Bilkent University

Department of Computer Engineering

CS 319 Term Project

Section 2

Group 2G, Oldies but Goldies

Q-bitz

System Design Report

Project Group Members

1. Burak Kırımlı
2. Cansu Canan Ceyhan
3. Emre Keskin
4. Mert Çerçiler
5. Yağmur Özkök

Supervisor: Eray Tüzün

Content

1. Introduction

1.1 Purpose of the System

Q-Bitz is a family board game. Our purpose is to implement Q-Bitz to computer in such a way that users can get the same and the highest satisfaction like the board game version. Our implementation is very similar with board game. In game there are 3 different game modes. First one is, user looks the design on the randomly opened card and tries to make the same design with using cubes. In the second game mode, user looks the design again and randomly rolls all cubes and tries to make the same design which is on the opened car. For the last game mode, user looks the design on the randomly opened card for 10 seconds and tries to make the same design with using cubes. Timing, memorizing and having fun is very important in our game.

1.2 Design Goals

The design step is very important about how the project will occur. In this step, we are going to analyze the project's needs and try to complete. To achieve that, we are going to use the non- functional requirements which we mentioned in the analysis report. In non-functional requirements, there are several subheadings which are usability, extendibility, performance and portability.

1.2.1 Usability

The most important requirement of usability is how easily a project (game) can be understood and used by the user. If a user has difficulty in understanding the game, he / she will not enjoy the game and the game loses its usability. In our project, the user interface is easily understandable. Switching between screens is set to be uncomplicated. The user can easily understand the game and play with fun. The user can play the game easily even without looking at “how to play”.

1.2.2 Extendibility

The way that we design “Q-Bitz” structure helps us add new features and also it supports to remove any undesirable content from the “Q-Bitz”. For example, in the third game mode (The Rolling Stones) user has 10 seconds to look the card and memorize it. We can decrease that time in the “hard” difficult level, so that “hard” level of “The Rolling Stones” mode will be more challenging.

1.2.3 Performance

Performance is very important criteria for every project. In “Q-Bitz”, GUI library is going to be used for better performance and subsystems will be designed carefully to not to affect the performance in a bad way.

1.2.4 Portability

In general, “portability” means that the ability of software to be easily transferred from one system to another. “Q-Bitz” is going to be implemented in Java. Is is known that Java is one of the most commonly used coding language, so that it can be said that “Q-Bitz” has portability.

2. System Architecture

2.1 Subsystem Decomposition

In this section, we will decompose our system into subsystems, that means the identification of subsystems, services, and their associations to each other to easier to conceive, understand, program and maintain. By decomposing our system, we can modify or extend our game easier.

We have decided to use MVC (Model-View-Controller) architectural pattern while decompose our system. By using Model-View-Controller architectural pattern, we divide our system into three subsystems, our user interface classes for view part, game classes for model part, and controller and manager classes for controller part, as shown in the Figure 1. In our user interface classes, meaning classes that have menu, is designed like user chooses options before the game starts such as choosing game mode or difficulty level, as shown in Figure 3. Our view subsystem is responsible for application domain knowledge. In game classes, our game objects will be created and games will be played in those sections, as shown in Figure 4. Our game subsystem is responsible for displaying application domain objects to user. Lastly, in our controller classes, control classes and manager classes will be implemented. Control classes will check to whether the players move accurate or not, or controls game is finished or not. Also, manager classes will manage the game play, as shown in Figure 5. Our controller subsystem is responsible for sequence of interactions with the user and notifying views of changes in the model.

As a result, our system decomposition provides us high cohesion and easier managing of our classes.

