CS 319

Object- Oriented Software Engineering

Design Report

IQ Puzzler Pro

Unknown - Group 1-I

Derviş Mehmed Barutçu

Elif Beril Şaylı

Giray Baha Kezer

Zeynep Ayça Çam

Table of Contents

1.Introduction

1.1. Purpose of the System

Iq Puzzler Pro is a combined mind game with its 2D and 3D extensions. Its design was implemented in way to broad your thinking. Users could get higher satisfaction if they pass the levels step by step from beginner to expert. Iq Puzzler Pro is thought to be created as user friendly and challenging game which consists of different boards and different paths. The player’s target is to accomplish level requirements in his way. His way should be in stage so as to do what related level requires, also there need to be faster time counts for scoreboard. Except for game’s high performance it also offers user-friendly and pleasing UI.

1.2. Design Goals

Design is another important point in order to create system, it is much helpful for guiding what to focus on that system. There are some lower efficient requirements as we wrote in our analysis report they are our focus objects. Following steps will be part of our design goals.

1.2.1 Criteria

End User Criteria

Usability:

Iq puzzler Pro is a form that combination of entertainment and mind tricks in 2D and 3D space. It can be seen as much more harder form of 3D tetris on a flat board. Our game will provide users user-friendly, funny interface, related noisy sounds for incorrect tries so it is funny to play and easy to use the game. For instance, imagine a understandable game home menu that shows how to play, settings, directly music on off buttons on settings button. In the game panel rotations can be seen by users, thus they dont have to imagine how to rotate in that piece in that way or in that way it is easier rotation for users. The section levels can be chosen easily by the user before the game beginning. The rotates and picked suitable holes can be seen on game illustration synchronously while playing with pieces and board holes. Ease of Learning for how to play can be counted as another ease of use for users, since pictures for different boards and pieces can be seen before the game via that button. User can use keyboard for exit for directly quit button for quitting easily.

Performance:

Performance could be seen as another important design goals either, which is necessary for games itself. We are using JAVAFX library for our game's 3D parts with JAVA source codes on NetBeans 8.2 environment. Thanks to using this programming language and its library support we can have better platform performance and ease of use. We will use JAVA codes to program our game with using JAVAFX in 2D and 3D modes.

Maintenance Criteria

Extendibility:

The game design provided for our game we will contact the users via e-mails related their valuable feedbacks to progress the game. We write our e-mail addresses on the game credit part to be easily connected them. Also we could advance the levels and pieces in difficulty for expert game users if they demand.

Modifiability:

The Iq Puzzler Pro game is designed for both multi or single gamers. Two game playing modes are offered them in different boards and levels. These features could be easily modified for different levels or different volume pieces. Thereby, thanks to extensions or directly changes our game can be modified without putting so much effort on it.

Reusability:

There are few subsystems in our game, which can be integrated to other different games with logic. For example our game can be seen as 3D flat board tetris with rotating and selecting but not falling pieces. It means our game logic and features can be integrated some part of likely tetris games. These kinds platforms and rotates and pieces selection games or directly mind trick games can be used for integration our features.

1.3 Definitions, Ancronyms and Abbreviations

UI[1]: User Interface

2. Software Architecture

2.1 Subsystem Decomposition

2.2 Hardware / Software Mapping

2.2.1 Software : Our project is being developed in JAVA programming language. We are using NetBeans 8.2 environment and JAVAFX libraries[1] to program and create UI and game interfaces, thereby we decided to use JAVA.

2.2.2 Hardware : As for hardware, the player uses mouse for selecting, dragging, placing, rotating and flipping. As to computer part, most of the computers should run our game because it does not require high specification.

2.3 Persistent Data Management

Although our program doesn't require any complex data systems, our program keeps players data, which are players names, time, moves and scores of that player. Our program keeps these data on text files. No database are not being used for this project.

2.4 Access Control and Security

Our project does not require internet connection so outside of that computer, nobody can change our game data or specifications of levels. As some of the necessary variables were defined as constant or private, other classes cannot see or modify that variables. In addition, our project doesn't hold sensitive information of the user, thus we don't provide any security.

