CS319 Object Oriented Software Engineering Project

Design Report

Iteration 2

IQ PUZZLER PRO

GROUP - 3G

Fatih Çelik

Cenk Er

Enes Yıldırım

Eren Yalçın

Burak Bayar

Ebru Kerem

1. Introduction

1.1 Purpose of the System

IQ Puzzler Pro is a puzzle board-game. Although how the game is played is very easy to learn, solving some puzzles can be incredibly difficult. Its system offers much more than the puzzle itself to make it a better experience. Our goal is to provide a system to the users in which they can have a fun brain exercises with various puzzles, compete with other players through online maps and get creative by creating their own puzzles.

1.2 Design Goals

End-user Criteria

Usability: Playing a new solo puzzle takes 3, continuing a puzzle takes 2, playing an online puzzle takes 4, opening custom puzzle mode takes 3 mouse clicks. User can swiftly open any functionalities from main menu. The gameplay has a big role in usability as well.Dragging puzzle pieces will be done in at least 40 fps. If the gameplay is not even slightly user friendly, it would be a frustrating experience to the player especially while they are trying to solve very hard puzzles.

Dependability Criteria

Reliability: Testing is done in at least every 50 lines of coding and there will be 0 bugs . Every boundary condition is taken into account to prohibit any user from coming across errors, which can exterminate the quality of the experience very quickly.

Maintenance Criteria

Extensibility: Extensibility is very important especially for games since a game must be changing and adapting time to prevent itself from becoming obsolete. Since there are lots of objects that are independent, no big changes in code will be needed to compose new things to the system. For example, adding a new puzzle to the system requires at most 6 lines of code and creating a new type of puzzle piece requires at most 10 lines of code.

Readability: There will be more than 500 lines of comments in order to prevent any confusion. Any programmer with intermediate coding ability will understand the created code with ease. The system involves a teamwork so when a team member checks another member's code, he/she should not be left confused so that implementation will be less arduous.

Performance Criteria

Response Time: The system will respond to any input in less than a second. The user can hang around the menu swiftly without having any problems. The response time is also very important during online gameplay considering the fact that time is important to get a great high score and have a better rank. Piece selections, drops and rotations will take less than 0.3 seconds

Tradeoffs

Functionality vs Ease of Learning: The game only requires a board and 12 pieces, and it has the very simple goal of putting pieces together in proper way to complete the board. For that reason, ease of learning was not one of our concerns. However, making the software version of the puzzle created the potential of adding many more functionalities to build a system that is much more interesting to the user and this is the path we took. Unlike

the way it is played in real life, there are features like time, highscores, hints, etc. Furthermore, there is an online mode in which users can create their own maps for other people to play and see their rankings among other players for each map.

Speed vs Memory: For any game, speed is an indispensable factor. Especially for simple games like ours, laggy input response can totally diminish the entertainment the user has. Therefore, we allocated as much memory as possible for our objects to decrease response time. This way the users can have a smooth gameplay in which they can quickly move and rotate objects, also view data like highscores and new maps in a matter of milliseconds.

Development Cost vs Portability: We decided to implement our system using C# since it is a language that is very strong for building desktop applications and games. The advantage that we gain from using C# reduces the portability of our system, given that unlike Java, it does not offer the bytecode converter that gives system the ability to run on any platform.

2. High-level Software Architecture

2.1 Subsystem Decomposition

The system compositions are divided into 3 high-level layers (Figure 1). The focus of each layer will be discussed in following section.

