Bilkent University

Department of Computer Engineering

CS 319 - Object-Oriented Software

CS319 Project

IQ Puzzler

Design Report

Group Name: Violet

Group Members

Ege Akın

Safa Aşkın

Betim Doğan

Alp Ertürk

Elif Gülşah Kaşdoğan

Overview

IQ Puzzler Pro is a logical computer game which is adapted from its board version. Our design aim is to create a flexible design to prevent bugs and handling the possible exceptions. Since we have limited time, we tried to make our implementation as easy as possible. For avoiding the complexity and handling the exceptions, we aim to create a design pattern with an elaborate detection

1. Introduction

In this section, we mainly describe our design choices and why did we choose these design choices. Also we discuss system features of our game.

1.1. Purpose of the system

IQ Puzzler Pro is a 2-D and 3-D logical puzzle game which is adapted from its board game version. Its design is implemented in such ways to meet the maximum user expectation. IQ Puzzler Pro aims to be portable, user-friendly and a brain-teasing challenging game which supports the development of cognitive skills. The goal of the player is to place geometric pieces of different colors and sizes to board or to create a specific shape with the given pieces.

1.2. Design Goals

Design goals are important steps for users to get the possible maximum satisfaction from the game. There are some non-functional requirements of the system as we mentioned before in our analysis report. These non-functional requirements are detailed below.

1.2.1. Criteria

Maintainability

Maintainability is an important point for attribution of quality in software. The design of the game is based on object-oriented programming concepts will enable the game to go through the changes easily. The multilayered design of the game provides flexibility that allows the system to be easily modified and meet the future demands.

Usability

Our aim is to create a game that is easy to use for the players. In general, usability is a substantial aspect for our game because it depends on effectiveness and efficiency of the game and affects satisfaction and comfort of the user. IQ Puzzler Pro has a user-friendly interface design to fulfill the user expectation. It will enable users to easily understand the directions of the game and to control with easily understandable buttons and control system. In addition to these, the game includes tutorial section, which is accessible from the main menu that explains and shows the game play in a detailed way.

Performance

Performance is another aspect that we mentioned before due to the fact that the players will not want to wait for a long time to get response while playing IQ Puzzler Pro. We will design our game with the aim for almost immediate response time. For avoiding decrease in computer's performance, we will not use high-resolution graphics in our game.

Reliability

Our aim is to provide users to a bug-free system, while we are designing the game. The system will be prepared for unexpected conditions such as unexpected inputs and while they happen, the game will not crash. For achieving this aim, we will use a testing procedure during the whole development stages. The testing procedure and the boundary conditions will be prepared elaborately for not missing any kind of unexpected situation which may cause system crashes.

Adaptability

Since we use Java as our programming language for development of our game, we will not have to worry about adaptability. Due to the fact that Java has cross-platform portability, users will be able to play IQ Puzzler Pro in all JRE installed platforms.

1.2.2. Trade-Offs

Memory vs Performance:

We will use object oriented concept and principles while we will be designing the IQ Puzzler Pro Game, so that it enables us to have efficient run time in the design. Therefore, we will not concern about memory usage.

Functionality vs Usability:

IQ Puzzler Pro is a game that aims to place the different puzzle pieces into correct places. We do not prefer to use complex interactions and game control system due to provide game to be easily understandable and we add tutorial option to improve usability. Due to the aim of our game is entertaining the users; we will spend more effort to make the system useful rather than developing the functionality more than necessary. The game will have simple instructions and a menu rather than complex and confusing ones.

Cost vs Reusability:

We changed our initial design slightly after first demo. Our code is generic and this allowed us to make desired changes in game easily. Time spent writing a generic code since we start coding benefit outweigh the cost. It saved us time during bug-fixing and code reviewing.

Functionality vs Robustness:

Since the main aim of IQ Puzzler Pro is to place various puzzle pieces in correct order, there can be various invalid actions while placement. Therefore, we will have some kind of precautions for handling with these exceptions and increasing the functionality. The exceptions will be determined carefully and the code will be developed with considering them.

Cost vs Portability:

Our game will only be implemented for desktop operating systems that run JRE, since it is implemented by Java. Consequently, the game will not be played in mobile platforms and the platforms that do not satisfy the requirements.

1.3. Definitions

Java: is a fast, secure and reliable programming language and computing platform that allows developers to write code for different systems [1].

JRE (Java Runtime Environment): is the combination of JVM (Java Virtual Machine), platform core classes and supported libraries and it is a part of the JDK (Java Development Kit) [2].

Cross-Platform: is a type of computer software that can be implemented on multiple computing platforms. A cross-platform application may run on every platform that satisfies the system requirements of the application [3].

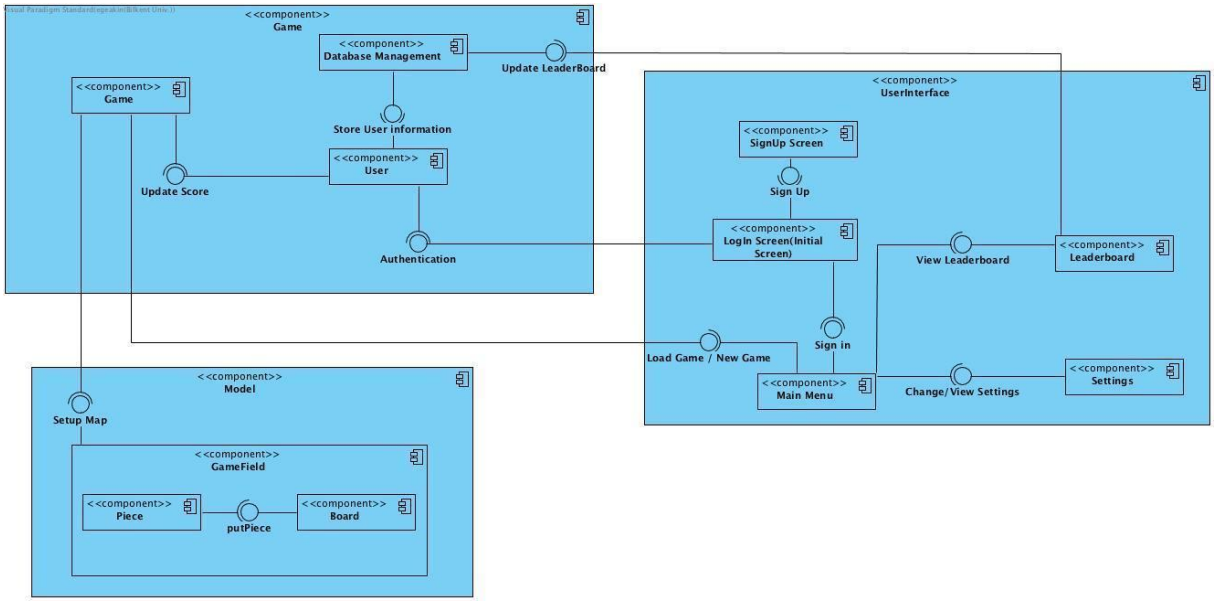
2. Software Architecture

In this section, we mainly describe software characteristics such as our design pattern , security issues, error handling and hardware-software mapping.