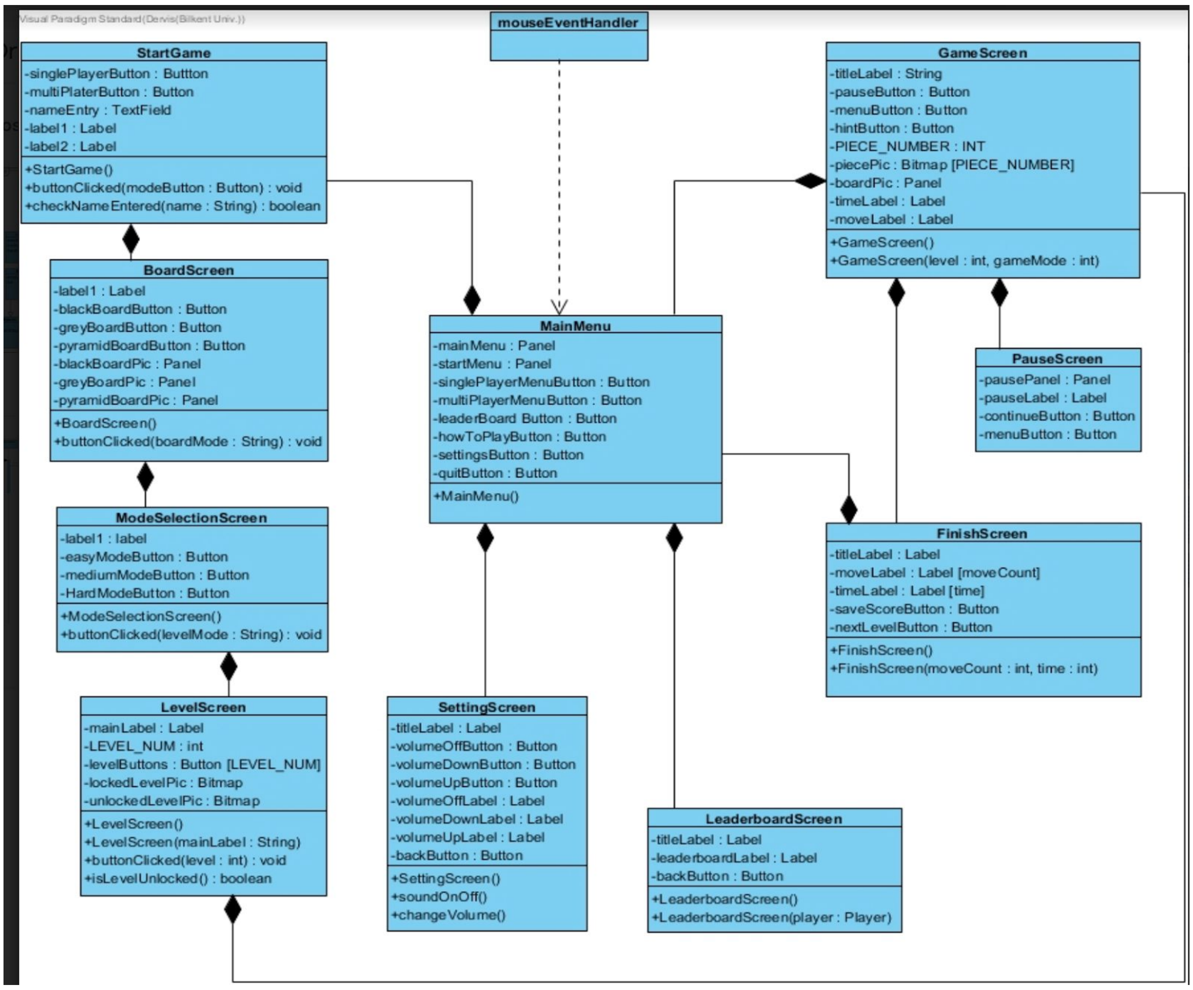
2.5 Boundary Conditions

When the player finishes the level which the player played, pop-up appears.With this pop-up , it shows the player's move count, time that spend in that level and score of the user.Moreover, it has two buttons to go next level or return to main menu.

3. Subsystem Services

The complete class diagram can be seen below. Our UI subsystem consists of “MainMenu”, “StartGame”, ”GameScreen”, ”BoardScreen”, ”PauseScreen”, ”FinishScreen”, ”ModeSelectionScreen”, ”LevelScreen”, ”SettingScreen”, ”LeaderBoardScreen” as panels. Game logic consists of different functional buttons, piece and board objects and different labels for titles. Our game logic has “TimeManager”, “SoundManager”,”LeaderBoard”,”Board”,”Pieces”,”Levels”,”SinglePlayerGame”,”MultiPlayerGame” and ”Player” game features to increase content. Lastly it has different board and pieces objects for complete logic to play.

3.1 Interface Layer



(Graph 1)

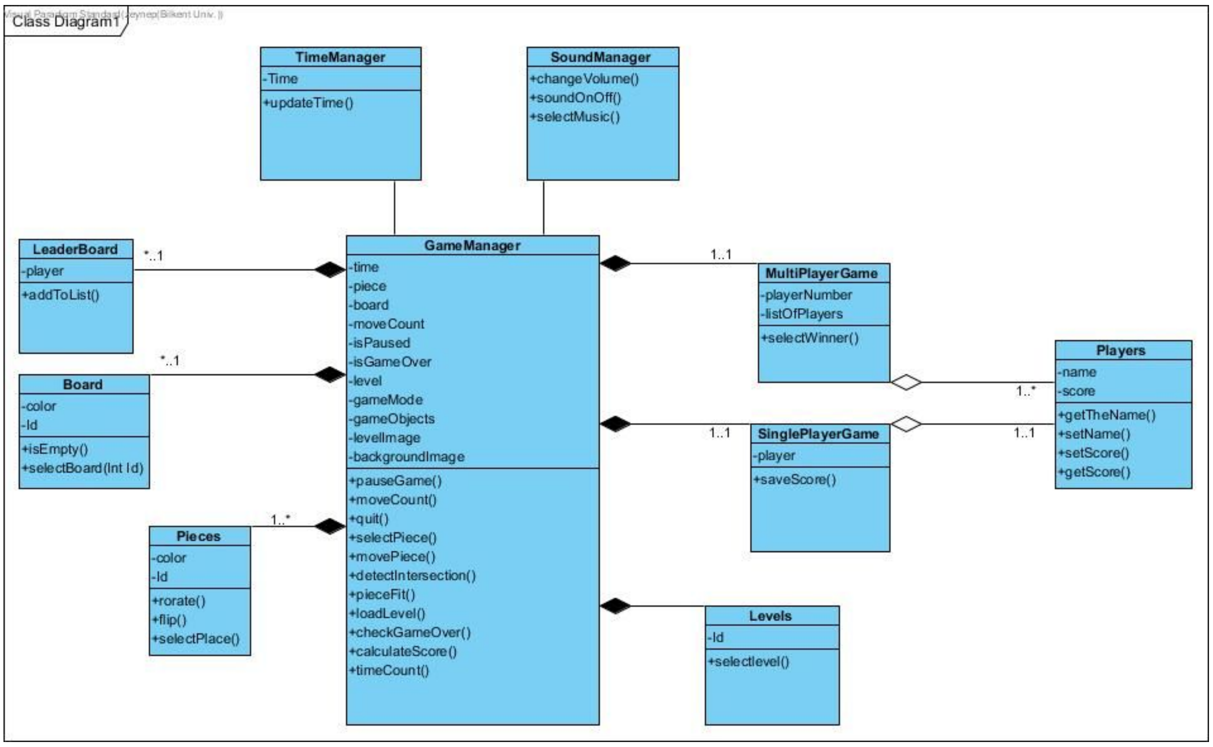
Our interface subsystem have ten classes and they are responsible for making connections between system and the user. MainMenu class is the top of the class chain of the user interface subsystem. The relationship can be easily seen from the graph. MainMenu class has relationship with almost every interface classes. StartGame is the parent class of BoardScreen, ModeSelectionScreen, LevelScreen, which are reached from main menu and GameScreen classes reached from after players select which level they want to play. SettingScreen has the sound settings. MainMenu class has also LeaderboarrdScreen class where the players can see their past scores, number of moves and how much time they spend.