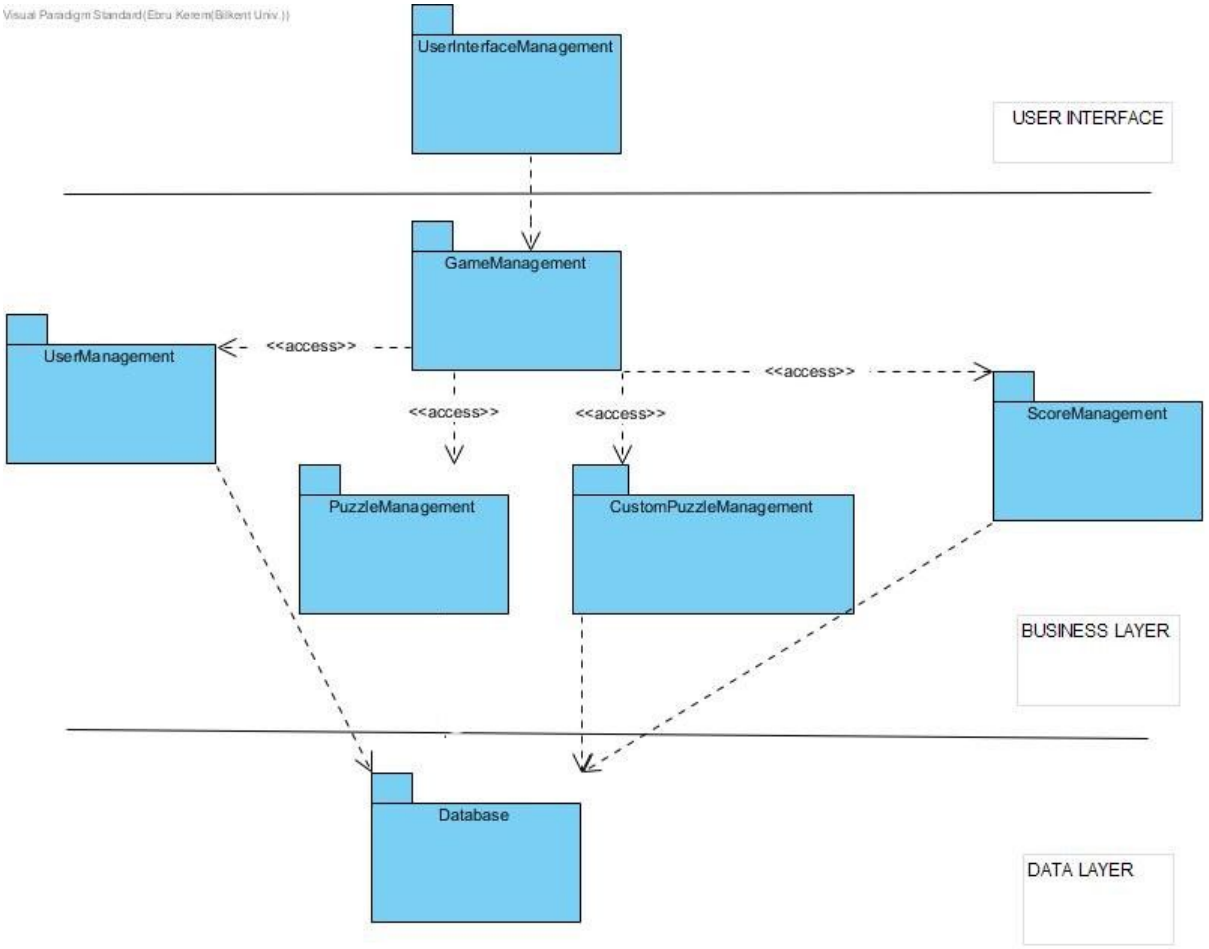

Figure 1- High-level Representation of Subsystem Decomposition

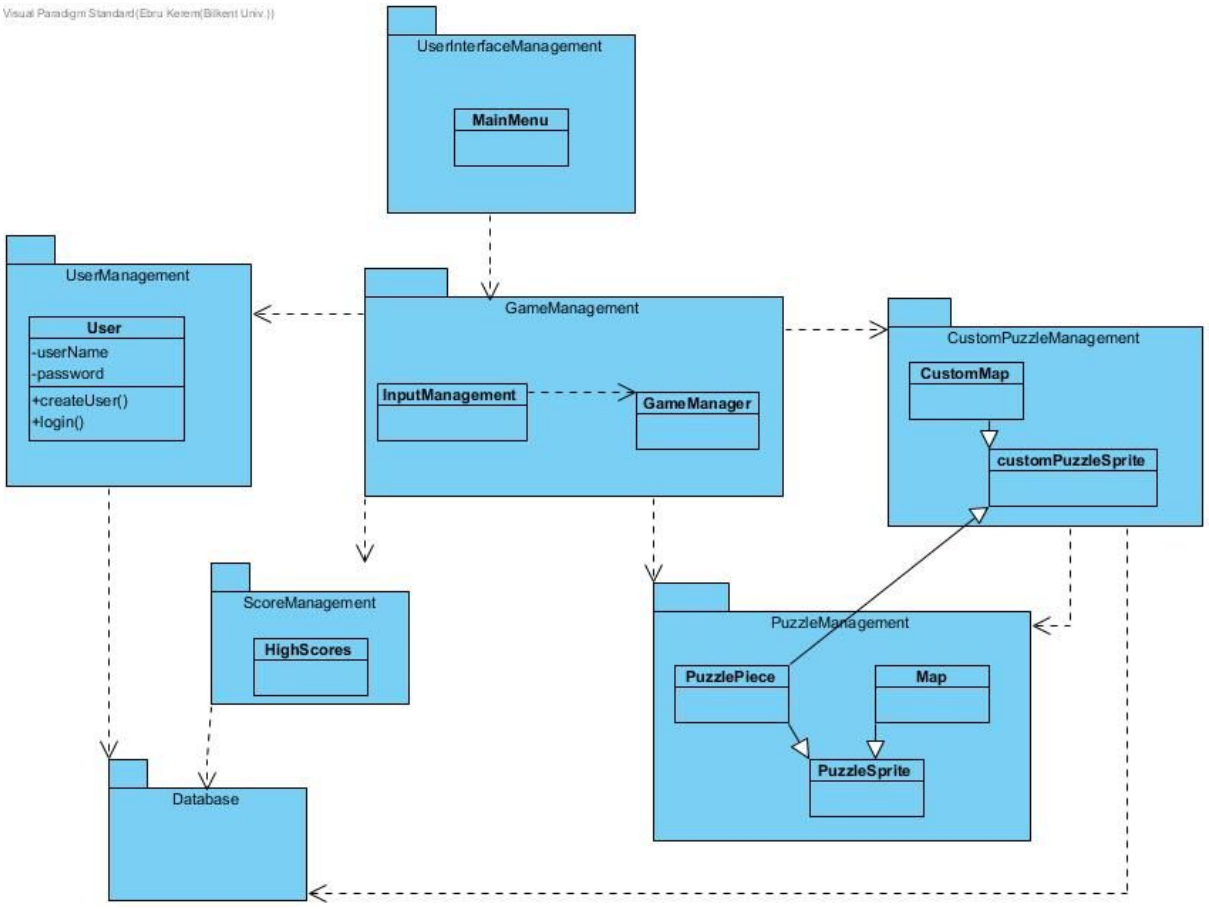

Figure 2 - Interaction between layers

2.1.1. User Interface

The user interface layer represents the front end of the game.The classes in this layer obtains the panels of the game. The goal of this layer is to allow effective operation and control of the game from the human end, whilst the game simultaneously feeds back information that aids the operators' game process.

2.1.2. Business Layer

The business layer handles the classes which contains both views and controllers. Business Logic Layer works as a bridge between user interface and data layer. All the user values received from the user interface layer are being passed to business layer. Business Logic Layer is the most important class in the whole architecture because it mainly contains all the business logic of the program. Whenever a user wants to update the business logic of the program only need to update this class through user interface.

2.1.3. Data Layer

Database Access Layer builds the data based on received parameters from the Business Logic Layer. And simple return results from the firebase to Business Logic Layer.

2.2 Hardware/Software Mapping

The architecture of our game is rather simple that it only uses our subsystems to operate. In terms of layers, the system has three main layers as the following; UI, Game Management and Game Entities. The names of these layers are self-explanatory enough but further explanations about these layers will be present in the following sections.

User Interface

This is the surface layer of our software. Here we will include all the visual and interactable components of our software. This layer will be constantly getting and sending information from other layers in order to keep the player updated regularly and take the players inputs to the places where they need to be processed.

Game Management

This is the layer where almost all the background work will be done from the puzzles construction to analysing the players moves and determining a resolution for the moves. Also the online features will be handles in this layer because the timing and scores will be calculated in this layer as well.

Game Entities

This layer is the layer where all the static information is stored. This ranges from map information to puzzle pieces and their shapes along with the scores and hints. Also the visuals and animations will be stored here. We will try to minimize most information by storing them as compressed forms and decompressing them later in game management layer.

2.3 Persistent Data Management

Since our game needs to keep data of the user in the online option, we decide to use Firebase database. Our game holds not only the users but also the maps which belong different levels, highscores, hints and custom maps. The levels and hints will be predefined in the database. Highscores, users' accounts and custom maps will be added when the users play the game and create new maps.

2.4 Access Control and Security

In terms of access control and security, our game has a login screen for online services yet there is no authentication for single player services because we think that the inconvenience and impracticality a login screen brings is greater than the security and self actualization benefits of it in terms of single player. There is nothing critical or private about it, however, we wanted it to be as practical as possible.

As for the online services, the login system is required since people may spend a relatively long time creating their custom puzzles and achieving high scores which they may want to secure and carry everywhere with their account. This way, whichever system a user logs in, they can see their highscores or custom maps.
On the other hand, the developers will have a special access without any restrictions in order to tweak minor things on the go or simply test something on the final product.

