Bilkent University

Department of Computer Engineering

CS319-OBJECT ORIENTED SOFTWARE ENGINEERING

Q-Bitz: Q-Bitz

Iteration 1 - Design Report

Sait Aktürk, Zafer Tan Çankırı, Berkin İnan, Halil Şahiner, Abdullah Talayhan

Supervisor: Eray Tüzün

1.Introduction

1.1 Purpose of the system

In *Q-bitz* players use their special cubes to recreate patterns on the cards, gain points and win the game. Although the original game has three rounds which will be played in order to win a game, our game will not be round based but mode based style which will consists of those rounds. We chose Q-bitz because it has the potential to be extended with new game modes and features that we will add.

In our game we have:

- Online Q-bitz
    - Multiplayer Game Modes
- Offline Q-bitz
    - Practice
    - Single Player Game
- Options
    - Adjustable color set
    - Sound options
    - Controls
- How to play

With this structure, the game will use the benefits of being digital, meaning that a user can choose multiplayer mode to enjoy the game with friends while in single player mode users can have a taste of an arcade game without the need of any other user.

1.2 Design goals

Design is one of the important process in the development of the system. It makes more clear the required systems that should be developed. As we also stated in our analysis report, the project has some non-functional requirements, and these requirements should become more clear in the design part. Particularly, these are the focuses of this paper. Following sections are the descriptions of the important design goals.

1.2.1 Criteria

End User Criteria

Usability:

One of the target user groups of the game are the children. Therefore, we paid attention to make the game simple as possible. We chose the controller parts of the game carefully, and we tried to make them modifiable as possible as we can. To reduce complexity of the game, we put the related screens, menus, and actions together. We tried to keep screens as clean as possible. The unnecessary components on the screens are ignored. The game can be both played with keyboard and mouse. We also thought that the game may be played with touch screens. Therefore, we are arguing about adding a gesture support to the game. There are also help menus in the game, and we prepared a "How to Play" module for the game. Therefore, the players can learn about the game and its controls.

Performance:

One of the main concerns of the project is the performance, since it a multiplayer game project. To overcome the performance issues we use a real time network communication protocol; a stable, well-known, and old database system; and a native GUI library for the project.

Maintenance Criteria

Extendibility:

Since the project will be developed in a fully object oriented and modular behaviour, all subparts of the project can work almost independently, and thanks to the object oriented design of the project, the abstractions and the composition of the project let us to extend the program easily by extending the existing classes. At the end, by writing some simple lines of code, the project can be extended easily.

Modifiability:

Since the project will be developed in a fully object oriented modular behaviour, all subparts of the project can work almost independently. Therefore, to modify the game, only thing need to be done is to change the parts related to the modifying process.

Reusability:

There are some subsystems which can be also used for other game projects, and similar kind of projects, since our program will be developed in a fully object oriented modular behaviour.

Portability:

Portability is also another important required for this project. Since it is a multiplayer playing supported game, it should also be easy to operate on a variety of systems. This situation is one of the main reasons why the project will be developed in Java language. Java programs run on Java Virtual Machine which is available for almost every system, and this situation also makes our game platform independent. As an extension to this, there are not any other dependencies which are required for our game to run. Therefore, it is an almost fully portable application.

Performance Criteria

Performance is one of the topics which argued on most. Since this is a multiplayer game which has time criteria. The playing of the game should be very smooth. We thought that there may be some performance issues, and to overcome these issues we try to use native functionalities of Java, we avoided to use third party libraries and extensions. Two of the problematic parts of the project are the database system we used, and the network communication parts, and for the database part of the system we chose MySQL since it is a well-known and old database system which can handle most problematic situations, and for the network part, we chose socket communication among the server and the clients, since it works in almost real time.

1.2.2 Design Patterns

The game will be implemented using Model View Controller(MVC) and Observer design patterns. MVC will be used for the user interface and classical game logic by using Manager classes as controllers. Observer pattern will be used for Client-Server interactions with callback mechanism, meaning that the clients will be notified for the changes depending on different states of the game.

2. High-Level Software Architecture

2.1 Subsystem decomposition

Our system is composed of four main subsystems which are network, user interface, game logic and user.

Network Subsystem: This subsystem contains client and server classes that implements the interaction between the game and the server for handling multiplayer games. The network subsystem also captures the interactions between the server and database with database connector classes.

User Interface Subsystem: This subsystem contains the menu classes that enables the navigation features. We classified all the non-game screens as menu, and the playable screens as the game instance.

Game Logic Subsystem: This subsystem contains the classes related to an instance of the game which captures all the mechanics and controller classes that enables the game to be played.

User Subsystem: This subsystem contains classes related to user informations and configuration files.

2.2 Hardware/software mapping

Q-Bitz will be implemented with Java and will use JavaFX for the game graphics and the menus.

The game will be controlled using mouse and keyboard. The mouse will be used to interact with the menus and the gameplay screens. The cubes can be selected by clicking on it with the mouse and the keyboard will be used to rotate the cube.

The game can be played both online and offline. To play the game with other players, the device that the game is installed needs to have internet access.

2.3 Persistent data management

Our system required databases that holds room and user's information. Any changes with user will be updated in databases. Cube patterns and information of single player stages will
hold in hard drive of the client, however, any progress in single player stage will also holds in user's information database to reach another computer in another game instance. Cube faces images and icons related to the game will hold in hard drive of the client as a ".png" format. Game musics and sounds will also hold in hard drive of the client as ".wav" format.

2.4 Access control and security

In order to play the game in multiplayer mode, the user needs to create an account. The account credentials will be stored in the database as previously mentioned in section 2.3. The passwords for each user will be stored in the database as salted SHA-256 hash values. During registration, an e-mail verification is required to prevent players to open redundant accounts for spamming game rooms or the database system itself.

Additionally the game has private rooms in multiplayer mode, these rooms are accessed via a special password dedicated to room. The system will request this password for each player that tries to join to any private room.

The inputs for login credentials will be checked for containing any kind of SQL keywords in order to prevent casual SQL injection attacks. The server can also blacklist a player based on IP address if there are multiple failed login attempts.

2.5 Boundary conditions

2.5.1 Starting of The Game

When the game is launched it checks for the local configuration files in the file system that the game is initialized. It will give an error if this configuration file is missing and proceed by creating the necessary configuration files for starting the game. If the configuration files are not missing, the game starts normally and initializes the single player settings directly.

2.5.2 Server Availability of Multiplayer System

The multiplayer system will check whether the server is currently responding or not. This can happen due to maintenance or any kind of failure that the server may have. The current situation will be prompted to the player and players will only be able to play in single player mode until the server starts responding again.

2.5.3 Server Failure During a Multiplayer Game

If the server crashes during a Multiplayer Game instance, all the players will be disconnected. The game rooms will be destroyed as well. Any kind of progress such as experience that is already gained after a game ends will be saved only if the game has properly ended. Any kind of server unavailability during a game will not be able to update player data since the game is not ended yet.

3.Subsystem Services

3.1 Network Subsystem

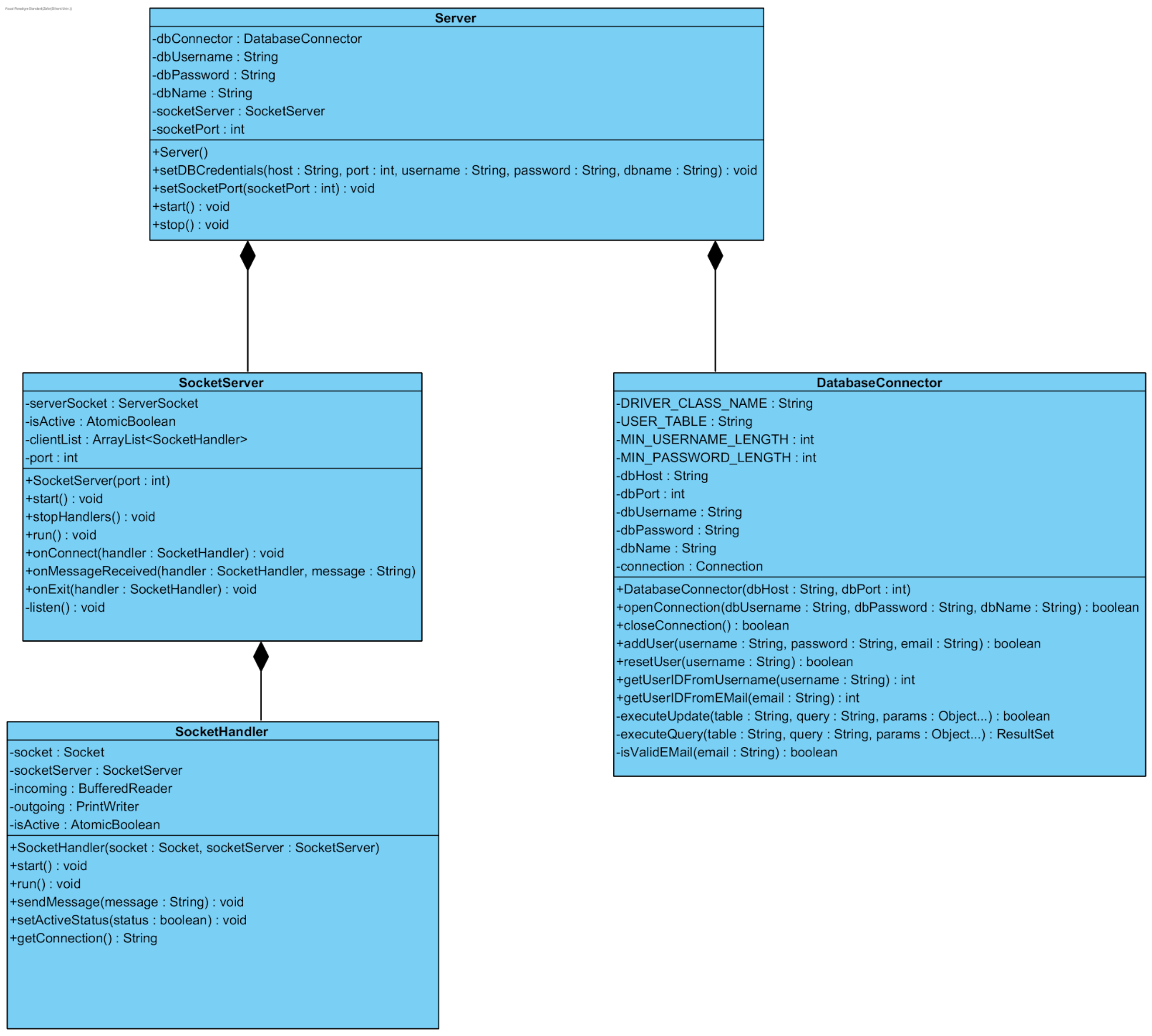

Server
-dbConnector : DatabaseConnector
-dbUsername : String
-dbPassword : String
-dbName : String
-socketServer : SocketServer
-socketPort : int
+Server()
+setDBCredentials(host : String, port : int, username : String, password : String, dbname : String) : void
+setSocketPort(socketPort : int) : void
+start() : void
+stop() : void
SocketServer
-serverSocket : ServerSocket
-isActive : AtomicBoolean
-clientList : ArrayList<SocketHandler>
-port : int
+SocketServer(port : int)
+start() : void
+stopHandlers() : void
+run() : void
+onConnect(handler : SocketHandler) : void
+onMessageReceived(handler : SocketHandler, message : String)
+onExit(handler : SocketHandler) : void
-listen() : void
DatabaseConnector
-DRIVER_CLASS_NAME : String
-USER_TABLE : String
-MIN_USERNAME_LENGTH : int
-MIN_PASSWORD_LENGTH : int
-dbHost : String
-dbPort : int
-dbUsername : String
-dbPassword : String
-dbName : String
-connection : Connection
+DatabaseConnector(dbHost : String, dbPort : int)
+openConnection(dbUsername : String, dbPassword : String, dbName : String) : boolean
+closeConnection() : boolean
+addUser(username : String, password : String, email : String) : boolean
+resetUser(username : String) : boolean
+getUserIDFromUsername(username : String) : int
+getUserIDFromEMail(email : String) : int
-executeUpdate(table : String, query : String, params : Object...) : boolean
-executeQuery(table : String, query : String, params : Object...) : ResultSet
-isValidEMail(email : String) : boolean
SocketHandler
-socket : Socket
-socketServer : SocketServer
-incoming : BufferedReader
-outgoing : PrintWriter
-isActive : AtomicBoolean
+SocketHandler(socket : Socket, socketServer : SocketServer)
+start() : void
+run() : void
+sendMessage(message : String) : void
+setActiveStatus(status : boolean) : void
+getConnection() : String

Figure 1: *Network Subsystem*

3.1.1 Server Class

This class is the main class which holds the Server properties of the game.
This class has instances of DatabaseConnector and SocketServer classes which are the main systems of the server of the game.