In main menu screen, there is six buttons, single player, multiplayer, leaderboard, how to play, settings and quit. Each screen has their own instances and they will be started according to these attributes.

3.2 Application Layer

As it can be seen Class Diagram below our classes are in connected with GameManager. In this hierarchy we could offer 2 game playing modes as MultiPlayer and SinglePlayer and through 2 board modes as 2D and 3D selection. Also, LeaderBoard datas can be stored via its class and time and sound checks and utilities can be changed via related classes belong GameManager. On the main logic there is levels,pieces and player objects programmed on their classes for game stats, utilities inside. Player with its name and scored stored inside can play game with taking different pieces and spaces to put.

(Graph 2)



3.3 Storage Layer

Our data can be accessed on computers via storage sources we use. Data can generate into that programmes' facilities and they can be gathered on easily. Thus exploring and data storing are easier and it can be synchronously updated by program and thus leaderboard can be holded. Computer will understand via the storing data system we used..

4. Low-Level Design

4.1 Object design trade-offs

Usability and Functionality :

Our main goal is about entertainment so we don't need to increase functionality.On the other hand, usability is very important for us.Game should be easy to play because kids will play.Also,we put how to play screen to prevent any confusions.We tried to eliminate complex and complicated things so we decrease many options.

For Facade design pattern provides us stabilize , modifiable and extensibility. It helps to solve errors.

Performance vs. Memory:

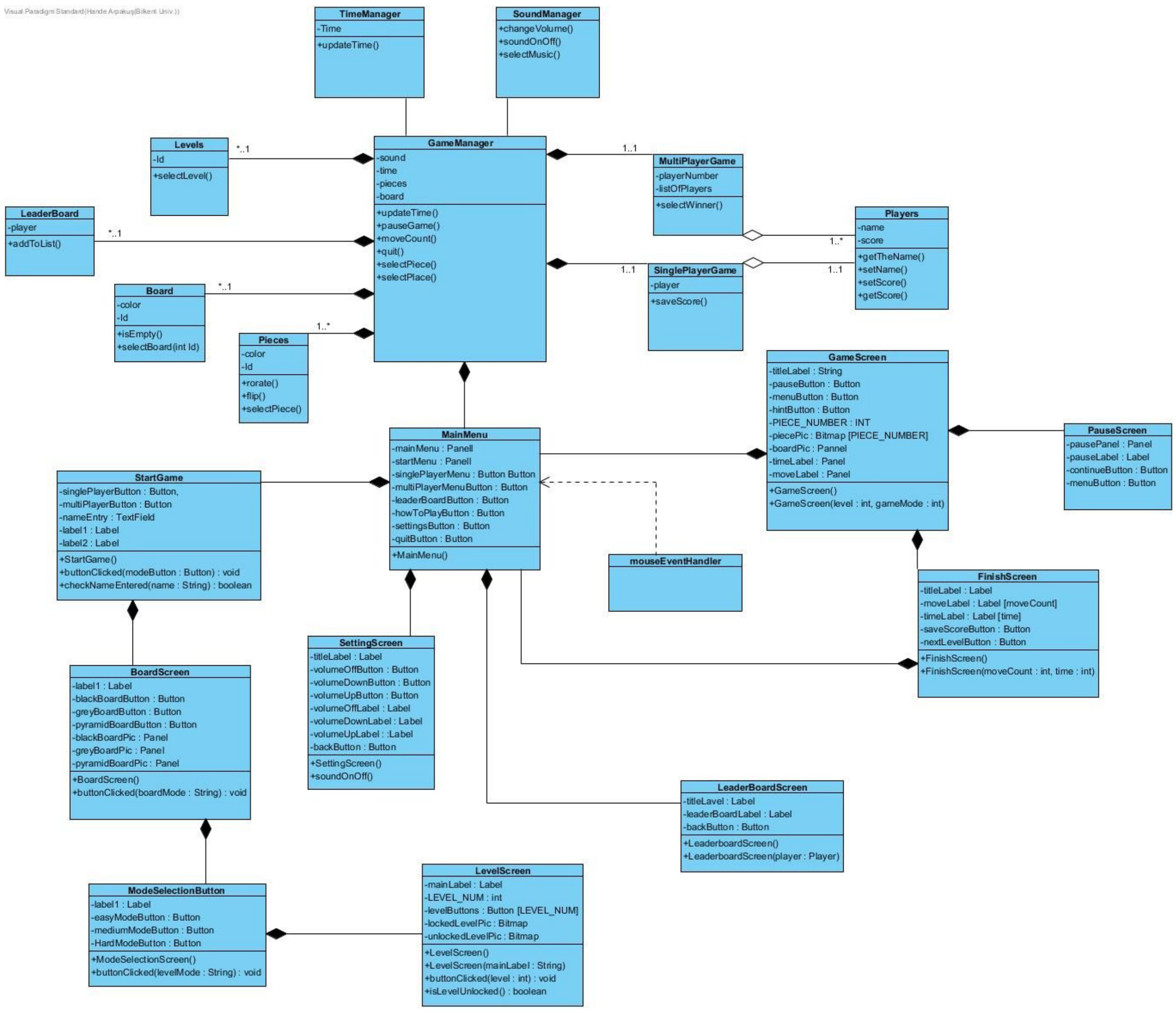
For memory, we prefer MS Access for database because it is portable.It doesn’t require any extra installations.It consists of one file.We choose JAVA programming language to use JAVAFX library for our game structures. It is created on platform with JAVA source codes on the NetBeans environment, also through its dynamic there is no unnecessary memory allocation to avoid time complexities and lower efficiency.