2.1. Subsystem Decomposition

We will introduce subsystems in this section. Decomposing the system into subsystems will reduce the coupling which in turn our system will be more concise and organized. Decomposing will allow us to construct a system which is more flexible to extensions and coherency will increase.

We used Façade design pattern for subsystem decomposition to reduce coupling. Incremented coherence and reduced coupling makes our decomposition easier to understand and maintain.



We used MVC pattern for decomposition, Game for Model, User Interface component represents view by providing the main user interface and Model component is a controller for system which will manipulate the Model.

We tried to divide systems according to their functionality to provide clarity about job definition of each component. Even if they are separated, each subsystem should communicate with others to keep system maintainable.

Controller component Game should communicate with the Model component to set up map for current game, Game Field arranges all items related to game session such as board and

pieces. Model component reaches other information related to game by communicating with the Game component.

User Interface component is constructed to provide communication between the Game component and the user. It also updates the database according to current score provided by the elements of the Game component. Other than those functionalities, User Interface component arranges sound and allows viewing changes made by use

2.2. Hardware / Software Mapping

Our game IQ Puzzler Pro will not need any specific built hardware in order to run properly. IQ Puzzler Pro will require keyboard and mouse for game play. Our game will work properly in different operating systems which require JavaFX libraries and Java Development kits. For the storage of the game as levels, board and pieces; we will store them in external MySQL database machine. Also our game will store user profiles of users and their scores in an external MySQL database machine.

2.3. Persistent Data Management

IQ Puzzler Pro will store user profiles and their high scores in a database. Our game will have user preferences settings, so these data will be able to modified during game by user.

Our game has a leaderboard which compares the scores of different player's scores for each level and creates a descending score list. Different users from different platforms should access leaderboard and scores, so for these reasons we choose to use a database for leaderboard. Our game stores the record of unlocked levels of each user and level score. We decide to use a database instead of a file system. Because multiple users access our system from different computers to look for scores of different users. If we decided to use a file

system, users would be able to manipulate the game so it is not secure to use a file system. For database we will use MySQL since most of the team members are familiar with MySQL rather than others.

2.4. Access Control and Security

There will be a user authentication, so users will enter the game with username and password. Initial screen forces user to either sign in or sign up. After sign in/up user will be able to play the game. Username and password will be stored in database and authentication check will be made according to stored data.

2.5. Global Control Software

IQ Puzzler Pro Access Matrix

In multi user systems, different actors may have access to different functions of classes in program. However, in our game we only have one actor which is player so our access matrix shows functions and classes which user can access directly.

Actors/ Classes	Board	Game	Piece	Settings	User
Player	putPiece( )	startGame() loadSelectedLevel(int levelID) accessLeaderboard( )	rotateMatrix()	changeSoundLevel() mute() changeTheme()	setPasswpord(string password)

2.6. Boundary Conditions

Initialization

IQ Puzzler Pro does not require an install because it does not have .exe extension file. Our game will start with an executable .jar file. The game can be transferred by copying .jar file to other computers.

Termination

Player can terminate the game by clicking “Exit” button or by closing .jar file. If player clicks the exit button, there will be a pop up window which asks to user to save the current game.

Error

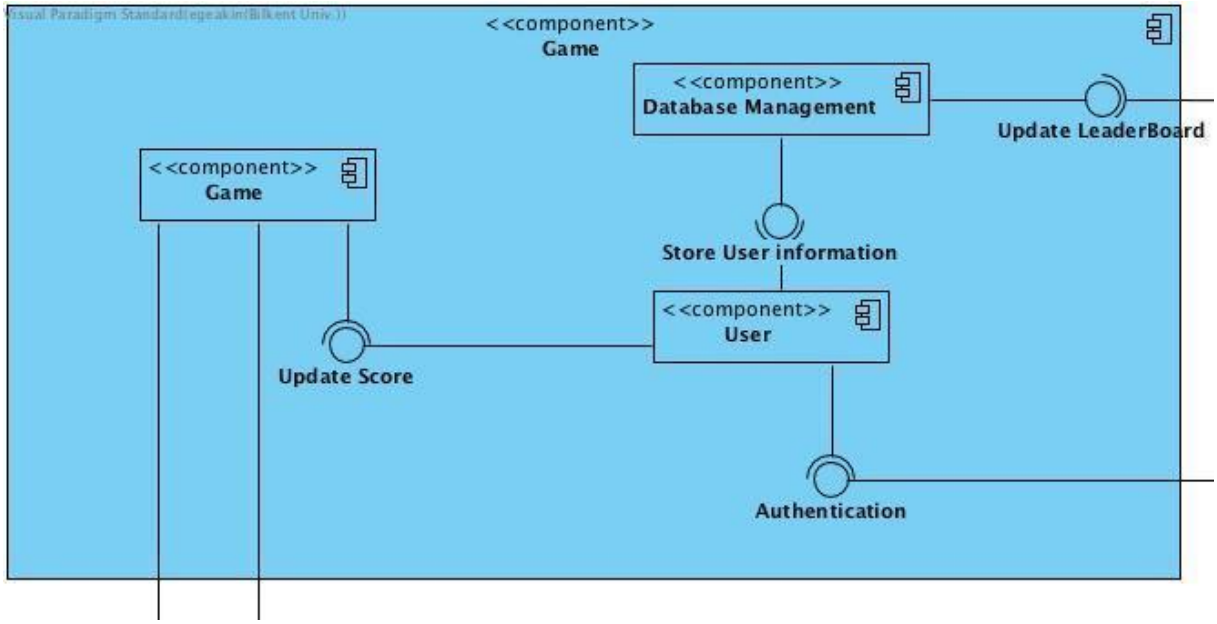
In case of an error while playing the game there will be a pop up screen which displays a message about loading to user. In such a case, the game should be restarted or data files should be changed.

3. Subsystem Services

In this section, we describe behavioral units in our system and classes that form these subsystems. Three main subsystems of our game are Game Subsystem, User Interface Subsystem, Model Subsystem.

3.1. Game Subsystem

Game System is responsible for connection between Game Field components, User Interface components and Database System which will record the user information and leaderboard data.



