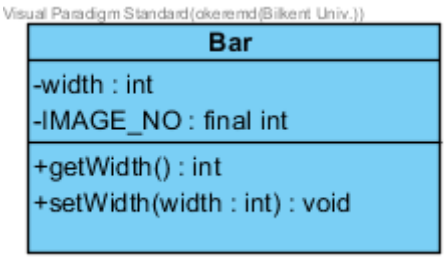
- **private 2DVelocity velocity:** The speed of character on the 2DPlane will be hold by this variable.
- **private int jumpPower:** This attribute will hold how high player can jump.
- **private final int IMAGE_NO:** This attribute will hold how many images are necessary to create the object in frame.

4.4.4 Wall



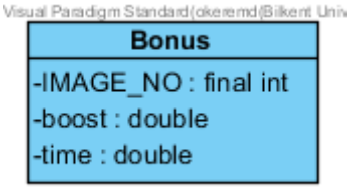
- **private final int IMAGE_NO:** This attribute will hold how many images are necessary to create the object in frame.

4.4.5 Bar



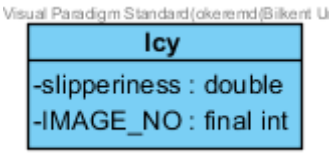
- **private final int IMAGE_NO:** This attribute will hold how many images are necessary to create the object in frame.
- **private int width:** Holds the width value of the bars

4.4.6 Bonus



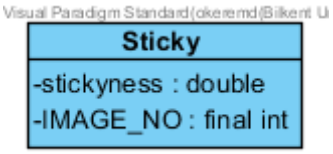
- **private final int IMAGE_NO:** This attribute will hold how many images are necessary to create the object in frame.
- **private double boost:** This attribute holds the amount of speed that bonus will give to character.
- **private double time:** This attribute holds the amount of time that bonus will applied to the character.

4.4.7 Icy



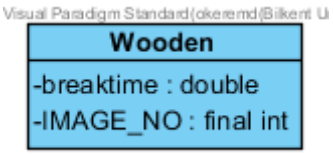
- **private double slipperiness:** The rate of slipperiness of the bar. On the icy bar, the character can accelerate and it may be hard to stop.
- **private final int IMAGE_NO:** This attribute will hold how many images are necessary to create the object in frame.

4.4.8 Sticky



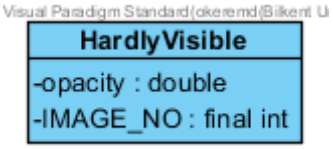
- **private double stickiness:** The rate of stickiness of the bar. Sticky bar makes the character moves slow down.
- **private final int IMAGE_NO:** This attribute will hold how many images are necessary to create the object in frame.

4.4.9 Wooden



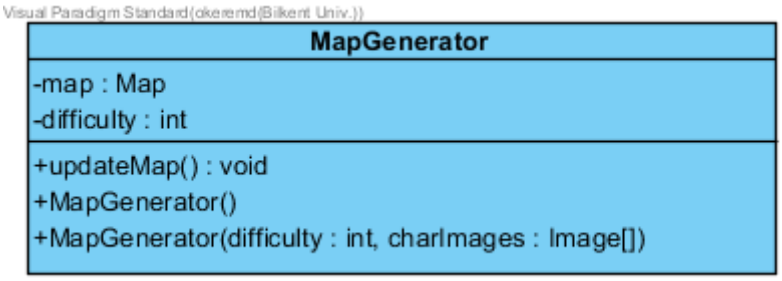
- **private double breaktime:** Wooden bar can be broken even though the character is on it.
- **private final int IMAGE_NO:** This attribute will hold how many images are necessary to create the object in frame.