We don’t want using unnecessary attributes to prevent unnecessary memory allocations.Performance is important for us because it makes the game playable.For whole codes , we avoid time complexities.

At the beginning , we chose double array as data structures but it can cause some problems about efficiency because our pieces can flip and turn when we check the board and whether game is over or not.On the other hand , we realize that other data structure doesn’t convenient for our games for example,tracing whole linked list is worse than array as complexity.

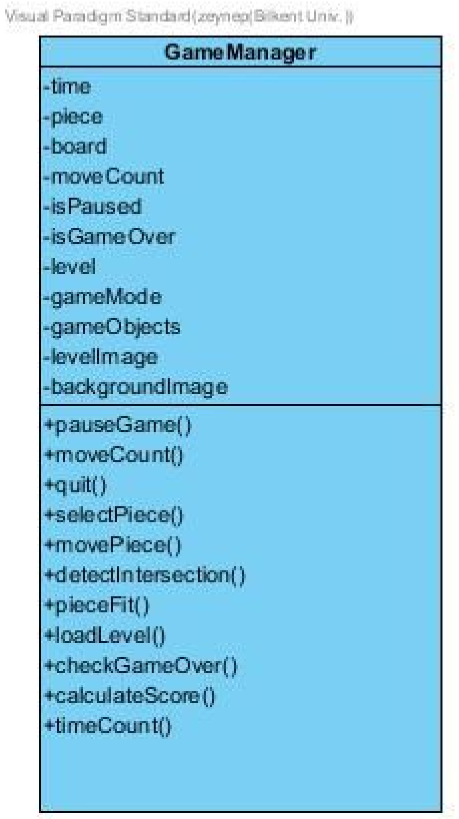
We decide to use again double arrays to build structure.Also,we are going to create some hash functions to maximize memory.In fact , we can create our own data structure for 3D board.We think to design triple array for that.

4.2 Final object design



4.2.1 Application Layer Design

Game Manager Class:



(Graph 3)

This class is the main class of the game. It takes two parameters in the constructor. By looking these parameters it determines game objects and images. It constructs the level, provide functionalities to the user such as move and place the piece, pause and quit the game. This class controls the main functionalities of the game.

Attributes:

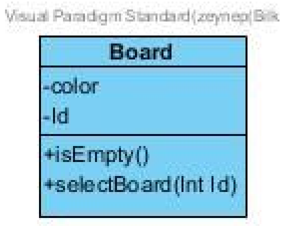
1. **time**: This integer attribute keeps tracks of the time. It is required to keep the user's time to finish each level. Its type is System.Timer.
2. **moveCount**: This integer attribute tracks of the count of the users move the pieces. It is increased each time user move and place a piece on the board.
3. **isPaused**: This attribute will keep the data whether the game is paused or not. It is required because the user can pause the game during the gameplay. Its type is boolean.
4. **isGameOver**: This attribute will keep the data whether the game is over or not. When the user completes the given level, the game is over. Its type is boolean.
5. **level**: This integer attribute specify the current level in the game. By this attribute, the game can determine which image to load.
6. **gameMode**: This string attribute specify the current mode in the game. By this attribute and with the level attribute, the program can determine which image to load.
7. **gameObjects**: This array attribute contains game objects according to the current game mode and level. This contains one board and 12 pieces. Board can change according to the level and mode, however, pieces remain same in each level and game mode.
8. **levelImage**: This attribute contains the image of the level. Pieces on the board determine according to this attribute.
9. **backgroundImage**: This attribute contains the background image of the game.
10. **piece**: This attribute is for one of the game objects. It provides piece objects and their functions.
11. **board**: This attribute is for one of the game objects. It provides one board object and its functions.