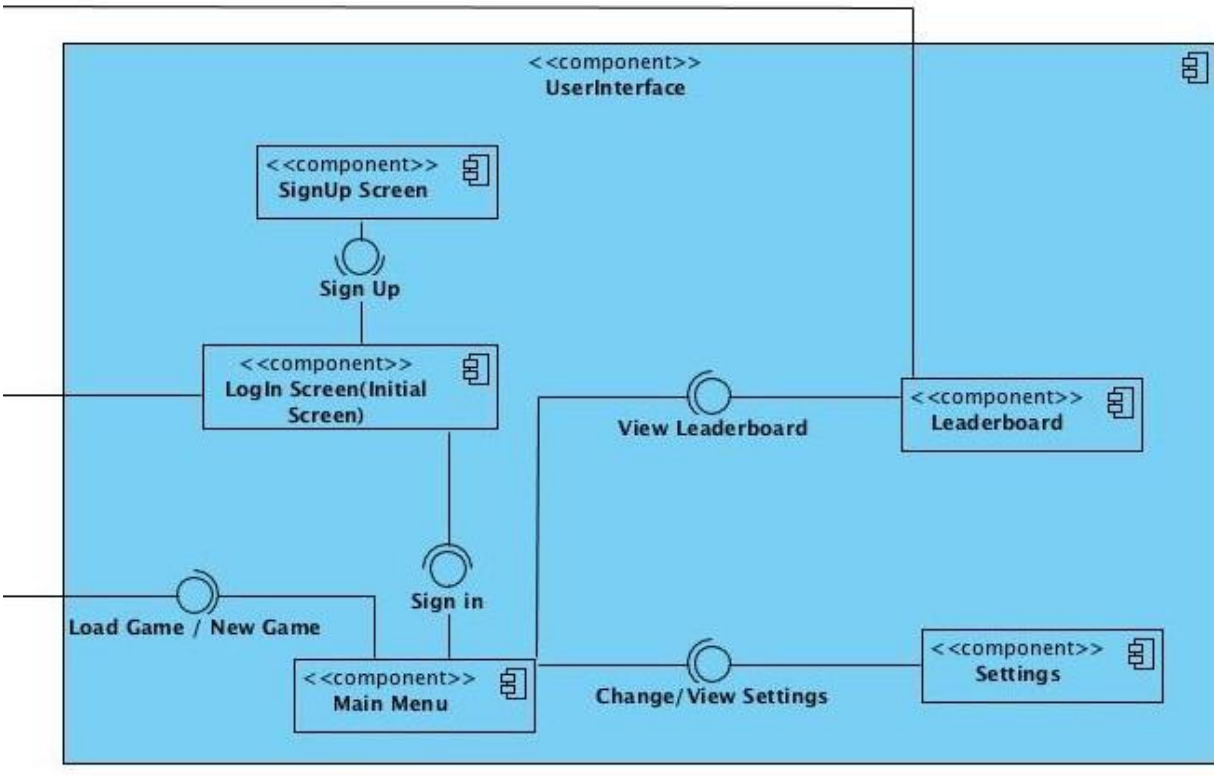
Game Class arranges operations like starting or finishing a game, changing settings by accessing User Interface subsystem components, sets the high score and loads a previously played game by accessing Database.

User class represents a user with a username and a valid password and the arrays representing passed levels. For each game array and score, it will be updated as needed by provided methods. User's data will be stored in database system.

Database stores scores and user information. Database Management updates leaderboard and stores the user information. User interface will access database for authentication during sign in process. User accesses the Game Class for getting new score and score stored in Database will be updated via User object. In short User class acts like a interface between Game and Database.

3.2. User Interface Subsystem

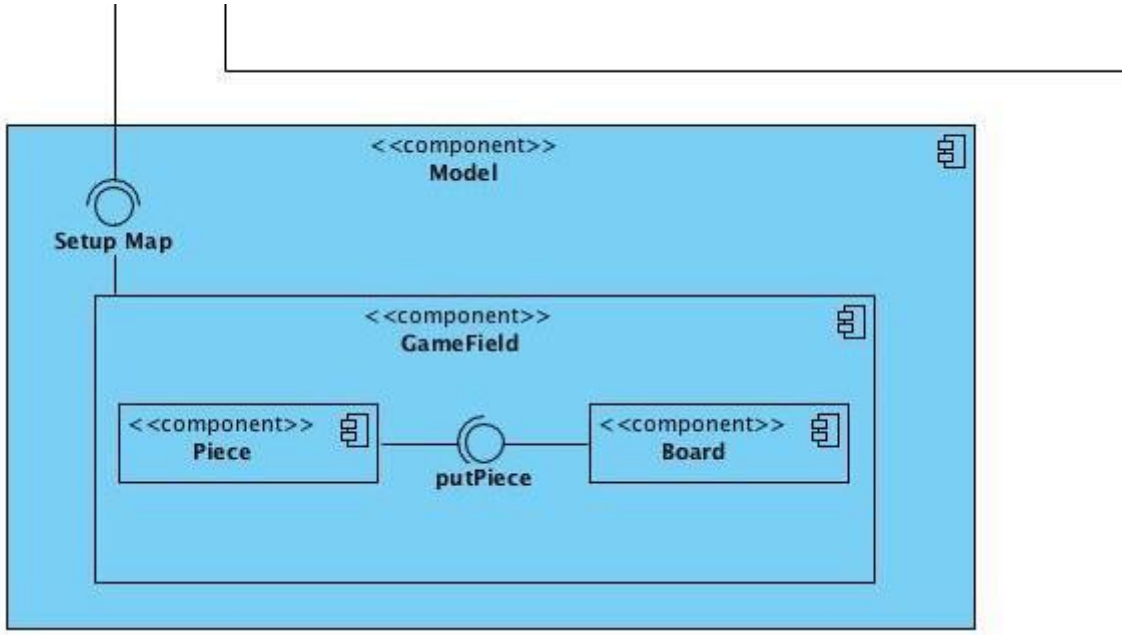
This subsystem shows the View part of MVC design pattern. Each component seen in User Interface subsystem represents a screen and interaction between screens and Game Subsystem.



LogIn Screen is the initial screen in game interface, user should either sign in or sign up to play a game. After signing in, user will access MainMenu Screen and can perform different actions such as start a new game, load a game, change and view the settings, view the leaderboard. UserInterface will be connected Game subsystem via loading or starting a new game. A new Game object will be instantiated by a call coming from UserInterface.

3.3. Model Subsystem

Model subsystem consists of game objects of type Board and Piece. All game components will be arranged by Game object via setting up map. Game subsystem acts like Controller of MVC design pattern. Board and piece objects will be drawn according to information coming from control model.