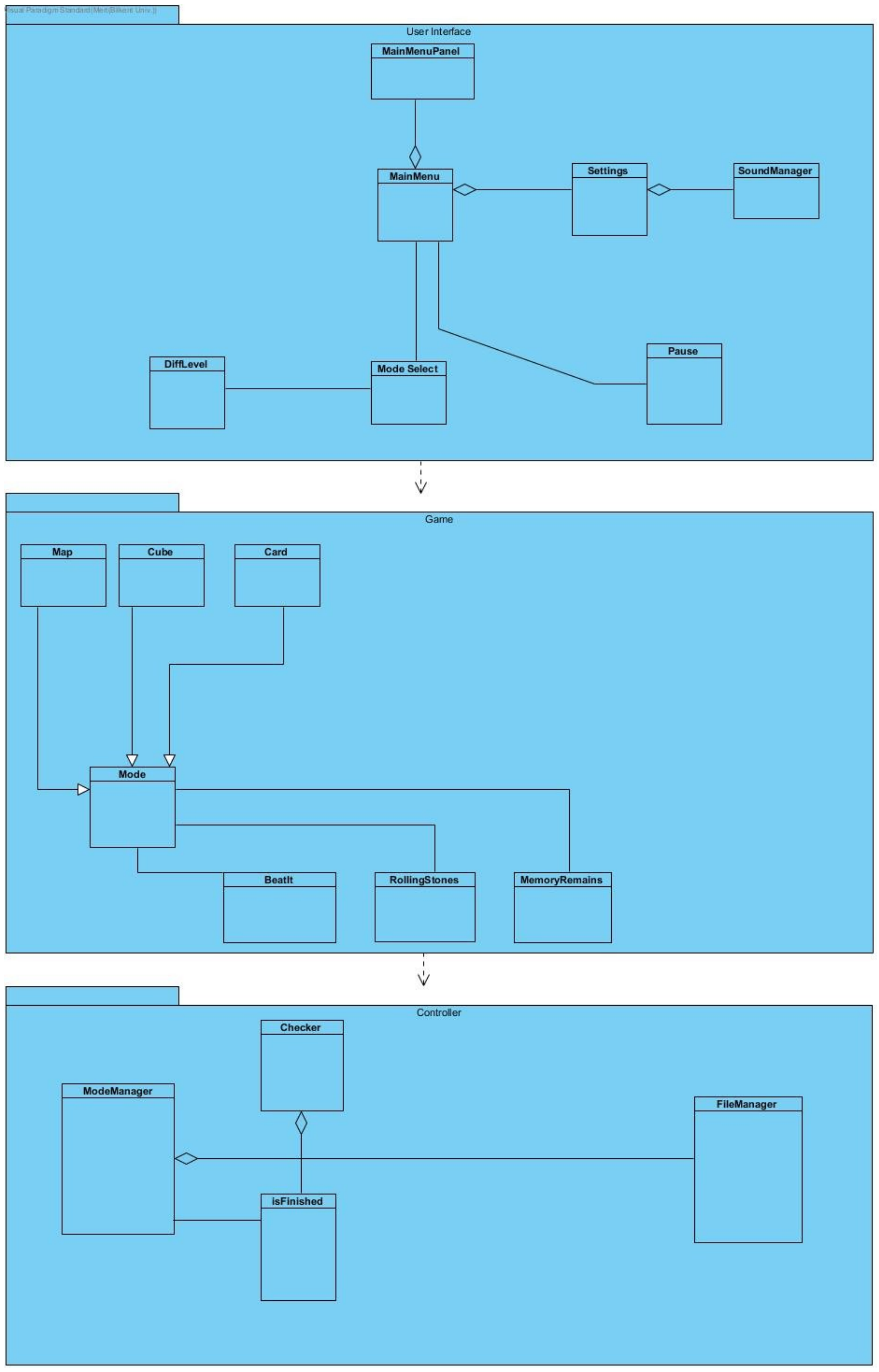


Figure 1: Basic Subsystem Composition

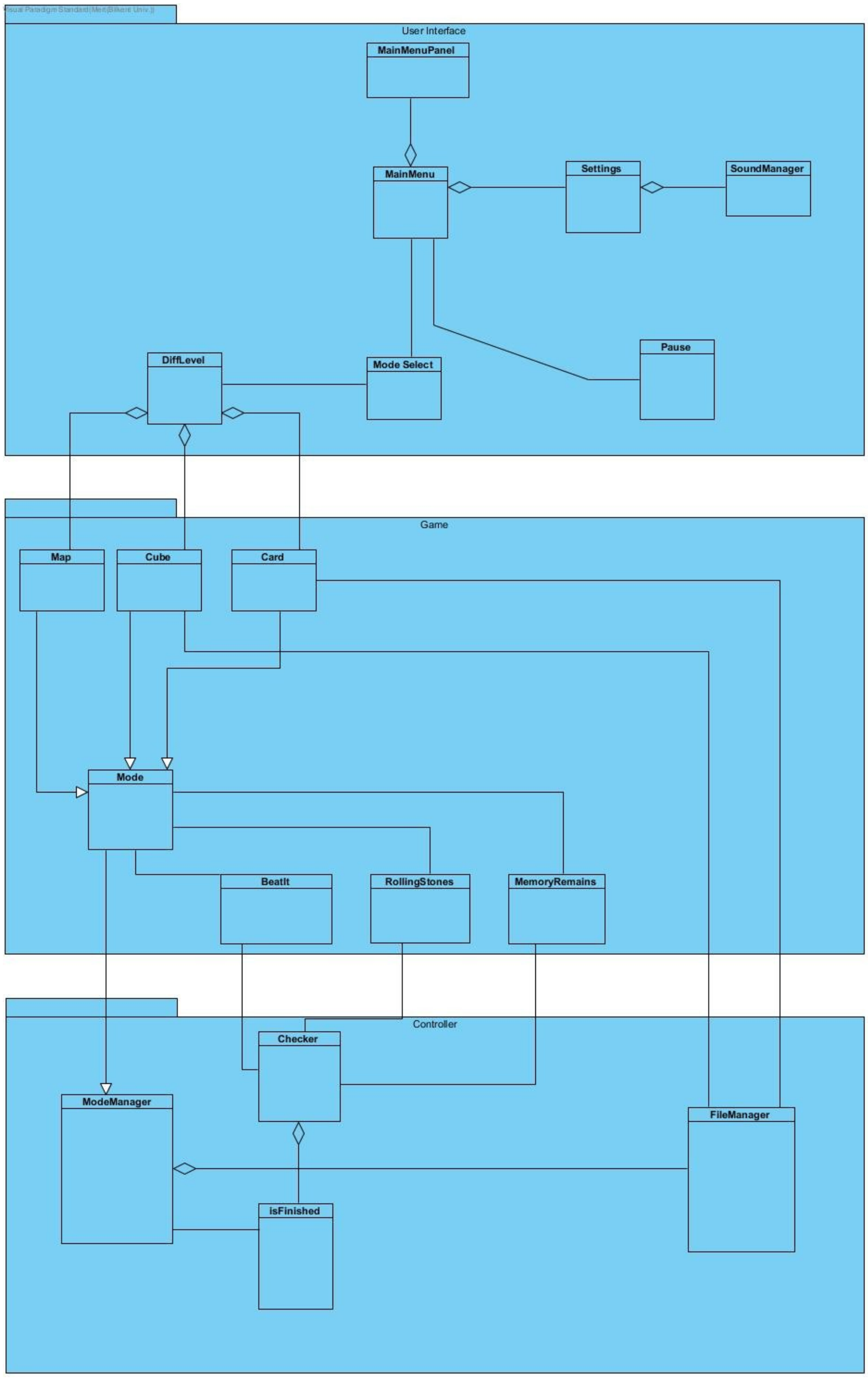


Figure 2: Detailed subsystem decomposition

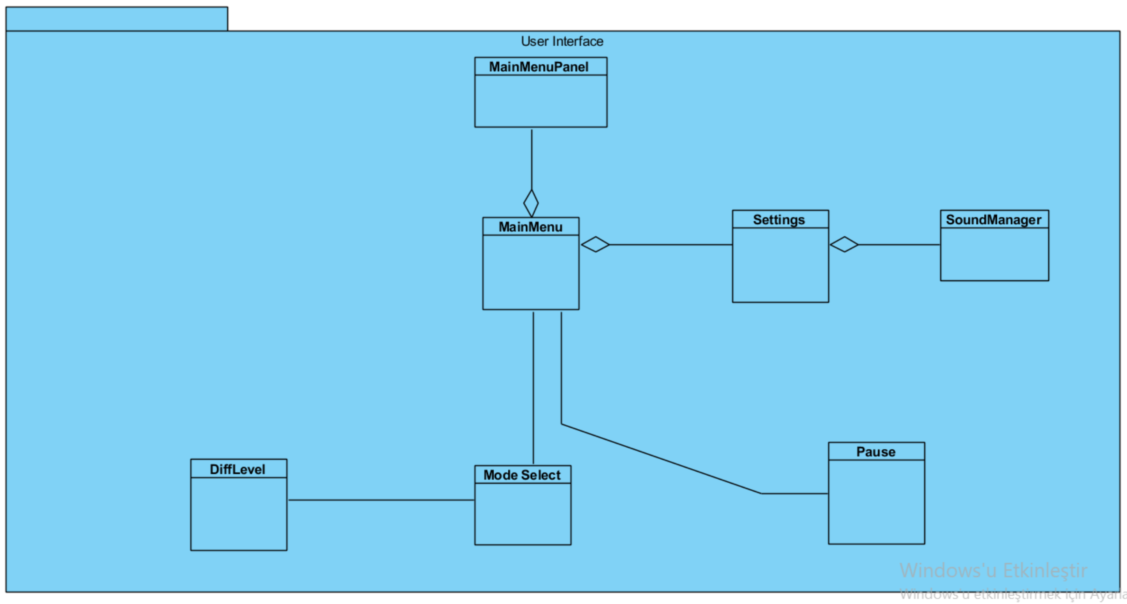


Figure 3: View subsystem (user interface)