Methods:

1. **detectIntersection**(): This method is detecting an intersection of the pieces if there is any. If there is any intersection detected, game not allow user to put piece in desired place. This method returns boolean.

2. **pieceFit**(): This method is placing pieces to the board. Board and pieces both have circle elements of objects. These elements have to fit each other in order to place the piece to the board. This method provide this functionality. This method returns void.
3. **moveCount**(): This method counts the move of each pieces in the game. Then, game record each moveCount until the level end. In addition, this information is recorded to the player object. This method returns integer.
4. **loadLevel**(): This method is load the game level. It is determined by the level, gameMode and playerType attributes. This method is also determine which images are loaded for the game. This method returns void.
5. **checkGameOver**(): This method is check if the game is over or not. Game can be over if and only if the level is completed or user quit the game. This method also lead the proper screen. This method returns boolean.
6. **pauseGame**(): This method pause the game. It also stops the time. This method returns void.
7. **selectPiece**(): This method is used by the user to select a piece. After user select a piece, he/she can use pieces methods. This method returns void.
8. **movePiece**(): This method is used by the user to move the piece. User can drag the piece and place it to the board's empty places. This method returns void.
9. **quit**(): This function is for quitting the game. This method returns void.
10. **timeCount**( ): This method gets time from the TimeManager class. It returns it to the int and return it. This method returns integer.
11. **calculateScore**( ): This method calculate the score of the player. It calculated by multiplying level by time and dividing it to the moveCount. It returns int.

Board Class:



(Graph 4)

This class contains board objects of the game.

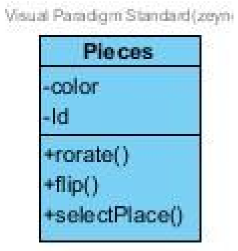
Attributes:

1. **color**: This string attribute determines the color of the board.
2. **Id**: This integer attribute is an integer representation of the board type.
3. **emptyPlaces**:
4. **fullPlaces**:

Methods:

1. **isEmpty**(): Return true if board has any empty place. Otherwise, it returns false.
2. **selectBoard** (int Id): Select board according to its Id. It returns void.

Pieces Class



(Graph 5)

This class contains piece objects of the game. There are 12 pieces on the game.

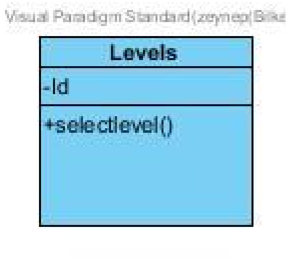
Attributes:

1. **color**: This attribute determines the color of the piece.
2. **Id**: This attribute is an integer representation of the piece type.

Methods:

1. **selectPiece** (int Id): Select piece according to its Id.
2. **rotate**(): It rotates piece right or left.
3. **flip**(): It flips the piece.

Levels Class



(Graph 6)

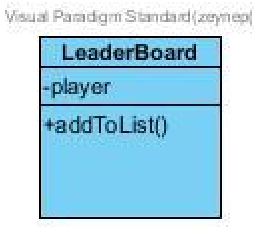
Attributes:

1. **Id**: This attribute is an integer representation of the piece type.

Methods:

1. **selectlevel** (int Id): Select level according to its Id.

Leaderboard Class



(Graph 7)

This class is to keep the scores of the players. At the end of the each level the player will be asked if he wants to save his score. If he want it, this class takes his name and score and adds to the score list's appropriate place considering other scores.

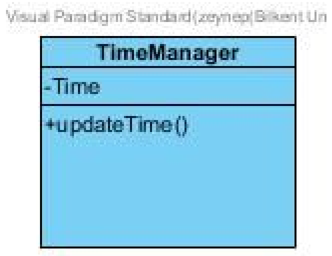
Attributes:

1. **player**: This attribute is an array which keeps the scores and the names as objects of the players in orders.

Methods:

1. **addToList**(): It adds player object to the list if user desire to record his score. It decides the place of the new player by comparing his score with the others.

Time Manager Class



(Graph 8)

This class is created to update the elapsed time until the player complete the game. Also, we provide to save this elapsed time with this class. End of every level, ıf the player wants to, his elapsed time is saved for the leaderboard.

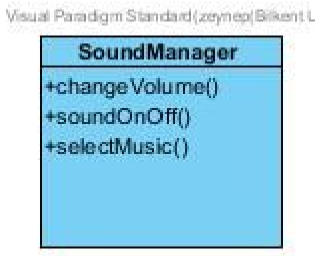
Attributes:

1. **Time**: System.Timer

Methods:

1. **updateTime**(): This method is to update the elapsed time. We use timer manager of C# language in this method. It returns this elapsed time.

Sound Manager Class



(Graph 9)

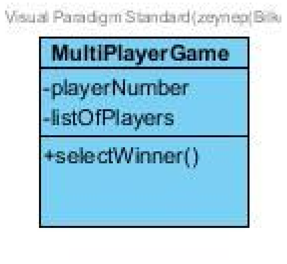
This class is written for the sound settings of the game. The player can choose the soundtrack which plays at the background of the game by this class. Also. the player can arrange the level of the sound or turn it on/off.

Methods:

1. **changeVolume**(): This method is for arranging the volume rate of the sound. The user can volume up or down. It doesn't return anything.
2. **soundOnOff**(): This method is used to turn on or off the sound of the game. It doesn't return anything.