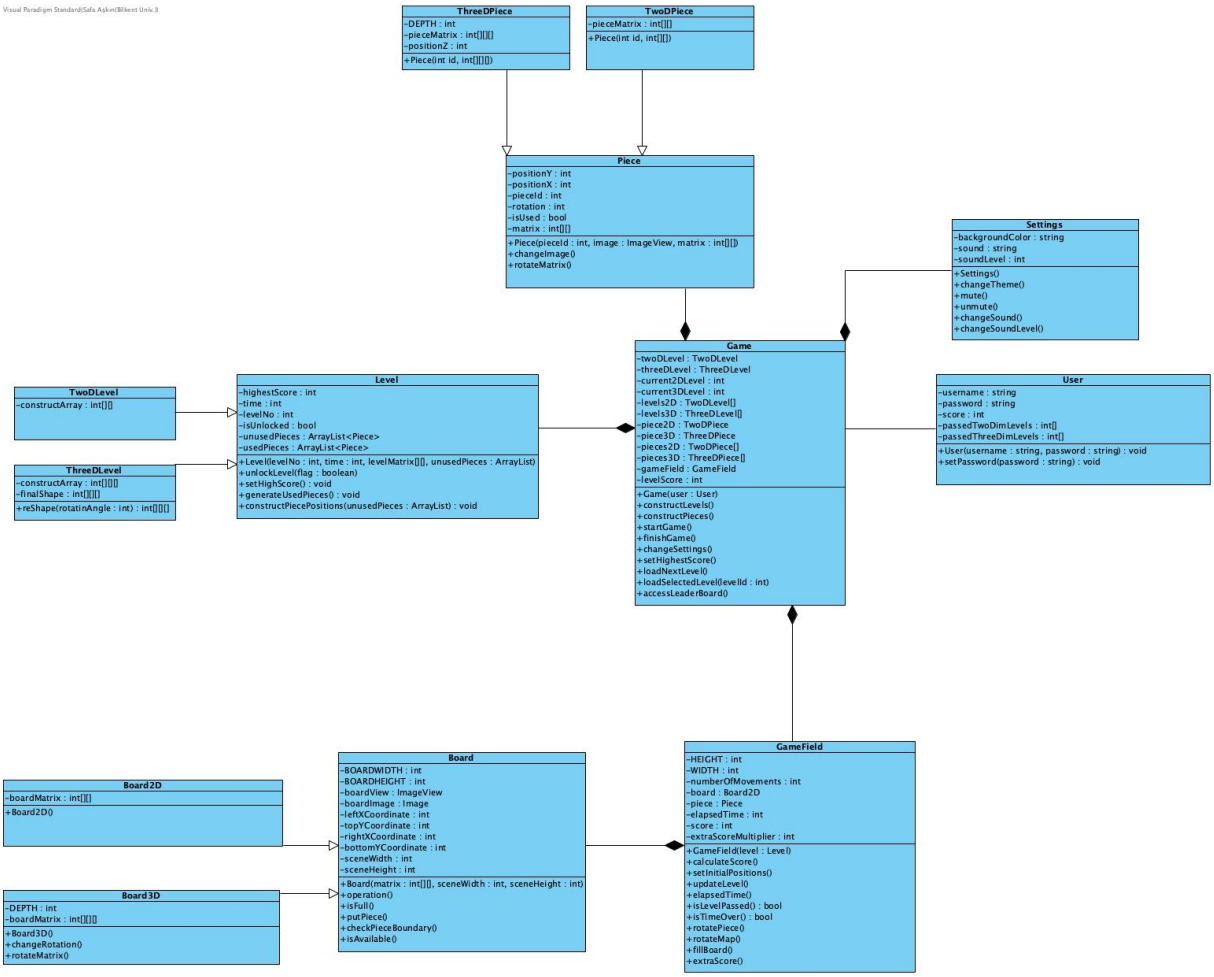


Piece class represents individual pieces which will be inserted to board. Piece has id, position, rotation and size attributes, images can be attached to a piece. Determining if a piece is used or not will be done by a boolean attribute isUsed inside a piece object for simplicity. Methods allow us to change rotation of a piece and image attached.
Game Field object has size attributes, a board object, a piece object, elapsed time for game set, current score of game and number of moves have been made so far. Game Field is the class which sets the game essential. Class organizes and updates the pieces, board, time and

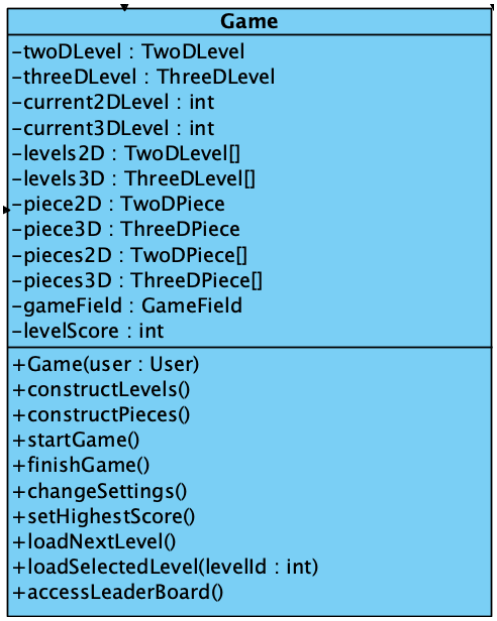
rotations regularly. Basically, each object manages its own data and Game manages objects and all game will be set properly. Score will be calculated in this class and will be recorded to database through Game subsystem.

Board is object representing board which player will insert pieces; it is actually the main play field. It has size attributes and functions to check existence of available positions and whether the board is full or not.

4. Class Interfaces



4.1. Game Class



Attributes:

- private Level level: This attribute is an initialization of Level object, which initializes attributes of a level.
- private int highestScore: This attribute stores the highest score of the user which is initially 0.
- private int current2DLevel: This attribute stores the index of current 2D level.
- private int current3DLevel: This attribute stores the index of current 3D level.
- private GameField gameField: This attribute is an instantiation of GameField class which stores physical objects of the game. It is also responsible with user interactions such as calculating and updating score and time.
- private int levelScore: This attribute stores the score of the level which is initially 0 at the beginning of the level.
- private Level [] levels2D: This attribute stores all previously generated maps of 2D levels which are generated by Game, in an array.
- private Level [] levels3D: This attribute stores all previously generated maps of 3D levels which are generated by Game, in an array.
- private Piece2D 2DPiece: This attribute initializes one of twelve distinct pieces.
- private piece3D 3DPiece: This attribute initialize the Piece3D class.

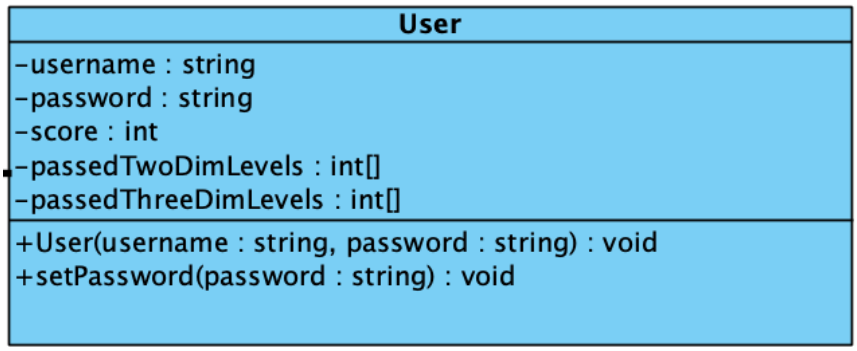
Constructors:

- public Game(User user): Default constructor for Game class, which simply initialize the game.