2.5 Boundary Conditions

2.5.1 Initialization

The game executes as the player clicks the .exe file. The necessary and fast access data will be gathered and stored accordingly in order to increase the efficiency of the gameplay. The file itself can be obtained online, wirelessly or any sort of physical usb device.

2.5.2 Termination

The player can terminate all the process by exiting the game from almost any window. The information that needs to be stored is stored as the information is created so any kind of loss is highly inconsiderable. When the player exits the software, nothing is left in cache and every data that the game uses to play efficiently will be terminated along with the window.

As for the subsystem termination, the subsystems are not allowed to terminate individually since the software is not that complex and terminating unused subsystems will not increase the performance considerably along with possibly increasing the risk of having errors regarding the termination or re-initialization of subsystems.

2.5.3 Failure

Since the subsystems are designed as closed boxed with limited and controlled inputs and outputs, any problem that occured on the system must also be under a subsystem which may fail depending on the error yet, since it is one of the design goals of subsystem design, the other subsystems are expected to keep working as long as they can without the previously failed subsystem. For example, if the highscore subsystem fails, the game will still start normally except for the section where the user sees the high score. On that part, it is expected to see an error message such as “Highscores are not available atm.”

If the error gets into the way of interrupting or avoiding the player to execute the base activities within the software, the system will terminate with a corresponding error message.

As for recovering subsystems, we rely on MonoGame frameworks recovery system. Other than that we are not planning to create any other recovery systems since it is way too complex to implement compared to expecting the player to restart the game when a fatal error occurs.

3. Subsystem Services

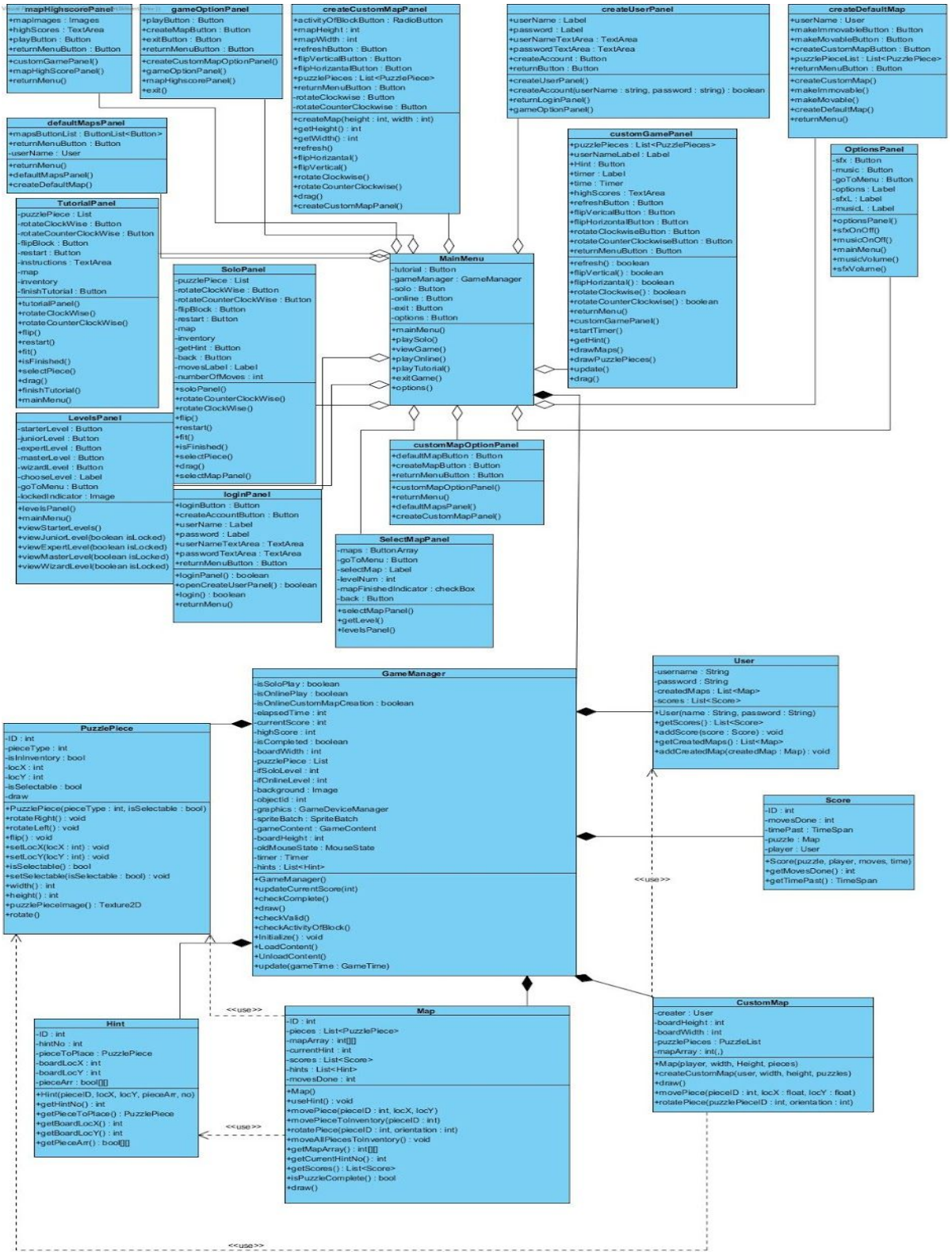

Figure 3 - Detailed Class Diagram

3.1. Packages

The game system has 6 packages. The packages are user interface management, game management, game management, user management, puzzle management, custom puzzle management, score management. The packages will be explained in details in the next section.