3. **selectMusic**(): the soundtrack can be changed by the user with this method. It doesn’t return anything.

Multiplayer Game Class



(Graph 10)

In our version of IQpuzzler pro we have multiplayer selection. If the user wants to, he can play the game with his friends. To provide this property ‘multiPlayerGame’ class is written. With this class at least two, at most four people can play the same level in order. At the end scores of the players are compared and determined the winner.

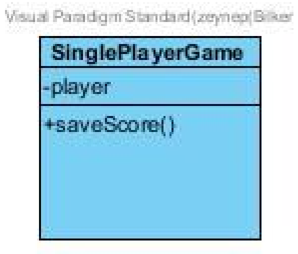
Attributes:

1. **listOfPlayers**: is an array of Players.It includes name and score of each player.
2. **playerNumber**: is an int which represents the number of people play the game.

Method:

1. **selectWinner**(): This method is to determine who the winner is. It takes the array listOfPlayer and compares the scores of the players and chooses the best score. Then determine the player with the chosen score as the winner. It returns the player who win.

SinglePlayer Game Class



(Graph 11)

This class is to keep the player's keep the properties by using Players class. Also, in this class the score of the player is saved, if the user wants to for the leaderboard.

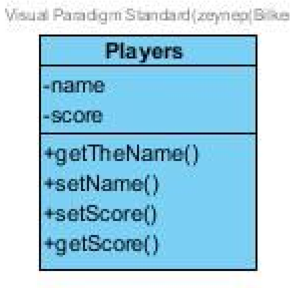
Attributes:

1. **player**: is an object of Players. It includes the name, time, move count, and the score of the player.

Methods:

1. **saveScore**(): If the player wants to, it takes the score of the player from the Players class and return it to send it to the leaderboard and add the player's score with his name to the list.

Players Class



(Graph 12)

This class is used to keep the name and score of players.

Attributes:

1. **name**: is a string for the name of the player.
2. **score**: is an integer.

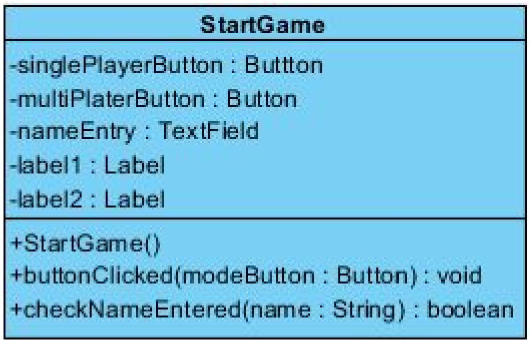
Methods:

1. **getTheName**(): This method asks the name of the player and it returns the string name.

2. **setTheName**(string name): This method is used to set the name as attribute name..
3. **getScore**(): This method is used to get score from the game manager class and returns it as integer.
4. **setScore**(int score): this method is used to set the score as attribute score.

4.2.2 UI Layer Design

Start Game Class:


StartGame
-singlePlayerButton : Button
-multiPlaterButton : Button
-nameEntry : TextField
-label1 : Label
-label2 : Label
+StartGame()
+buttonClicked(modeButton : Button) : void
+checkNameEntered(name : String) : boolean

(Graph 13)

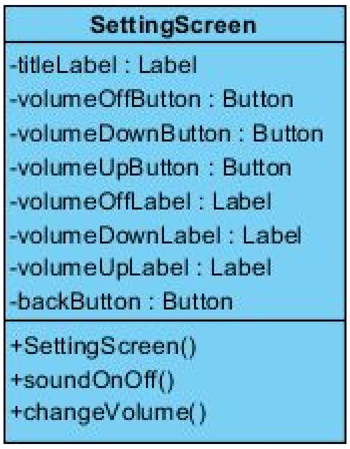
Attributes:

1. **Button singlePlayerButton** :This button helps to user to choose single player game mode.
2. **Button multiPlayerButton** : This button helps to user to choose multiplayer game mode.
3. **TextField nameEntry** :This text field takes user’s name.
4. **Label label1** :This label is for title for ” Start Game ”.
5. **Label label2** :This label is for title for ” Enter your name ”.

Methods:

1. **startGame** () : This method starts the game and open main game screen.This function returns void.
2. **buttonClicked** (Button ModeButton) :This method checks which button is clicked.Then,it loads panel according to clicked button.This function returns void.
3. **checkNameEntered** (String name) :This method checks whether user writes his name or not.This function returns boolean.

Setting Screen Class



(Graph 14)

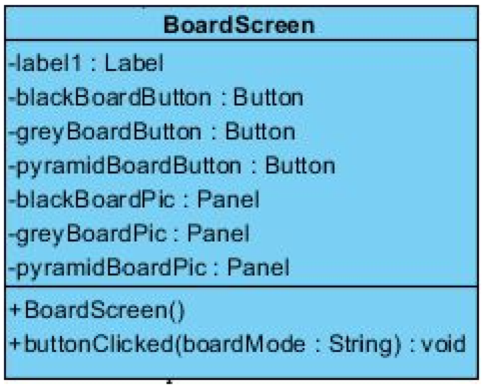
Methods:

1. **SettingsScreen**(): This method provides setting screen for game utility. This function returns void.
2. **SoundOnOff**(): When the sound button inside setting screen is tapped sounds' volume can be muted or unmuted.This method helps user to check sounds settings.This method is returned boolean, if it is on it returns true, reverse it returns false.
3. **ChangeVolume**(): When the change volume button inside setting screen is tapped sounds' volume can be changed so as to increase or decrease it. This method helps user to check sounds settings either. Method returns void.