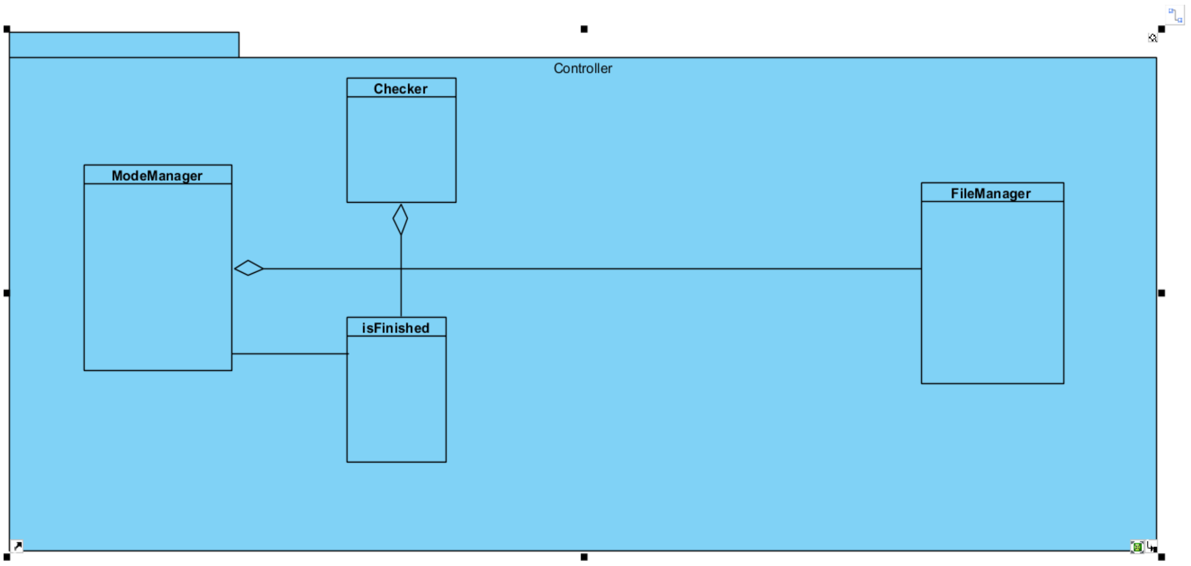


Figure 4: Controller subsystem

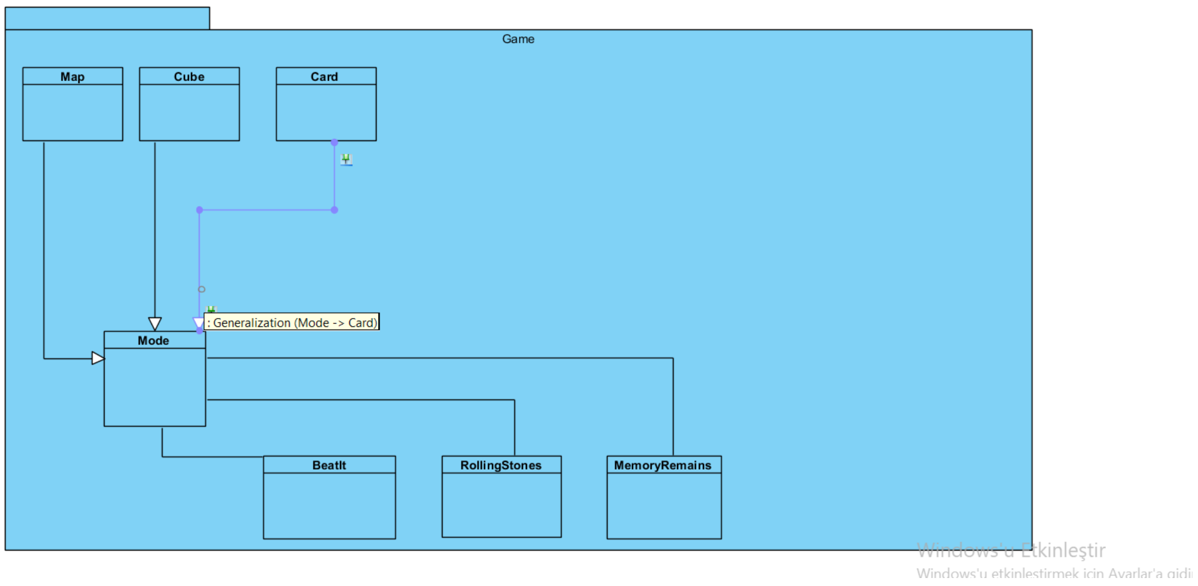

Game
Map
Cube
Card
: Generalization (Mode -> Card)
Mode
BeatIt
RollingStones
MemoryRemains
Windows'u Etkinleştir
Windows'u etkinleştirmek için Ayarlar'a gidi

Figure 5: Model subsystem (Game)

2.2 Hardware/Software Mapping

As a group, we decide to implement our Project in Java programming language. Graphical user interface (view part) of the Project will be implemented using JAVAFX library.

As a result of our choice, system requirement of Q-bitz will be Java Development Kit 8 or any newer version. If user has compatible version of the JDK and a mouse, he/she can easily compile and run our game.

2.3 Persistent Data Management

In order to run properly, Q-bitz needs to store some data, yet it will not require any internet connection or data base operation system, it will manage data in hard drive of the user. Objects which are crucial to the game such as map, cubes will be stored during run time phase and they are not modifiable. Moreover, in the same document as the game source code, there will be another file to handle image and sound. Images will be used for the puzzle that will be represented to the user, face of the cubes so that user can choose another cube by looking at the top of the cube. When user is in main menu and he/she clicks to see the settings panel, user has the ability to change music that is playing in the background. Sound options that we provide to the user will be in the same folder as the Images. Moreover, images will be stored using .jpg format and sound will be kept as .waw format.

2.4 Access Control and Security

As mentioned in the sections 2.2 and 2.3, our Project will not require internet connection nor database, therefore there will be nothing for access control. It makes sense considering there will not be any attributes user would like to store. There is only high score and it shows users high score starting from initial running of the program. So, every user will have unique high score when he/she runs the program. Therefore, access control and security are not necessary for our program.