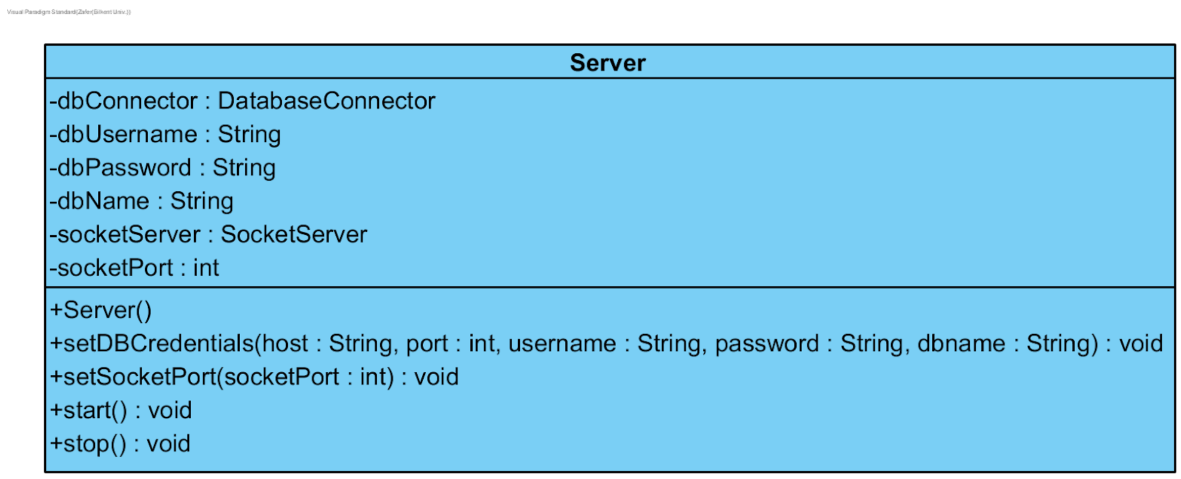

Visual Paradigm Standard(Zafer(Bilkent Univ.))
Server
-dbConnector : DatabaseConnector
-dbUsername : String
-dbPassword : String
-dbName : String
-socketServer : SocketServer
-socketPort : int
+Server()
+setDBCredentials(host : String, port : int, username : String, password : String, dbname : String) : void
+setSocketPort(socketPort : int) : void
+start() : void
+stop() : void

Figure 2: *Server Class*

Server:

Attribute:

private DatabaseConnector db : Holds the instance of DatabaseConnector object.

private String dbUsername : Holds the database connection username.

private String dbPassword : Holds the database connection password.

private String dbName : Holds the database name.

private SocketServer socketServer : Holds the instance of SocketServer object.

private int socketPort : Holds the socket connection port.

Constructor:

Server() : Constructor for Server Class.

Methods:

void setDBCredentials(String host, int port, String username, String password, String dbname)
This method initializes the database connection credentials.

host : Hostname of the Database Server
port : Port of the Database Server
username : Username for the Database Server
password : Password for the Database Server
dbname : Database Name on the Database Server

void setSocketPort(int socketPort)
This method initializes the port of socket connections.
socketPort : The connection port for the Socket Server.

void start()
This method starts the main systems of the server. (Database and SocketServer)

void stop()
This method stops the main systems of the server. (Database and SocketServer)

3.1.2 SocketServer Class

This class is used to hold socket operations.
It has a collection of ServerSocketHandlers and it is responsible for managing the socket connections with the clients.
It receives operation requests from clients, processes them, and sends responses.
This class is also an extension of Thread class. Therefore, it runs on a different thread other than the main thread of the server application.

Visual Paradigm Standard(Zafer(Bilkent Univ.))

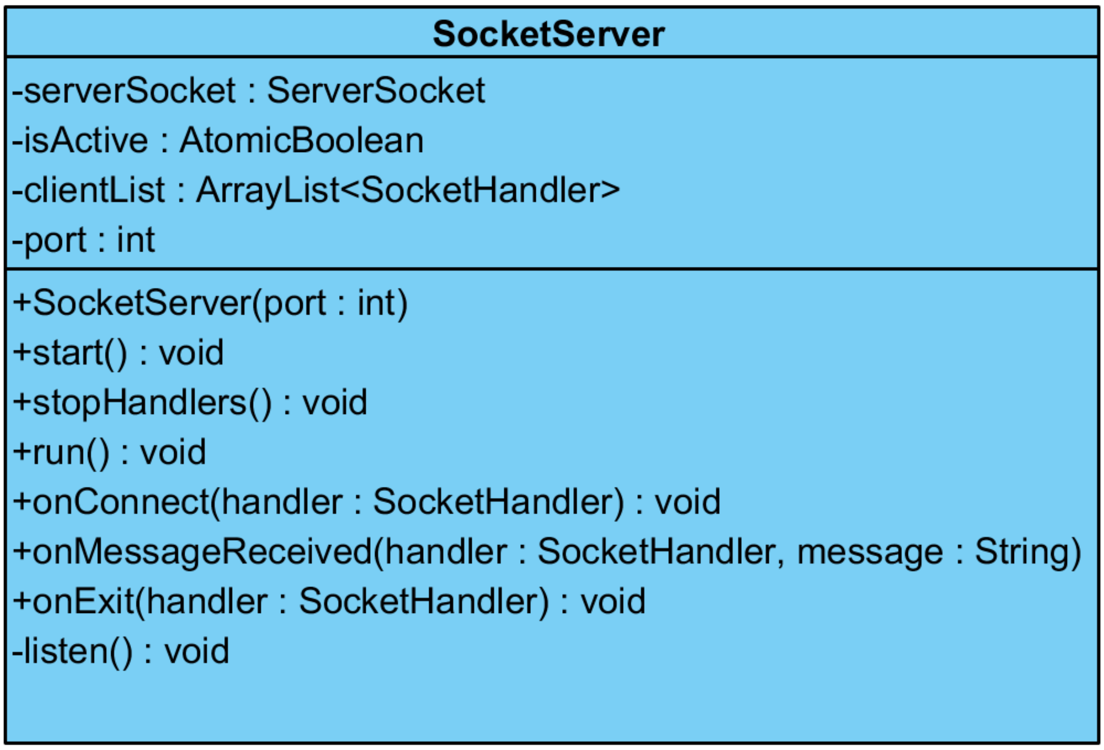

SocketServer
-serverSocket : ServerSocket
-isActive : AtomicBoolean
-clientList : ArrayList<SocketHandler>
-port : int
+SocketServer(port : int)
+start() : void
+stopHandlers() : void
+run() : void
+onConnect(handler : SocketHandler) : void
+onMessageReceived(handler : SocketHandler, message : String)
+onExit(handler : SocketHandler) : void
-listen() : void

Figure 3: *SocketServer Class*

<u>**SocketServer:**</u>

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

private ServerSocket serverSocket : Holds the instance of socket server object.

private AtomicBoolean isActive : Holds the status of the socket handler.

private ArrayList<ServerSocketHandler> clientList : Holds the collection of ServerSocketHandler instances.

private int port : Holds the socket connection port.

<u>**Constructors:**</u>

SocketServer(int port)
Constructor for SocketServer Class.
port : The connection port of the socket connections.

Methods:

public synchronized void start()
This method start the thread of SocketServer Class, also it initializes the isActive property as true to start

void stopHandlers()
This method stops the ServerSocketHandlers in the collection.

public void run()
This method runs the operations of the Thread of SocketServer class.

void onConnect(ServerSocketHandler handler)
This method is a callback method which is called when a client connected.
handler : The handler for the socket connection with the client.

void onMessageReceived(ServerSocketHandler handler, String message)
This method is a callback method which is called when a client sent a message through the socket connection.
handler : The handler for the socket connection with the client.
message : The message came through the socket connection.

void onExit(ServerSocketHandler handler)
This method is a callback method which is called when a client disconnected.
handler : The handler for the socket connection with the client.

private void listen()
This method is responsible for listening the incoming socket connections.
It accepts the incoming socket connection requests, and adds them to the collection.
It also starts different threads for each socket connection.

3.1.3 ServerSocketHandler Class

This class is used to handle socket connections.

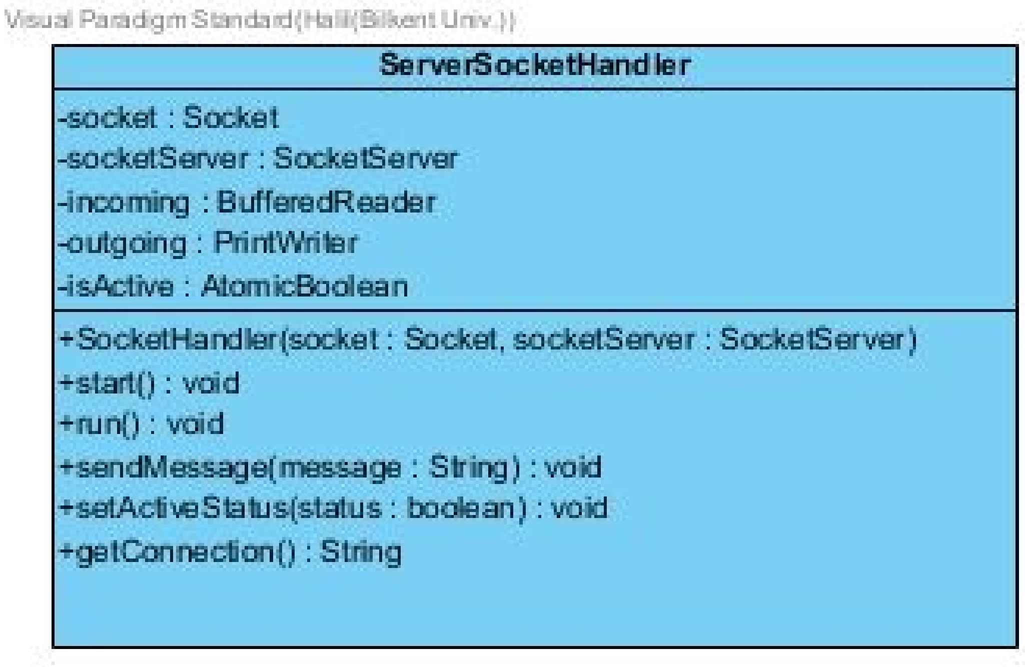

Visual Paradigm Standard(Halil(Bilkent Univ.))
ServerSocketHandler
-socket : Socket
-socketServer : SocketServer
-incoming : BufferedReader
-outgoing : PrintWriter
-isActive : AtomicBoolean
+SocketHandler(socket : Socket, socketServer : SocketServer)
+start() : void
+run() : void
+sendMessage(message : String) : void
+setActiveStatus(status : boolean) : void
+getConnection() : String

Figure 4: *ServerSocketHandler Class*

ServerSocketHandler:

Attributes:

private Socket socket : Holds the instance of socket object of the client.

private SocketServer socketServer : Holds the instance of socket server object.

private BufferedReader incoming : Holds the incoming stream from the client.

private PrintWriter outgoing : Holds the outcoming stream to the client.

private AtomicBoolean isActive : Holds the status of the socket handler.

Constructors:

ServerSocketHandler(Socket socket, SocketServer socketServer)
Constructor for ServerSocketHandler Class.
socket : The socket object for the connection.
socketServer : The instance of the SocketServer which holds the collection of ServerSocketHandlers.

Methods:

public synchronized void start()

This method starts the thread of ServerSocketHandler.

public void run()
This method handles the incoming messages over socket connection.