3.1.1. User Interface Management Package

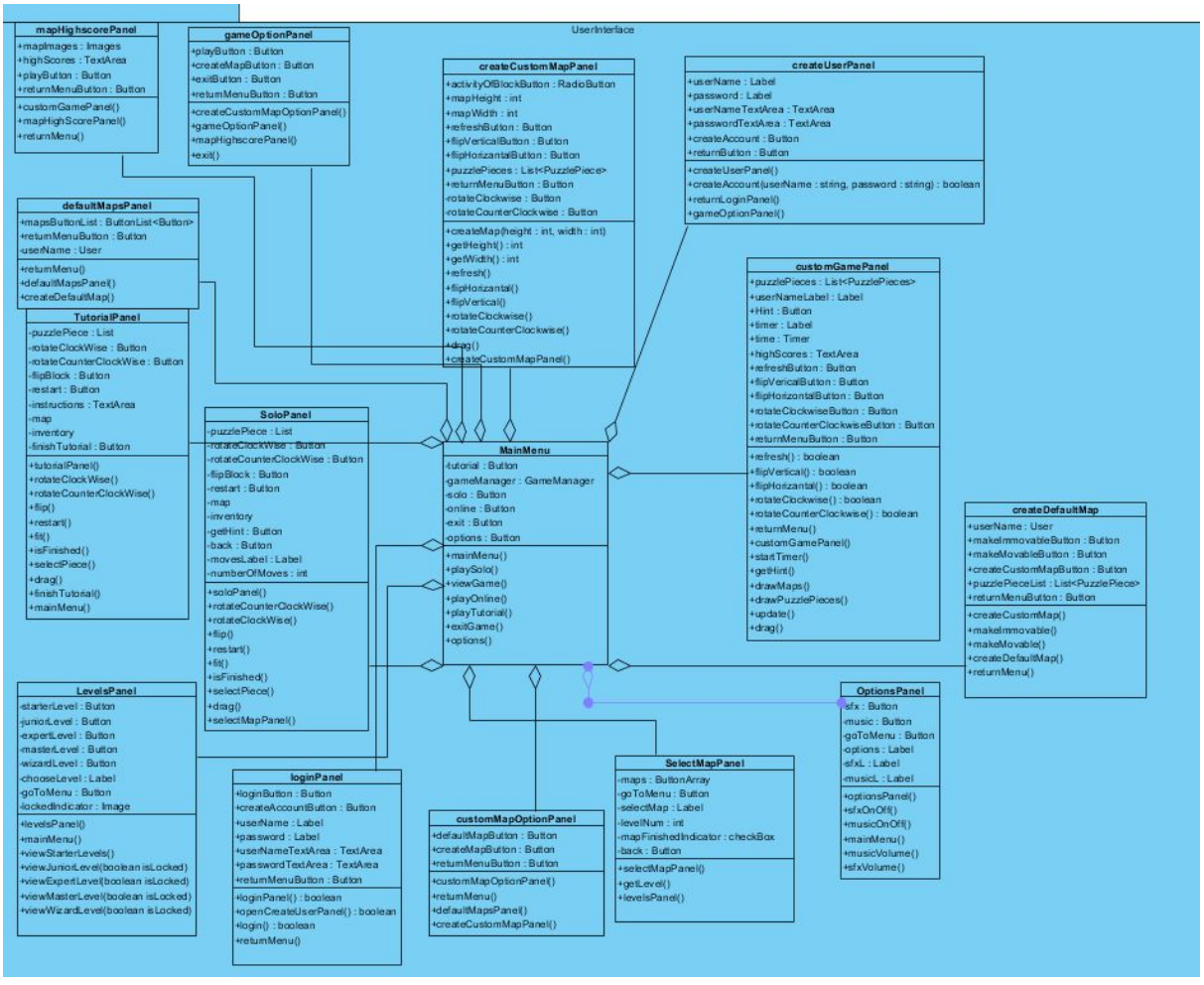

Figure 4 : User Interface Management Package

MainMenu Class

MainMenu class is the first panel which will shown when the player open the game. It consists of the options such as online or solo game, tutorial, options and exit.

Figure 5: MainMenu Class

Attributes:

Button tutorial: This attribute is a button which is used to open tutorial page.

GameManager gameManager: This attribute takes the variable of gameManager. Thus, the system creates game objects and initialize the game.

Button solo: This attribute is a button. It directs the user to the solo option's panels.

Button online: This attribute is a button. It directs the user to the online option's panels.

Button exit: When user wants to quit the game, this button helps him/her.

Button options: This attribute is a button. It directs the user to options panel.

Constructors:

Void MainMenu: This constructor initializes the main menu panel of the game.

Methods:

Boolean playSolo: This method open the solo game panels when the button is pressed.

Boolean playOnline: This method open the online game panels when the button is pressed.

Boolean playTutorial: When the tutorial button is pressed, the system leads the user to the game tutorial. This method opens tutorial panel.

Void exitGame: This method helps the user to exit the game.

Void Options: This method opens the option panel so that user can change the sound effects and volume of these effects.

TutorialPanel Class

This class shows the user the tutorial panel. The user canlearn the playing the game with help of the this class.

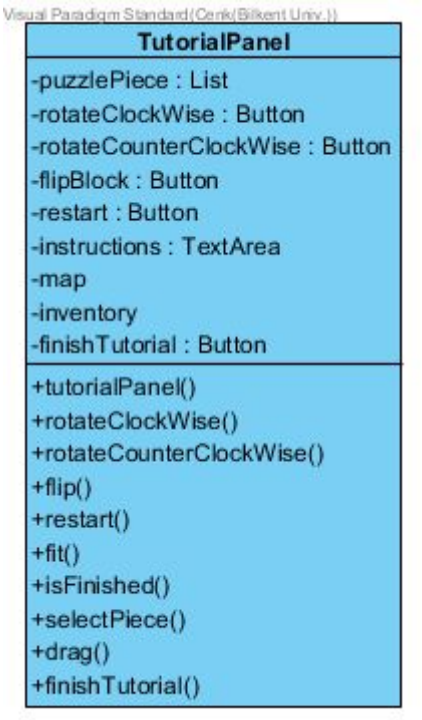

Figure 6: Tutorial Panel Class

Attributes:

List puzzlePiece: This attribute is a button which enables to restart the game and clear user's previous actions.

Button rotateClockWise: This attribute is a button which enables to rotate the selected puzzle piece in the direction of clockwise.

Button rotateCounterClockWise: This attribute is a button which enables to rotate the selected puzzle piece in the direction of counterclockwise.

Button flipBlock: This attribute is also a button which enables to flip the selected puzzle piece.

Button restart: Starts the tutorial from the start again.

TextArea instructions: Helps player figuring out how to play

Button finishTutorial: Ends the tutorial mode and closes the panel.

Constructors:

tutorialPanel() : Initializes the tutorial panel.

SoloPanel Class

This class shows solo game panel to the user. All the properties of the game are in this panel.

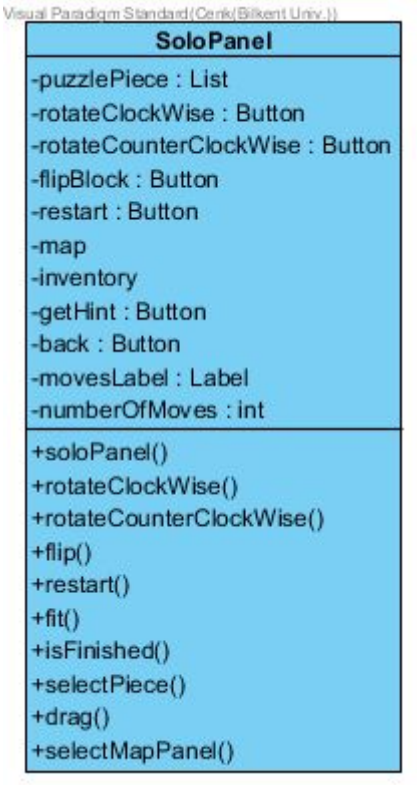

Figure 7:Solo Game Class

Label movesLabel: In this label it writes “Number of moves:” to indicate the counter for the moves the user make.

int numberOfMoves: This integer is the counter for the number of moves made by the user.