2.5 Boundary Conditions

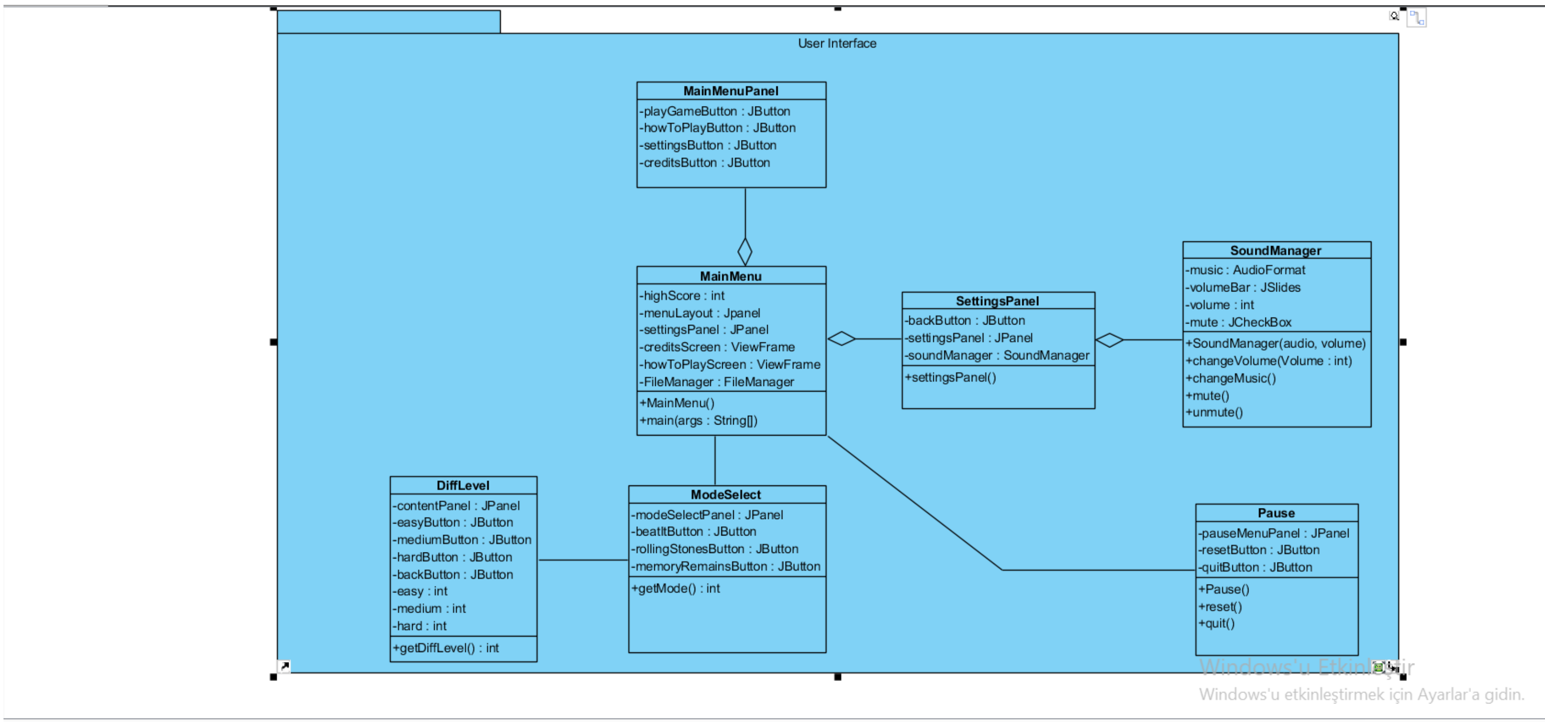
Q-bitz will not require any install or download, in the final stage, it will be in an executable .jar and anyone who has that folder can run our project.

Q-bitz can be terminated just by clicking the “X” button on the top left corner. It does not matter if the user presses that button in main menu, pause screen or during the gameplay, game will be terminated.

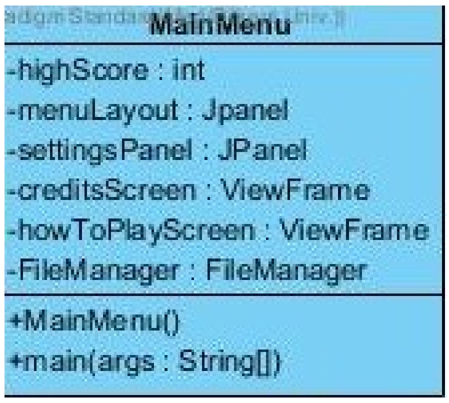
If there is a run time error regarding time, image or high score, regarding folders in the main Project folder must be investigated in order to solve the problem. Moreover, if the game stops executing during gameplay any data will be lost.

3. Subsystem Services

3.1 User Interface Subsystem



User interface subsystem has 7 classes. They are all menus that user chooses options such as game mode or difficulty level. There are also Pause class, which user can either reset the game or quit the game. Settings class, that user change sound settings in it. Also, MainMenuPanel provides panel for the MainMenu class, which user has 4 options. (Play Game, Credits, HowToPlay and Settings)



3.1.1 MainMenu

attributes:

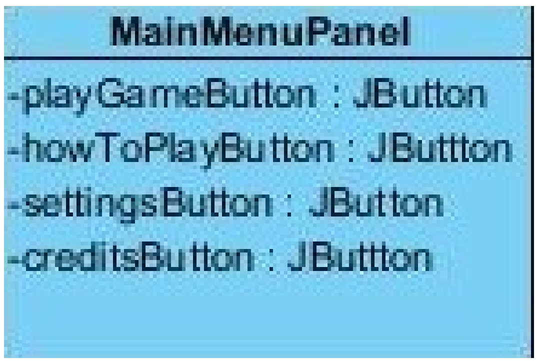
- **private int highScore:** This will be used to show the high score in the game.
- **private JPanel menuLayout:** This will provide main menu layout via JPanel.
- **private JPanel settingsPanel**: This will provide settings screen layout via JPanel.
- **private viewFrame creditsScreen:** This will create the frame for credits screen.
- **private viewFrame howToPlayScreen:** This will create the frame for how to play screen.
- **private FileManager fileManager:** This attribute will be used in order to call the fileManager from FileManager class.

Constructor:

- **public MainMenu():** Default constructor for the MainMenu class.

Methods:

- **public void main(args: String[]):** main method of the MainMenu class.


MainMenuPanel
-playGameButton : JButton
-howToPlayButton : JButtton
-settingsButton : JButton
-creditsButton : JButtton

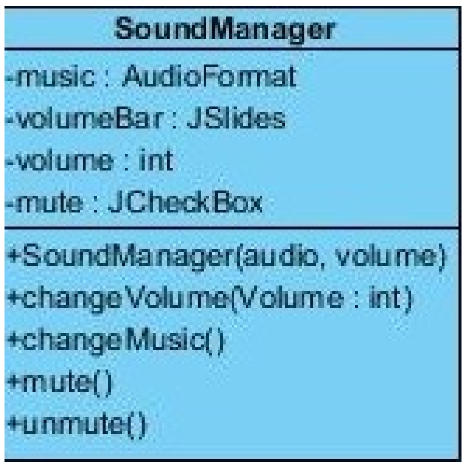
3.1.2MainMenuPanel

attributes:

- **private JButton playGameButton:** This will provide a button in the main screen of the game, user can click it to start the game.
- **private JButton howToPlayButton:** This will provide a button in the main screen of the game, user can click it to display game rules.
- **private JButton settingsButton:** This will provide a button in the main screen of the game, user can click it to display settings of the game.
- **private JButton creditsButton:** This will provide a button in the main screen of the game, user can click it to display credits of the game.

Constructor:

- **public MainMenuPanel():** Default constructor for the MainMenuPanel class.



3.1.3 SoundManager

attributes:

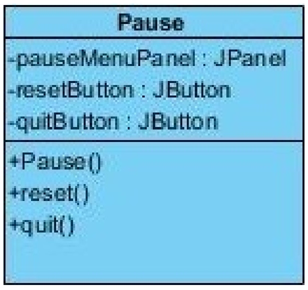
- **private AudioFormat music:** This will provide music of the game, which user can change if he/she desires.
- **private JSlider volumeBar:** This will enable user to adjust the volume of the music playing in the background of the game.
- **private JCheckBox mute:** This will enable the option to mute the music.
- **private int volume:** This will hold the current volume value for the music.

Constructor:

- **public SoundManager(audio: AudioFormat, volume: int):** Chooses default music and default volume.