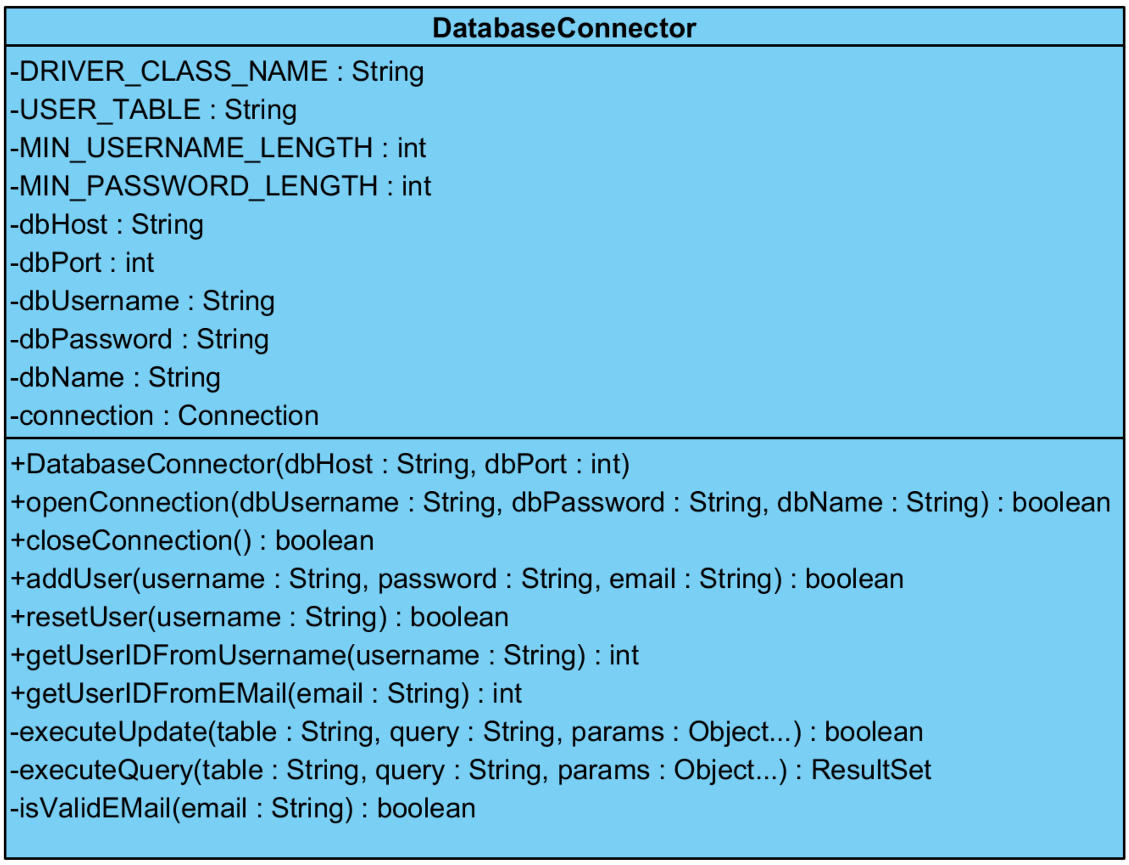
void sendMessage(String message)
This method sends message to the client over the socket connection.
message : The string message.
void setActiveStatus(boolean status)
This method sets the active status of the ServerSocketHandler.
status : The active status.

String getConnectionIP()
This method returns the IP of the client.
Returns the IP of the client.

3.1.4 DatabaseConnector Class

This class is used to handle database operations.
It has specific methods for predefined database operations.
It also has dynamic query methods which can be used querying DB server easily.

Visual Paradigm Standard(Zafer(Bilkent Univ.))



```
DatabaseConnector
-DRIVER_CLASS_NAME : String
-USER_TABLE : String
-MIN_USERNAME_LENGTH : int
-MIN_PASSWORD_LENGTH : int
-dbHost : String
-dbPort : int
-dbUsername : String
-dbPassword : String
-dbName : String
-connection : Connection
+DatabaseConnector(dbHost : String, dbPort : int)
+openConnection(dbUsername : String, dbPassword : String, dbName : String) : boolean
+closeConnection() : boolean
+addUser(username : String, password : String, email : String) : boolean
+resetUser(username : String) : boolean
+getUserIDFromUsername(username : String) : int
+getUserIDFromEMail(email : String) : int
-executeUpdate(table : String, query : String, params : Object...) : boolean
-executeQuery(table : String, query : String, params : Object...) : ResultSet
-isValidEMail(email : String) : boolean
```

Figure 5: *DatabaseConnector Class*

DatabaseConnector:

Attributes:

private static final String DRIVER_CLASS_NAME = "com.mysql.cj.jdbc.Driver" : Holds the database connector driver class name.

private static final String USER_TABLE = "users" : Holds the name of users table on the database.

private static final int MIN_USERNAME_LENGTH = 4 : Holds the minimum length for the username of a new record.

private static final int MIN_PASSWORD_LENGTH = 8 : Holds the minimum length for the password of a new record.

private String dbHost : Holds the hostname for the database connection.

private int dbPort : Holds the connection port for the database connection.

private String dbUsername : Holds the username for the database connection.

private String dbPassword : Holds the password for the database connection.

private String dbName : Holds the database name for the database connection.

Constructors

DatabaseConnector(String dbHost, int dbPort)
Constructor for DatabaseConnector Class.
dbHost : The hostname for the Database Connection.
dbPort : The port for the Database Connection.

Methods:

boolean openConnection(String dbUsername, String dbPassword, String dbName)
This method opens the database connection with the required credentials.
dbUsername : The username for the database connection.
dbPassword : The password for the database connection.
dbName : The database name for the database connection.
Returns True if the connection is successful, otherwise is returns False.

boolean closeConnection()
This method closes the database connection.
Returns True if the connection is closed successfully, otherwise is returns False.

boolean addUser(String username, String password, String email)
This method adds a new user to the database.
username : The username for the new record.
password : The password for the new record.
email : The e-mail for the new record.
Returns True if the operation is completed successfully, otherwise is returns False.

boolean resetUser(String username)
This method resets the progress of the user.
username : The username of the user.
Returns True if the operation is completed successfully, otherwise is returns False.

int getUserIDFromUsername(String username)
This method finds the SQL ID of the user from its username.
username : The username of the user.
Returns the SQL ID of the user if the operation is completed successfully, otherwise is returns -1.

int getUserIDFromEMail(String email)
This method finds the SQL ID of the user from its e-mail.
email : The username of the user.
Returns the SQL ID of the user if the operation is completed successfully, otherwise is returns -1.

private boolean executeUpdate(String table, String query, Object... params)
This method queries manipulating queries on the desired table.
table : The table name which the operation runs on.
query : The SQL query.
params : The constraint parameters for the SQL query.
Returns True if the operation is completed successfully, otherwise is returns False.

private ResultSet executeQuery(String table, String query, Object... params)
This method queries selecting queries on the desired table.
table : The table name which the operation runs on.
query : The SQL query.
params : The constraint parameters for the SQL query.
Returns a ResultSet object with the desired data in it if the operation completed successfully, otherwise it returns null.

private boolean isValidEMail(String email)
This method checks if an e-mail address is valid.
email : The e-mail address.
Returns True if the e-mail address is valid, otherwise it returns False.

3.1.5 ClientSocketHandler

This class is used to handle socket connections.

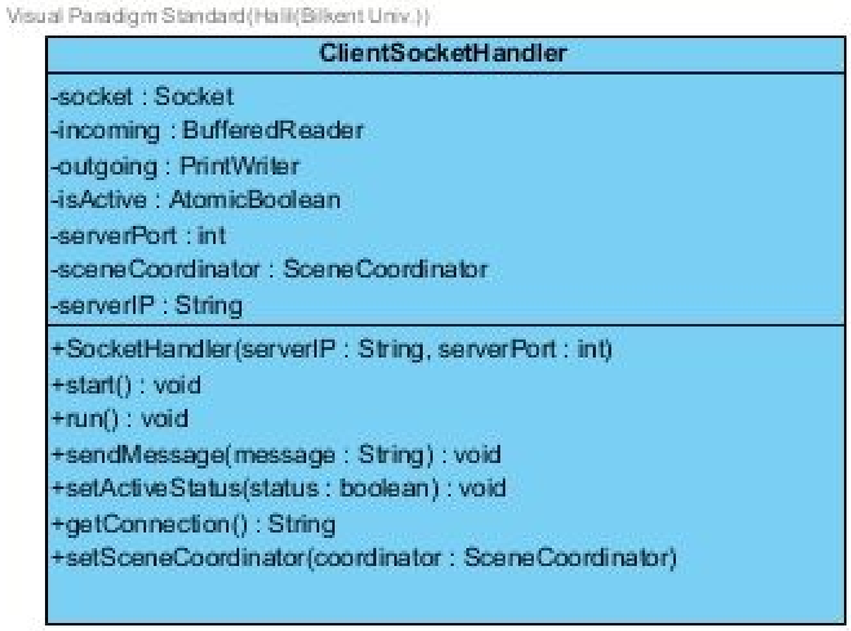

Visual Paradigm Standard(Halil(Bilkent Univ.))
ClientSocketHandler
-socket : Socket
-incoming : BufferedReader
-outgoing : PrintWriter
-isActive : AtomicBoolean
-serverPort : int
-sceneCoordinator : SceneCoordinator
-serverIP : String
+SocketHandler(serverIP : String, serverPort : int)
+start() : void
+run() : void
+sendMessage(message : String) : void
+setActiveStatus(status : boolean) : void
+getConnection() : String
+setSceneCoordinator(coordinator : SceneCoordinator)

Figure 6: *ClientSocketHandler Class*

ClientSocketHandler:

Attributes:

private String serverIP : Holds the server IP.

private int serverPort : Holds the server port.

private Socket socket : Holds the instance of socket object of the client.

private BufferedReader incoming : Holds the incoming stream from the client.

private PrintWriter outgoing : Holds the outcoming stream to the client.

private AtomicBoolean isActive : Holds the status of the socket handler.

private SceneCoordinator sceneCoordinator : Holds scene coordinator instance.

Constructors:

ClientSocketHandler(String serverIP, int serverPort)
Constructor for SocketHandler Class.
serverIP : IP of the server.
serverPort : Connection port for the socket connection.

Methods:

public synchronized void start()
This method starts the thread of ClientSocketHandler.

public void run()
This method handles the incoming messages over socket connection.

void sendMessage(String message)
This method sends message to the client over the socket connection.
message : The string message.

void setActiveStatus(boolean status)
This method sets the active status of the ServerSocketHandler.
status : The active status.

void setSceneCoordinator(SceneCoordinator coordinator)
This method sets the scene coordinator for socket handler.
coordinator : SceneCoordinator instance.

3.1.6 SceneCoordinator

This class holds socket connection and the active scene.
Therefore, it can act as a bridge between server updates and the game screens.

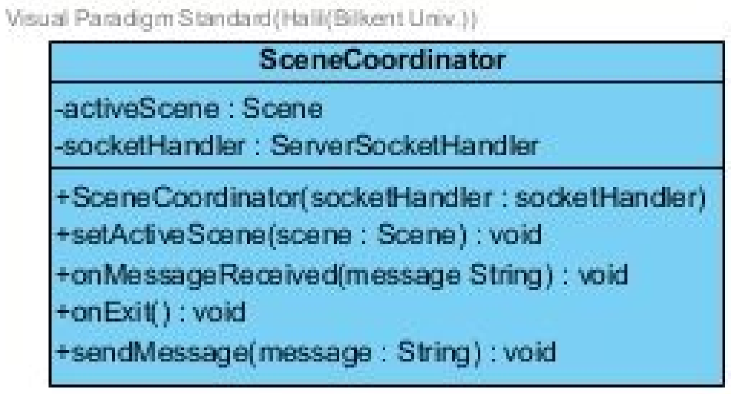

Visual Paradigm Standard(Halil(Bilkent Univ.))
SceneCoordinator
-activeScene : Scene
-socketHandler : ServerSocketHandler
+SceneCoordinator(socketHandler : socketHandler)
+setActiveScene(scene : Scene) : void
+onMessageReceived(message String) : void
+onExit() : void
+sendMessage(message : String) : void

Figure 7: SceneCoordinator

SceneCoordinator:

Attributes:

private Scene activeScene : Holds reference to active scene instance.

private SocketHandler socketHandler : Holds reference to socket handler instance.

<u>Constructors:</u>

public SceneCoordinator(SocketHandler socketHandler)
Constructor for SceneCoordinator
socketHandler : Socket handler instance.

<u>Methods:</u>

public void setActiveScene(Scene scene)
This method sets the active scene of the coordinator.
scene : Active scene.

public void onMessageReceived(String message)
This method is a callback method which is called when a client sent a message through the socket connection.
message : The message came through the socket connection.

public void onExit()
This method is a callback method which is called when a client disconnected.

public void sendMessage(String message)
This method sends message to the client over the socket connection.
message : The string message.

3.2 Menu Subsystem

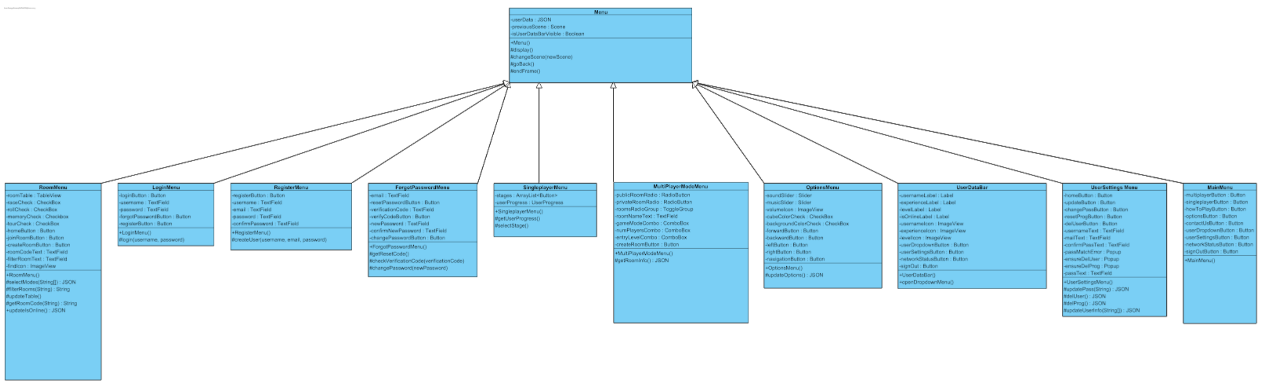


Figure 8: Menu class diagram

3.2.1 User Settings Menu

This class is used to handle user settings menu and send to changes about user settings to Menu Class.