Button restart: This attribute is a button which enables to restart the game and clear user's previous actions.

Button flipBlock: This attribute is also a button which enables to flip the selected puzzle piece.

Button rotateClockWise: This attribute is a button which enables to rotate the selected puzzle piece in the direction of clockwise.

Button rotateCounterClockWise: This attribute is a button which enables to rotate the selected puzzle piece in the direction of counterclockwise.

Button back: This attribute is a button which enables to return the previous page.

List puzzlePieces: This attribute holds the puzzle objects as an array. Thus, the system can use them.

Button Hint: This attribute helps user to get hint by pressing it.

TextArea highScores: This attribute shows the highscores of the current map.

Constructors:

void SoloPanel: Initializes the SoloPanel: puzzlePieces, map, buttons.

void selecetMapPanel: Initializes the selectMapPanel.

Methods:

void returnMenu: When the user wants to return previous page, this method calls the previous panel.

boolean restart: This method restart the whole game.

boolean flipVertical: This method makes the selected puzzle piece flipped vertically.

boolean flip: This method makes the selected puzzle piece flipped horizontally.

boolean rotateClockwise: This method rotates the selected puzzle piece in the direction of clockwise.

boolean rotateCounterClockwise: This method rotates the selected puzzle piece in the direction of counterclockwise.

boolean fit: This method checks if piece that is chosen by the user and dragged to the map can fit in the desired place.

Hint getHint: This method gets the hint and shows to the user.

void drawPuzzlePieces: This method initializes the puzzle pieces and draw it in the proper locations on the screen.

void drag: This method is used for changing location of specified puzzle piece.

SelectMapPanel Class

This class shows the maps to the user. Thus, the user can choose which map he/she wants to play.

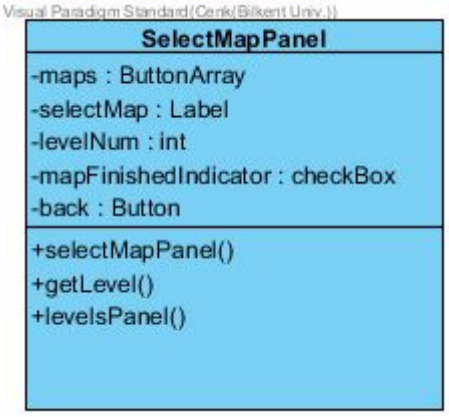

Figure 8: Map Selecting Panel Class

Attributes:

ButtonArray maps: An array of buttons will hold the maps each button will open a map.

Label selectMap: In this label it will write “Select Map”.

checkBox mapFinishedIndicator: If the user have finished a map in that level pack there, the check box will have a “check” on it.

Button back: This button will return the user to the levels panel.

Constructors:

void selectMapPanel: This constructor initializes the select map panel.

void levelsPanel: This constructor initializes the levels panel when the back button is pressed..

Methods:

int getLevel: This method will return an integer that represents the map when pressed to a map button.

OptionPanel Class

This class shows the options to the user. User can change the volumes of the sound of the game or user can close the sounds if he/she wants.

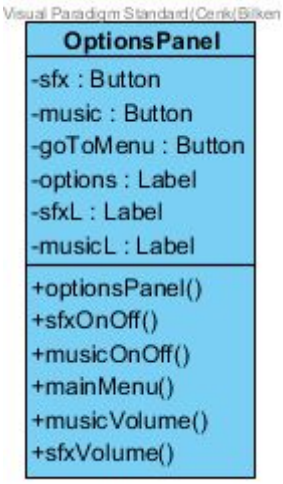

Figure 9: Option Panel Class

Attributes:

Button sfx: This button is for user to turn on and of the SFX of the game.

Button music: This button is for user to turn on and of the music of the game.

Button goToMenu: If the user wants to return to main menu panel from the options panel they can press this button to go back.

Label options: To show that user is on the option panel of the game.

Label sfxL: It is for the implication of which button is for SFX.

Label musicL: It is for the implication of which button is for music.

Constructors:

Void optionsPanel: This constructor initializes the options panel.

Methods:

LevelsPanel Class

This class shows the levels to the player. Thus, the player can choose which level he/she wants to play.

Figure 10 : Levels Class

Attributes:

Button starterLevel: This is for user to pick the starter level of the game and open it.

Button juniorLevel: This is for user to pick the junior level of the game and open it.

Button expertLevel: This is for user to pick the expert level of the game and open it.

Button masterLevel: This is for user to pick the master level of the game and open it.

Button wizardLevel: This is for user to pick the wizard level of the game and open it.

Button goToMenu: This attribute will be used to return the menu when the user press the button.

image lockedIndicator: this is image will indicate if the level pack is locked or not.

Constructors:

void levelsPanel: This constructor will initialize the levels panel of the game.

void mainMenu: This constructor will initialize the main menu panel of the game if the user presses the go back to menu button.

Methods:

boolean viewStarterLevels: If the user presses the starter level button it will open the maps inside the starter level pack.

boolean viewJuniorLevel: If the user presses the junior level button it will open the maps inside the junior level pack. Only if the pack not locked.

boolean viewExpertLevel: If the user presses the expert level button it will open the maps inside the expert level pack. Only if the pack not locked.

boolean viewMasterLevel: If the user presses the master level button it will open the maps inside the master level pack. Only if the pack not locked.

boolean viewWizardLevel: If the user presses the wizard level button it will open the maps inside the wizard level pack. Only if the pack not locked.

3.1.2 Online Game User Interface

loginPanel Class

When the user select online game option for the first time, the game system will want account to continue. This class helps users to open their account. If they do not have an account, the game system will give them a chance to open a new one by pressing createUser button. This button will lead them to createUserPanel.

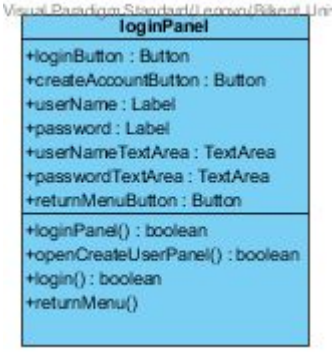

Figure 11: Login Panel

Attributes:

Button loginButton : This attribute will be used to open the game with the account when the user enter password and the username correctly.

Button createAccountButton: If the users wants to create new account, this button will lead them to createAccountPanel. This attribute holds that button.

Label userName: This attribute is just a label which shows the user where he/she can write his/her username.