Methods:

- **public void changeVolume(volume: int):** This will be used when user is adjusting volume using volumeBar slider stated above.
- **public void changeMusic():** This will choose and play naother music from already defined options.
- **public void mute():** This will be used for muting the music.
- **public void unmute():** This will be used for unmuting the music.



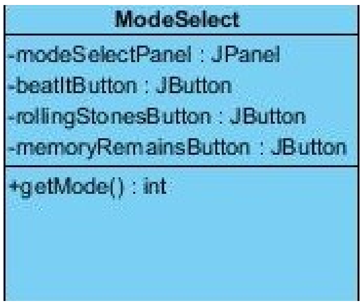
3.1.4 Pause

attributes:

- **private JPanel pauseMenuPanel:** This attribute will create panel for pause menu
- **private JButton resetButton:** Reset button for reset the game
- **private JButton quitButton:** Quit button for quit the game.

Contructor:

- **Pause():** constructor of pause
- **public void reset():** method for reseting the game.
- **Public void Quit():** method for quitting the game.


ModeSelect
-modeSelectPanel : JPanel
-beatItButton : JButton
-rollingStonesButton : JButton
-memoryRemainsButton : JButton
+getMode() : int

3.1.5 ModeSelect

attributes:

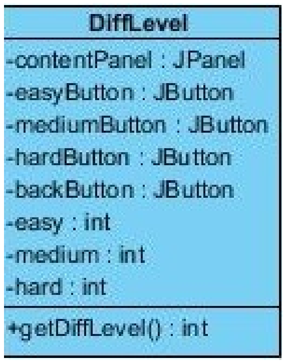
- **private JPanel modeSelectPanel:** This attribute creates panel for mode select menu
- **private JButton beatItButton:** beatItButton for the Beat It mode.
- **private JButton rollingStonesButton:** rollingStonesButton for the Rolling Stones mode.
- **private JButton memoryRemainsButton**: memoryRemainsButton for the Memory Remains mode.
- **Private int beatIt:** all modes are numbered. Number of Beat It is 1.
- **Private int memoryRemains:** all modes are numbered. Number of Memory Remains is 2.
- **Private int rollingStones:** all modes are numbered. Number of Rolling Stones is 3.

Constructor:

- **ModeSelect():**default constructor for ModeSelect class

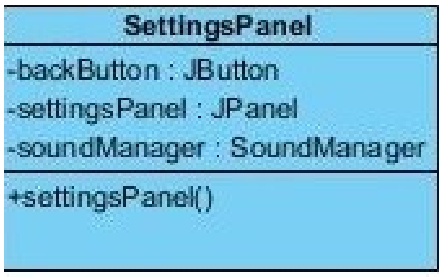
Methods:

- **public int getMode():** getter method of ModeSelect class. Returns selected mode.



3.1.6 DiffLevel

Attributes:
private JPanel contentPanel: This attribute will be used for creating "Content" panel
private JButton easyButton: This attribute will be used for creating "easy" button to let the user select the easy game mode
private JButton mediumButton: This attribute will be used for creating "medium" button to let the user select the medium game mode
private JButton hardButton: This attribute will be used for creating "hard" button to let the user select the hard game mode
private JButton backButton: This attribute will be used for creating "back" button to let the user go back the previous page
private int easy: This attribute will be used to represent the easy game mode with using "int" type
private int medium: This attribute will be used to represent the medium game mode with using "int" type
private int hard: This attribute will be used to represent the hard game mode with using "int" type
Methods:
public int getDiffLevel(): Method to get the user's difficulty level selection.



3.1.7 Settings

Attributes:

private JButton backButton: This attribute will be used for creating “back” button to let the user go back the previous page

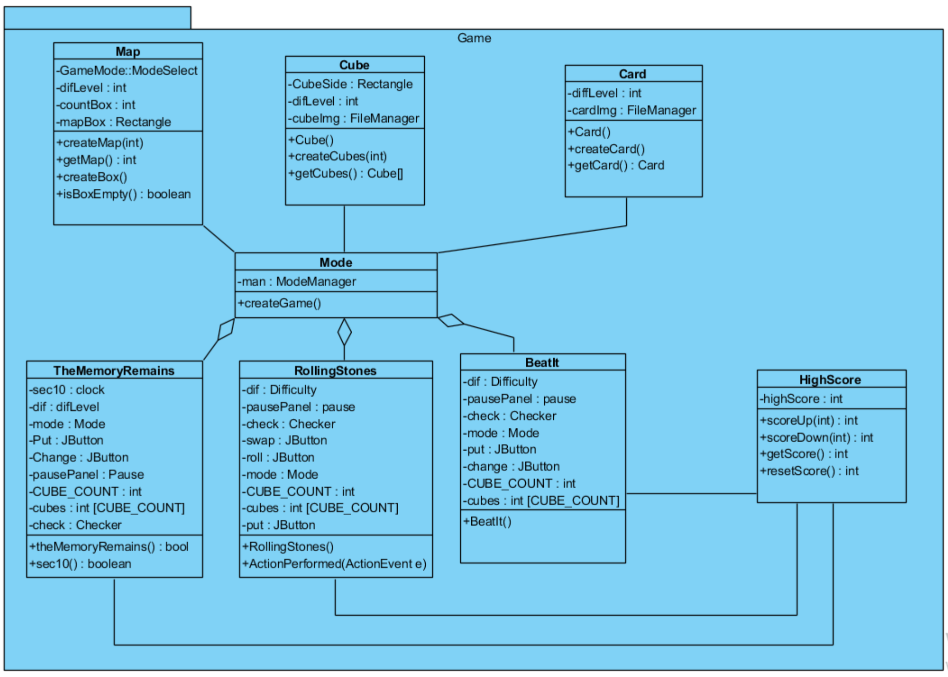
private JPanel settingsPanel: This attribute will be used for creating JPanel for “Settings” menu

private SoundManager soundManager: This attribute will be used to call soundManager object from SoundManager class. In “Settings”, user can select the background music.

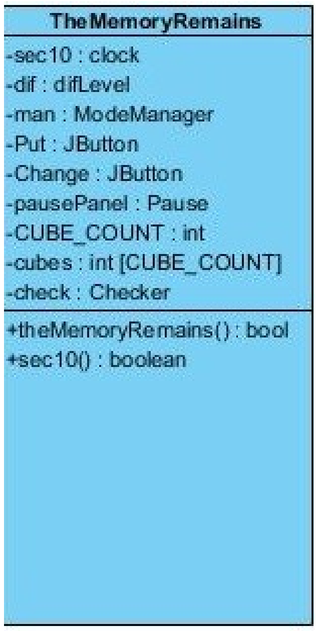
Constructor:

public settingsPanel():default constructor for the settingsPanel.

3.2 Game Subsystem



Game subsystem is the subsytem where game objects are created. We have 3 Game Objects which are Cube, Card and Map. Mode class is creating the game by doing put the commands and game objects into the panel. Also, in the game subsystem, we have game mode classes, which are theMemoryRemains, RollingStones and BeatIt. Player will play the game mode which he/she selected earlier. Lastly, the highscore class that hold the score of the player and the highscore.