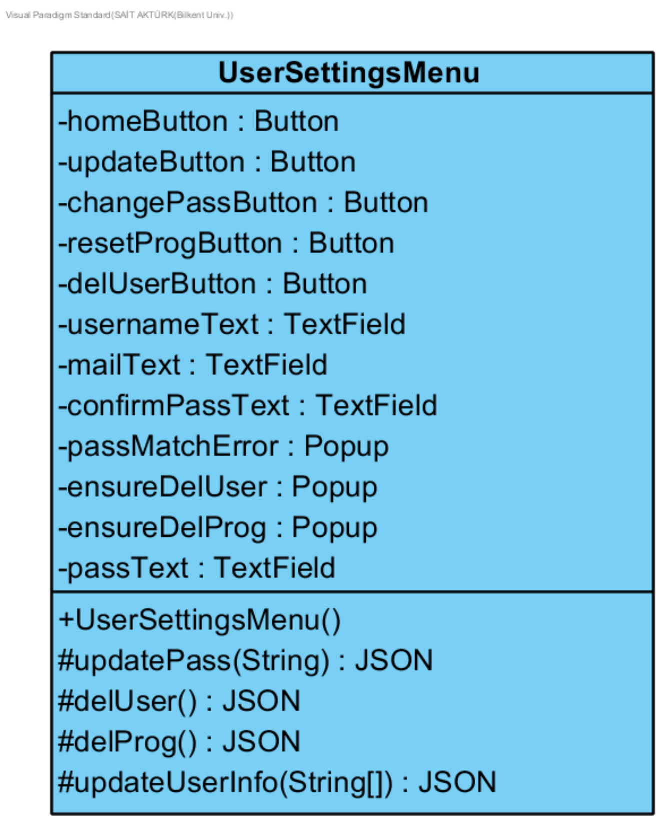


Figure 9: UserSettingsMenu

UserSettingsMenu:

Attributes:

private Button homeButton : This menu item will enable to endFrame() function, when this item is clicked.

private Button updateButton : This menu item will update information about valid changed username and mail. This item will activate updateUserInfo(String[]) to send updated info.

private Button changePassButton : This menu item will update user's password about valid new password. This item will activate updatePass(String) to send new password.

private Button resetProgButton : This menu item will activate delProg(), when this button is clicked. Also, a pop-up will activate to ensure about deletion.

private Button delUserButton : This menu item will activate delUser(), when this button is clicked. Also, a pop-up will activate to ensure about deletion.

private TextField usernameText: This menu item will get new valid username.

private TextField mailText : This menu item will get new valid mail.

private TextField confirmPassText : This menu item will get new password with second time to ensure user to set new password.

private Popup passMatchError: If user’s new passwords does not match then menu initialize this popup screen to give information about unmatched error.

private Popup ensureDelUser: This menu will be appear, when user wants to delete his/her account to alert user.

private Popup ensureDelProg: This menu will be appear, when user wants to delete his/her progress to alert user.

Constructors:

public UserSettingsMenu(): This is the constructor of userSettingsMenu, basically, all menu item will be initialized and screen’s scene will be designed.

Methods:

protected JSON updatePass(String): This function sends update password as a JSON file.

protected JSON updateUserInfo(String, String) : This function sends updated username or mail or both as a JSON file.

protected JSON delUser(): This function sends flag for deleting user’s account as a JSON file.

protected JSON delProg(): This function sends flag for deleting user’s progress as a JSON file

3.2.2 Options Menu

This class is used to handle options menu and send to changes about options to Menu Class.

Visual Paradigm Standard(SAİT AKTÜRK(Bilkent Univ.))

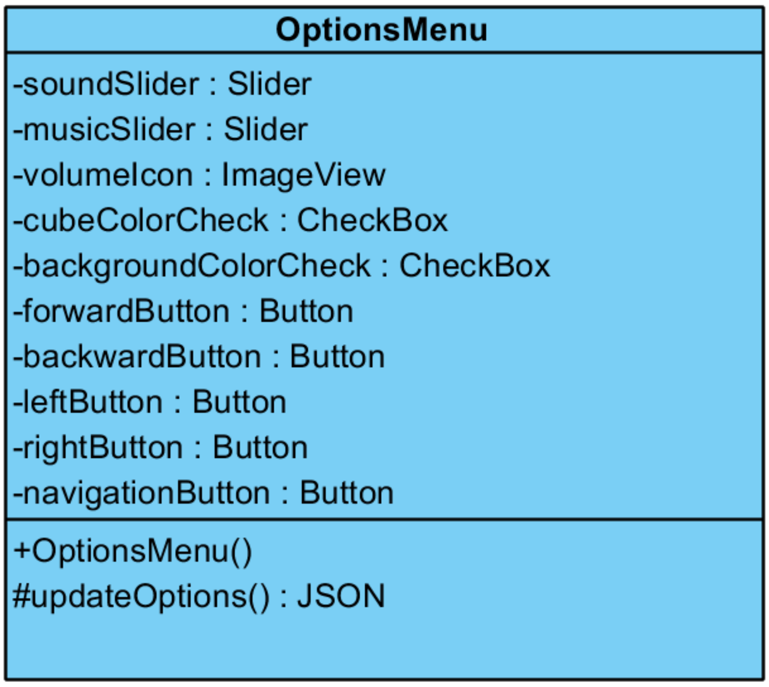


Figure 10: Options Menu Class

<u>OptionsMenu:</u>

<u>Attributes :</u>

private Slider soundSlider : This menu item will control the volume of sounds in the game. User could adjust to sound volume between 0 and 100.

private Slider musicSlider : This menu item will control the volume of musics in the game. User could adjust to music volume between 0 and 100.

private ImageView volumeIcon : This menu item will show volume of sounds and musics. There will be two different volumeIcon that one of the icons will belong sounds of the game and other icon will belong musics of the game. User could pressed these icons to mute sounds or musics. The user could also press these icons to turn on volume of musics and sounds.

private CheckBox cubeColorCheck : This menu item will help user to select color of cubes that user wants to change.

private CheckBox backgroundColorCheck : This menu item will help user to select color of background of the game that user wants to change.

private Button forwardButton : This menu item will help user to determine keyboard key that responsible forward turn of components of cube.

private Button forwardButton : This menu item will help user to determine keyboard key that responsible forward turn of components of cube.

private Button backwardButton : This menu item will help user to determine keyboard key that responsible backward turn of components of cube.

private Button leftButton : This menu item will help user to determine keyboard key that responsible left turn of components of cube.

private Button rightButton : This menu item will help user to determine keyboard key that responsible right turn of components of cube.

private Button navigationButton : This menu item will help user to determine mouse button that responsible selection of grid in board.

<u>Constructors:</u>

public OptionsMenu() : This is the constructor of optionsMenu, basically, all menu item will be initialized and screen’s scene will be designed.

<u>Methods:</u>

protected JSON updateOptions() : This function will return any changes related to OptionsMenu as a JSON file.

3.2.3 Multiplayer Mode Menu

This class is used to handle multi player menu and send to changes about options to Menu Class.

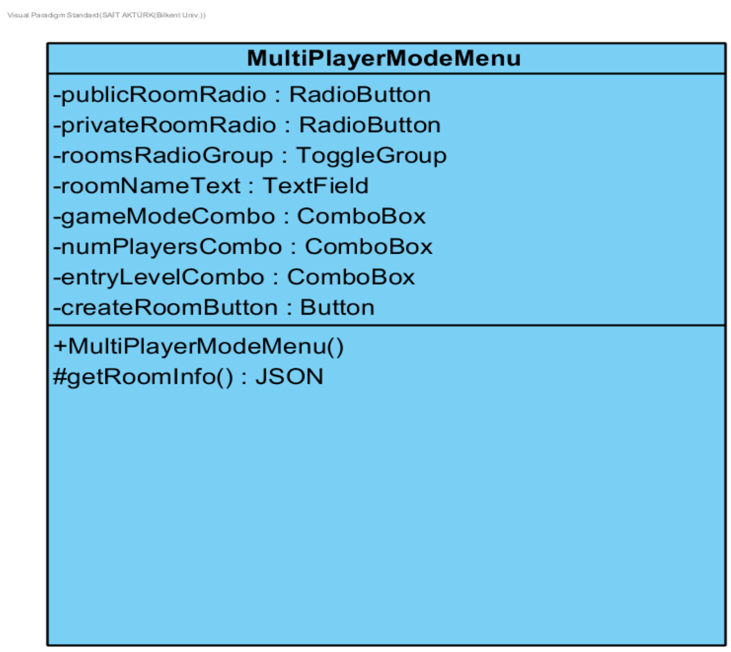

Visual Paradigm Standard(SAİT AKTÜRK(Bilkent Univ.))
MultiPlayerModeMenu
-publicRoomRadio : RadioButton
-privateRoomRadio : RadioButton
-roomsRadioGroup : ToggleGroup
-roomNameText : TextField
-gameModeCombo : ComboBox
-numPlayersCombo : ComboBox
-entryLevelCombo : ComboBox
-createRoomButton : Button
+MultiPlayerModeMenu()
#getRoomInfo() : JSON

Figure 11: MultiplayerModeMenu Class

MultiPlayerModeMenu:

Attributes:

private RadioButton publicRoomRadio : This menu item will help user to select room type that whether user wants to create public room or not. When user selects this button, user could not select private room radio button.

private RadioButton privateRoomRadio : This menu item will help user to select room type that whether user wants to create private room or not. When user selects this button, user could not select public room radio button.

private ToggleGroup roomRadioGroup : This menu item will enforce two radio buttons that privateRoomRadio and publicRoomRadio to select only one button.

private TextField roomNameText : This menu item will specify room’s name that will be created after all specifications filled. This items could not be blank text.

private ComboBox gameModeCombo : This menu item will specify room’s mode that could be “Roll”, “Memory”,   “Race” and “Tournament”..

private ComboBox numPlayersCombo : This menu item will specify number of players for room that will be created. This menu item has value between 2 and 8.

private ComboBox entryLevelCombo : This menu item will specify minimum level of attendants of rooms.This menu item could be at least level of creator of room.

<u>Constructors:</u>

public MultiPlayerModeMenu(): This is the constructor of multiPlayerModeMenu, basically, all menu item will be initialized and screen's scene will be designed.

<u>Methods:</u>

protected JSON getRoomInfo(): This function will return room's specifications as JSON file.

3.2.4 Room Menu

This class is used to handle room menu and send to changes about options to Menu Class.