4.4.10 HardlyVisible



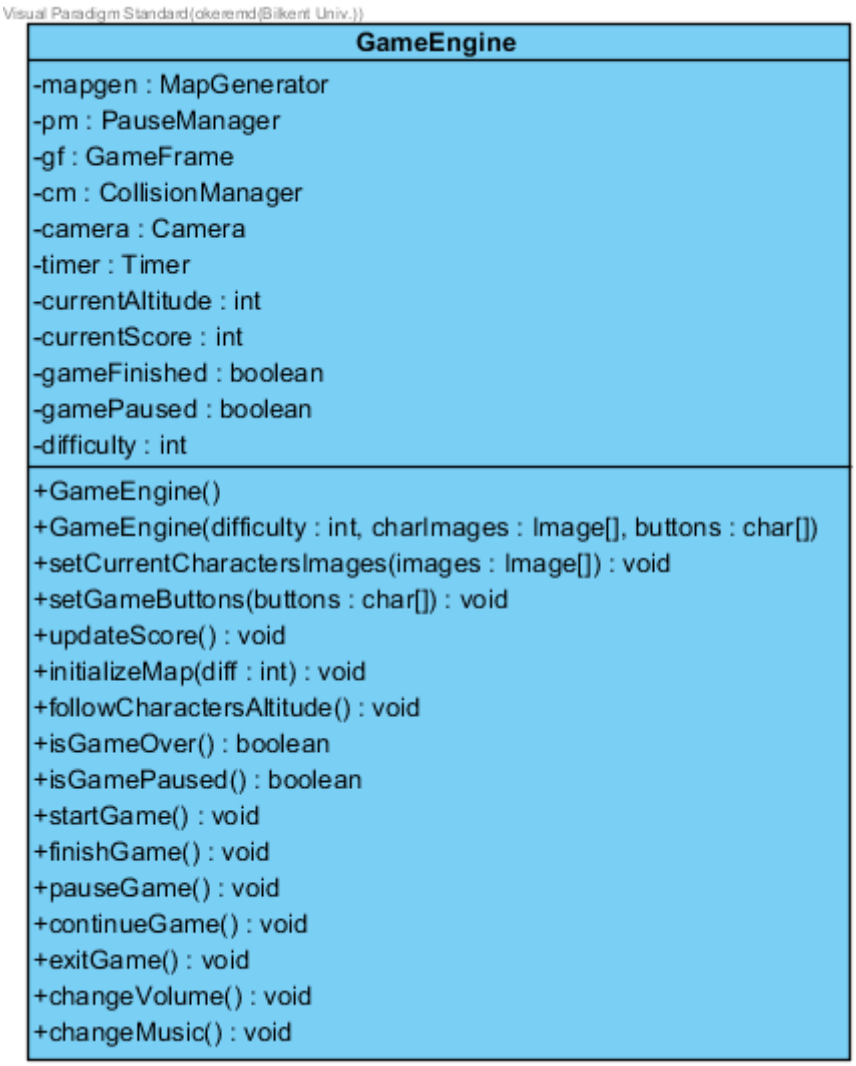
- **private double opacity:** HardlyVisible bar will be opaque so that it will hard to see by the players.
- **private final int IMAGE_NO:** This attribute will hold how many images are necessary to create the object in frame.

4.4.11 MapGenerator



- **private Map map:** Map object that represents the game area
- **private int difficulty:** Chosen difficulty level (easy/medium/hard) by the player
- **public MapGenerator():** Default constructor for the MapGenerator class
- **public MapGenerator(difficulty: int, charImages: Image[]):** Constructor that takes the difficulty level and character image chosen by the player as a parameter
- **public void updateMap():** Updates the map according to the character's current movements

4.4.12 GameEngine

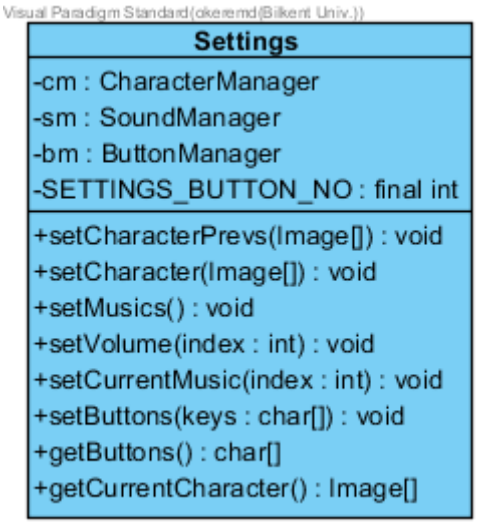


- **private MapGenerator mapgen:** MapGenerator instance in order to update and initialize map.
- **private PauseManager pm:** PauseManager instance in order to display pause menu.
- **private GameFrame gf:** GameFrame instance in order to send current orientation of the map.
- **private CollisionManager cm:** CollisionManager instance for checking validity of character's movement.