3.2.1 TheMemoryRemains

attributes:

- **private Clock sec10:** This will be used for 10 sec time limit user has for seeing the puzzle in this mode.
- **private Clock timeRem:** This will be used for regular time limit checking of the game.
- **private Mode mode:** This will be get commands like put and change buttons, game objects like maps, cubes and card, and general rules like time checker.
- **private DiffLevel dif:** This will hold the current difficulty level.

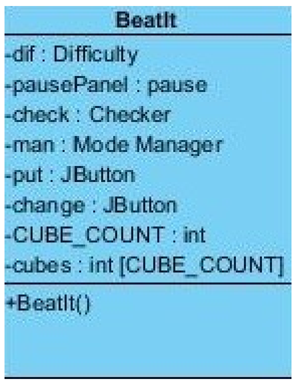
- **private JButton put:** This will be used for the” put” button.
- **private JButton change:** This will be used for the” change” button.
- **private Pause pausePanel:** This will be used for pausing the game in this mode.
- **private final CUBE_COUNT:** This will be used in the creation of the Cube array, with size differing in each difficulty level.
- **private Cube[] cubes:** This will be an array of cubes which user can use in this game mode.

Constructor:

- **public TheMemoryRemains ():** Default constructor for the TheMemoryRemains class.

Methods:

- **public boolean sec10(volume: int):** This will be used at the start of the game, it will return false after 10 seconds and user will not be able to see the puzzle anymore.

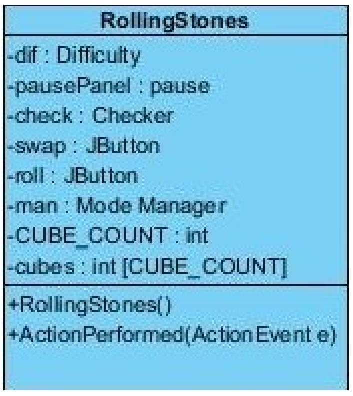


3.2.2 BeatIt

attributes:

- **private int diff:** This attribute will be used to get selected difficulty level.

- **Private Pause pausePanel:** This attribute will be used to call the pausePanel object from pause class.
- **Private Checker check:** This attribute will be used to call the check object from Checker class.
- **private Mode mode:** This will be get commands like put and change buttons, game objects like maps, cubes and card, and general rules like time checker.
- **private JButton put:** This will be used for the” put” button.
- **private JButton change:** This will be used for the” change” button.
- **private final CUBE_COUNT:** This will be used in the creation of the Cube array, with size differing in each difficulty level.
- **private Cube[] cubes:** This will be an array of cubes which user can use in this game mode.



3.2.3 Rolling Stones

Attributes:

private Difficulty difficulty : This attribute will be used to reach to get the size of the map. As maps size is related to difficulty.

private Pause pause: This attribute brings pause button with it’s attribute.

Private Mode mode: Mode brings methods for roll put swap and roll.

private Checker check: Checker checks some properties like time, or if player did right or wrong.

private JButton put: Creates canvas for put method from ModeManager class.

private JButton swap: Creates canvas for swap method from ModeManager class.

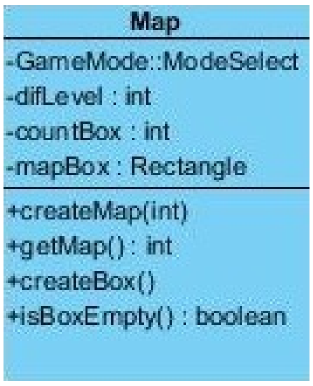
private JButton roll: Creates canvas for roll method from ModeManager class.

private Cube[] cubes: Creates identical cubes for a given size.

Methods:

public void Rolling Stones(): Creates a constructor to recall in main JFrame.

public actionPerformer(actionEvent e): Methods for mouse actions in GUI.



3.2.4 Map

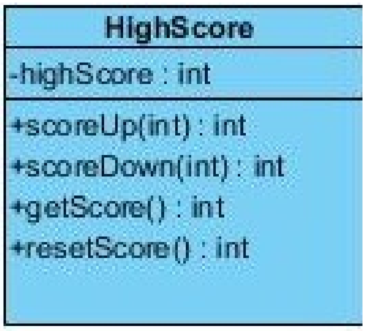
attributes:

- **private int gameMode:** This attribute will be used to get the selected game mode (BeatIt, Rolling Stones, Memory Remains)
- **private int difLevel:** This attribute will be used to get the selected difficulty level.
- **Private int countBox:** This attribute will be used to hold number of boxes in the game.
- **Private Rectangle mapBox:** This attribute will be used for each rectangle box in the game.

Methods:

- **public void createMap():** This method will be used to create map
- **public Map getMap():** getter method of the map class.
- **Public void createBox():** This method will be used for creating boxes in the map.

- **Public boolean isBoxEmpty():** this method will be used to check boxes whether they are empty or not. Return true if the box is empty, return false otherwise. If box is empty, user will not be albe to put cube into box.



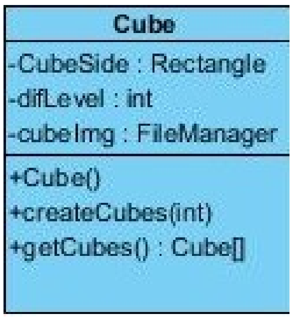
3.2.5 HighScore

Attributes:

- **private int highScore:** This attribute will be used to show highscore of player.
- **Private int score:** This attribute will be used to hold current score.

Methods:

- **public int scoreUp(int):** method for increase the score.
- **Public int scoreDown(int):** method for decrease the score.
- **Public int getScore():** returns the score.
- **Public int resetScore():** resets the score.



3.2.6 Cube

attributes:

- **private Rectangle cubeSide:** This attribute draws each side of a cube.
- **Private int difLevel:** This attribute gets selected difficulty level. Creates cubes according to selected difficulty level.

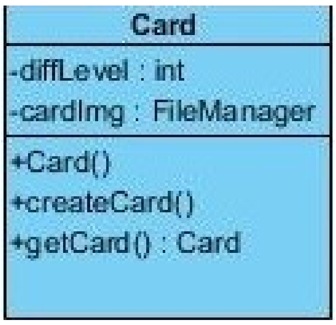
- **Private FileManager cubeImg:** This attribute gets scanned image for the one side of the cube.

Constrctor:

- **Cube(int difLevel):** Default constructor of the Cube class.

Methods:

- **public void createCubes():** This method will be used to create cubes and put scanned image into cube sides.
- **Public Cube[] getCubes():** This method is getter method of the Cube class. Returns Cubes
- **Public Cube getCube():** This method returns just one Cube.

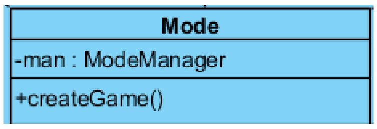


3.1.7 Card

Attributes:
private int difLevel: This attribute will be used to represent difficulty level which is selected by user(easy, medium, hard)
private FileManager cardImage: This attribute will be used to represent card image which are going to use in game screen. These images are in FileManager.

Constructor:
public Card(): default constructor for the Card.

Methods:
public void createCard(): method is going to be used for creating cards for game screen so that user can see it and start to play game.
public void getCard(): method to get card and place it in game screen.