Label password: This attribute is also just a label which shows the user where he/she can write his/her password.

TextArea userNameTextArea: This attribute enables the users to write their username in given text area.

TextArea passwordTextArea: This attribute enables the users to write their password in given text area.

Button returnMenuButton: This attribute will be used to return the menu when the user press the button.

Constructors:

Boolean loginPanel: Initializes the login panel : username, password sections and buttons.

Methods:

Boolean openCreateUserPanel: This method is used for open the createUserPanel.

Boolean login: This method controls the user's password and username and decide whether the user can enter his/her account or not.

Void returnMenu: When the user wants to return menu, this method calls the returnMenuPanel.

createUserPanel Class

This class helps player to create new account in the game.

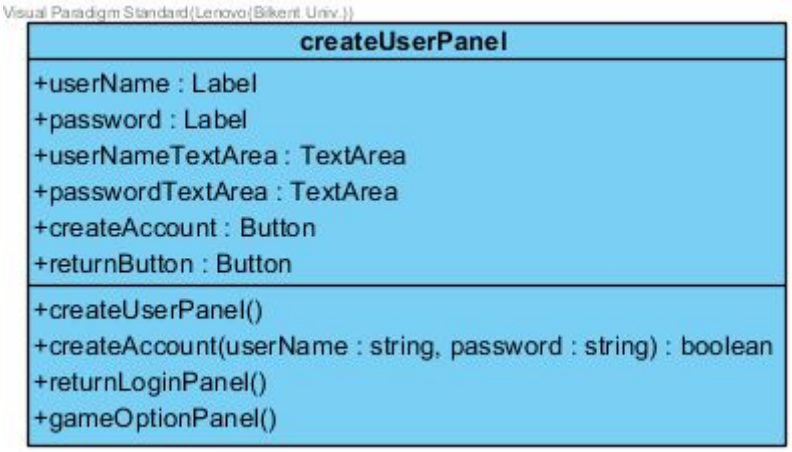

Figure 12 : Create User Panel

Attributes:

Button returnButton: This attribute is a button which enables to return the previous page.

Button createAccount: This attribute is also a button which enables to create account when the user gives valid username and password.

Label userName: This attribute is just a label which shows the user where he/she can write his/her username.

Label password: This attribute is also just a label which shows the user where he/she can write his/her password.

TextArea userNameTextArea: This attribute enables the users to write their username in given text area.

TextArea passwordTextArea: This attribute enables the users to write their password in given text area.

Constructors:

void createUserPanel: Initializes the createUser panel : username and password sections and buttons.

Methods:

boolean createAccount: This method is used to create a new account in the system..

void returnLoginPanel: When the user wants to return previous page, this method calls the loginPanel

void gameOptionPanel: When the user create an account, this method directs him/her to the gameOptionPanel.

customMapOptionPanel Class

Creating custom maps has more than one way. This class shows the options to the user.

Therefore, he/she can choose which one he/she wants to create.

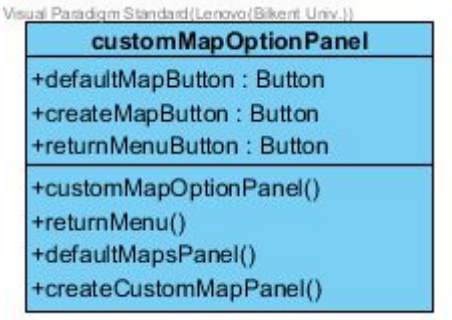

Figure 13 : Custom Map Option Panel

Attributes:

Button defaultMapButton: This attribute is a button which enables to play the game. If the user press this button, the system directs the user to the game.

Button createMapButton: This attribute is also a button which enables to enter createMapPanel when the user press the button.

Button returnMenuButton: This attribute is a button which enables to return the previous page.

Constructors:

void customMapOptionPanel: Initializes the gameOptionPanel : buttons and panels in it.

Methods:

void returnMenu: When the user wants to return previous page, this method calls the previous panel.

void defaultMapsPanel: This method is used to lead the user to the panel which shows the default maps.

void createCustomMapPanel: This method leads the user the panel which has property creating a custom map from the scratch.

customGamePanel Class

The custom game main screen will be showed in this panel. User can play the selected

game with the help of this class.

Figure 14 : Custom Game Panel

Attributes:

Button refreshButton: This attribute is a button which enables to restart the game and clear user's previous actions.

Button flipVerticalButton: This attribute is also a button which enables to flip the selected puzzle piece vertically.

Button flipHorizontalButton: This attribute is also a button which enables to flip the selected puzzle piece horizontally.

Button rotateClockwiseButton: This attribute is a button which enables to rotate the selected puzzle piece in the direction of clockwise.

Button rotateCounterClockwiseButton: This attribute is a button which enables to rotate the selected puzzle piece in the direction of counterclockwise.

Button returnMenuButton: This attribute is a button which enables to return the previous page.

List<PuzzlePieces> puzzlePieces: This attribute holds the puzzle objects as an array. Thus, the system can use them.

Label userNameLabel: This attribute is a label which shows the username above the screen.

Button Hint: This attribute helps user to get hint by pressing it.

Label timer: This attribute works with Timer object and shows the time when the game continues.

Timer time: This attribute is a timer which calculates the time user spend to solve the puzzle.

TextArea highScores: This attribute shows the highscores of the current map.

Constructors:

void customGamePanel: Initializes the customGamePanel: puzzlePieces, map, time, highscores, buttons.

Methods:

void returnMenu: When the user wants to return previous page, this method calls the previous panel.

boolean refresh: This method restart the whole game.

boolean flipVertical: This method makes the selected puzzle piece flipped vertically.

boolean flipHorizontal: This method makes the selected puzzle piece flipped horizontally.

boolean rotateClockwise: This method rotates the selected puzzle piece in the direction of clockwise.

boolean rotateCounterClockwise: This method rotates the selected puzzle piece in the direction of counterclockwise.

void startTimer: This method starts the timer from 0.

Hint getHint: This method gets the hint and shows to the user.

void drawMaps: This method initializes the map and draw it on the screen.

void drawPuzzlePieces: This method initializes the puzzle pieces and draw it in the proper locations on the screen.

void update: This method update the map when the user makes a move or change the locations of the puzzle pieces. Thus, the system can track the process of the game.

void drag: This method is used for changing location of specified puzzle piece.

mapHighscorePanel Class

This class shows the highscores of the maps. Thus, user can select among of them.

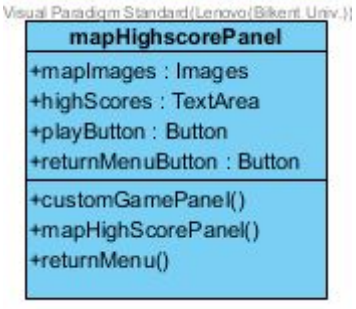

Figure 15 : Highscore Panel

Attributes:

Images mapImages: This attribute holds the images of the maps in the system.