- **private Camera camera:** Camera instance for showing the correct area of the map to player.
- **private Timer timer:** Timer instance for handling fps, and randoms.
- **private int currentAltitude:** This attribute counts the levels passed by the character.
- **private int currentScore:** This attribute holds the score of the character.
- **private boolean gameFinished:** This attribute holds the status of the game.
- **private boolean gamePaused:** This attribute holds the pause status of the game.
- **private int difficulty:** This attribute holds the difficulty level of the game.
- **public GameEngine():** Default constructor for the GameEngine class.
- **public GameEngine(difficulty: int, charImages: Image[], buttons: char[]):** GameEngine class constructor which takes the difficulty level, character image and buttons that are chosen by the user as a parameter
- **public void setCurrentCharactersImages(images: Image[]):** By the player's character choice, the appereance of the chosen character is set to the character object that will be played with
- **public void setGameButtons(buttons: char[]):** According to the player's button choice, these buttons are set to the buttons object that will be played with
- **public void updateScore():** Updates the player's score
- **public void initializeMap(diff: int):** By taking the difficulty level as a parameter, it prepares and initialize the map which is compatible with the chosen difficulty
- **public void followCharactersAltitude():** This method determines the altitude of the character.
- **public boolean isGameOver():** When the game is over, this method returns true

- **public boolean isGamePaused():** When the game is paused, this method returns true.
- **public void startGame():** This method starts the game.
- **public void finishGame():** This method finishes the game.
- **public void pauseGame():** This method pauses the game.
- **public void continueGame():** This method is for returning from paused game.
- **public void exitGame():** This method exits to main menu.
- **public void changeVolume():** This method changes the level of volume.
- **public void changeMusic():** This method changes the currently playing music.

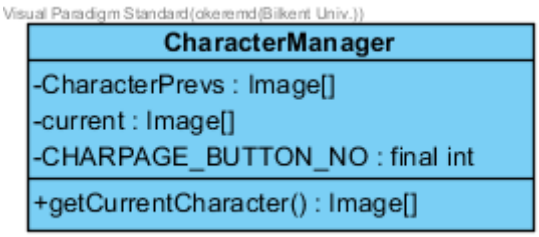
4.4.13 Settings



- **private CharacterManager cm:** CharacterManager instance in Settings.
- **private SoundManager sm:** SoundManager instance in Settings.
- **private ButtonManager bm:** ButtonManager instance in Settings.
- **private final int SETTINGS_BUTTON_NO:** This constant holds how many frames will be shown on frame.

- **public void setCharacterPrevs(Image[]):** This method changes the preview of the playable characters.
- **public void setCharacter(Image[]):** This method changes the selected character to play the game.
- **public void setMusics():** This method initializes the musics.
- **public void setVolume(index:int):** This method changes the level of the volume.
- **public void setCurrentMusic(index: int):** This method changes the currently playing music.
- **public void setButtons(keys: char[]):** This method changes the ingame buttons of the game.
- **public char[] getButtons():** This method returns the current ingame buttons.
- **public Image[] getCurrentCharacter():** This method returns the current Character's images.

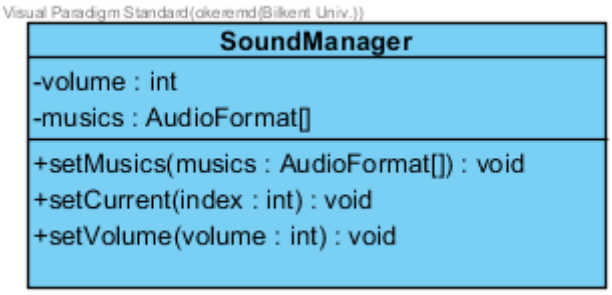
4.4.14 CharacterManager



- **private Image[] characterPrevs:** This attribute holds the preview images of the all selected characters.
- **private Image[] current:** This attribute holds the character images selected by the user. If user did not selected any character, this will hold default character's images.