3.1.8 Mode

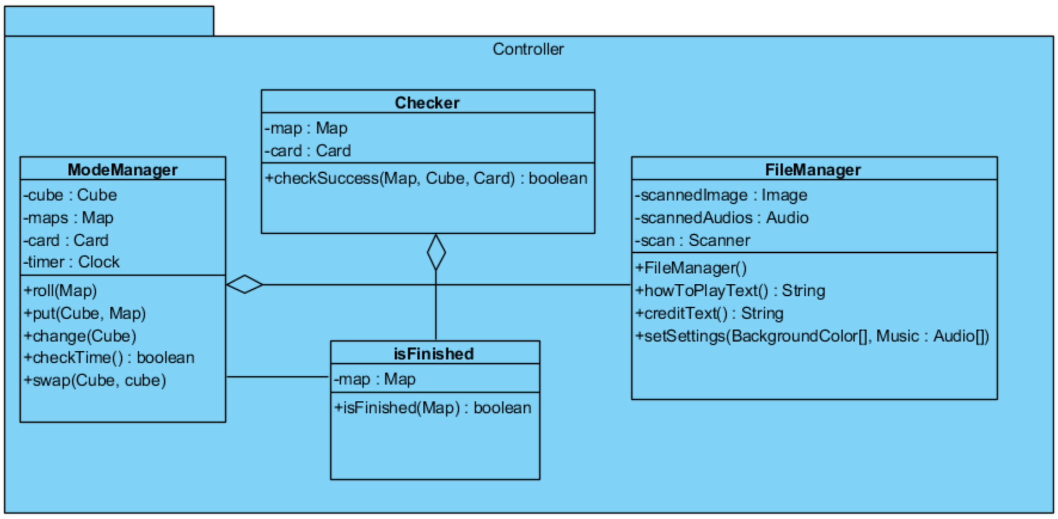
attributes:

private ModeManager man: This attribute gets instance of Mode Manager class to hold commands and timeChecker.

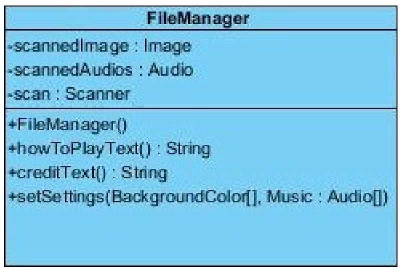
Methods:

public void createGame(): This method creates game. Put maps, commands and timeChecker.

3.3 Controller Subsystem



In our controller subsystem, we have 4 classes. Mode manager is the class where game commands will be implemented. Our commands are change (for the Memory Remains and BeatIt game modes), swap and roll (for the Rolling Stones game mode), and change (common for all game modes). Also, time checker will be implemented in this class, which will check whether time is up or not. Checker class will controll the user actions, whether he or she put the cube correctly or not. isFinished class will control whether the map is finished or not. Lastly, FileManager class will scan images(for the cubes and cards) and audios (for background music).



3.3.1 FileManager

attributes:

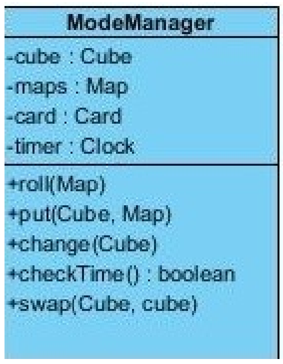
- **private Images scannedImages:** This attribute will be used to scan images, that are going to be used in entire game.
- **Private Audio sacnnedAudios:** This attribute will be used to scan audios, that are going to be used in entire game.
- **Private Scanner scan:** This attribute will be used to call the scan object from Scanner class.

Constructor:

- **public FileManager():** default constructor of the File Manager.

Methods:

- **public String howToPlayText():** This method will return the text for "How to Play" section.
- **Public String creditsText():** This method will return the text for "Credits" section.
- **Public void setSettings(colorArray, audioArray):** method to set background color and background audio for the game.



3.3.2 ModeManager

Attributes:

private Card card: This attribute will be used to reach the properties of the card. As users main goal is to combine the figures on the cubes by looking at the cards. Cards are essential for every mode. So Mode Manager takes it as an attribute.

private Cube cubes: In every mode there are number of cubes related to difficulty. So cubes are used as an attribute for every mode.

private Map maps: When cubes are combined they either swapped or putted in a map to check if they resemble to card. So as it is in every mode mode manager takes Map as an attribute.

private Clock time: As it is a time based game, timer must be used to see if the time is up or not.

Methods:

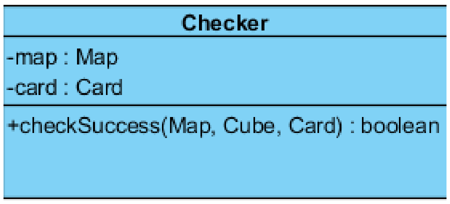
public void roll(Cube c): Roll method rolls all of the unfixed cubes. When user push the roll button, this method will be executed. Then sides of the cubes will change.

public void put(Cube c): Put method is for fixing the cube. When user press the put button, cube is fixed and then checks if it is in the right place.

public void swap(Cube c1, Cube c2): When user selects two cubes and press' swap button two cubes swapped.

public void change(Cube c): When user press the change button, side of the cube will be changed.

public boolean checkTime(): This method checks if time is up or not. Related to this game will continue or will end.



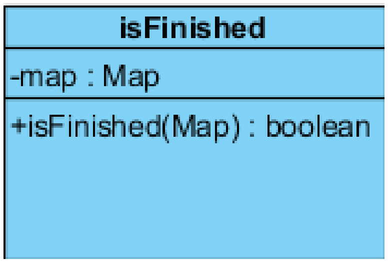
3.3.3 Checker

Attributes:

Private Map map: This attribute gets the map from map class.
Private Card card: This attribute gets card from Card class.

Methods:

public Boolean checkSucccess(Map, Card): Checks cubes place on the map and if successful returns true else false.



3.3.4 isFinished

Attributes:

private Map map: This attribute gets map object from Map class.

Methods:

public boolean isFinished: This method checks whether the map is completed or not.

4. Low-level Design

4.1 Object Design Trade-Offs

Understandability vs Functionality:

In terms of understandability, our aim is to make our game easy to understand by our users and not implement a complicated game in terms of use. Therefore, it is important for us to decrease the complexity in the game modes and provide a simple control mechanism for our users so that they can understand the concept of the game easily and play enjoyably.

