CS 319 Object-Oriented Software Engineering

Spring 2019

Design Report

My Little Quadrillion - Group 1C

- Alemdar Salmoor 21500430
- Ece Çanga 21600851
- Erkin Beşer 21400961
- Gökçe Sefa 21400596
- Mustafa Azyoksul 21501426
- Talha Şen 21502663

Instructor: Eray Tüzün

1. Introduction

1.1. Purpose of the system

“My Little Quadrillion” is a digital, single-player, puzzle, board game which aims to improve the player’s analytical and visual skills. What makes “My Little Quadrillion” different from the original “Quadrillion” that is being inspired from, is this version of the game is implemented and therefore played on a digital platform. Apart from its virtualization, there exists plentiful improvements and innovations that challenges the player (achievements, different modes and themes etc.) by destroying the boring time perception since it provides an enjoyable time whilst playing it.

1.2. Design goals

Below are the design goals that are focused during the development of My Little Quadrillion.

1.2.1. Performance

1.2.1.1. Response time

The responses should be fast enough to provide smooth user experience. For this reason, the responses should be delivered within one second. The change of windows in the game should also be less one second.

1.2.1.2. Throughput

The game should not lag or freeze because of the multiple processes executing at the same time. This should be achievable since the only background process will be database access. Database, on the other hand should be able to satisfy thousands of requests at the same time. That is the database should be able to handle more than 1000 requests at the same time.

1.2.1.3. Memory

My Little Quadrillion should be Memory efficient. The motivation for this design goal is to be able to run the game on many machines including low end machines with less memory. The goal is to be able to run the game on machines with volatile memory as low as 256 MB and available disk space of 100 MB.

1.2.5. Robustness

My Little Quadrillion should be Robust. Invalid user entries during registration and authentication should be all catched and warned to the user. Other than authorization, the game does not offer many scenarios where the user can enter invalid input. The user always interacts within the predefined objects with definite functionality.

1.2.6. Availability

The user should be able to launch the game and authorize and play the game any time they wish so. This requires the machine the database runs on the be online at all times.

1.2.7. Development Cost

Since the game is developed by poor students. Development cost should be of the design goals. This dictates that whenever there is an option - free, open source or no copyright resources should be preferred over paid ones even if paid ones are easier to incorporate, or offer wider range of functionality. The aim is to have development cost below 100$.

1.2.8. Portability

My Little Quadrillion should be Portable. The game should be able to be played on wide range of operating systems and hardware configurations. This is achieved by Developing a Java application. The executable then can be run on any

system that run Java Runtime Environment. The game should be runnable on machines with Windows, Mac, and Linux operating systems.

1.2.9. Usability

1.2.9.1. Color-blind Mode

Color-blind mode will be helpful for players who suffer from some sort of color-blindness. This is very critical in the game where a successful completion of the game depends on the distinguishing among 12 colorful pieces. Color-blind mode makes the game more comfortable and easier to play for players with color blindness.

1.2.9.2. Tutorial

The Game Tutorial is mandatory for the first time play to demonstrate how to play the game and how the puzzles pieces are interact in game. In the tutorial, the settlement of twelve game pieces and 4x4 grid will be shown. Also, mouse clicks which are hold, drop and drag the pieces will be animated in tutorial before playing the game. Pause button for freeze the game can be provided by “P” button on the keyboard. That is, there are right and left mouse buttons and 1 button on the keyboard to interact with the game.

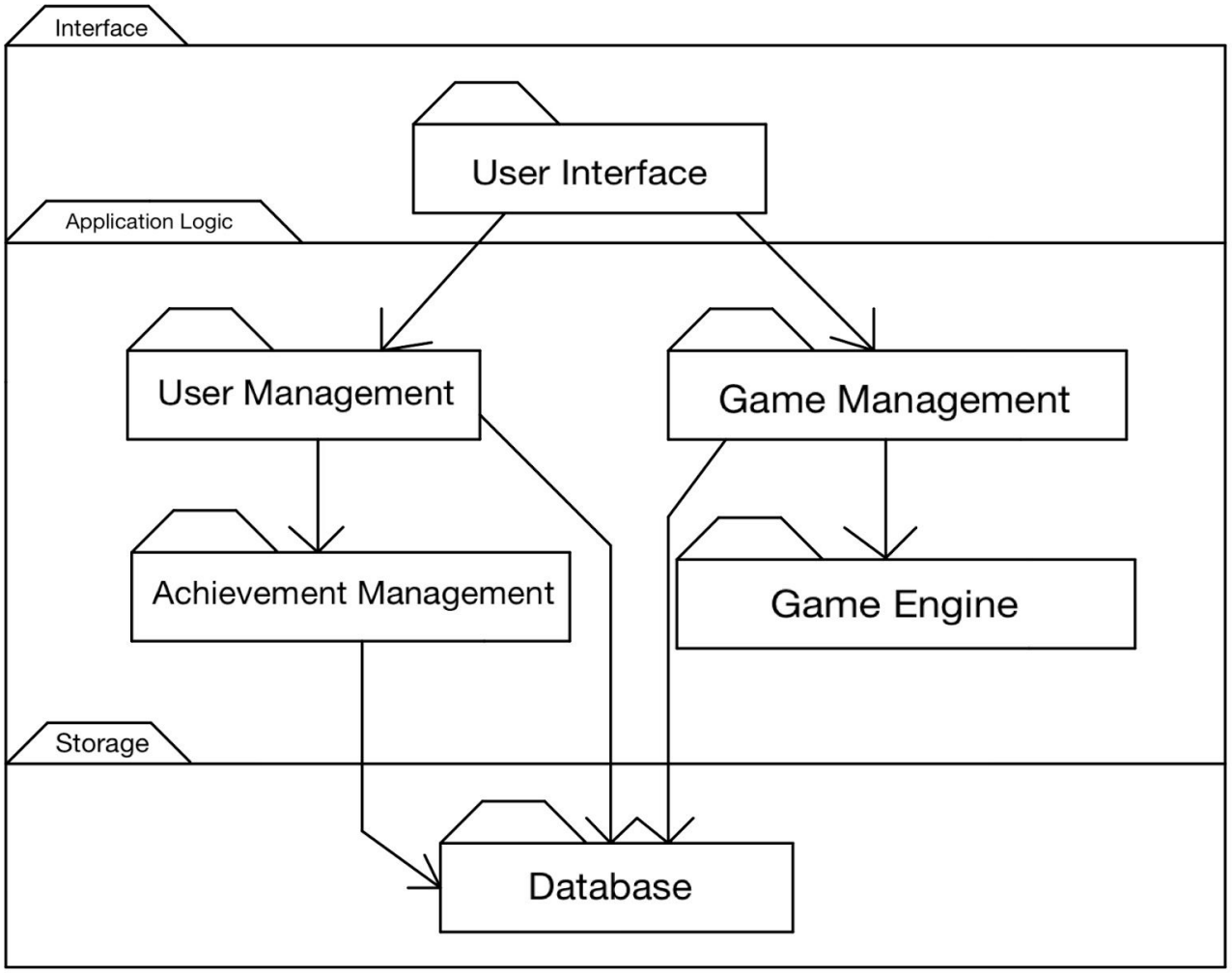
1.2.9.3. Changing Theme

My Little Quadrillion is a desktop game in the digital platform. For optional visualization while playing game can be provided by three different themes in the options part of the game. Night mode will reduce the half of the game light for providing darker screen. Also, user experiences the game more like when they play the game on a wooden table owing to wooden theme in the theme option.