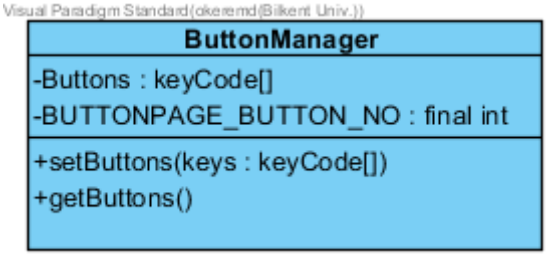
- **private final int CHARPAGE_BUTTON_NO:** This constant will hold the button number will be shown on the CharacterManager's frame.
- **public Image[] getCurrentCharacter():** This method will return the images of the current character.

4.4.15 SoundManager



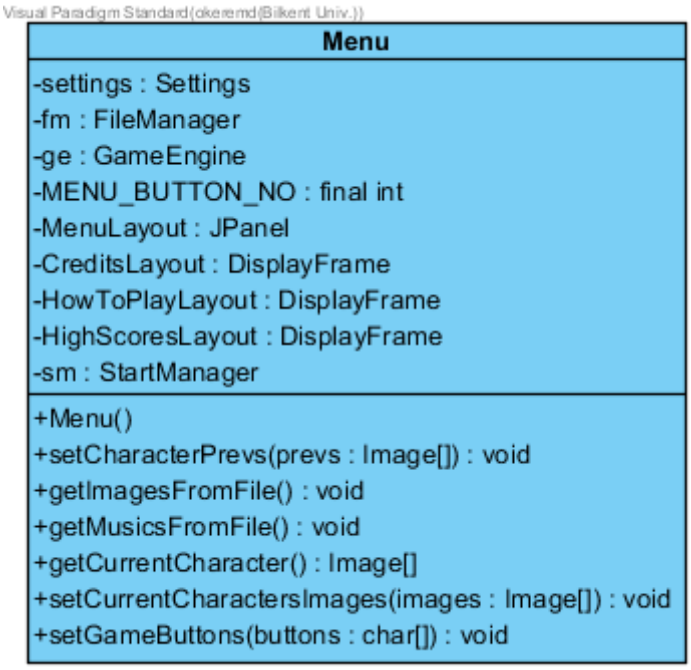
- **private int volume:** This attribute will hold the level of the volume.
- **private AudioFormat[] musics:** This attribute will hold the playable musics in the game.
- **public void setMusics(musics: AudioFormat[]):** This method changes the playable musics in the game.
- **public void setCurrent(index: int):** This method changes the current playing music.
- **public void setVolume(volume: int):** This method changes the volume.

4.4.16 ButtonManager



- **private keyCode[] buttons:** This attribute holds the buttons to control the character.
- **private final int BUTTONPAGE_BUTTON_NO:** This constant will hold the button number will be shown on the ButtonManager's frame.

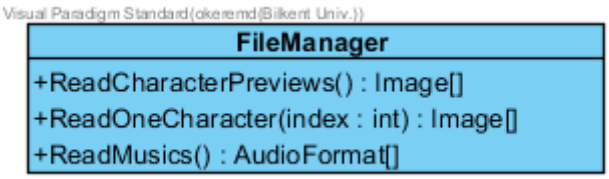
4.4.17 Menu



- **private Settings settings:** Settings instance in Menu.
- **private FileManager fm:** FileManager instance in Menu.
- **private GameEngine ge:** GameEngine instance in Menu.
- **private final int MENU_BUTTON_NO:** This constant will hold the button number will be shown on the Menu's frame.
- **private JPanel MenuLayout:** This attribute holds the layout of Menu screen.
- **private DisplayFrame CreditsLayout:** This attribute holds the layout of Credits screen.

- **private DisplayFrame HowToPlayLayout:** This attribute holds the layout of How To Play screen.
- **private DisplayFrame HighScoresLayout:** This attribute holds the layout of High Scores screen.
- **private StartMenu sm:** StartMenu instance in Menu.
- **Menu():** Default constructor.
- **public void setCharacterPrevs(prevs: Image[]):** This method changes the preview of the character's images.
- **public void getImagesFromFile():** This method gets the all game related images from the file.
- **public void getMusicsFromFile():** This method gets the musics from the file.
- **public Image[] getCurrentCharacter():** This method returns the all images of the current character.
- **public void setCurrentCharacterImages(images: Image[]):** This method changes the current character's images.
- **public void setGameButtons(buttons: char[]):** This method changes the ingame buttons.