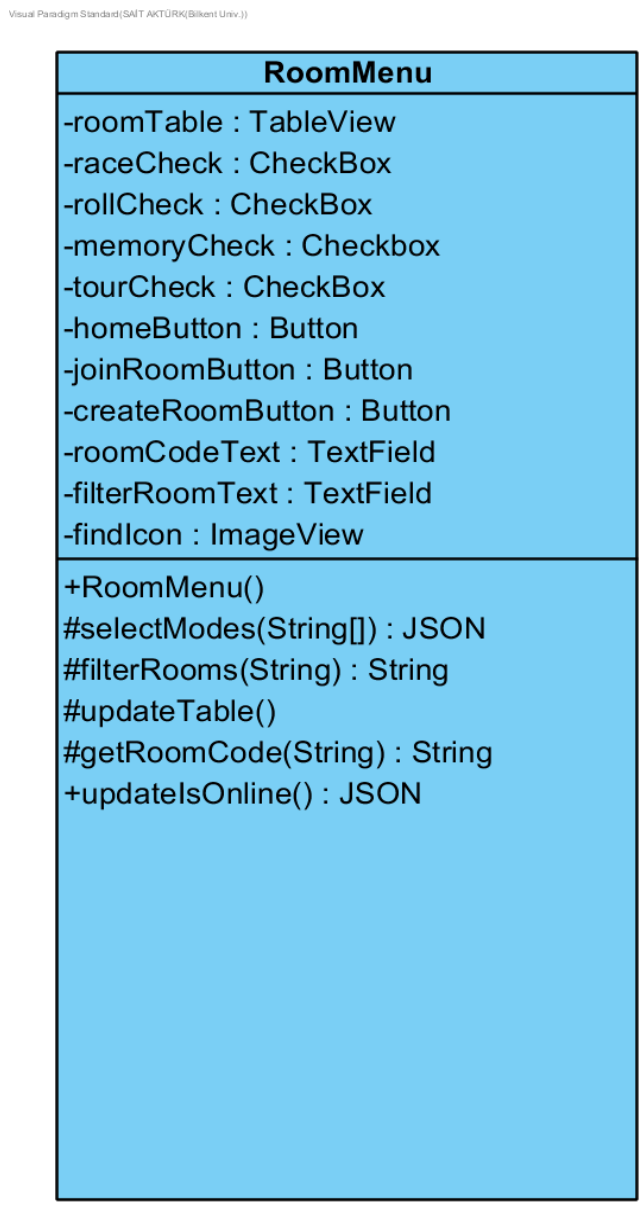


Figure 12: RoomMenu class

RoomMenu:

Attributes:

private TableView roomTable : This menu item will show all room created in server. This table view clickable to select rooms. In this roomTable, there will be columns called “Room’s Name”, “Game Mode”, “Owner”, “Players” and “Entrance Level”.

private CheckBox raceCheck : This menu item will filter room’s list to help user to find rooms that related to “race” mode. This menu item will be preselected as default option.

private CheckBox rollCheck : This menu item will filter room’s list to help user to find rooms that related to “roll” mode. This menu item will be preselected as default option.

private CheckBox memoryCheck : This menu item will filter room’s list to help user to find rooms that related to “memory” mode. This menu item will be preselected as default option.

private CheckBox tourCheck : This menu item will filter room’s list to help user to find rooms that related to “tournament” mode. This menu item will be preselected as default option.

private Button homeButton : This menu item will call “endFrame()” in the RoomMenu to return home menu.

private Button joinRoomButton : This menu item has property that if a room selected from roomTable, then user will click this button to attend to room section.

private Button createRoomButton: This menu item will open MultiPlayerModeMenu() to specify room properties.

private TextField roomCodeText : This menu item could be editable with user to find specific private room that friends could play together with privately.

private TextField filterRoomText : This menu item could be editable to find rooms that has exact same or partial same name will be shown in roomTable(). This search only completes when user click to findIcon clicked.

private ImageView findIcon: This menu item will be clickable to find given room’s name with filterRoomText and activate updateTable() to show matched rooms.

__Constructors:__

public RoomMenu(): This is the constructor of RoomMenu, basically, all menu item will be initialized and screen’s scene will be designed.

__Methods:__

protected JSON selectModes(String [ ]): This function will take selected modes as an input and return a JSON file to update roomTable.

protected String filterRooms(String) : This function will take user’s speficied room name that user searches with, then will return that String input to updateTable.

protected void updateTable(String [ ]): This function will update roomTable with given input String filters.

protected String getRoomCode(String) : This function will getRoomCode to find private room in server.

public JSON updateIsOnline(): This function will send connection status to server

3.2.5 Login Menu

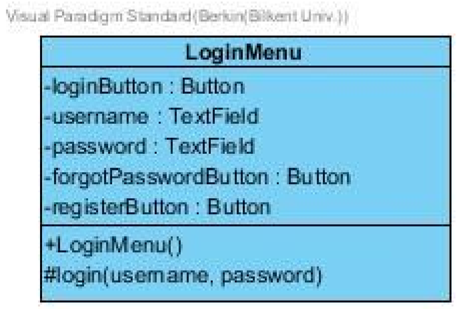


Figure 13: LoginMenu Class

<u>**LoginMenu:**</u>

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

private Button loginButton: This button triggers the login() method.

private TextField username: This text field holds username of the user.

private TextField password: This text field holds password of the user.

private Button forgotPasswordButton: This button directs the user to the forgot password menu.

private Button registerButton: This button directs the user to the register menu.

<u>**Constructors:**</u>

public LoginMenu(): Constructor for login menu.

<u>**Methods:**</u>

protected boolean login(username, password): This method send the username and password to the server for logging in.

3.2.6 Register Menu

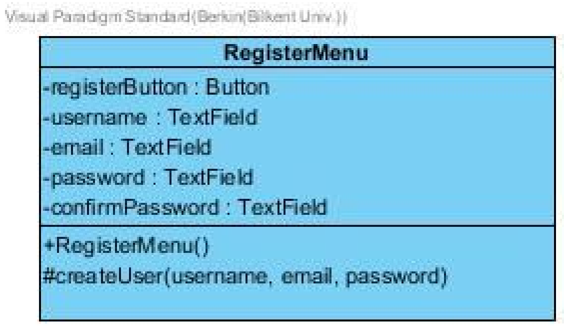

Visual Paradigm Standard(Berkin(Bilkent Univ.))
RegisterMenu
-registerButton : Button
-username : TextField
-email : TextField
-password : TextField
-confirmPassword : TextField
+RegisterMenu()
#createUser(username, email, password)

Figure 14: RegisterMenu Class

RegisterMenu:

Attributes:

private Button registerButton: This button submits the registration form and triggers the createUser() method.

private TextField username: This text field holds username of the new account.

private TextField email: This text field holds email of the new account.

private TextField password: This text field holds password of the new account.

private TextField confirmPassword: This text field is for the user to re-enter their password to reduce the password errors

Constructors:

public RegisterMenu(): Constructor of register menu.

Methods:

protected boolean createUser(username, email, password): This method sends the account information in the registration form to the server.

3.2.7 Forgot Password Menu

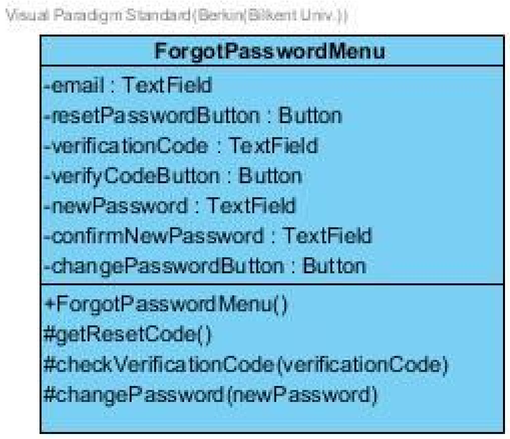

Visual Paradigm Standard(Berkin(Bilkent Univ.))
ForgotPasswordMenu
-email : TextField
-resetPasswordButton : Button
-verificationCode : TextField
-verifyCodeButton : Button
-newPassword : TextField
-confirmNewPassword : TextField
-changePasswordButton : Button
+ForgotPasswordMenu()
#getResetCode()
#checkVerificationCode(verificationCode)
#changePassword(newPassword)

Figure 15:ForgotPasswordMenu Class

ForgotPasswordMenu:

Attributes:

private TextField email: This text field is for user to enter their email for the verification code.

private Button resetPasswordButton: This button sends the email to the server.

private TextField verificationCode: This button is for the user to enter the verification code that they received in the mail.

private Button verifyCodeButton: This button sends the verification code the user entered to the server.

private TextField newPassword: This text field is for user to enter a new password for their account.

private TextField confirmNewPassword: This text field is for user to re-enter the new password for confirmation.

private Button changePasswordButton: This button sends the new password to the server.

Constructors:

public ForgotPasswordMenu(): Constructor for forgot password menu.

Methods:

protected boolean getResetCode(): This method is called by the resetPasswordButton and requests a verification code from the server.
protected boolean checkVerificationCode(verificationCode): This method is called to check if the entered verification code is correct.

protected boolean changePassword(newPassword): This method called by the changePasswordButton to send the new password to the server.

3.2.8 Main Menu

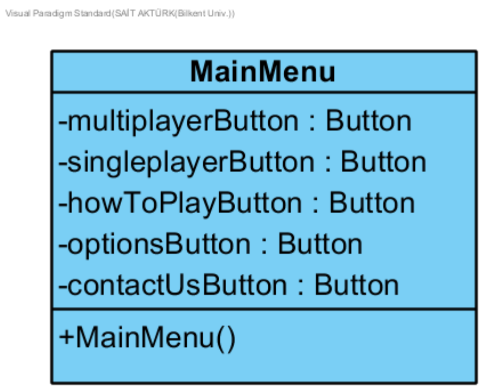

Visual Paradigm Standard(SAİT AKTÜRK(Bilkent Univ.))
MainMenu
-multiplayerButton : Button
-singleplayerButton : Button
-howToPlayButton : Button
-optionsButton : Button
-contactUsButton : Button
+MainMenu()

Figure 16: MainMenu Class

MainMenu:

Attributes:

private Button multiplayerButton: This button directs the user to the multiplayer menu.

private Button singlePlayerButton: This button directs the user to the singleplayer menu.

private Button howToPlayButton: This button directs the user to the how to play menu.

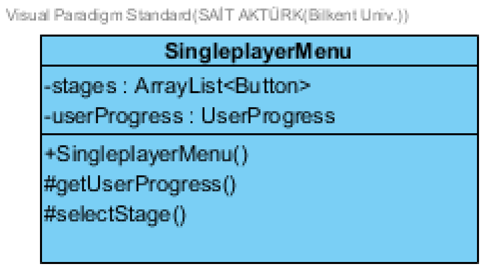
private Button optionsButton: This button directs the user to the options menu.

private Button contactUsButton: This button creates a form to send a message to the developers.

Constructors:

public MainMenu(): Constructor of main menu.

3.2.9 Singleplayer Menu



Visual Paradigm Standard(SAİT AKTÜRK(Bilkent Univ.))
SingleplayerMenu
-stages : ArrayList<Button>
-userProgress : UserProgress
+SingleplayerMenu()
#getUserProgress()
#selectStage()

Figure 17: SingleplayerMenu Class

MainMenu:

Attributes:

private ArrayList<Button> stages: This attribute contain the Stage objects to reach every stage in SingleplayerGame.

private UserProgress userProgress: This attribute holds the user's progress data in single player game.

Constructors:

public SingleplayerMenu(): Constructor of single player menu.

Methods:

protected void getUserProgress(): This method loads the user progress to the single player menu.

protected Stage selectStage(): This method loads the stage after selecting it on the menu.

3.2.10 UserDataBar

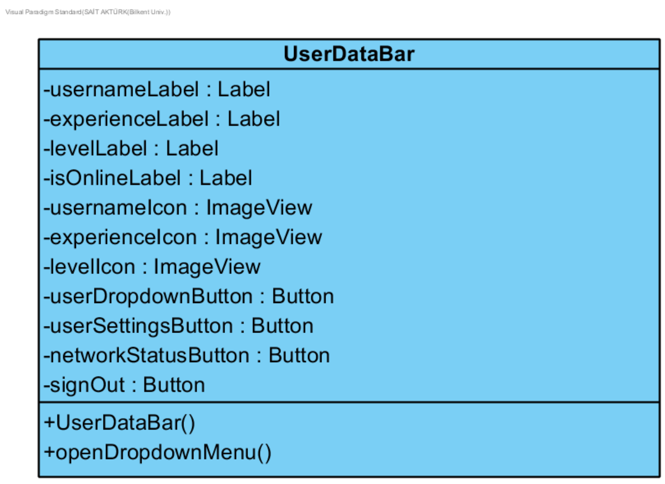

Visual Paradigm Standard(SAIT AKTÜRK(Bilkent Univ.))
UserDataBar
-usernameLabel : Label
-experienceLabel : Label
-levelLabel : Label
-isOnlineLabel : Label
-usernameIcon : ImageView
-experienceIcon : ImageView
-levelIcon : ImageView
-userDropdownButton : Button
-userSettingsButton : Button
-networkStatusButton : Button
-signOut : Button
+UserDataBar()
+openDropdownMenu()

Figure 18: UserDataBar Class

UserDataBar:

Attributes:

private Label usernameLabel: This label shows the username of the user.

private Label experienceLabel: This label shows the current experience points of the user.

private Label levelLabel: This label shows the experience level of the user.

private Label isOnlineLabel: This label shows the network status of the user. (Online or offline)

private ImageView usernameIcon: This image is the username icon.

private ImageView experienceIcon: This image is the experience icon.

private ImageView levelIcon: This image is the level icon.

private Button userDropdownButton: This button displays the dropdown menu when clicked on the username icon.

private Button userSettingsButton: This button is in the dropdown menu and directs to the user settings menu.

private Button networkStatusButton: This button is in the dropdown menu and changes the network status of the user.

private Button signOut: This button is in the dropdown menu and logs out of the account of the user.

Constructors:

public UserDataBar(): Constructor of user data bar.

Methods:

public openDropdownMenu(): Displays the dropdown menu in the user bar.