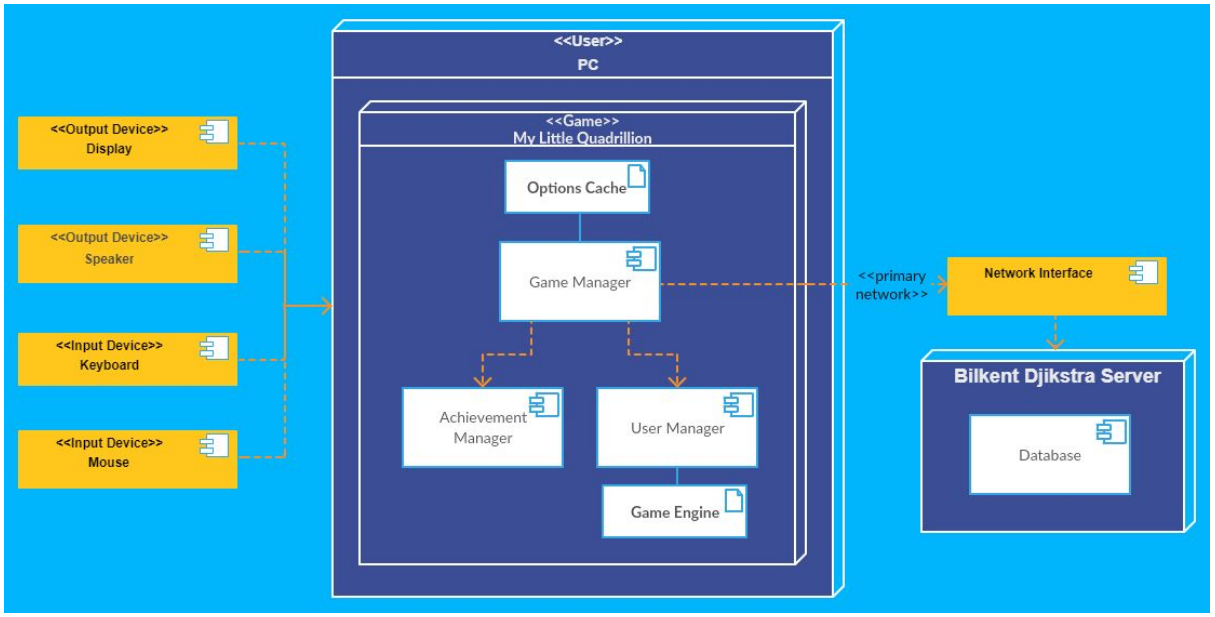
2. High-level software architecture

2.1. Subsystem decomposition

We will deal with our system into subsystems. We have three layers, interface, application logic and storage. Our first layer is only consist of user interface to see how designed game. Our second layer is application logic which have user management, game management, achievement management and game engine. Also, third and last layer is storage which has database.



2.2. Hardware/software mapping



My Little Quadrillion can be launched on any machine that runs Java Runtime Environment. The user interface of the application is realized through mouse and keyboard for user entries, monitor to display the visual game responses to the user input, speakers for audio output. The user related data is stored on the database that runs on the Bilkent Dijkstra Server. All the instances of the game communicate with this machine to access those data.

2.3. Persistent data management

Persistent data of application will be stored in the one common relational database. All instances of the applications will access the central database. Username and password combinations are also stored in the database for each user. For example user progress in the Arcade Mode will be saved in the database. This enables users to access and start playing on one machine and then continue their progress on other machine. Additionally, having one central relational database enables more complex data queries than if local files were used. Use of relational database enables the users to have concurrent access.

The motivation behind using database is because the database provides more convenient way to serve multiple users concurrently. Additionally, the data is

ported across multiple systems and using database permits developers not to worry about the file systems of any particular operating system.

2.4. Access control and security

No user can interfere with data of other users or access their username or passwords. That is, user can not see other user progress or modify user data in any way. This is realized with SQL queries to the database at the time of the authorization. Only those tuples of progress tables related to game modes are accessible to specific user that are associated with that user or explicitly made to be accessible to every user. As an example, consider Level Editor mode. When the user finalizes the creation of custom level this level is created in the table corresponding to the Created Levels. After this everyone can access this table and everyone can play the level because Created Levels are available to everyone.

2.5. Boundary conditions

2.5.1. Start-up

The system is started by double clicking the executable file. The user needs to authorize within the game. On successful authentication user information is fetched from the database and the game instance is initialized with those values. Database is always online. The user information includes user credentials and progress in each game mode as well as the unlocked game achievements. The initial graphical user interface that user is presented is Register/Login window.

2.5.2. Shut Down

My Little Quadrillion has ‘X’ button to exit from the system. The game will have ‘X’ button on all of the pages. If the player clicks ‘X’ button during the game, error message will be displayed on the screen because if players exits while playing, the game progress will be lost. The error message serves as a reminder to the users of possible consequences. Additionally the main menu has ‘Quit’ button to exit the

game. In error free scenario all of the game progress up to the last completed level gets stored in the database.

2.5.3. Failures

2.5.3.1. Power outage

There is no explicit behaviour that is to be executed in case of power outage. All of the progress that happened after the last database update gets lost. That is if the user was in the process of completing a level that progress is lost. However, all of the completed levels in that particular session until the last uncompleted level get stored in the database. If the power outage happens on the machine where the database is launched all of the uncompleted transactions will be rolled back.

2.5.3.2. Network outage

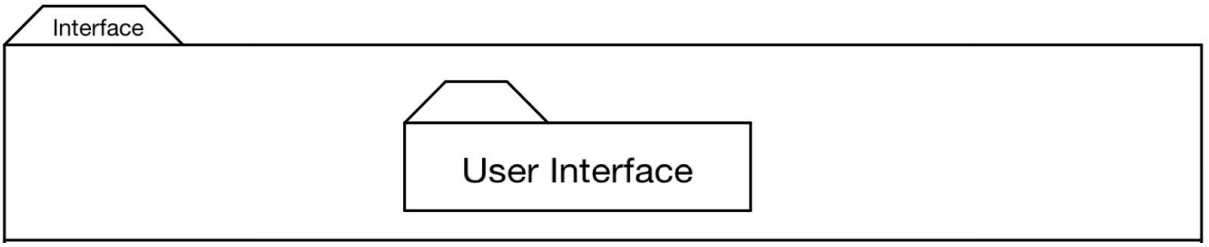
In case of the network outage the user will be notified that their current game progress might not be saved due to network outage. Furthermore, notification will advise user to seek network connection as soon as possible and warn that game progress might be lost in case of current game session termination.

2.5.3.3. Software failure

In case the software becomes unresponsive the user will be advised to close the application and relaunch the game.

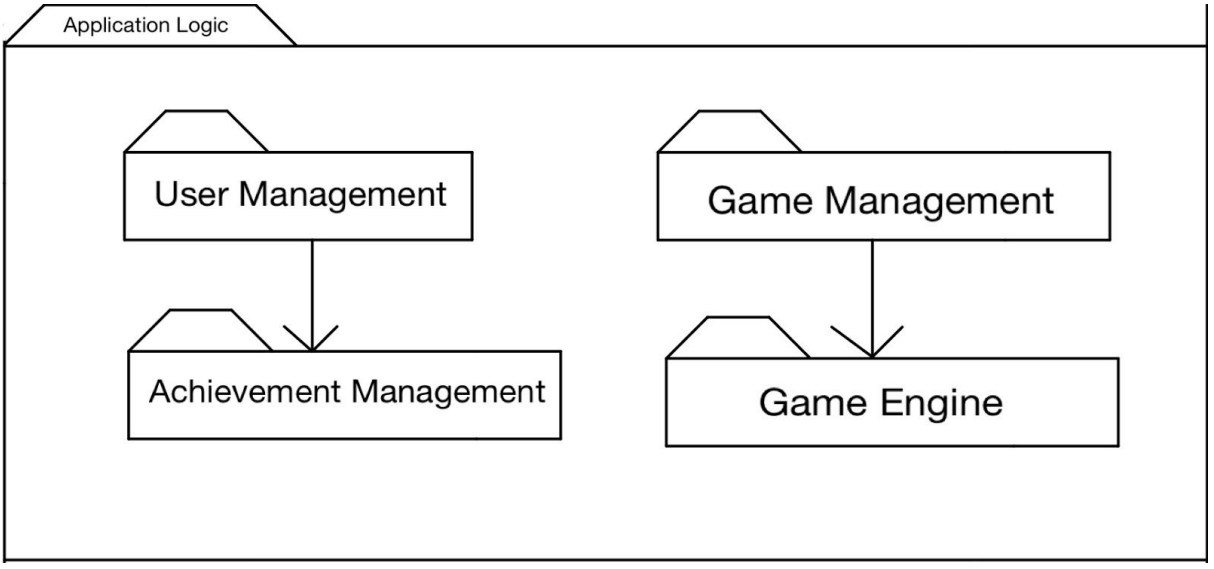
3. Subsystem services

3.1. User Interface Subsystem



Main class has a main stage as attribute that will be created upon launch and will be kept until application is terminated. Main stage is a javafx window functionality. This instance is going to be the same throughout the game session. This stage manages all the scenes, data management between scenes and scene transitions. Scenes can raise events to change scenes or other attributes of the main stage. This interface system combines all the functionalities together.