4.4.18 FileManager



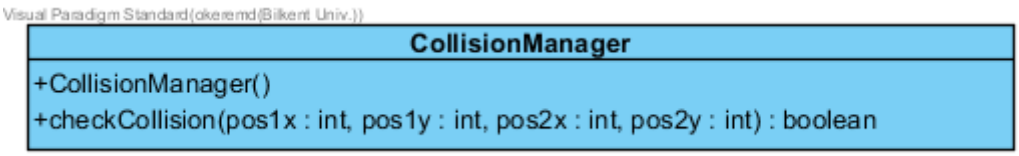
- **public Image[] ReadCharacterPreviews():** This method returns the preview of the all characters by first reading them from file.

- **public Image[] ReadOneCharacter(index : int):** This method returns the all images of one character by first reading them from file.
- **public AudioFormat[] ReadMusics():** This method returns the all musics in game folder.

4.4.19 StartManager

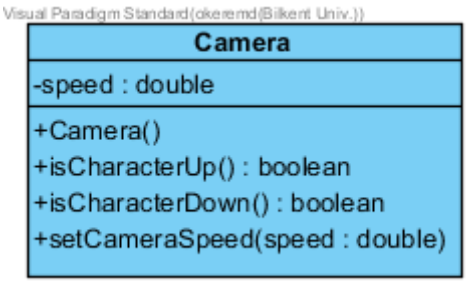
- **private final int DIF_BUTTON_NO:** This constant holds the button count in the StartManager's frame.
- **public void startGame(diff:int):** This method starts the game after taking the difficulty value.

4.4.20 CollisionManager



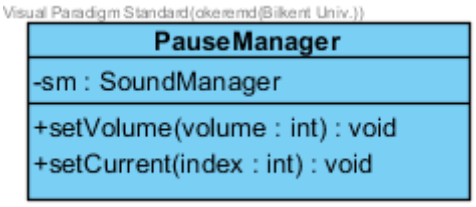
- **public CollisionManager():** Default constructor of the CollisionManager.
- **public boolean checkCollision(pos1x:int, pos1y:int, pos2x:int, pos2y:int):** This method returns true if two objects are colliding.

4.4.21 Camera



- **private double speed:** This attribute holds the vertical speed of camera since there will be no horizontal speed. Also speed will never be negative since the camera will never move down.
- **public Camera():** The default constructor of the camera.
- **public boolean isCharacterUp():** This method will return true if the speed of character exceeds the speed of camera.
- **public boolean isCharacterDown():** This method will return true if the camera is not containing character due to character is down, which is the ending case of the game.
- **public void setCameraSpeed(speed: double):** This method changes the speed of the camera.

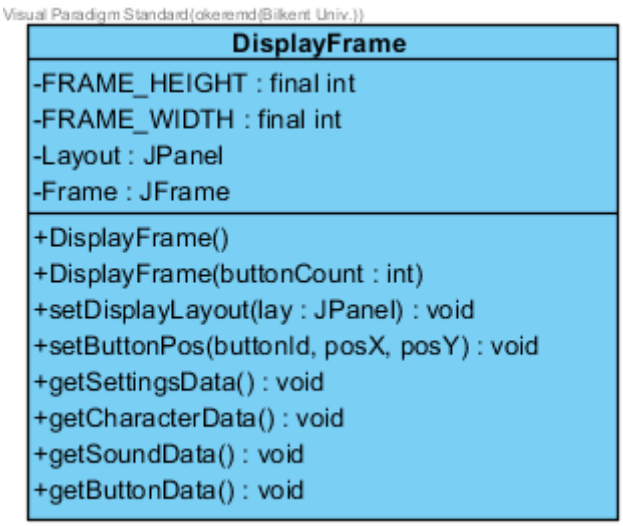
4.4.22 PauseManager



- **private SoundManager sm:** SoundManager instance in PauseManager.
- **public void setVolume(volume: int):** This method changes the level of the volume.

- **public void setCurrent(index: int):** This method changes the currently playing music.