TextArea highScores: This attribute shows the highscores of the each map.

Button returnMenuButton: This attribute is a button which enables to return the previous page.

Button playButton: This attribute is a button which enables to select the map which user wants to play and direct the user play panel.

Constructors:

void mapHighScorePanel: Initializes the mapHighScorePanel: maps, buttons, high score tables.

Methods:

void returnMenu: When the user wants to return previous page, this method calls the previous panel.

void customGamePanel: This method is used to lead the user to the panel which the game will be played.

defaultMapsPanel Class

The user can choose one of the default maps and create new map according to this map.

This panel makes possible this option for user.

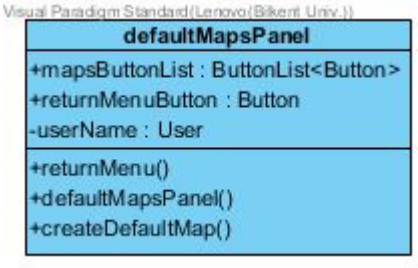

Figure 16: Default Map Panel

Attributes:

ButtonList<Button> mapButtonList: This attribute is a button list which holds the maps in it. When the user selects one of them, the selected default map will be opened.

User userName: This attribute holds the user's account in the current panel.

Button returnMenuButton: This attribute is a button which enables to return the previous page.

Constructors:

void defaultMapsPanel: Initializes the defaultMaosPanel: maps, buttons.

Methods:

void returnMenu: When the user wants to return previous page, this method calls the previous panel.

void createDefaultMap: This method directs the user to the panel which the user can design a default map.

createDefaultMap Class

After selecting the default map which user wants to create, this panel helps the user to choose which puzzle piece will be in the puzzle and which are in inventory.

Figure 17: Create Default Map Class

Attributes:

Button makeImmovableButton: This attribute is a button which enables to make the selected puzzle piece immovable.

User userName: This attribute holds the user account for the current panel.

Button returnMenuButton: This attribute is a button which enables to return the previous page.

Button makeMovableButton: This attribute is a button which enables to make the selected puzzle piece movable if the puzzle piece is immovable.

Button createCustomMapButton: This attribute is a button which enables to create custom map and add into the database.

List<PuzzlePiece> puzzlePieceList: This attribute consists of the puzzle pieces which are entity of the map.

Constructors:

void createDefaultMap: Initializes the createDefaultMap: map, puzzle pieces, instructions, buttons and panels.

Methods:

void returnMenu: When the user wants to return previous page, this method calls the previous panel.

void makeMovable: This method is used to make the selected puzzle piece movable.

void makeImmovable: This method is used to make the selected puzzle piece immovable.

void createCustomMap: This method is used to create map and add into the database.

createCustomMapPanel Class

This class enables to create a map from the scratch. User can choose every little detail of the map and create them.

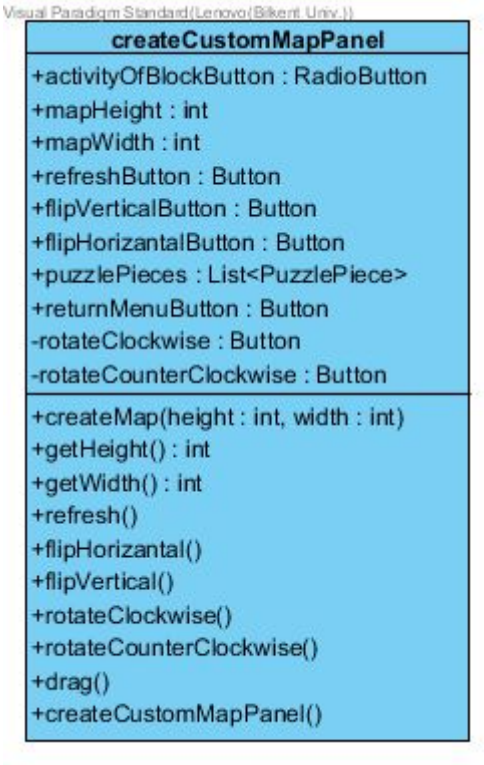

Figure 18: Create Custom Map Class

Attributes:

RadioButton activityOfBlockButton: This attribute is a button which enables to control the status of the puzzle pieces - whether the selected puzzle piece movable or not-

int mapHeight: This attribute holds the desired height of the map.

Button refreshButton: This attribute is a button which enables to restart the game and clear user's previous actions.

Button flipVerticalButton: This attribute is also a button which enables to flip the selected puzzle piece vertically.

Button flipHorizontalButton: This attribute is also a button which enables to flip the selected puzzle piece horizontally.

Button rotateClockwiseButton: This attribute is a button which enables to rotate the selected puzzle piece in the direction of clockwise.

Button rotateCounterClockwiseButton: This attribute is a button which enables to rotate the selected puzzle piece in the direction of counterclockwise.

Button returnMenuButton: This attribute is a button which enables to return the previous page.

List<PuzzlePieces> puzzlePieces: This attribute holds the puzzle objects as an array. Thus, the system can use them.

int mapWidth: This attribute holds the desired width of the map.

Constructors:

void createCustomMapPanel: Initializes the createCustomMapPanel: map, puzzle pieces, activity of the puzzle pieces, buttons and panels.

Methods:

boolean refresh: This method restart the whole game.

boolean flipVertical: This method makes the selected puzzle piece flipped vertically.

boolean flipHorizontal: This method makes the selected puzzle piece flipped horizontally.

boolean rotateClockwise: This method rotates the selected puzzle piece in the direction of clockwise.

boolean rotateCounterClockwise: This method rotates the selected puzzle piece in the direction of counterclockwise.

void returnMenu: When the user wants to return previous page, this method calls the previous panel.

Map createMap(int height, int width): This method is used to create a map with desired scale.

int getHeight: This method is used to get desired map height from the user.

int getWidth: This method is used to get desired map width from the user.

void drag: This method is used for changing location of specified puzzle piece.