3.3 Game Logic Subsystem

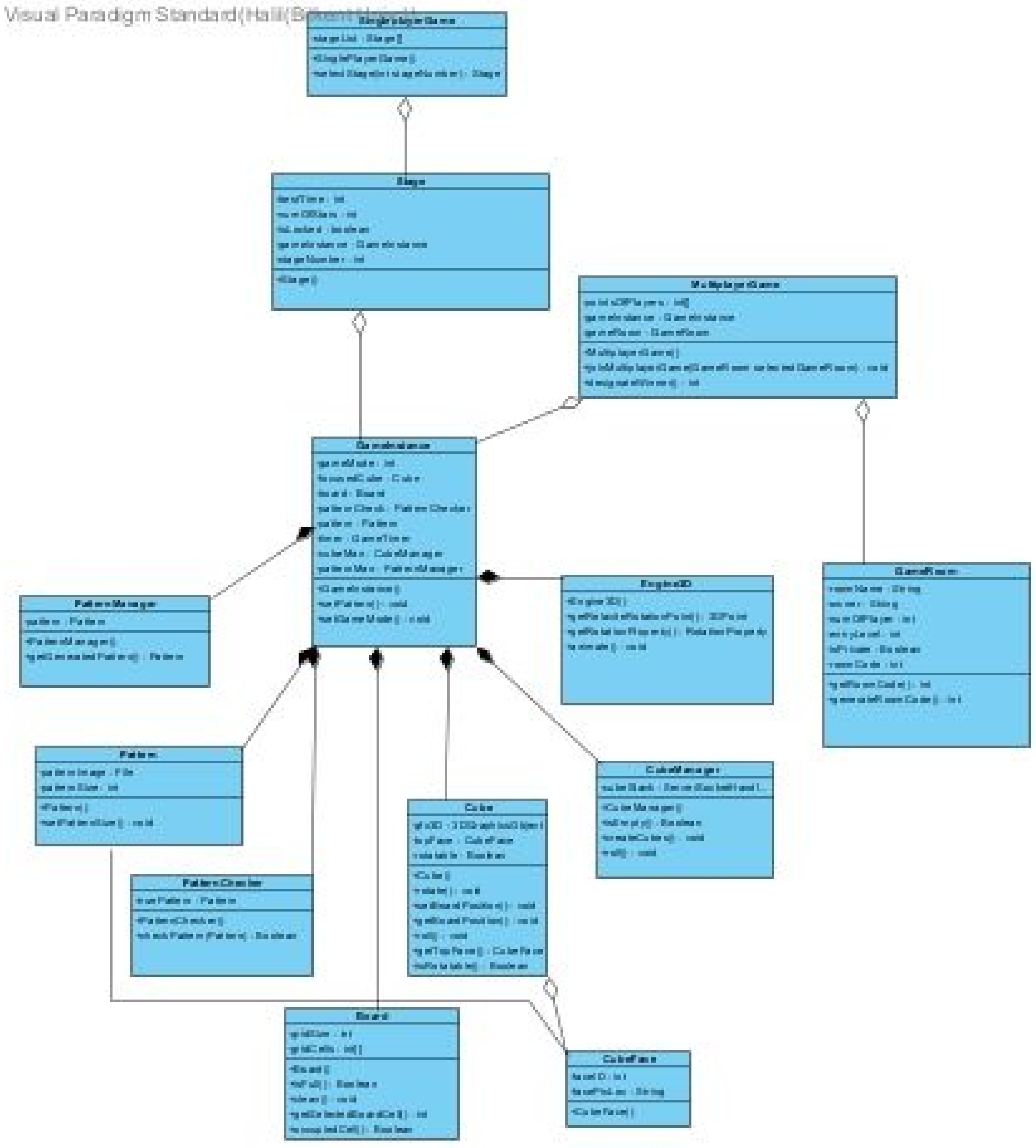


Figure 19: GameLogic Class Diagram

3.3.1 Stage

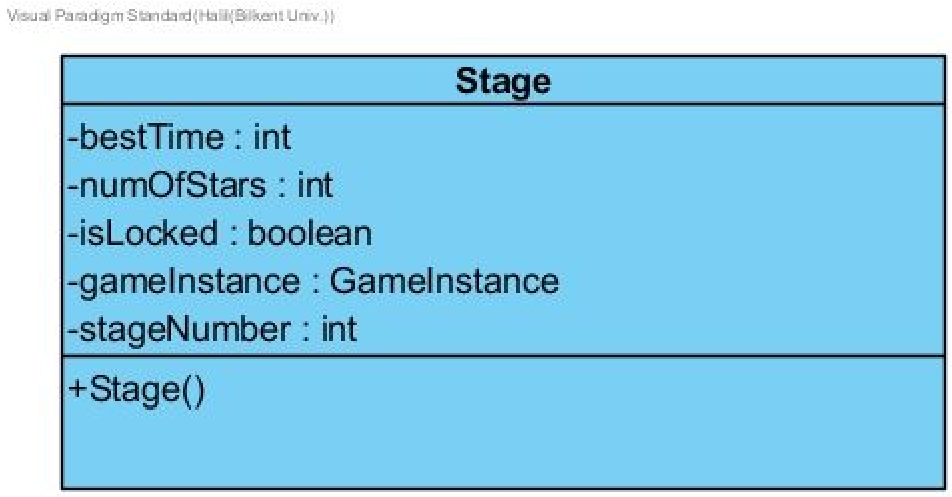


Figure 20: Stage Class

Stage:

Attributes:

private int bestTime: This attribute will be used to keep the best time of the stage that user did.

private int numOfStars: This attribute will be used to keep the number of stars of the stage according to time of the stage.

private boolean isLocked: This attribute will be used to keep the status of the stage about being locked for the user.

private GameInstance gameInstance: This attribute will be used to create a game instance for the stage with the specifications.

private int stageNumber: This attribute will be used as ID of the stage to keep track of the stages.

Constructor:

public Stage(): Constructor of stage of the singleplayer game to create a stage object.

3.3.2 Singleplayer Game

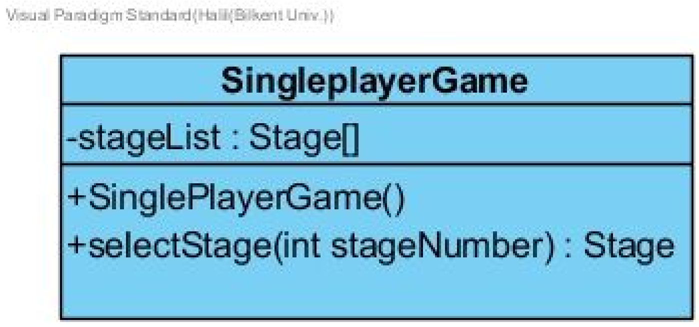

Visual Paradigm Standard(Halil(Bilkent Univ.))
SingleplayerGame
-stageList : Stage[]
+SinglePlayerGame()
+selectStage(int stageNumber) : Stage

Figure 21: SingleplayerGame Class

<u>SingleplayerGame:</u>

<u>Attributes:</u>

private Stage[] stageList: This attribute contain the Stage objects to reach every stage in SingleplayerGame

<u>Constructors:</u>

public SingleplayerGame(): Constructor of SingleplayerGame to create a singleplayer game object that provides singleplayer functionality

<u>Methods:</u>

public static Stage selectStage(int stageNumber): A function to select the stage by its stage number to and return that Stage object.

3.3.3 Multiplayer Game

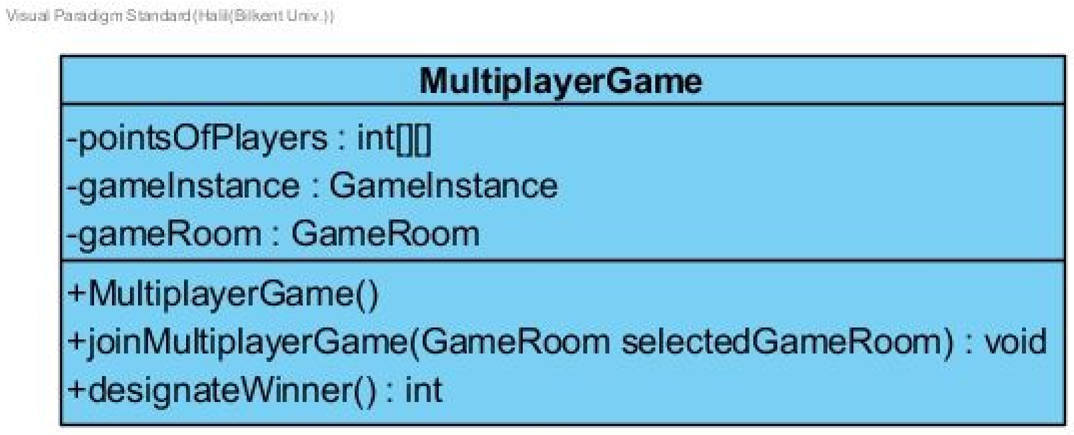

Visual Paradigm Standard (Halil(Bilkent Univ.))
MultiplayerGame
-pointsOfPlayers : int[][]
-gameInstance : GameInstance
-gameRoom : GameRoom
+MultiplayerGame()
+joinMultiplayerGame(GameRoom selectedGameRoom) : void
+designateWinner() : int

Figure 22: MultiplayerGame Class

MultiplayerGame:

Attributes:

private int[] pointsOfPlayers: It is an 2D array which is created to keep the points of players who are entered that multiplayer game with the user. It keep the points by summing them up in the index of a user.

private GameInstance gameInstance: This attribute will be used to create a game instance for a game room with its specifications.

private GameRoom gameRoom: This attribute will be used to get the specifications of game room which user entered for creating the game instance

Constructors:

public MultiplayerGame(): Constructor of multiplayer game to create the multiplayer game

Methods:

public static int designateWinner(): A function to get the user number of the winner when the game ends by selecting from pointsOfPlayers array.

public void joinMultiplayerGame(GameRoom selectedGameRoom): A function to be invoked when user select a game room to pass the selected game room information to multiplayer game.

3.3.4 Game Instance

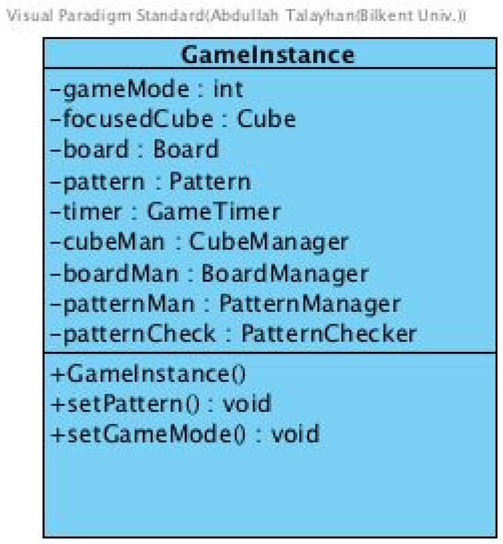


Figure 23: GameInstance Class

GameInstance:

Attributes:

private int gameMode: This attribute is used for selecting the game mode, it is represented as an integer.

private Cube focusedCube: the object that is currently selected.

private Board board: The board that the player is going to place the cubes.

private Pattern pattern: the correct pattern that the players are trying to match.

private GameTimer timer: Timer for the game.

private CubeManager cubeMan: Cube manager class that controls the cubes.

private BoardManager boardMan: Board manager class that controls the board.

private PatternManager patternMan: Pattern manager class that initializes the pattern.

private PatternChecker patternCheck: Pattern checker class that checks the pattern.

Consturctors:

public GameInstance(): constructor for the GameInstance class.

Methods:

public void setPattern(): sets the pattern of this game instance.

public void setGameMode(): sets the game mode of this game instance.

3.3.5 Cube

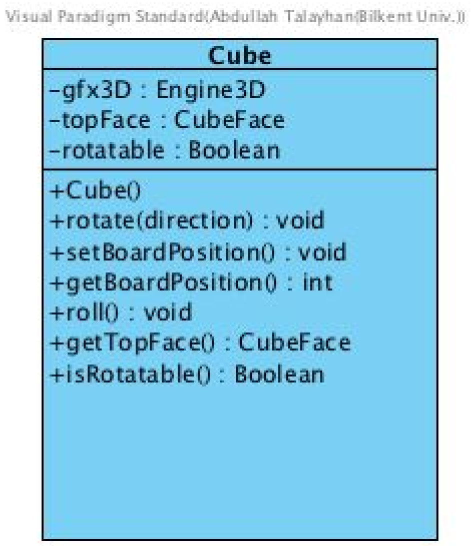

Visual Paradigm Standard(Abdullah Talayhan(Bilkent Univ.))
Cube
-gfx3D : Engine3D
-topFace : CubeFace
-rotatable : Boolean
+Cube()
+rotate(direction) : void
+setBoardPosition() : void
+getBoardPosition() : int
+roll() : void
+getTopFace() : CubeFace
+isRotatable() : Boolean

Figure 24: Cube Class

Cube:

Attributes:

private Engine3D gfx3D: this is going to be the 3D graphics objects.

private CubeFace topFace: contains the top cubeFace

private boolean rotatable: stores whether the cube is rotatable or not.