Attributes:

1. **VolumeOffButton**: This button helps user to change volume in settings screen as on or off.
2. **VolumeDownButton**: This button helps user to decrease volume when tapping its method.
3. **VolumeUpButton**: This button helps user to increase volume when tapping its method.
4. **VolumeOffLabel/VolumeDownLabel/VolumeUpLabel**: This label shows user volume label.
5. **BackButton**: This button helps user to get back while he is checking the settings screen.

BoardScreen


BoardScreen
-label1 : Label
-blackBoardButton : Button
-greyBoardButton : Button
-pyramidBoardButton : Button
-blackBoardPic : Panel
-greyBoardPic : Panel
-pyramidBoardPic : Panel
+BoardScreen()
+buttonClicked(boardMode : String) : void

(Graph 15)

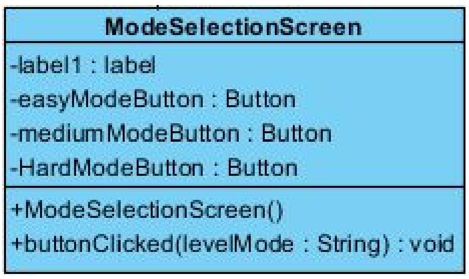
Methods:

1. **BoardScreen():** This methods returns void. It helps user to show board screen for game choices.
2. **ButtonClicked(boardMode: String):** This method returns void. It checked which board you choose, grey or black for different levels.

Attributes:

1. **BlackBoardButton:** This button is choosing black flat board, it helps user for that choose.
2. **GreyBoardButton:** This button is choosing grey flat board, it helps user for that choose.
3. **PyramidBoardButton:** This button is selection for pyramid board, it helps user for that.
4. **BlackBoardPic:** This attribute helps user to take black board pic.
5. **GreyBoardPic:** This attribute helps user to take black grey pic.
6. **PyramidBoardPic:** This attribute helps user to take pyramid board pic.

Mode Selection Screen



(Graph 16)

Constructor:

1. **ModeSelectionScreen ()** : This constructor creates the mode selection screen

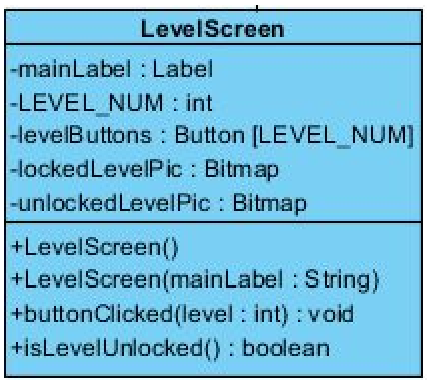
Methods :

1. **buttonClicked (String levelMode)** :This method checks which button is clicked.Then,it loads panel according to clicked button.This function returns void.

Attributes:

1. **Button easyModeButton**:This button helps to user to choose easy level game.
2. **Button hardModeButton**: This button helps to user to choose hard level game.
3. **Button mediumModeButton**:This button helps to user to choose medium level game.
4. **Label label**:This label is for title for ” Select the Mode ”.

Level Selection Screen:



```
LevelScreen
-mainLabel : Label
-LEVEL_NUM : int
-levelButtons : Button [LEVEL_NUM]
-lockedLevelPic : Bitmap
-unlockedLevelPic : Bitmap
+LevelScreen()
+LevelScreen(mainLabel : String)
+buttonClicked(level : int) : void
+isLevelUnlocked() : boolean
```

(Graph 17)

Constructor:

1. **LevelScreen():** This constructor display new level selection screen.

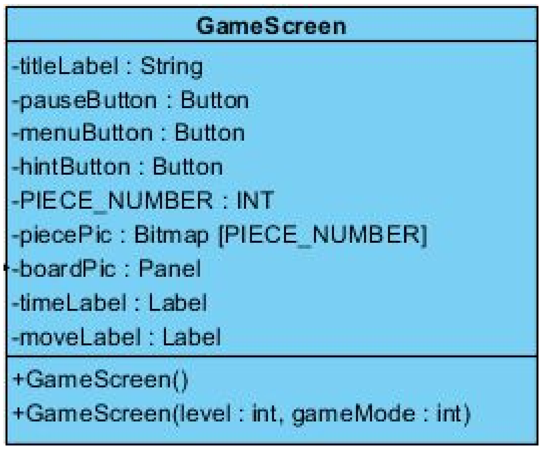
Methods:

1. **levelScreen():** This methods returns void. This method helps user to choose level for playing. There are different level stages from beginner to expert.
2. **levelScreen(mainLabel: String):** This method returns the choice as easy,medium or expert. Thereby it displays game as user's choice.
3. **buttonClicked(level: int):** This method returns level difficulty which level difficulty is selected.
4. **isLevelUnlocked():** This method returns boolean if new level is unlocked or not. Whenever levels are passed new levels are unlocked.

Attributes:

1. **LEVEL_NUM:** This label holds level num integers.
2. **LevelButtons:** This button holds level num inside that button.
3. **LockedLevelPic:** This attribute holds locked level pictures.
4. **UnlockedLevelPic:** This attribute holds unlocked level pictures.