4.2 Final Object Design

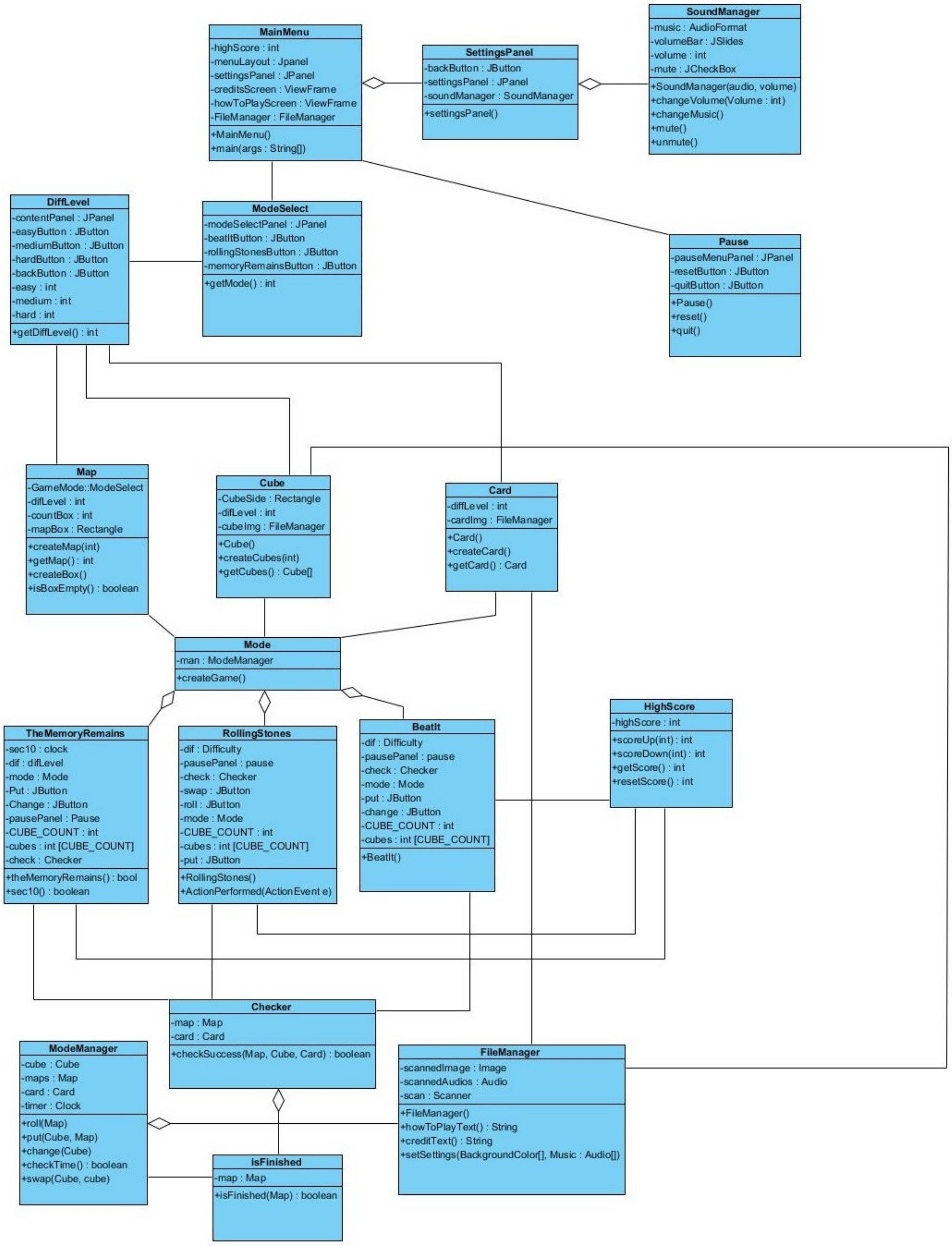

MainMenu
-highScore : int
-menuLayout : Jpanel
-settingsPanel : JPanel
-creditsScreen : ViewFrame
-howToPlayScreen : ViewFrame
-FileManager : FileManager
+MainMenu()
+main(args : String[])
SettingsPanel
-backButton : JButton
-settingsPanel : JPanel
-soundManager : SoundManager
+settingsPanel()
SoundManager
-music : AudioFormat
-volumeBar : JSlides
-volume : int
-mute : JCheckBox
+SoundManager(audio, volume)
+changeVolume(Volume : int)
+changeMusic()
+mute()
+unmute()
DiffLevel
-contentPanel : JPanel
-easyButton : JButton
-mediumButton : JButton
-hardButton : JButton
-backButton : JButton
-easy : int
-medium : int
-hard : int
+getDiffLevel() : int
ModeSelect
-modeSelectPanel : JPanel
-beatItButton : JButton
-rollingStonesButton : JButton
-memoryRemainsButton : JButton
+getMode() : int
Pause
-pauseMenuPanel : JPanel
-resetButton : JButton
-quitButton : JButton
+Pause()
+reset()
+quit()
Map
-GameMode::ModeSelect
-difLevel : int
-countBox : int
-mapBox : Rectangle
+createMap(int)
+getMap() : int
+createBox()
+isBoxEmpty() : boolean
Cube
-CubeSide : Rectangle
-difLevel : int
-cubeImg : FileManager
+Cube()
+createCubes(int)
+getCubes() : Cube[]
Card
-diffLevel : int
-cardImg : FileManager
+Card()
+createCard()
+getCard() : Card
Mode
-man : ModeManager
+createGame()
HighScore
-highScore : int
+scoreUp(int) : int
+scoreDown(int) : int
+getScore() : int
+resetScore() : int
TheMemoryRemains
-sec10 : clock
-dif : difLevel
-mode : Mode
-Put : JButton
-Change : JButton
-pausePanel : Pause
-CUBE_COUNT : int
-cubes : int [CUBE_COUNT]
-check : Checker
+theMemoryRemains() : bool
+sec10() : boolean
RollingStones
-dif : Difficulty
-pausePanel : pause
-check : Checker
-swap : JButton
-roll : JButton
-mode : Mode
-CUBE_COUNT : int
-cubes : int [CUBE_COUNT]
-put : JButton
+RollingStones()
+ActionPerformed(ActionEvent e)
BeatIt
-dif : Difficulty
-pausePanel : pause
-check : Checker
-mode : Mode
-put : JButton
-change : JButton
-CUBE_COUNT : int
-cubes : int [CUBE_COUNT]
+BeatIt()
Checker
-map : Map
-card : Card
+checkSuccess(Map, Cube, Card) : boolean
ModeManager
-cube : Cube
-maps : Map
-card : Card
-timer : Clock
+roll(Map)
+put(Cube, Map)
+change(Cube)
+checkTime() : boolean
+swap(Cube, cube)
FileManager
-scannedImage : Image
-scannedAudios : Audio
-scan : Scanner
+FileManager()
+howToPlayText() : String
+creditText() : String
+setSettings(BackgroundColor[], Music : Audio[])
isFinished
-map : Map
+isFinished(Map) : boolean

Figure 6- Final Object Design

4.3 Packages

4.3.1 java.util

This package contains concurrent which contains time unit methods.

We
will use these package for checking time.

4.3.2 java.awt.gridLayout

We will use this layout for maps to place the cubes.

4.3.3 java.awt.actionEvent

Implements ActionListener interface for events.

4.3.4 javax.swing.*

For using JLabels, JButtons, JFrames etc. we need this package. Because our game is highly visual all packages in swing is useful.

4.3.5 javafx.scene.image

This package is useful for importing pictures. As we use figures on the cubes and a picture on the card this package is essential.

4.4 Class Interfaces

4.4.1. ActionListener

Especially, as shown in our mock up, there are too many buttons used in the game. For adding function to those buttons, action listener interface used.

4.4.2. MouseListener

We will use mouse listener for selecting cubes. As selected cubes will be used as parameters for functions. Mouse listener interface is also essential.