Consturctors:

public Cube(): constructor for the Cube class.

<u>Methods:</u>

private void rotate(int): rotates the cube depending on the input.

private void setBoardPosition(): sets the board position of the cube.

private int getBoardPosition(): returns the board position.

private void roll(): rotates the cube randomly.

private CubeFace getTopFace(): returns the top face of the cube.

private boolean isRotatable(): returns whether the cube is rotatable or not.

3.3.6 CubeFace

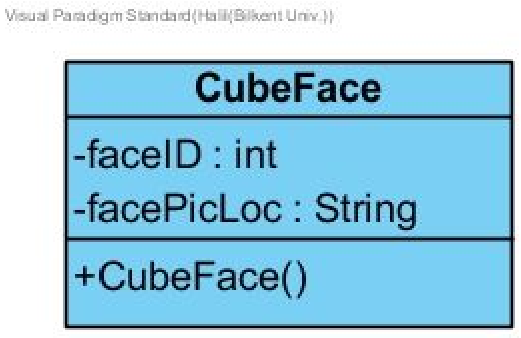


Figure 25: CubeFace Class

<u>CubeFace:</u>

<u>Attributes:</u>

private int faceID: ID of the picture that belongs to this face.

private String facePicLoc: File location of the face picture.

<u>Constructors:</u>

public CubeFace(): constructor for the CubeFace class.

3.3.7 CubeManager

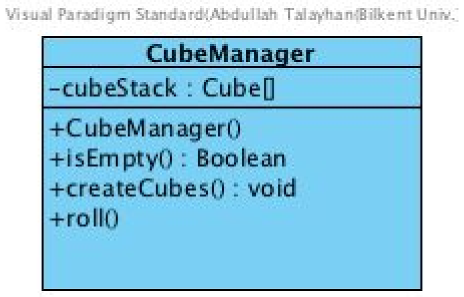


Figure 26: CubeManager Class

<u>CubeManager</u>:

<u>Attributes:</u>

private Cube[] cubeStack: array consisting of Cubes

<u>Constructors:</u>

public CubeManager(): constructor for the CubeManager class.

<u>Methods:</u>

public boolean isEmpty(): checks whether the cubeStack is empty or not.

public void createCubes(): initializes the cubes.

public void roll(): rolls all the cubes inside cubeStack.

3.3.8 Board

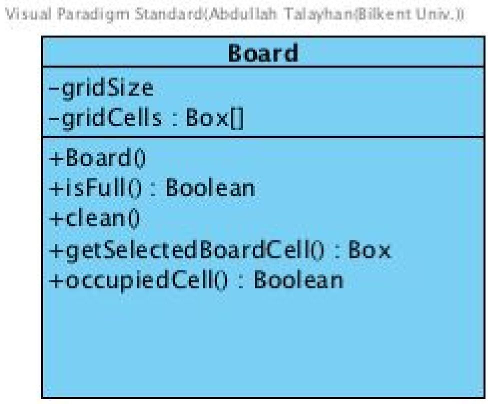

Visual Paradigm Standard(Abdullah Talayhan(Bilkent Univ.))
Board
-gridSize
-gridCells : Box[]
+Board()
+isFull() : Boolean
+clean()
+getSelectedBoardCell() : Box
+occupiedCell() : Boolean

Figure 27: Board Class

Board:

Atributes:

private int gridSize: Contains the board size(either 3x3, 4x4 or 5x5).

private Box[] gridCells: keeps the status of each boardCell as a Box object.

Consturctors:

public Board(): constructor for the Board class.

Methods:

public boolean isFull(): Checks whether the board is full or not, this is necessary for checking a pattern.

public void clean(): Resets the gridCells array.

public Box getSelectedBoardCell(): returns the selected cell in the grid.

public boolean occupiedCell(): tells whether a cell is occupied a cube or not.

3.3.9 Pattern

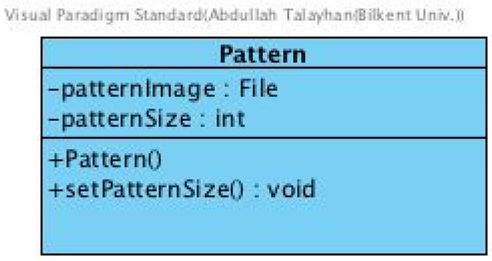


Figure 28: Pattern Class

Pattern:

Attributes:

private File patternImage: stores the pattern as an image file.

private int patternSize: stores the dimension of the grid.

Consturctors:

public Pattern(): constructor for the Pattern class.

Methods:

public setPatternSize(): sets the pattern size.

3.3.10 PatternChecker

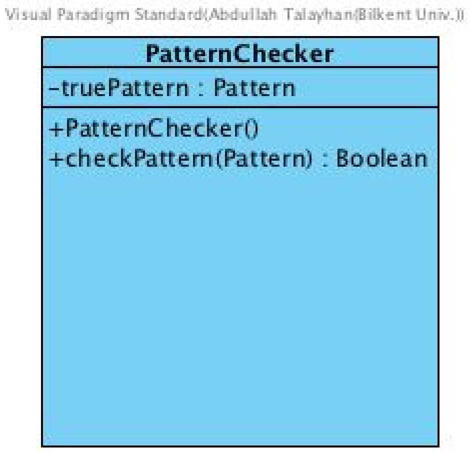

Visual Paradigm Standard(Abdullah Talayhan(Bilkent Univ.))
PatternChecker
-truePattern : Pattern
+PatternChecker()
+checkPattern(Pattern) : Boolean

Figure 29: PatternChecker Class

PatternChecker:

Attributes:

private Pattern truePattern: pattern to be matched

Consturctors:

public PatternChecker(): constructor for the PatternChecker class.

Methods:

private boolean checkPattern(Pattern): compares the true pattern with the given pattern.

3.3.11 PatternManager

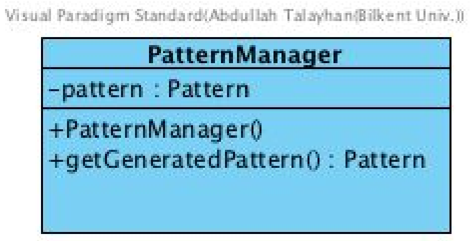

Visual Paradigm Standard(Abdullah Talayhan(Bilkent Univ.))
PatternManager
-pattern : Pattern
+PatternManager()
+getGeneratedPattern() : Pattern

Figure 30: PatternManager Class

PatternManager:

Attributes:
pattern: stores the Pattern object.

Consturctors:

public PatternManager(): constructor for the PatternManager class.

Methods:

public getGeneratedPattern(): generates and initializes the pattern object.

3.4 User Subsystem

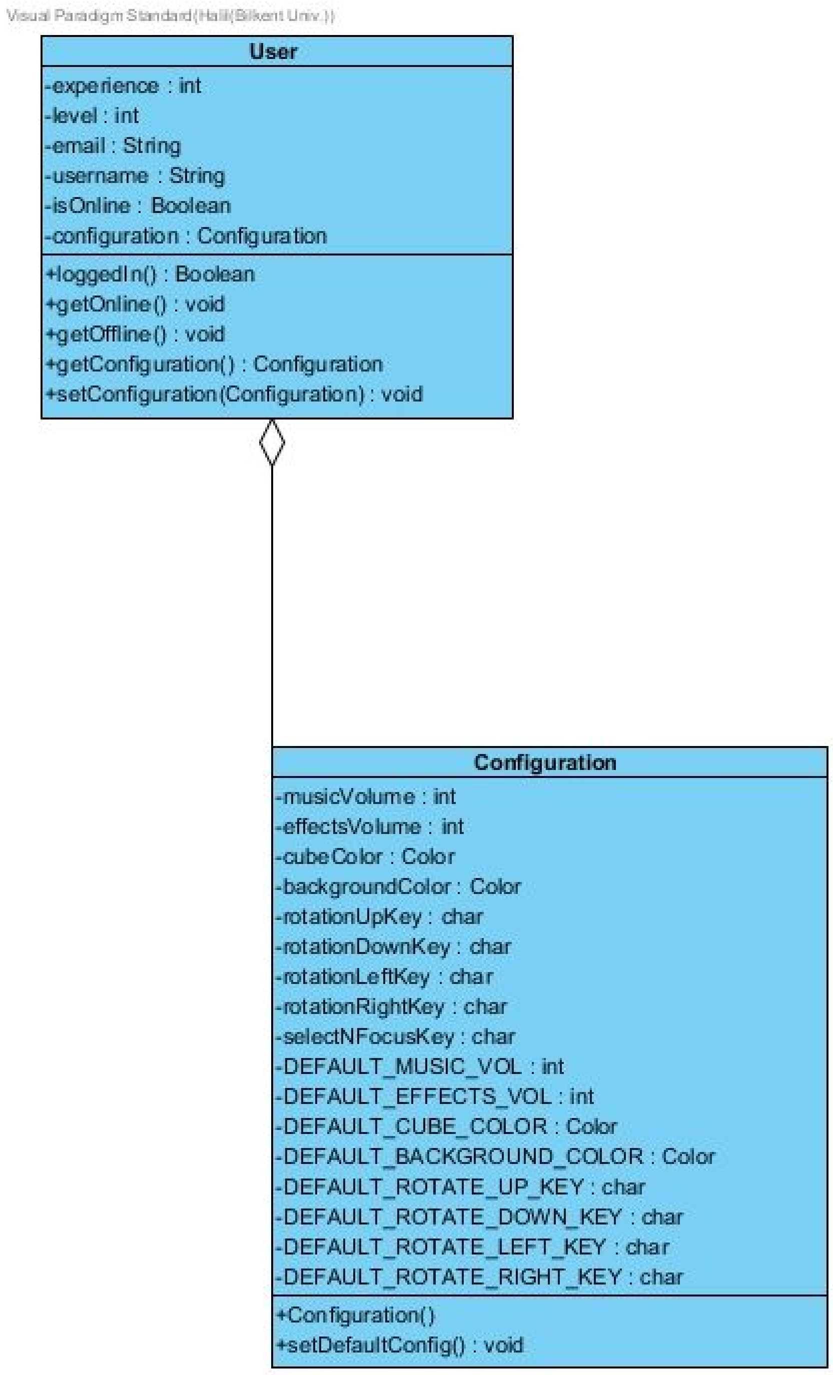


Figure 31: User Class Diagram

3.4.1 User

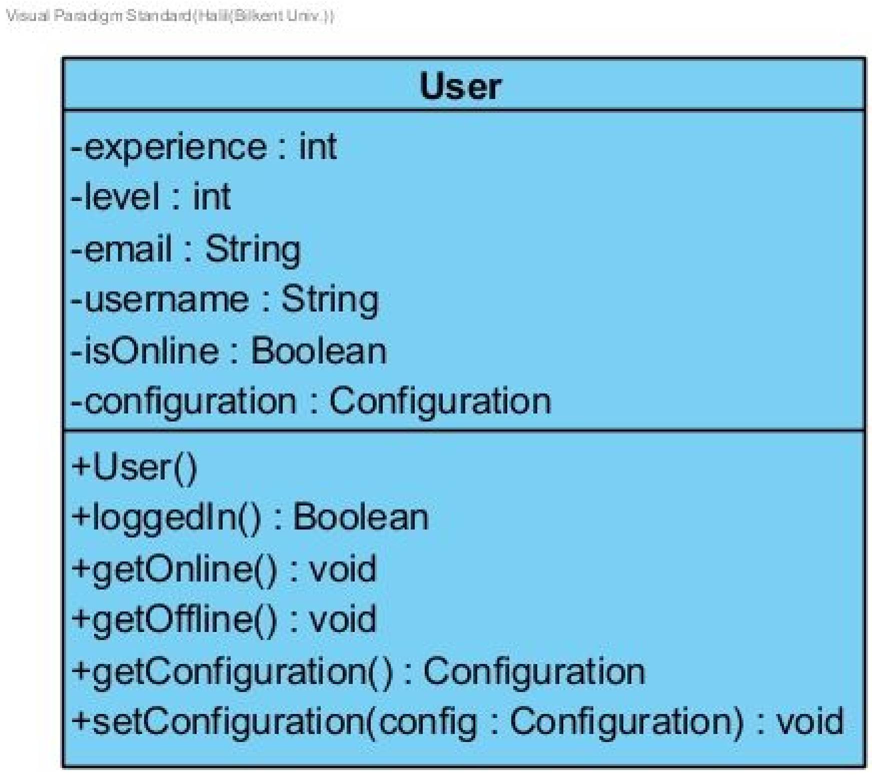

Visual Paradigm Standard(Halil(Bilkent Univ.))
User
-experience : int
-level : int
-email : String
-username : String
-isOnline : Boolean
-configuration : Configuration
+User()
+loggedIn() : Boolean
+getOnline() : void
+getOffline() : void
+getConfiguration() : Configuration
+setConfiguration(config : Configuration) : void

Figure 32: User Class

User:

Attributes:

private int experience : This attribute keeps the experience of the user.

private int level : This attribute keeps the level of the user.

private String email : This attribute keeps the email of the user.

private String username : This attribute keeps the username of the user.

private boolean isOnline : This attribute keeps the state of the user whether it is online or not.

private Configuration configuration: This attribute keeps the configuration of the user.