Methods:

- public void startGame: This method simply starts the game.
- public void finishGame(): This method simply finishes the game.
- public void constructLevels(): This method instantiates all levels with appropriate views.
- public void constructPieces(): This method instantiates all pieces with appropriate views and matrixes.
- public void changeSettings(): This method provides user to change settings of the game which is open sound or not.
- public void loadSelectedLevel(): This method simply loads specific level that user selects from load game screen.
- public void setHighestScore(int highestScore): If user passes the previous highest score, this method updates his/her highest score.
- public boolean isGameFinished(): This method simply checks if the game is over or not and returns 1 if game is over and 0, otherwise.
- public void nextLevel(): This method determines the next level and passes on next level.
- public void loadGame(): This method provides user to load a previous game.
- public void accessLeaderBoard(): This method simply accesses leaderboard.

4.2. User Class



Attributes:

- private string username: This attribute stores the name of user.
- private string password: This attribute stores the password of user.
- private int score: This attribute stores the score of user.
- private int [] passedTwoDimLevels: This attribute stores the passed 2D levels.
- private int [] passedThreeDimLevels: This attribute stores the passed 3D levels.

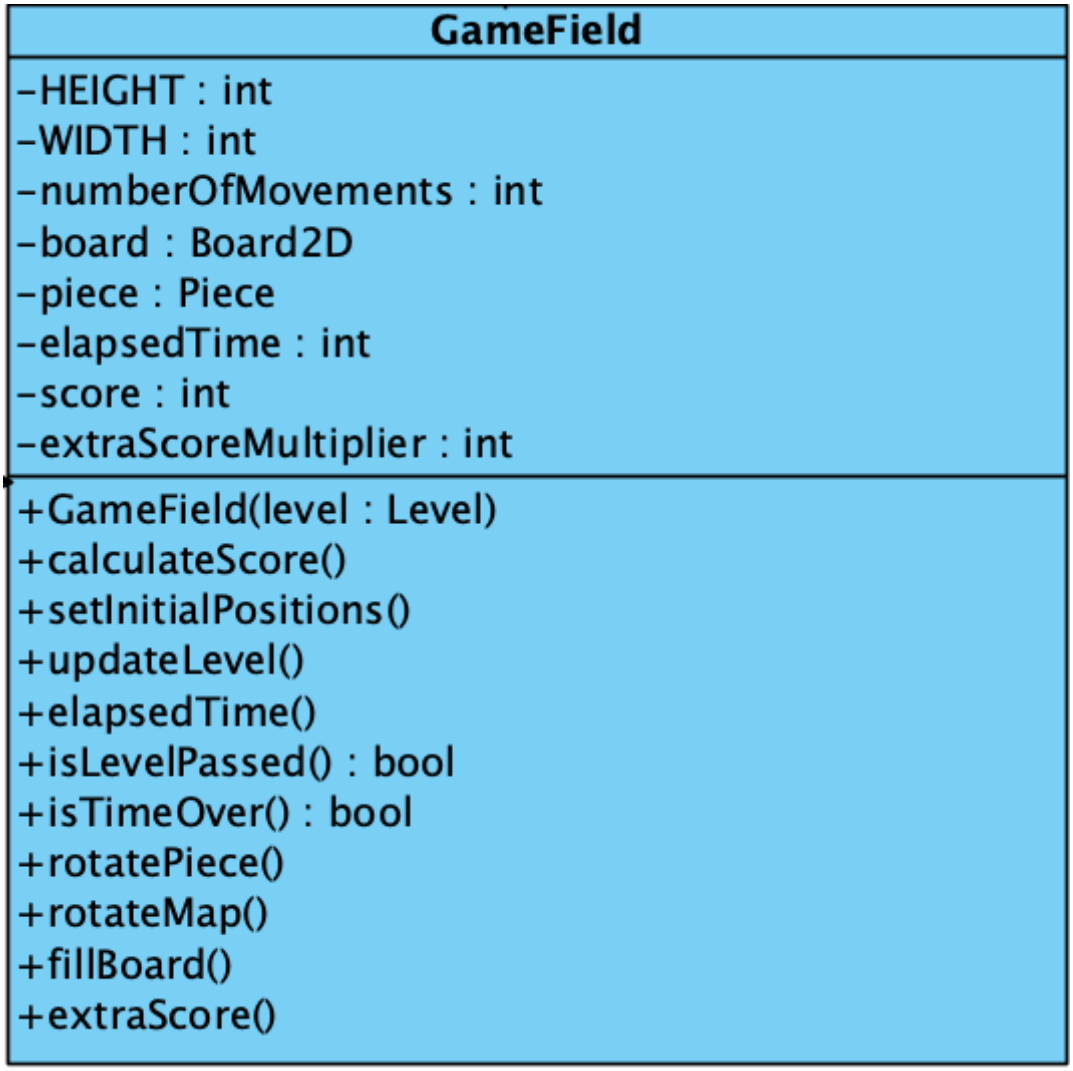
Constructors:

- public User(string username, string password): This constructor initializes the user class with given username and password.

Methods:

- public void setPassword(string password): This method simply sets the new password of user with given parameter.

4.3. GameField Class



Attributes:

- private const int HEIGHT: This attribute stores the height of the game window.
- private const int WIDTH: This attribute stores the width of the game window.
- private board Board2D: This attribute initializes the Board2D object which is for initializing the physical map of the current level in the 2D game mode.
- private Piece piece: This attribute is for initialization of Piece class.
- private int elapsedTime: This attribute is for keeping track of elapsed time in the game which is 0 at the beginning of a level.
- private int score: This attribute is for keeping track of score of current level.

- private int extraScoreMultiplier : This attribute simply keeps the score multiplier for extra scoring calculations.
- private int numberOfMovements : This attribute counts the number of user movements to calculate a game score at the end.

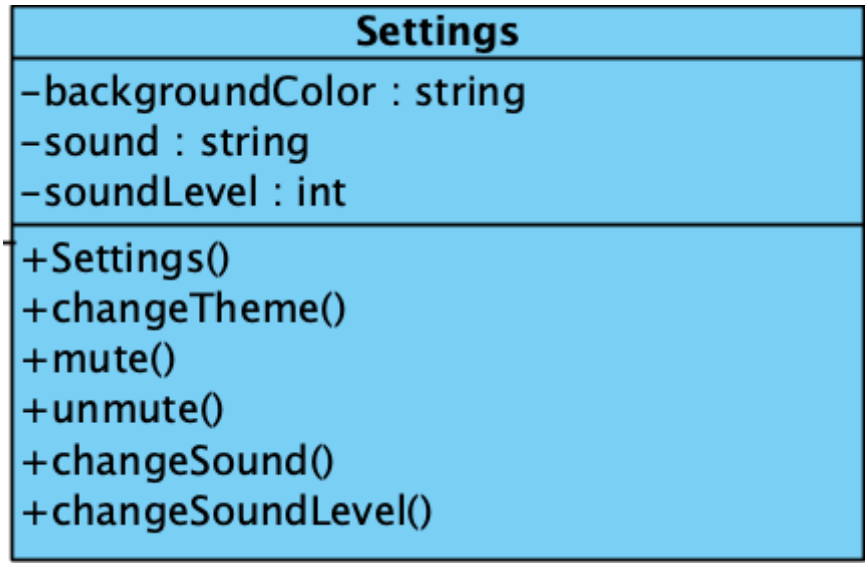
Constructors:

- public GameField(level Level): Default constructor which initializes the physical map of Game by using level's properties.