3.2 Application Logic Subsystem

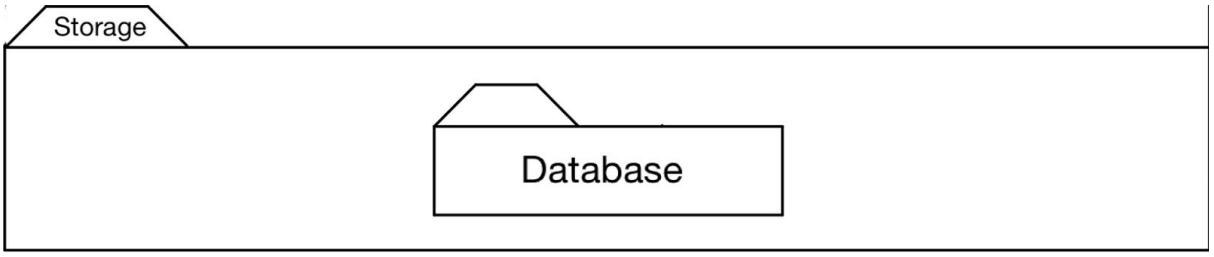


When user enters the game, he/she lands on the login page. Application navigates user with convenient and easy to use buttons and controls. After

authentication, user lands on the menu screen. From there, button and control takes the user to respective menu and game screens. If user takes an unapproved action such as entering wrong username or password, game warns user with a popup screen.

When you enter any game screen, the game loads a game board. In this game board, depending on the game mode, there will be grids and pieces. The game engine enables users to grad and drop grids and pieces on the game board. Engine keeps track of coordinate and rotation of grids, obstacles on grids and pieces both placed and non-placed. When user drops grids and pieces on the game board, engine checks if this possible and if the action is possible updates that game board state. Additionally, when game over conditions are met, engine raises an event to the interface so that player can advance to next level of see his/her statistics again depending on the game mode.

3.3 Storage Subsystem



We are going to use a MySQL database for the application. Database is going to store usernames and passwords of the players for authentication. Moreover, the database is going to store each player’s progress in terms of achievements, arcade game level star information, rush mode high scores and custom created levels. Our program is going to access and manipulate the data on the database during the game session. If internet connection fails, user will be unable to save progress.

4. Low-level design

4.1. Object design trade-offs

4.1.1. Throughput vs Latency

My Little Quadrillion deploying one common database enables many users to authenticate and use the game at the same time from different machines which increases the throughput. This however decreases latency since the requests are needed to be made and responses needed to be received to and from the central database which adds overhead and increases latency

4.1.2. Memory Space vs Response Time

Persistent data is stored in the relational database. This increases the response time by a small margin in bottleneck situations like authorization. But since the game is not response time critical and overheads in milliseconds would not affect user experience. Therefore, for My Quadrillion it is possible to to prioritize Memory Space over Response Time.

4.1.3. Efficiency vs. Portability

Even though Java programming Language may not be the most efficient one, it offers the ease of Portability which ensures that My Little Quadrillion can be launched on virtually any machine capable of running Java Runtime Environment.

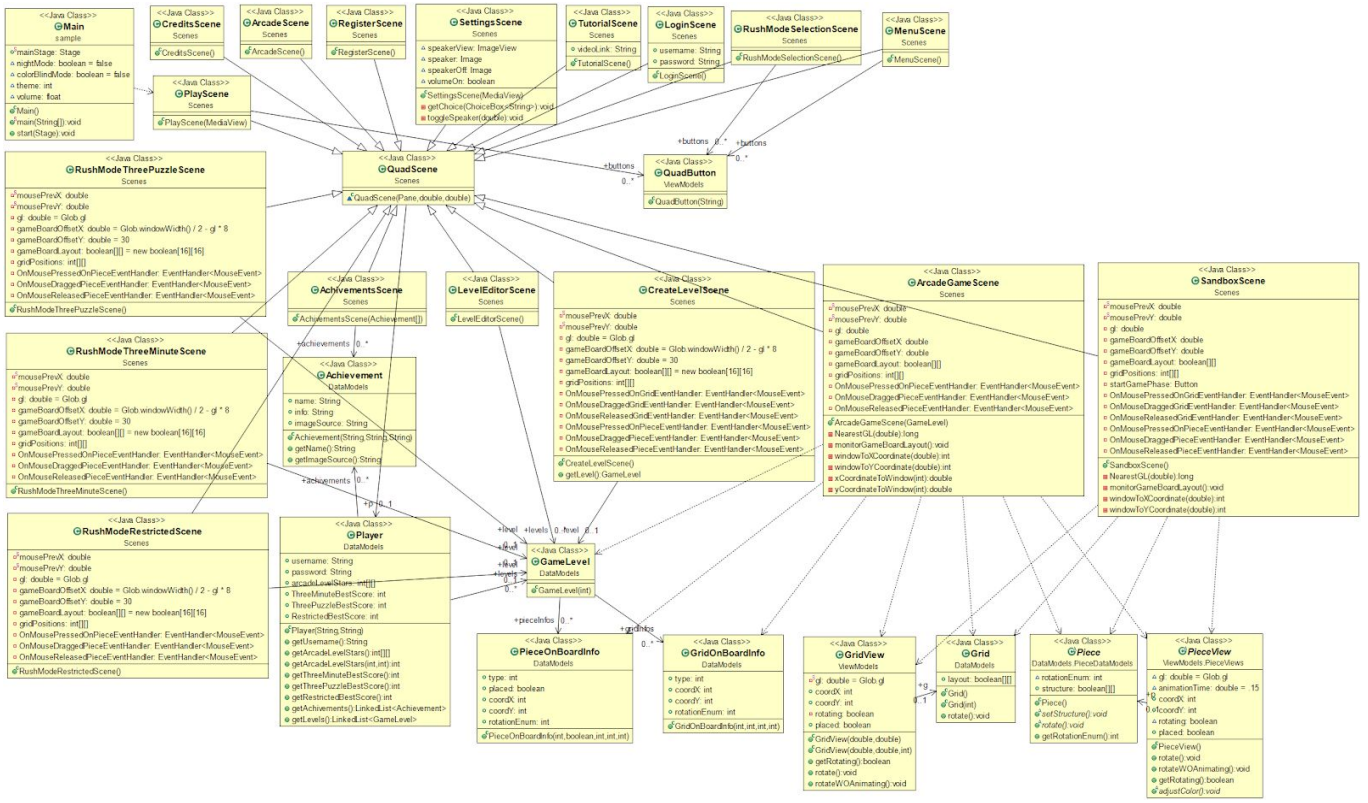
4.1.4. Rapid Development vs Functionality

Rapid Development is of critical importance to the My Little Quadrillion because the dates are strict and explicit. Nonetheless, the game has fair amount of features and game modes to offer to the end user. During the development all of the possible new features that can be added to the game should be considered with taking time constraints into account.

4.1.5. Security vs Usability

The target audience of My Little Quadrillion, as underlined by the course instructor, Eray Tüzün, are children. Additionally, the game does not store personal information or any other information of high vulnerability. For the reasons above, it is not rational to try to make the game exceptionally secure. Still, the game offers basic security precautions such as md5 hashing of passwords. Passwords are not required to obey some strict rules such as inclusion of both uppercase and lowercase letters, apart from some invalid characters. Reiterating, Usability is more important than security in the scope of this particular game.

4.2. Final object design



4.3. Packages

4.3.1. java

The util package of java is used to save values in the arrayList structure. By this way handling with the object is more efficient.

4.3.2. javafx

4.3.2.1. javafx.animation

Unlike in java, javafx provides the dynamic handling of the visual elements. By using the animation package, handling the pieces is done.