Constructors:

public User() : Constructor of the user of the game to create an object that can change and keep the user information in it.

Methods:

private boolean loggedIn() : This function checks the user is logged in or not.

private void getOnline() : This function connects the user to the server to make it online.

private void getOffline() : This function cuts the connection of the user to the server to make it offline in system.

private Configuration getConfiguration() : This function gets the configuration of the user.

private void setConfiguration(Configuration config) : This function set the configuration for the user with the given Configuration object through parameter.

3.4.2 Configuration

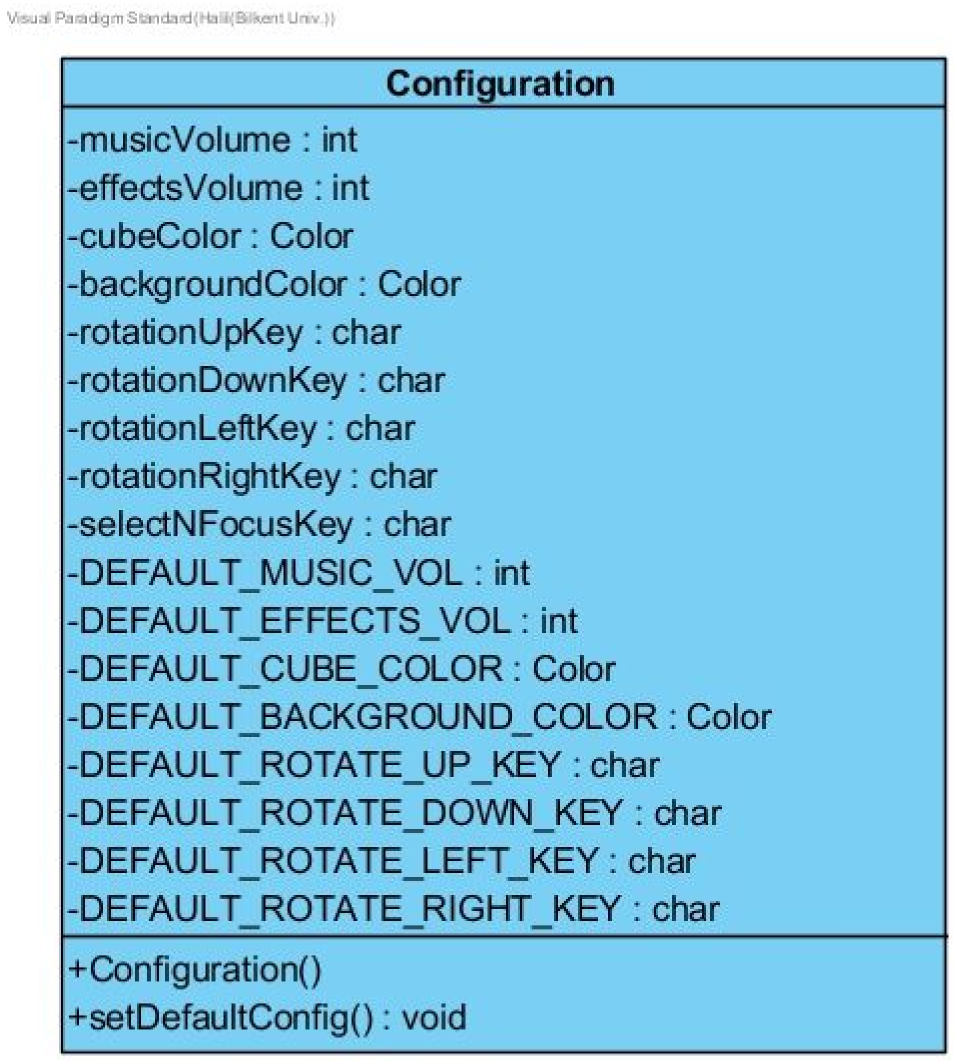

Visual Paradigm Standard(Halil(Bilkent Univ.))
Configuration
-musicVolume : int
-effectsVolume : int
-cubeColor : Color
-backgroundColor : Color
-rotationUpKey : char
-rotationDownKey : char
-rotationLeftKey : char
-rotationRightKey : char
-selectNFocusKey : char
-DEFAULT_MUSIC_VOL : int
-DEFAULT_EFFECTS_VOL : int
-DEFAULT_CUBE_COLOR : Color
-DEFAULT_BACKGROUND_COLOR : Color
-DEFAULT_ROTATE_UP_KEY : char
-DEFAULT_ROTATE_DOWN_KEY : char
-DEFAULT_ROTATE_LEFT_KEY : char
-DEFAULT_ROTATE_RIGHT_KEY : char
+Configuration()
+setDefaultConfig() : void

Figure 33: Configuration Class

Configuration:

Attributes:

private int musicVolume: This attribute will be used to store the level of music volume of the game.

private int effectsVolume: This attribute will be used to store the level of effects volume of the game.

private Color cubeColor: This attribute will be used to store the color of cube in game screens.

private Color backgroundColor: This attribute will be used to store the color of background in game screens.

private char rotationUpKey: This attribute will be used to store the keyboard binding for up rotation.

private char rotationDownKey: This attribute will be used to store the keyboard binding for down rotation.

private char rotationLeftKey: This attribute will be used to store the keyboard binding for right rotation.

private char rotationRightKey: This attribute will be used to store the keyboard binding for left rotation.

private char selectNFocusKey: This attribute will be used to store the keyboard binding for selecting and focusing a cube.

private final int DEFAULT_MUSIC_VOL: This attribute will be used to store the default level of music volume of the game.

private final int DEFAULT_EFFECTS_VOL: This attribute will be used to store the default level of effects volume of the game.

private final Color DEFAULT_CUBE_COLOR: This attribute will be used to store the default cube color of the game.

private final Color DEFAULT_BACKGROUND_COLOR: This attribute will be used to store the default background color of the game.

private final char DEFAULT_ROTATE_UP_KEY: This attribute will be used to store the default up rotation key of the game.

private final char DEFAULT_ROTATE_DOWN_KEY: This attribute will be used to store the default down rotation key of the game.

private final char DEFAULT_ROTATE_LEFT_KEY: This attribute will be used to store the default left rotation key of the game.

private final char DEFAULT_ROTATE_RIGHT_KEY: This attribute will be used to store the default right rotation key of the game.

<u>Constructors:</u>

public Configuration(): Constructor of the configurations of the game to create an object that can modify and set default of configurations of the game.

Methods:
public static void setDefaultConfig(): A function to set the configuration to their default settings.

4.Low-level Design

4.1 Object design trade-offs

4.1.1 Understandability vs. Functionality:

As we discussed before, our system should be easy to play and understand the game by user for every age. Therefore, we tried to create optimal design that we

4.1.2 Memory vs Maintainability:

The collection of Socket Handlers for each client should be kept in SocketServer object, since the connection to the client is established through these socket connections. Therefore, there were two ways of holding this collection. One of them is doing it by Map, and the other one is doing it by List. If it was a map, then sending a message through all connections was problematic since getting the keys of all sockets in the map. On the other hand, getting a specific socket from list by one of its properties requires traversing the all list. Since, status update operation which requires sending messages to all connected clients is a more common operation. The list implementation is chosen.

4.1.3 Development Time vs. User Experience:

During the analysis of the “Q-Bitz” game, we came to a consensus that we will use JAVAFX to create user interfaces and3D graphics because JAVAFX is native library of JAVA and also JAVAFX does not enforce us to use predefined design like some libraries do. Since JAVAFX is the standard library of JAVA for interfaces, there are a lots of sources to enhance user interfaces and graphics that will help us to create modern, modular and easy-to-play game. As a result, we believed that these properties of JAVAFX will enhance user experience.

4.2 Final Object Design

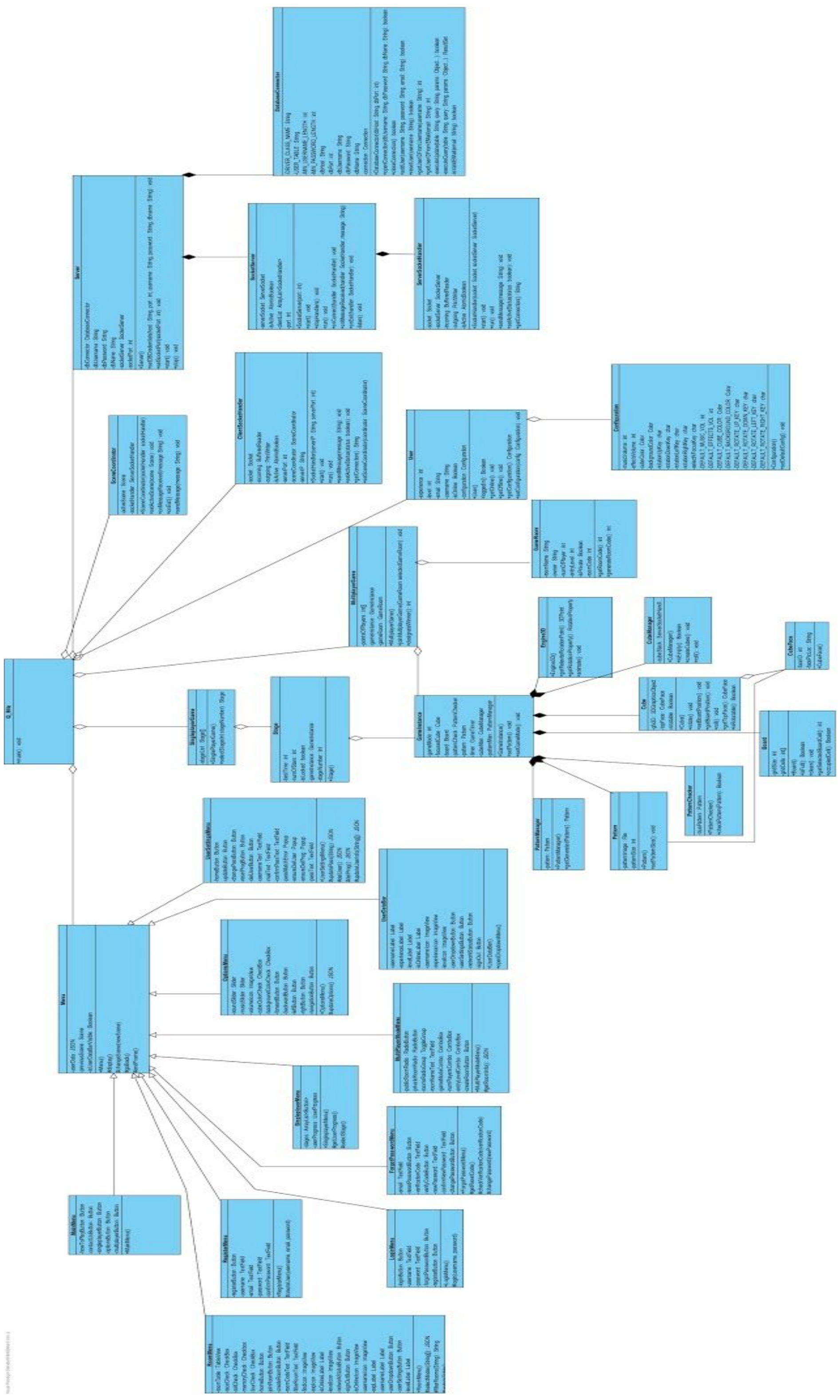


Figure 34: Final Object Design

4.3 Packages

javafx.* : For all the user interfaces and 3D graphics, we used JAVAFX package.

java.util.* : Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes

java.sql : Provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java^{TM} programming language.

java.io : Provides for system input and output through data streams, serialization and the file system.

java.net : Provides the classes for implementing networking applications.

org.json : We use JSON for the messages sent between the server and client for communication.

4.4 Class Interfaces

** There is no interface design in this iteration.

5.Glossary & references

[1] "Getting Started with JavaFX 3D Graphics." *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, docs.oracle.com/javase/8/javafx/graphics-tutorial/javafx-3d-graphics.htm.
[2] Capital, Board Game. "Q-Bitz Game Rules." *Board Game Capital*, www.boardgamecapital.com/q-bitz-rules.htm.