Methods:

- public void calculateScore(): This method calculates and updates the score when the level is passed by using elapsed time and number of movements.
- public void setInitialPositions(): This method sets the initial positions of components in the window.
- public void fillBoard(): This method fills the occupied positions of board view when game starts.
- public void updateLevel(): If user passes the current game, this method updates the level.
- public boolean isLevelPassed(): This method determines whether the game is passed or not.
- public boolean isTimeOver: This method determines whether elapsed time is passed at the boundary time or not.
- public void rotatePiece(): This method simply rotates the martrix and view of selected piece.
- public void rotateMap(): This method simply rotates the map. This method is for 3D game mode because in 2D game mode there is no need to rotate map.
- public void extraScore(): This method updates multiplier when user completes a extra score objective.
- public void elapsedTime(): This method basically keeps the elapsed time during game period to calculate score and finish game if time is over.

4.4. Settings Class



Attributes:

- private String backgroundColor: This attribute stores the color of the background in a string
- private String sound: This attribute stores the sound of the game.
- private int soundLevel: This attribute keeps track of level of sound in the game.

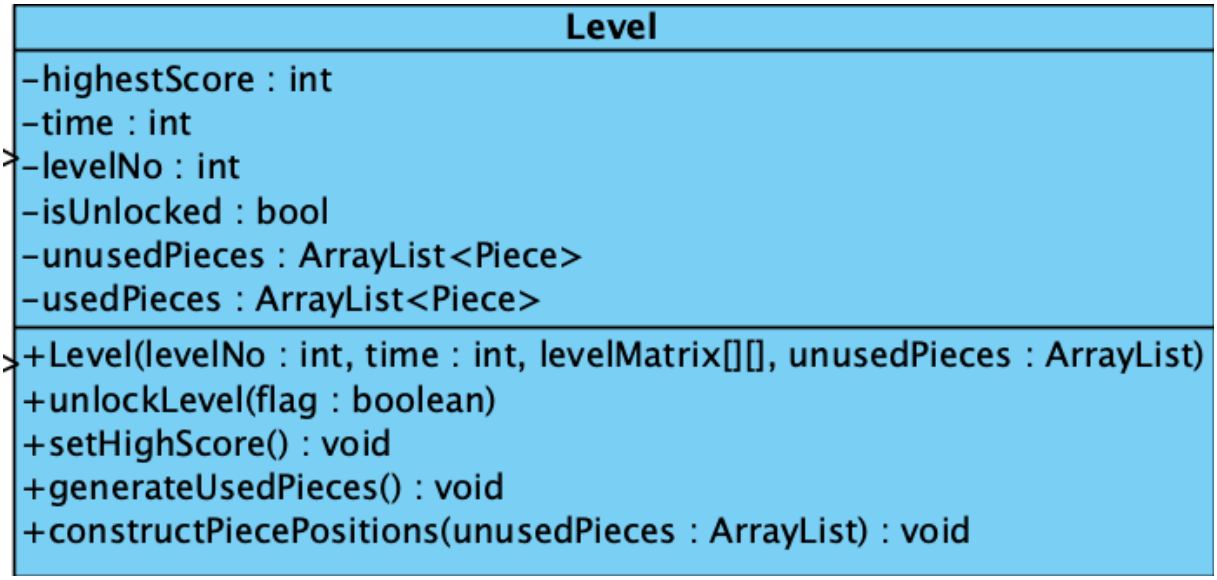
Constructors:

- public Settings(): Default constructor which initializes the Settings class.

Methods:

- public void changeTheme(): This method changes the theme of the board.
- public void mute(): This method mutes the sound in the game.
- public void unmute(): This method unmutes the sound if the sound is muted before.
- public void changeSound(): This method changes the sound of game with another sound.
- public void changeSoundLevel(): This method increases or decreases the level of sound.

4.5. Level Class



Attributes:

- private int highestScore: This attribute stores the highest score that is made in a specific level. It's zero as default and when user passes the level, Game updates it.
- private int time: As the user passes levels, the game will become more difficult. Therefore, every level has its unique time limit and this attribute keeps the time limit of a level.
- private int levelNo: This attribute stores the number of the levels.
- private bool isLocked: This attribute stores whether the current level is passed before or not. If it is passed before, or it is the last played level that is not finished yet, isLocked is false and true otherwise.
- private ArrayList<> unusedPieces: This attribute stores the pieces that is not used, in other words the pieces that is not on the board.
- private Piece ArrayList<> usedPieces: This attribute stores the usedPieces in the board. For example if a player moves a piece and puts the piece to the board, this piece becomes a usedPiece. So usedPieces stores the pieces that are on the board.

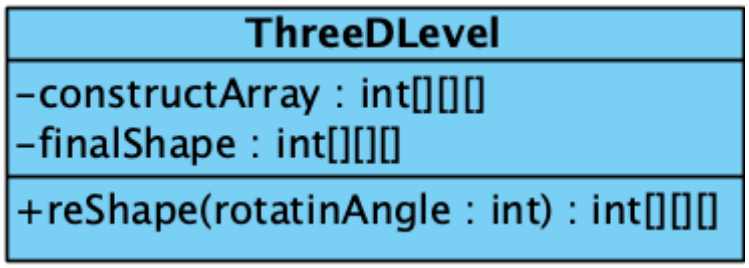
Constructors:

- public Level(): Default constructor that initializes the Level object.

Methods:

- public void updateScore(): This method updates the highest score.
- public void unlockLevel(): If user passes a level, this method unlocks the level so that user can access this unlockedLevel.
- setHighScore(): Method, which updates user's highest score for current level.
- generateUsedPieces(): Fill board with pieces, which can differ for each level in the beginning of current level. These pieces will not be able to move by user interaction.
- constructPiecePositions(unusedPieces: ArrayList): Assign pieces' positions on screen which user will use in the beginning of current level.

4.6. ThreeDLevel Class



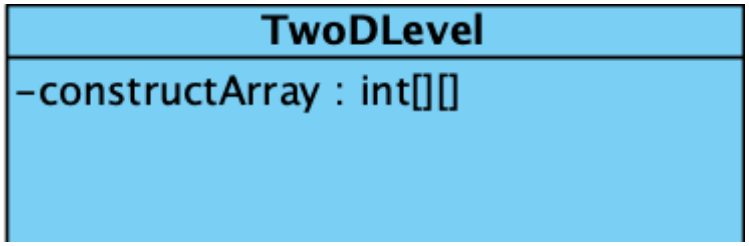
Attributes:

- private int[][][] constructArray: This array keeps the used pieces and occupied positions for each level specifically.
- private int[][][] finalShape: This attribute keeps the 3D matrix of final shape to decide whether the game finished or not.