Game Screen:


```
GameScreen
-titleLabel : String
-pauseButton : Button
-menuButton : Button
-hintButton : Button
-PIECE_NUMBER : INT
-piecePic : Bitmap [PIECE_NUMBER]
-boardPic : Panel
-timeLabel : Label
-moveLabel : Label
+GameScreen()
+GameScreen(level : int, gameMode : int)
```

(Graph 18)

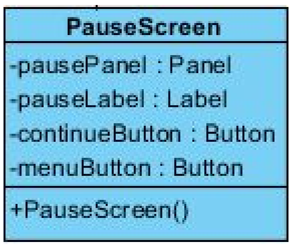
Constructor:

1. **GameScreen():** It display game screen.
2. **GameScreen(level : int, gameMode : int):** Game Screen method returns void. It display new game screen when selections are done.

Attributes:

1. **PauseButton:** This button holds pause function, if it is tapped game needs to be paused.
2. **MenuButton:** This button holds menu, if it is tapped game needs to display menu.
3. **HintButton:** This button holds hints related game levels for guiding user when he stucked somewhere in game.
4. **PIECE_NUMBER:** This attribute holds piece numbers as integers.
5. **PiecePic:** This attribute holds piece pic inside.
6. **BoardPic:** This attribute holds board pic inside.
7. **TimeLabel:** This attribute holds time labels inside for time information related leaderboard score check.
8. **MoveLabel:** This label holds move labels inside,which moves are taken.

Pause Screen class



(Graph 19)

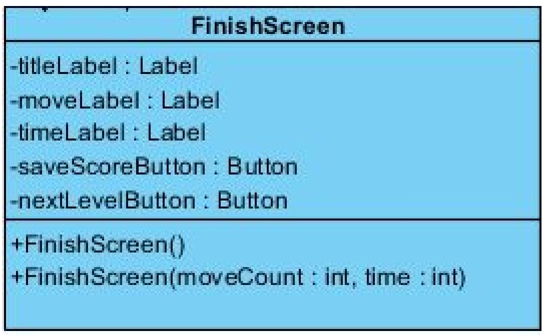
Constructor :

1. **PausePanel()**:This constructor displays pause screen.

Attributes:

1. **Button menuButton**:: This button helps to user to go menu button.
2. **Button continueButton**:This button helps to user to continue game.
3. **Label pauseLabel**:This label is for title for " PAUSE ".

Finish Screen Class


FinishScreen
-titleLabel : Label
-moveLabel : Label
-timeLabel : Label
-saveScoreButton : Button
-nextLevelButton : Button
+FinishScreen()
+FinishScreen(moveCount : int, time : int)

(Graph 20)

Constructor:

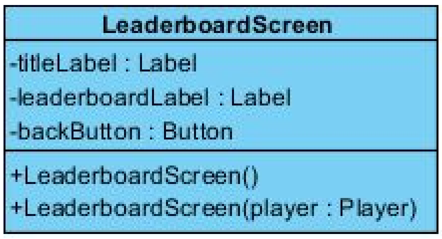
1. **FinishScreen ()** : This constructor creates finish screen at the end of the game.
2. **FinishScreen (int moveCount , int time) :** This constructor creates the screen according to the move count and time in the parameter.

Attributes:

1. **Label titleLabel**:This label is for title for "Congratulations :D ".
2. **Label moveLabel:**This label is for title to show move count.
3. **Label timeLabel** :This label is for title to show time.
4. **Button saveScoreButton**:This button helps to user to save score button with text "Do you want to save score?".

5. **Button nextLevelButton:**This button helps to user to access next level with “Next Level”.

LeaderBoardScreen:



(Graph 21)

Constructor:

1. **LeaderboardScreen ()** : This constructor creates the screen default.
2. **LeaderboardScreen (Player player) :** This constructor creates the screen according to the player in the parameter.

Attributes:

1. **Label titleLabel**:This label is for title for ” Start Game ”.
2. **Label leaderboarLabel:**This label is for title for ” Enter your name ”.
3. **Button backButton** :This button helps to user to turn back.

Mouse Event Handler

(Graph 22)

This class is to control mouse events in UI.

ROTATION WITH MOUSE WHEEL

MOUSE CLICKED FOR FLIPPING

MOUSE TAPPED FOR SELECTED HOLES

4.3 Packages

4.3.1 JAVA.util

Util package can be thought as a data structures container, so we need ArrayList to hold multiple variables inside, such as board types and members inside. Thus, we could use these lists to integrate our game into language environment.

4.3.2 java.fx.scene.layout

This package could give us User Interface layouts to integrate our Graphical User Interface objects. JAVAFX is thought as beneficial support for game’s 3D object mode among group members.

4.3.3 java.fx.scene.paint

This package could beneficial for visualization our game in JAVA programming language.

4.3.4 java.fx.application ???

It provides the application life cycle for our game project.

4.3.5 javafx.embed.swing

It provides set of classes to use JAVAFX inside the Swing applications for Graphical User Interface support for visualization.

4.3.6 java.fx.geometry

It provides the sets for 2D objects of 2D game mode. These are usable for defining and performing operations among objects related to two-dimensional geometry spaces.

4.3.7 javafx.util

It provides various utilities and helper classes by containing inside.

5 Glossary & References

[1]JAVAFX Documents

https://docs.oracle.com/javafx/2/api/overview-summary.html