4.3.2.2. javafx.geometry

The set of 2D classes are provided in this package. The 2D objects in “My Little Quadrillion” are composed of lines, rectangles, circles and squares.

4.3.2.3. javafx.event

Event-handling is done using this package which contains the control of the mouse, the keyboard input and the click input.

4.3.2.3. javafx.util

Duration is calculated and set with this package. In rush mode, time constraints should be considered.

4.3.2.4. javafx.stage

Top level design is done using the stage package. This package includes the layout and the design of the whole visualization.

4.3.2.5. javafx.scene

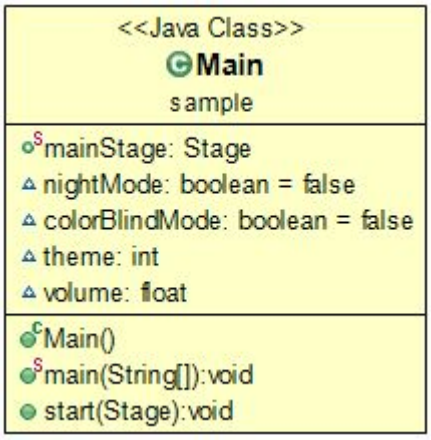
This package is used for the visual scenes in “My Little Quadrillion”. All of the elements that creates the view are stored in the scene container which are transform, shape, Scene (class), layout, control, input and paint.

4.3.3. mySQL

java.sql package is used in order to connect the database with the app.

4.4. Class Interfaces

4.4.1. Main



Attributes:

- private player: This is the player object that we will be using throughout the application.
- boolean nightMode: This is a boolean flag that indicates if the night mode is active or not.
- boolean colorBlindMode: This is a boolean flag that indicates if the blind mode is active or not.
- int theme: This is an enumeration that will keep track of which theme is selected by the user.
- float volume: This float indicates the volume of the music and effects that are playing.

Constructor:

- Main class does not have a constructor.

Methods:

- public static void main(String[] args): javafx main method that launches the application.
- public void start(Stage primaryStage) javafx start method that initializes and opens the window. This method will set the window dimensions, window icon,

window title, resizability, and other necessary window attributes. Then instantiates a LoginScecne object and sets that scene to it. Finally, shows the window.

4.4.2. Register Scene

This scene shows input fields for username and password and register button. When register button is pressed, credentials are saved to the database and player is automatically logged in to the game. Menu scene gets loaded.

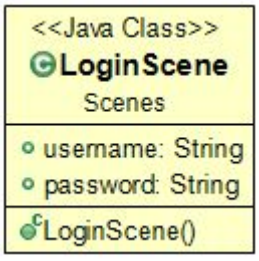
Attributes:

- We do not need attributes for this class.

Constructor:

- public RegisterScene(): This constructor constructs the register scene interface. The register button on this scene fetches user info from database so that same username cannot be created. If registration phase is successful, raises an event to main stage that will load the menu scene and updates the database

4.4.3. Login Scene



This scene is entry scene to the application. This scene shows input fields for username and password and login button. When login button is pressed, credentials are checked and player is logged in to the game. Menu scene gets loaded.

Attributes:

- string: username: username

- string password: password. There is no security measures for this game. Password is just stored in memory.

Constructor:

- public LoginScene(): This constructor constructs the login scene interface. The login button on this scene fetches user info from database for comparison. If login is successful, raises an event to main stage that will load the menu scene.

Methods:

- We do not need methods for this class.

4.4.4. Credits Scene

This scene shows some simple information about developers.

Attributes:

- We do not need attributes for this class.

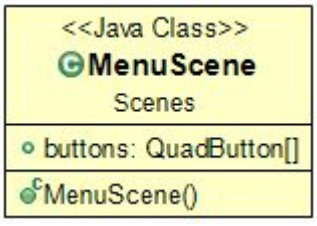
Constructor:

- public CreditsScene(): This constructor constructs the credits scene interface. Details are hard coded into this constructor.

Methods:

- We do not need methods for this class.

4.4.5. Menu Scene



This scene is the main navigational scene of the application. Users land on this scene when they log into the menu scene.This scene allows users to go to play scene, achievements scene and options scene.

Attributes:

- We do not need attributes for this class.

Constructor:

- public CreditsScene(): This constructor constructs the credits scene interface. Details are hard coded into this constructor.

Methods:

- We do not need methods for this class.

4.4.6. Play Scene

This scene is the main navigational scene for game modes. Users land on this scene when they click to play button.This scene allows users to go to tutorial scene, arcade scene, rush mode selection scene, level editor scene, and sandbox scene.

Attributes:

- We do not need attributes for this class.

Constructor:

- public PlayScene(): This constructor constructs the play scene interface. All buttons raises event in main stage that takes the application to corresponding game scene.

Methods:

- We do not need methods for this class.

4.4.7. Tutorial Scene

Attributes:

- string videoLink

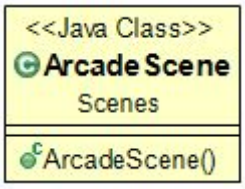
Constructor:

- public CreditsScene(): This constructor constructs the arcade level selection scene interface. The levels are locked/unlocked depending on user's progression.

Methods:

- We do not need methods for this class.

4.4.8. Arcade Scene



Attributes:

- We do not need attributes for this class.

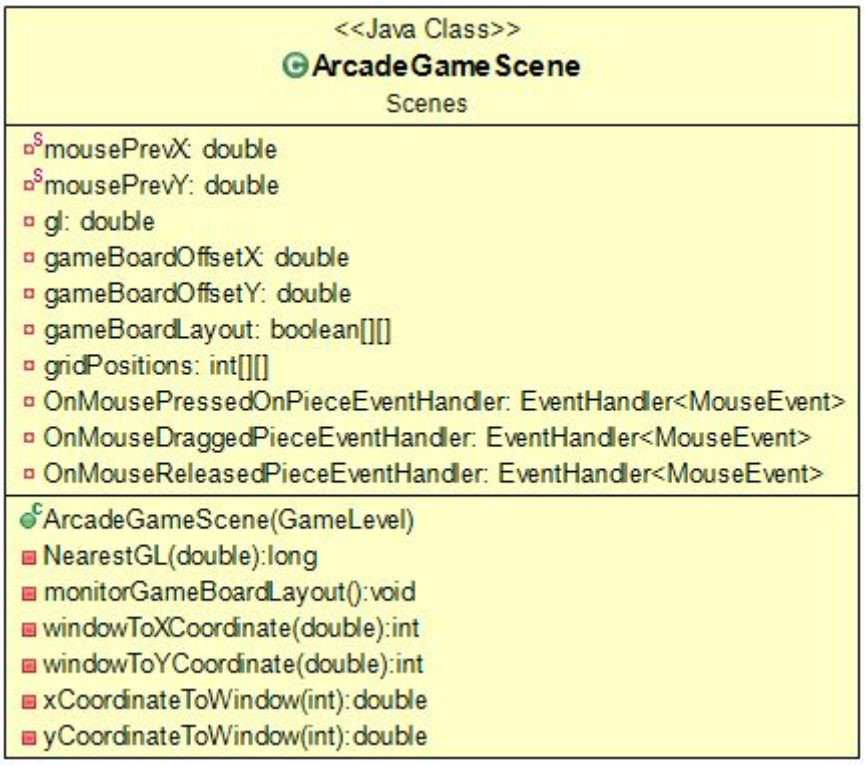
Constructor:

- public ArcadeScene(): This constructor constructs the credits scene interface. Details are hard coded into this constructor.

Methods:

- We do not need methods for this class.

4.4.9. Arcade Game Scene



Attributes:

- private static double mousePrevX, mousePrevY: These attributes save the mouse position between MouseDrag events.
- private double gl = Glob.gl: Grid length of squares on the game board.
- private double gameBoardOffsetX = Glob.windowWidth() / 2 - gl * 8
- private double gameBoardOffsetY = 30: These two are the game board position on the scene since game logic uses scene coordinates of the pieces.
- private boolean[][] gameBoardLayout = new boolean[16][16]: This is the array that holds the gameboard states. Every position corresponds to every square on the game board. If position is true, the square is available, if position is false, square is out of game board or occupied by a piece.
- private int[][] gridPositions: This is the coordinate of the grid on the game board.