Methods:

- void setReShape(): This method changes 3D shapes mapping in the screen.

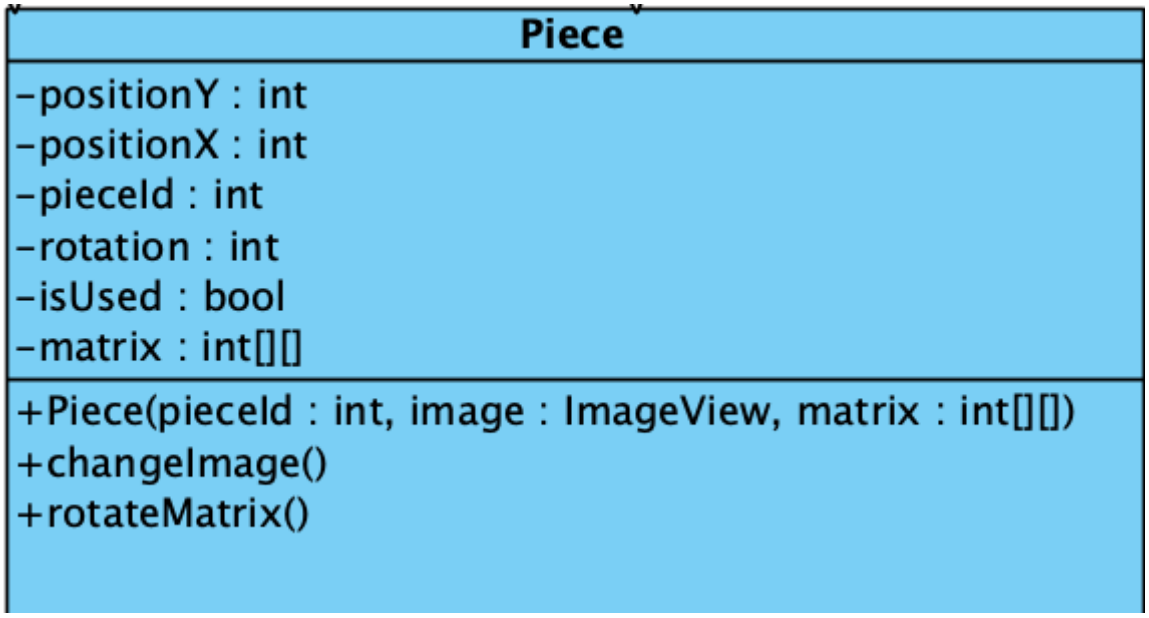
4.7. TwoDLevel Class



Attributes:

- private int[][] constructArray: This array keeps the used pieces and occupied positions for each level specifically.

4.8. Piece Class



Attributes:

- private int rotation: This attribute stores the current rotation of the piece as integer to decide which image of the piece will be used.
- private int positionX: This attribute holds the X coordinate of a piece object to place it into proper position when the game starts.
- private int positionY: This attribute holds the Y coordinate of a piece object to place it into proper position when the game starts.

- private int pieceID: This attribute holds the unique piece id to distinguish pieces.
- private bool isUsed: This attribute stores the information of a piece about its current usage situation. If current piece is in the boundary of board, isUsed attribute is true. Else isUsed attribute is false.
- private int[][]: This attribute simply stores the piece's view features by reflecting it to a square matrix.

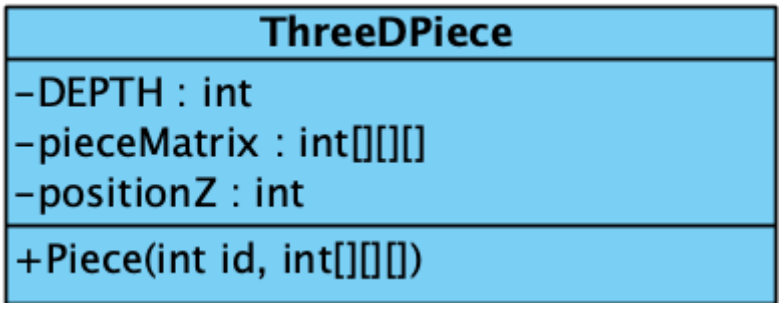
Constructor:

- public Piece(int pieceId, ImageView image, int matrix[][]) : This constructor basically constructs Piece from given pieceId, piece image and a piece matrix.

Methods:

- void changeRotation(): This method sets the new matrix and width-height values after its rotation.
- void changeImage(): This method sets the new image of piece after rotation.
- void rotateMatrix(): When we rotate a piece, we must also rotate its matrix that describes the view of current piece.. This method provides rotating piece's matrix when a rotation occurs.

4.9. 3DPiece Class



Attributes:

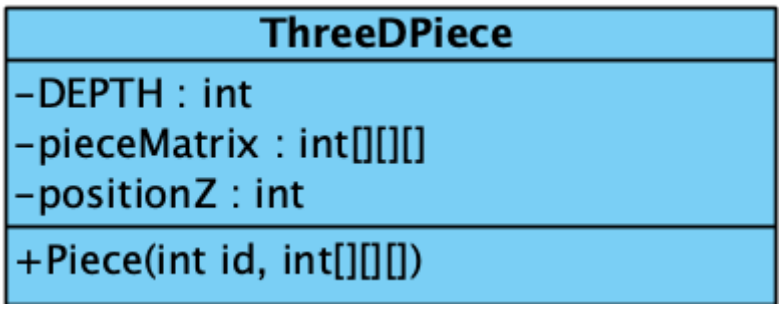
- private int DEPTH: This attribute keeps the depth of piece matrix as constant.
- private int[][][] pieceMatrix: Keeps the size and boarders of piece in a 3D array, to properly place it on game board.

- private int positionZ: In addition to its 2D version, 3D pieces also keeps the Z coordinate.

Constructors:

- Piece(): It initializes the piece with given images and values.