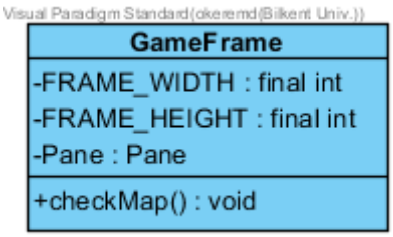
4.4.23. Display Frame



- **private final int FRAME_HEIGHT :** This constant will be determine the preferred height of the frame.
- **private final int FRAME_WIDTH :** This constant will be determine the preferred width of the frame.
- **private JPanel layout:** This attribute holds the current layout of the frame.
- **private JFrame frame:** This attribute holds the current shown frame.
- **public DisplayFrame():** Default constructor for the DisplayFrame class. It creates an empty frame.
- **public DisplayFrame(buttonCount: int):** Constructor with buttonCount. It creates the amount of buttons on the page from model class.
- **public void setDisplayLayout(lay: JPanel):** This method changes the layout to lay.
- **public void setButtonPos(buttonId, posX, posY):** This method can change the buttons position on layout.

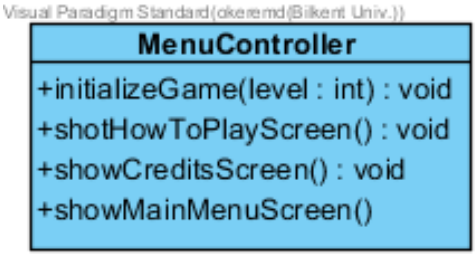
- **public void getSettingsData():** This method gets the information and buttons which will be shown on Settings screen.
- **public void getCharacterData():** This method gets the information and buttons which will be shown on Character Manager screen.
- **public void setSoundData():** This method gets the information and buttons which will be shown on Sound Manager screen.
- **public void getButtonData():** This method gets the information and buttons which will be shown on Button Manager screen.

4.4.24. GameFrame



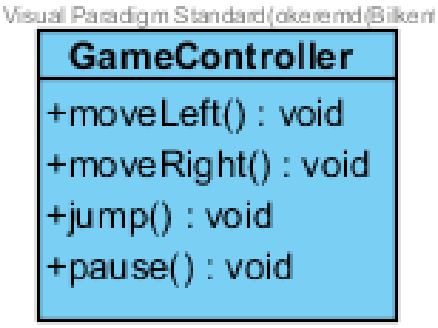
- **private final int FRAME_HEIGHT :** This constant will be determine the preferred height of the game frame.
- **private final int FRAME_WIDTH :** This constant will be determine the preferred width of the game frame.
- **private Pane pane:** This attribute will be used to place the images on frame with predetermined rules.
- **private void checkMap():** This method will be continously checks map in order to display current map.

4.4.25 MenuController



- **public void initializeGame():** This method communicates with the GameEngine to initialize the game.
- **public void showHowToPlayScreen():** This method opens the *How to Play* frame.
- **public void showCreditsScreen():** This method opens the *Credits* frame.
- **public void showMainMenuScreen():** This method opens the MainMenu frame.

4.4.26 GameController



- **public void moveLeft():** Sends a message to the GameEngine to character's going left.
- **public void moveRight():** Sends a message to the GameEngine to character's going right.
- **public void moveUp():** Sends a message to the GameEngine to character's jumping up.

- **public void pauseGame():** Sends a message to the GameEngine to pause the game.

4.4.27 ActionListener

This interface is for receiving action events which are occured by mouse clicks. When the action event occurs, ActionListener interface will be invoked. This interface is dependent to the DisplayFrame class.

4.4.28 MouseListener

This interface is for receiving the mouse event which is clicking. When the mouse event is received, therefore, the player clicks to the mouse, MouseListener interface will be invoked. This interface will be dependent to the DisplayFrame class.

4.4.29 KeyListener

This interface is for receiving keyboard events. When a key is pressed by the player, keyboard event is received and KeyListener interface will be invoked. This interface will also be dependent to the DisplayFrame class.

5. Glory & references

- Java Platform SE 7, docs.oracle.com/javase/7/docs/api/.
- Bruegge, Bernd, and Allen H. Dutoit. Object-Oriented software engineering: using UML, patterns, and Java. Pearson, 2014.