Constructor:

- public ArcadeGameScene(GameLevel level): This constructor constructs the game scene with the given game level object.

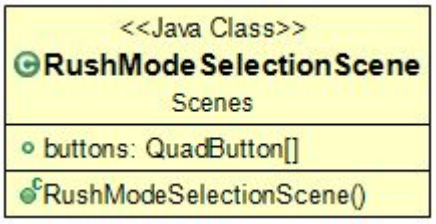
Methods:

- We do not need methods for this class.

Events:

- private EventHandler<MouseEvent> OnMousePressedOnPieceEventHandler: This event is raised when a piece on this scene is pressed.
- private EventHandler<MouseEvent> OnMouseDraggedPieceEventHandler: This event is raised when a piece on this scene is dragged.
- private EventHandler<MouseEvent> OnMouseReleasedPieceEventHandler: This event is raised when a piece on this scene is released.

4.4.10. Rush Mode Selection Scene



Attributes:

- We do not need attributes for this class.

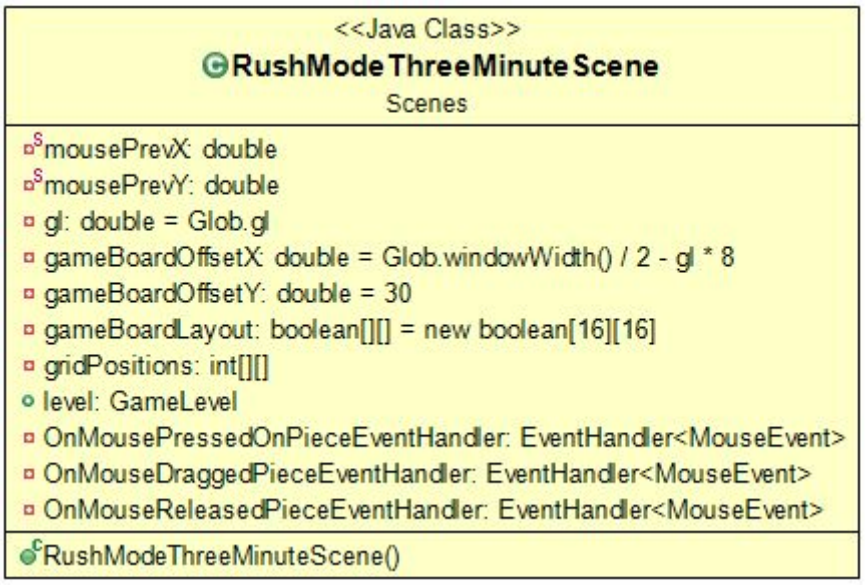
Constructor:

- public RushModeSelectionScene(): This constructor constructs the rush mode selection scene interface. There are 3 buttons that take you to corresponding game mode.

Methods:

- We do not need methods for this class.

4.4.11. Rush Mode Three Minute Scene



Attributes:

- private static double mousePrevX, mousePrevY: These attributes save the mouse position between MouseDrag events.
- private double gl = Glob.gl: Grid length of squares on the game board.
- private double gameBoardOffsetX = Glob.windowWidth() / 2 - gl * 8
- private double gameBoardOffsetY = 30: These two are the game board position on the scene since game logic uses scene coordinates of the pieces.
- private boolean[][] gameBoardLayout = new boolean[16][16]: This is the array that holds the gameboard states. Every position corresponds to every square on the game board. If position is true, the square is available, if position is false, square is out of game board or occupied by a piece.
- private int[][] gridPositions: This is the coordinate of the grid on the game board.

Constructor:

- public RushModeThreeMinuteScene(GameLevel level): This constructor constructs the game scene with the given game level object.

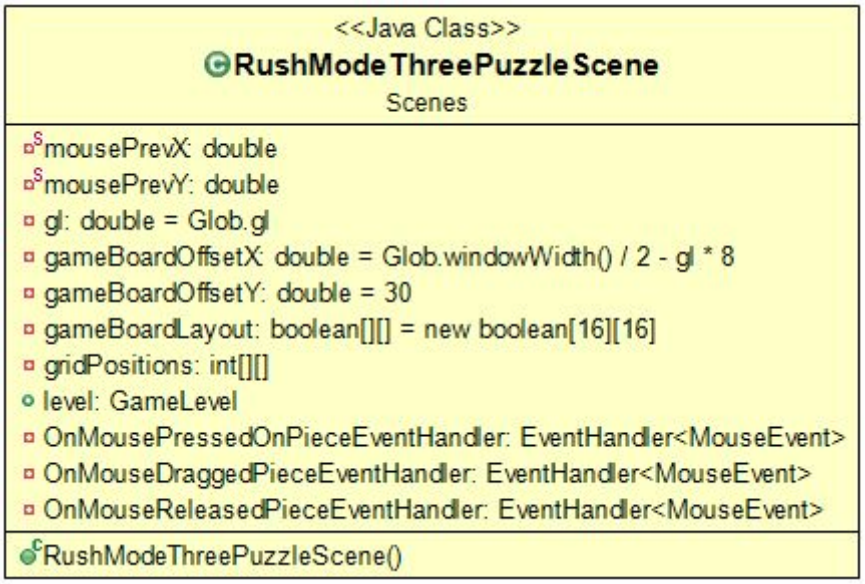
Methods:

- We do not need methods for this class.

Events:

- private EventHandler<MouseEvent> OnMousePressedOnPieceEventHandler: This event is raised when a piece on this scene is pressed.
- private EventHandler<MouseEvent> OnMouseDraggedPieceEventHandler: This event is raised when a piece on this scene is dragged.
- private EventHandler<MouseEvent> OnMouseReleasedPieceEventHandler: This event is raised when a piece on this scene is released.

4.4.12. Rush Mode Three Puzzle Scene



Attributes:

- private static double mousePrevX, mousePrevY: These attributes save the mouse position between MouseDrag events.
- private double gl = Glob.gl: Grid length of squares on the game board.
- private double gameBoardOffsetX = Glob.windowWidth() / 2 - gl * 8
- private double gameBoardOffsetY = 30: These two are the game board position on the scene since game logic uses scene coordinates of the pieces.
- private boolean[][] gameBoardLayout = new boolean[16][16]: This is the array that holds the gameboard states. Every position corresponds to every square on the game board. If position is true, the square is available, if position is false, square is out of game board or occupied by a piece.

- private int[][] gridPositions: This is the coordinate of the grid on the game board.

Constructor:

- public RushModeThreePuzzleScene(GameLevel level): This constructor constructs the game scene with the given game level object.

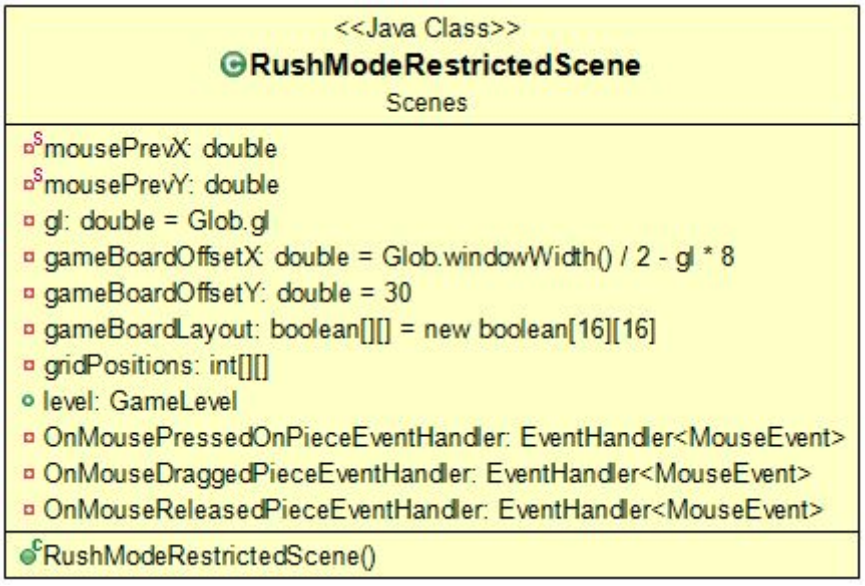
Methods:

- We do not need methods for this class.