4.10. 2DPiece Class



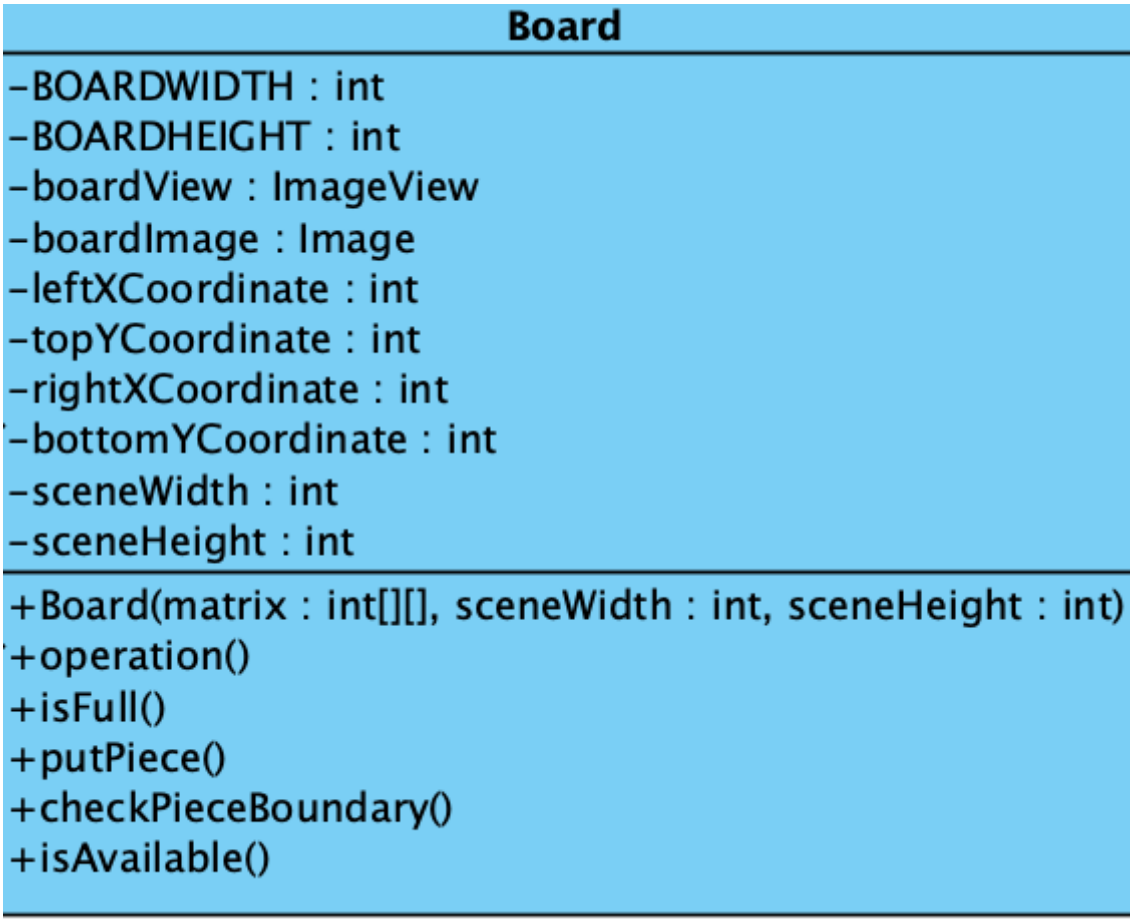
Attributes:

- private int[][] pieceMatrix: Keeps the size and borders of piece in a 2D array, to properly place it on game board.

Constructors:

- Piece(): It initializes the piece with given images and values.

4.11. Board Class



Attributes:

- private int BOARDWIDTH: This attribute keeps the width of Board matrix as constant.
- private int BOARDHEIGHT: This attribute keeps the height of Board matrix as constant.
- private boardImage: This attribute keeps boards image which can accessed by url of an image.
- private boardView: This attribute stores the board image as image view.
- private int leftXCoordinate: This attribute simply stores the left X coordinate of the board.
- private int topYCoordinate: This attribute simply stores the top Y coordinate of the board.
- private int rightXCoordinate: This attribute simply stores the right X coordinate of the board.

- private int bottomYCoordinate: This attribute simply stores the bottom Y coordinate of the board.
- private int sceneWidth: This attribute stores the width of scene to position the board properly to the scene.
- private int sceneHeight: This attribute stores the height of scene to position the board properly to the scene.

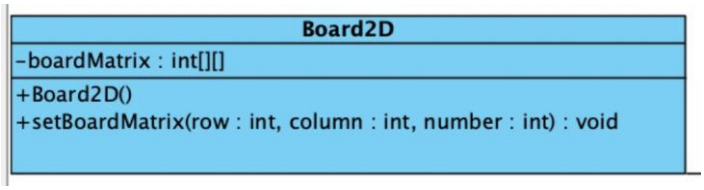
Constructors:

- public Board(int matrix[][], int sceneWidth, int sceneHeight): This constructor initializes the Board.

Methods:

- public boolean isFull(): This method checks whether the board is full or not by using board matrix and returns boolean.
- public boolean isAvailable(): Checks whether the position where user trying to place the piece is available or not.
- public void putPiece(): This method fixes the position of the piece on the board if put by user is not proper enough.
- public boolean checkPieceBoundary(): This method checks whether the piece that is using by user is inside the board boundary.

4.12. Board2D Class



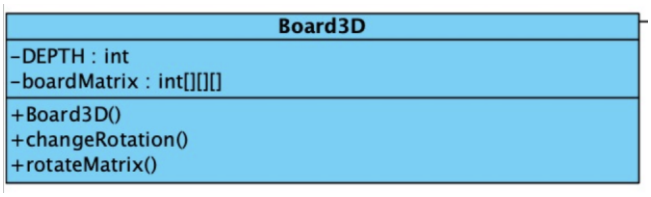
Attributes:

- private int[][] boardMatrix: This attribute keeps the board matrix in 2D array.

Methods:

- public int[][] getBoardMatrix(): This method returns the board matrix.

4.13. Board3D Class



Attributes:

- private int[][][] boardMatrix: This attribute keeps the board matrix in 3D array.
- private int DEPTH : Other than 2D board, this attribute keeps the depth of the 3D board matrix.

Methods:

- public int[][][] getBoardMatrix(): This method returns the 3D board matrix.
- public void setBoardMatrix(): This method sets the board matrix with initial conditions.
- public void changeRotation() : This method changes the 3D boards rotation with given commands.

5. Improvement Summary

We have done considerable changes in the second iteration of our design report. In our first implementation we used fixed size of components. We realized that this design may cause problems in computers which have different monitor size. We changed the component sizes accordingly, program now determines the size of board and pieces by looking at the current PC's screen size. We also provide the rotation functionality for pieces.

We planned to use file system and database but the way we use it might be problematic, we reconsidered the conditions and decided to use only database. Database will be sufficient for our game.

We review each other's code to make our code easy to read and easy to debug. We changed some classes' internal design but since we follow object oriented design principles these changes did not affect other components that much.

We've fixed some object oriented problems and replace some of the methods into proper classes to avoid maintenance problems and to ease future improvements.

We've added an extra scoring functionality to our game in order to make it more attractive. A random square at game board is shining in random time and if user would place the proper piece on this shining square, user will gain extra score.

6. References

[1] Oracle."What is Java technology and why do I need it?". Online: https://java.com/en/download/faq/whatis_java.xml [Nov. 10, 2018].

[2] Technopedia. "Java Runtime Environment (JRE)". Online: https://www.techopedia.com/definition/5442/java-runtime-environment-jre [Nov. 8, 2018]

[3] "Cross-platform software" Online: http://www.wikizeroo.net/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvQ3Jvc3MtcGxhdGZvcm1fc29mdHdhcmU [Nov. 8, 2018].