Events:

- private EventHandler<MouseEvent> OnMousePressedOnPieceEventHandler: This event is raised when a piece on this scene is pressed.
- private EventHandler<MouseEvent> OnMouseDraggedPieceEventHandler: This event is raised when a piece on this scene is dragged.
- private EventHandler<MouseEvent> OnMouseReleasedPieceEventHandler: This event is raised when a piece on this scene is released.

4.4.13. Rush Mode Restricted Scene



Attributes:

- private static double mousePrevX, mousePrevY: These attributes save the mouse position between MouseDrag events.
- private double gl = Glob.gl: Grid length of squares on the game board.
- private double gameBoardOffsetX = Glob.windowWidth() / 2 - gl * 8

- private double gameBoardOffsetY = 30: These two are the game board position on the scene since game logic uses scene coordinates of the pieces.
- private boolean[][] gameBoardLayout = new boolean[16][16]: This is the array that holds the gameboard states. Every position corresponds to every square on the game board. If position is true, the square is available, if position is false, square is out of game board or occupied by a piece.
- private int[][] gridPositions: This is the coordinate of the grid on the game board.

Constructor:

- public RushModeRestrictedScene(GameLevel level): This constructor constructs the game scene with the given game level object.

Methods:

- We do not need methods for this class.

Events:

- private EventHandler<MouseEvent> OnMousePressedOnPieceEventHandler: This event is raised when a piece on this scene is pressed.
- private EventHandler<MouseEvent> OnMouseDraggedPieceEventHandler: This event is raised when a piece on this scene is dragged.
- private EventHandler<MouseEvent> OnMouseReleasedPieceEventHandler: This event is raised when a piece on this scene is released.

4.4.15. Level Editor Scene

Attributes:

- We do not need attributes for this class.

Constructor:

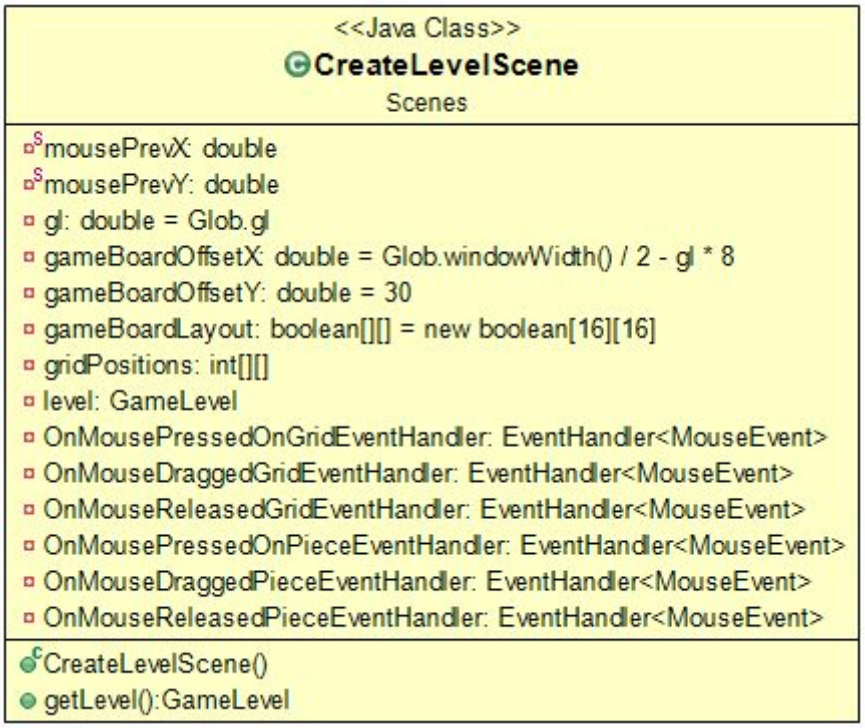
- public LevelEditorScene(): This constructor constructs the level editor scene interface. There are panes that when clicked open the previously created

levels and also a new level button. These components raise events on main stage that takes you to corresponding game scene.

Methods:

- We do not need methods for this class.

4.4.16. Create Level Scene



Attributes:

- private static double mousePrevX, mousePrevY: These attributes save the mouse position between MouseDrag events.
- private double gl = Glob.gl: Grid length of squares on the game board.
- private double gameBoardOffsetX = Glob.windowWidth() / 2 - gl * 8
- private double gameBoardOffsetY = 30: These two are the game board position on the scene since game logic uses scene coordinates of the pieces.
- private boolean[][] gameBoardLayout = new boolean[16][16]: This is the array that holds the gameboard states. Every position corresponds to every square on the game board. If position is true, the square is available, if position is false, square is out of game board or occupied by a piece.

- private int[][] gridPositions: This is the coordinate of the grid on the game board.

Constructor:

- public CreateLevelScene(): This constructor constructs the create level scene. Here, initially, four grids are on the scene. Button on the scene takes us to next phase of the level creation process once it is valid to do so.

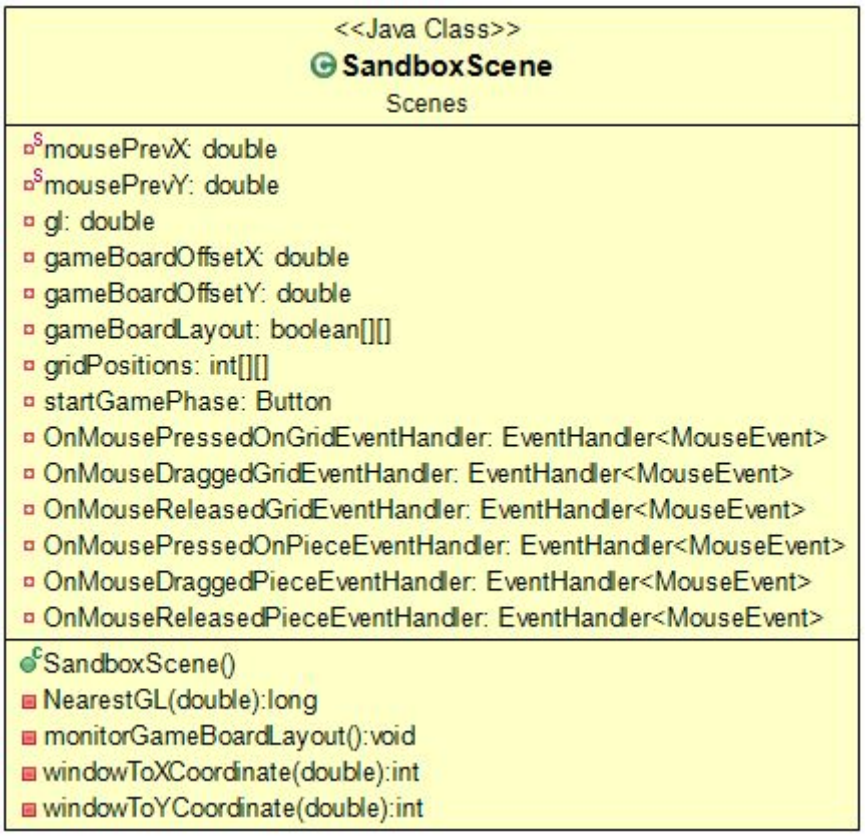
Methods:

- We do not need methods for this class.

Events:

- private EventHandler<MouseEvent> OnMousePressedOnPieceEventHandler: This event is raised when a grid on this scene is pressed.
- private EventHandler<MouseEvent> OnMouseDraggedPieceEventHandler: This event is raised when a grid on this scene is dragged.
- private EventHandler<MouseEvent> OnMouseReleasedPieceEventHandler: This event is raised when a grid on this scene is released.
- private EventHandler<MouseEvent> OnMousePressedOnPieceEventHandler: This event is raised when a piece on this scene is pressed.
- private EventHandler<MouseEvent> OnMouseDraggedPieceEventHandler: This event is raised when a piece on this scene is dragged.
- private EventHandler<MouseEvent> OnMouseReleasedPieceEventHandler: This event is raised when a piece on this scene is released.

4.4.17. Sandbox Scene



Attributes:

- private static double mousePrevX, mousePrevY: These attributes save the mouse position between MouseDrag events.
- private double gl = Glob.gl: Grid length of squares on the game board.
- private double gameBoardOffsetX = Glob.windowWidth() / 2 - gl * 8
- private double gameBoardOffsetY = 30: These two are the game board position on the scene since game logic uses scene coordinates of the pieces.
- private boolean[][] gameBoardLayout = new boolean[16][16]: This is the array that holds the gameboard states. Every position corresponds to every square on the game board. If position is true, the square is available, if position is false, square is out of game board or occupied by a piece.
- private int[][] gridPositions: This is the coordinate of the grid on the game board.

Constructor:

- public SanboxScene(): This constructor constructs the sandbox scene. Here there are grids and pieces that can be duplicated.

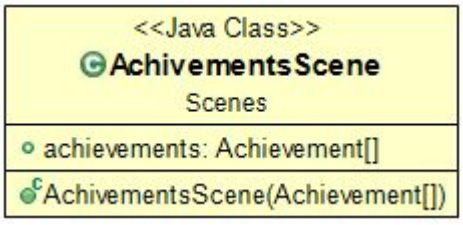
Methods:

- We do not need methods for this class.

Events:

- private EventHandler<MouseEvent> OnMousePressedOnPieceEventHandler: This event is raised when a grid on this scene is pressed.
- private EventHandler<MouseEvent> OnMouseDraggedPieceEventHandler: This event is raised when a grid on this scene is dragged.
- private EventHandler<MouseEvent> OnMouseReleasedPieceEventHandler: This event is raised when a grid on this scene is released.
- private EventHandler<MouseEvent> OnMousePressedOnPieceEventHandler: This event is raised when a piece on this scene is pressed.
- private EventHandler<MouseEvent> OnMouseDraggedPieceEventHandler: This event is raised when a piece on this scene is dragged.
- private EventHandler<MouseEvent> OnMouseReleasedPieceEventHandler: This event is raised when a piece on this scene is released.

4.4.18. Achievement Scene



Attributes:

- private Achivement[] achievements: This is the array that has all the achivement information of the current user.

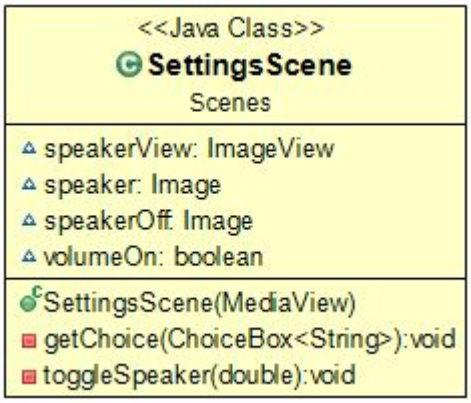
Constructor:

- public AchievementScene(): This constructor constructs the achievement scene interface. There is a list view here that shows all the achievements on the screen.

Methods:

- We do not need methods for this class.

4.4.19. Settings Scene



Attributes:

- We do not need attributes for this class.

Constructor:

- public SettingsScene(): This constructor constructs the settings scene interface.There are dropdowns, tick boxes and slides that change different settings.

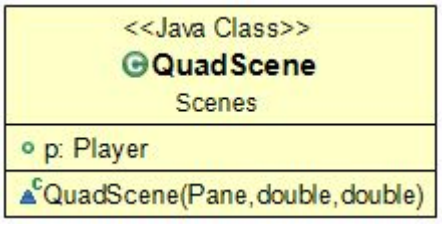
Constructor:

- public SettingsScene(): This constructor constructs the achievement scene interface. There is a list view here that shows all the achievements on the screen.

Methods:

- We do not need methods for this class.

4.4.20. QuadScene



Attributes:

- We do not need attributes for this class.

Constructor:

- public QuadScene(): Base constructor. Similar to javafx.application.Scene

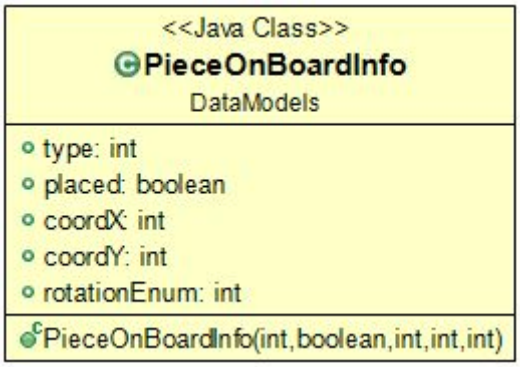
Constructor:

- public SettingsScene(): This constructor constructs the achievement scene interface. There is a list view here that shows all the achievements on the screen.

Methods:

- We do not need methods for this class.

4.4.21. Piece On Board Info



This class is a struct that keeps the piece's type, position on the board, rotation and whether or not if the piece is placed or not.

Attributes:

- public int type: Type of the piece
- public boolean placed: Whether or not the piece is placed
- public int coordX: x coordinate of the piece
- public int coordY y coordinate of the piece
- public int rotationEnum: rotation of the piece

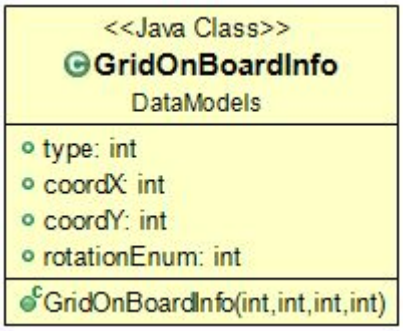
Constructor:

- public PieceOnBoardInfo(int type, boolean placed, int coordX, int coordY, int rotationEnum): Instantiates the object.

Methods:

- We do not need methods for this class.

4.4.22. Grid On Board Info



This class is a struct that keeps the grid's type, position on the board, and rotation.

Attributes:

- public int type: Type of the grid
- public int coordX: x coordinate of the grid
- public int coordY y coordinate of the grid
- public int rotationEnum: rotation of the grid

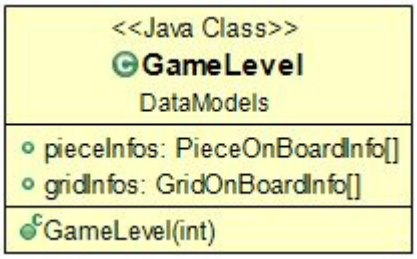
Constructor:

- public PieceOnBoardInfo(int type, int coordX, int coordY, int rotationEnum): Instantiates the object.

Methods:

- We do not need methods for this class.

4.4.23. GameLevel



This class represents a game level. It has grid and piece positions and rotations informations for scene to construct a game on the scene.

Attributes:

- public PieceOnBoardInfo[] pieceInfos: Array of piece informations.
- public GridOnBoardInfo[] gridInfos: Array of grid informations.

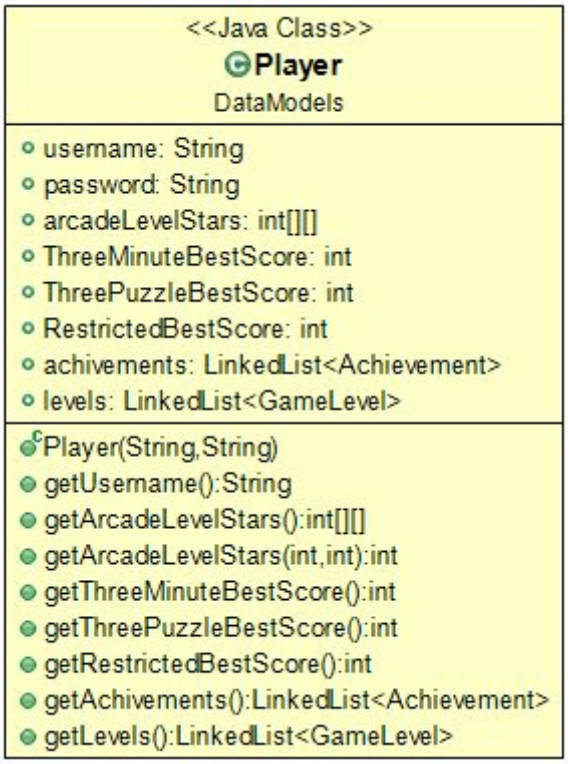
Constructor:

- public GameLevel(): Fetches the game level information from database.

Methods:

- We do not need methods for this class.

4.4.24. Player



This class represents a game level. It has grid and piece positions and rotations informations for scene to construct a game on the scene.

Attributes:

- public String username: Username
- public String password: Password
- public int[][] arcadeLevelStars: Stars of player for every level in arcade.

- public int ThreeMinuteBestScore: Best score of the user in three minutes mode
- public int ThreePuzzleBestScore: Best score of the user in three puzzle mode
- public int RestrictedBestScore: Best score of the user in restricted mode
- public LinkedList<Achievement> achievements: Achievement information of the player
- public LinkedList<GameLevel> levels: Created game levels of the user.

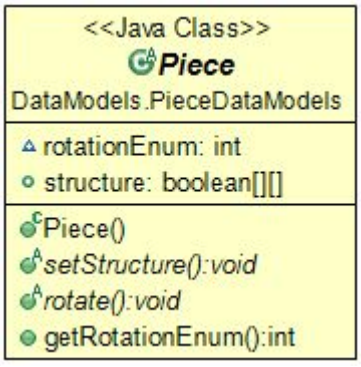
Constructor:

- public Player(): Initializes the attributes

Methods:

- public String getUsername(): Get username
- public int[][] getArcadeLevelStars(): get arcade level stars
- public int setArcadeLevelStars(int difficulty, int level): Sets arcade level stars
- public int getThreeMinuteBestScore(): Get three minute score
- public int getThreePuzzleBestScore(): Get three puzzle score
- public int getRestrictedBestScore(): Get restricted score
- public LinkedList<Achievement> getAchivements(): Get achievements.
- public LinkedList<GameLevel> getLevels(): Get custom created levels.

4.4.25. Piece



This is an abstract base class for all piece classes. There are specific piece classes that inherit from this class such as LPiece and TPiece. These classes represents a piece with its informations.

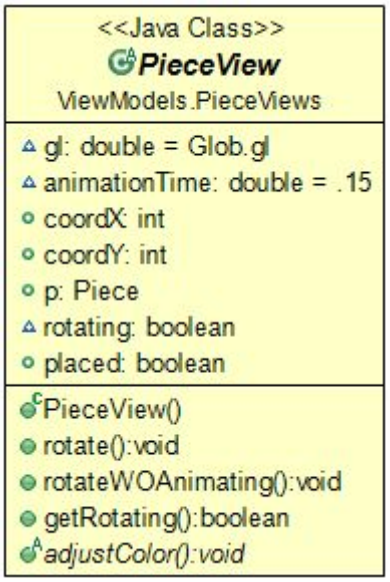
Attributes:

- private int rotationEnum: This gives piece's rotation. Different pieces can have different rotations. Some of them have 8 different rotations while some can have less.
- public boolean[][] structure: This is a 2D boolean array that defines the structure of the piece. If a place is true, piece occupies the position, if the place is false, it is not applying the position.

Methods:

- public abstract void setStructure(): Sets the structure of the piece when it rotates.
- public abstract void rotate(): Rotates the piece. Changes the rotation enum and sets the structure.
- public int getRotationEnum(): Get rotation enum.

4.4.26. PieceView



Attributes:

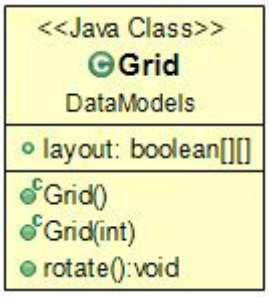
- private double gl: Length of side of every square.
- private double animationTime: Time for rotation animation.
- public int coordX: x coordinate of the piece.
- public int coordY: y coordinate of the piece.
- public Piece p: Piece object of the view.

- boolean rotating: Boolean flag that becomes true when rotating. So that rotate event is not send while rotation animation.
- public boolean placed: If the piece is placed or not.

Methods:

- public void rotate(): Rotates the polygon view itself.
- public void rotateWOAnimating(): Rotates the piece while constructing the level.
- public boolean getRotating(): Gets the boolean flag isRotating.
- public abstract void adjustColor(): Changes the color of the piece when it is placed or displaced.

4.4.27. Grid



Attributes:

- public boolean[][] layout: 2D boolean array that resembles the layout of the grid. False places indicated obstacles.

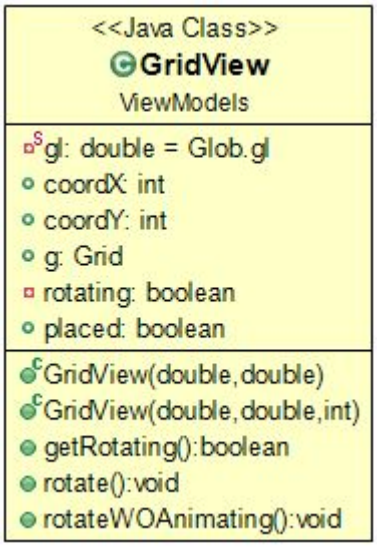
Constructor:

- public Grid(): Creates a default grid with no obstacle for the sandbox mode.
- public Grid(int type): Creates a grid that have unique obstacles depending on the type of the gird.

Methods:

- public void rotate(): Rotates the grid view.

4.4.28. GridView



Attributes:

- public int coordX: x coordinate of the grid.
- public int coordY: y coordinate of the grid.
- public Grid g: Abstract grid data model.
- private boolean rotating: Boolean flag that becomes true when rotating. So that rotate event is not send while rotation animation.
- public boolean placed: Is grid placed or not

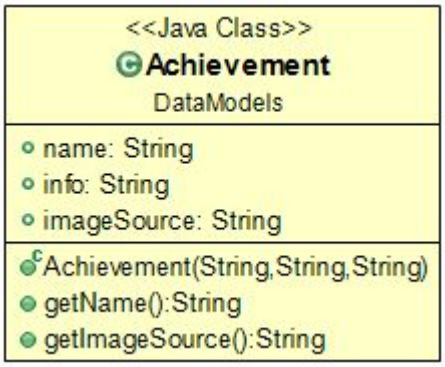
Constructor:

public GridView(double x, double y, int type)

Methods:

- public boolean getRotating(): Get the boolean flag isRotating.
- public void rotate(): Rotate the view
- public void rotateWOAnimating(): Rotates the grid while constructing the level.

4.4.29. Achievement



Attributes:

- public String name: name of the achievement
- public String info: Explanation of the achievement.
- public String imageSource: Image source of the achievement

Constructor:

- public Achievement(String name, String info, String imageSource): Initiates the attributes.

Methods:

- public String getName(): Get name
- public String getImageSource(): Get image source

5. Improvement Summary

- Design goals are updated to reflect the non-functional requirements in the Analysis Report. The Design Goals are updated for the goals to be testable.
- Deployment diagram is added to the Hardware Software Mapping section.
- Motivation for choosing Database is added to Persistent Data Management section.
- Boundary Conditions part is revised.
- Class diagram of Object Model is updated.
- Class descriptions are clarified.
- UML diagrams with respect to each class is added.

6. References

[1] “Quadrillion - SmartGames”, [Online] Available: https://www.smartgames.eu/uk/one-player-games/quadrillion , Accessed March 09, 2019

[2] “user1444_Quadrillion_ CS 319 Term Project - Spring 2019”, [Online] Available: https://github.com/user1444/Quadrillion , Accessed March 10, 2019

[3] B. Bruegge, A. H. Dutoit, Object-Oriented Software Engineering 3rd Edition, Using UML, Patterns, and Java, Prentice-Hall, 2010

[4] “Ideal Modeling & Diagramming Tool for Agile Team Collaboration”, [Online] Available: https://www.visual-paradigm.com/ , Accessed March 10, 2019

[5] “Balsamiq. Rapid, effective and fun wireframing software.”, [Online] Available: https://balsamiq.com/ , Accessed March 10, 2019

[6] “Enabling Open Innovation & Collaboration _ The Eclipse Foundation”, [Online] Available: https://www.eclipse.org/ , Accessed March 10, 2019

[7] “IntelliJ IDEA_ The Java IDE for Professional Developers by JetBrains”, [Online] Available: https://www.jetbrains.com/idea/ , Accessed March 10, 2019