3.1.2. Game Management Package

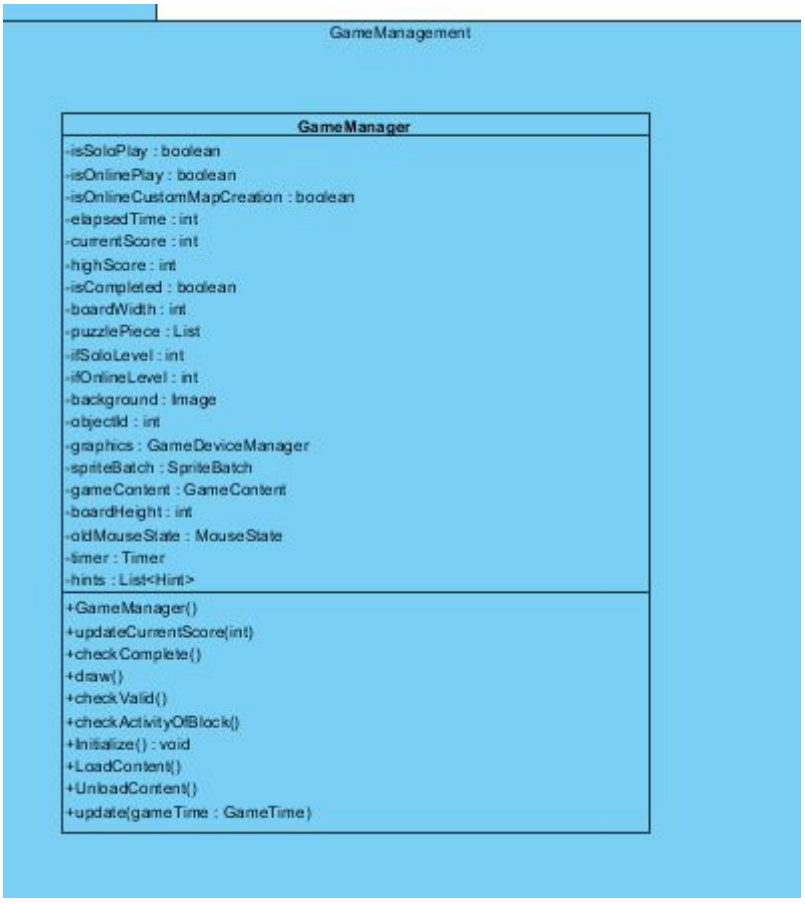

Figure 19: Game Management Package

Game Manager Class

This class controls the objects' moves, rotations. Every control mechanisms first pass this class.

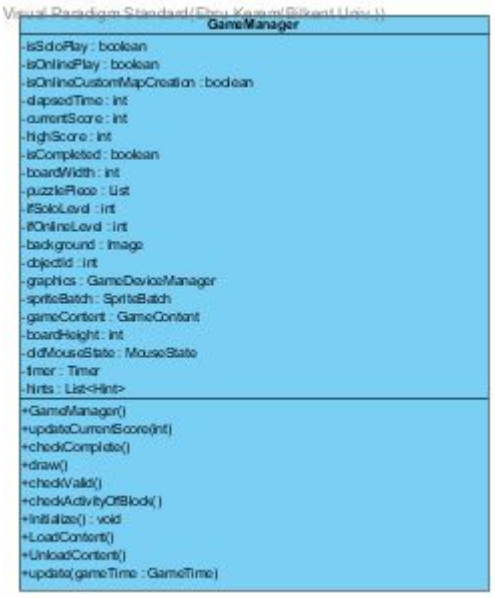

Figure 20: Game Manager

Attributes:

boolean isSoloPlay: This attribute is controls whether the game is solo or not.

boolean isOnlinePlay: This attribute is controls whether the game is online or not.

int elapsedTime: This attribute holds elapsed time for custom game.

int currentScore: This attribute shows current high score of the player.

int highScore: This attribute shows high score of the game.

boolean isCompleted: This attribute determines if the game finished or not.

int boardWidth: Board width will be hold in this attribute.

int boardHeight: Board height will be hold in this attribute.

List<PuzzlePieces> puzzlePieces: This attribute holds the puzzle objects as an array. Thus, the system can use them.

Image background: Holds the background image of the game.

Graphics gameDeviceManager: This attribute controls the process of the game in general.

SpriteBatch spriteBatch: This attribute controls the process of drawing the objects in general.

GameContent gameContent: This attribute holds the items which will be used in the game.

List<PuzzlePieces> hints: This attribute holds the puzzle objects as an array. Thus, the system can use them when it tries to give hint.

Constructors:

GameManager gameManager: Initializes the game manager class: map, puzzle pieces, activity of the puzzle pieces.

Methods:

int updateCurrentScore(): This method updates the score of the game.

boolean checkComplete(): This method checks if the puzzle is complete or not.

draw(): This method helps to draw all of the objects in the puzzle.

LoadContent(): This method loads the object which will be used.

UnloadContent(): If any object is unnecessary, this method helps to unload them.

update(GameTime gameTime): This method constantly updates the game.

3.1.3. Game Management Package

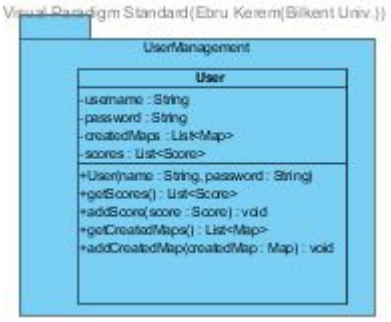

Figure 21: User Management Package

User Class

This class controls the users' activity and properties such as username, password, created maps and scores.

Figure 22: User Class

Attributes:

String username: This attribute holds the name of the user.

String password: This attribute holds the password of the user.

List<Map> createdMaps: This attribute holds the created maps in a list.

List<Score> scores: This attribute holds the scores in a list.

Constructors:

User user(string name, string password): Initializes the user with his/her name and with his/her password

Methods:

List<Score> getScore(): This method holds the scores of the game.

addScore(int score): This method add score into the score list of the game.

List<Map> getCreatedMap(): This method gets the maps which are created by the users.

addMap(Map map): This method add map into the map list of the game.

3.1.4. Score Management Package

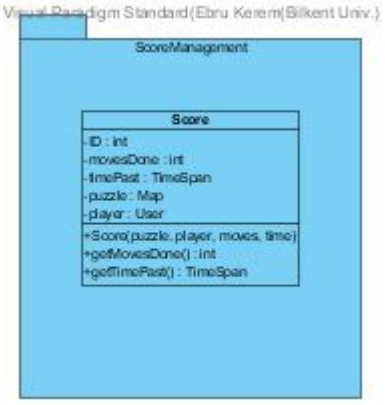

Figure 23: Score Management Package

Score Class

This class controls the scores' activity and properties.

Figure 24: Score Class

Attributes:

int moveDone: This attribute holds the moves done by the user.

TimeSpan timePast: This attribute holds the time span of the game.

Map puzzle: This attribute holds which puzzle the user plays.

Player user: This attribute holds the user who play the game..

Constructors:

Score score(puzzle, player, moves, time): Initializes the score with the given properties.

Methods:

int getMovesDone(): This method counts the number of move the user made.

int getTimePast(): This method holds the passing time in the game.

3.1.5. Puzzle Management Package

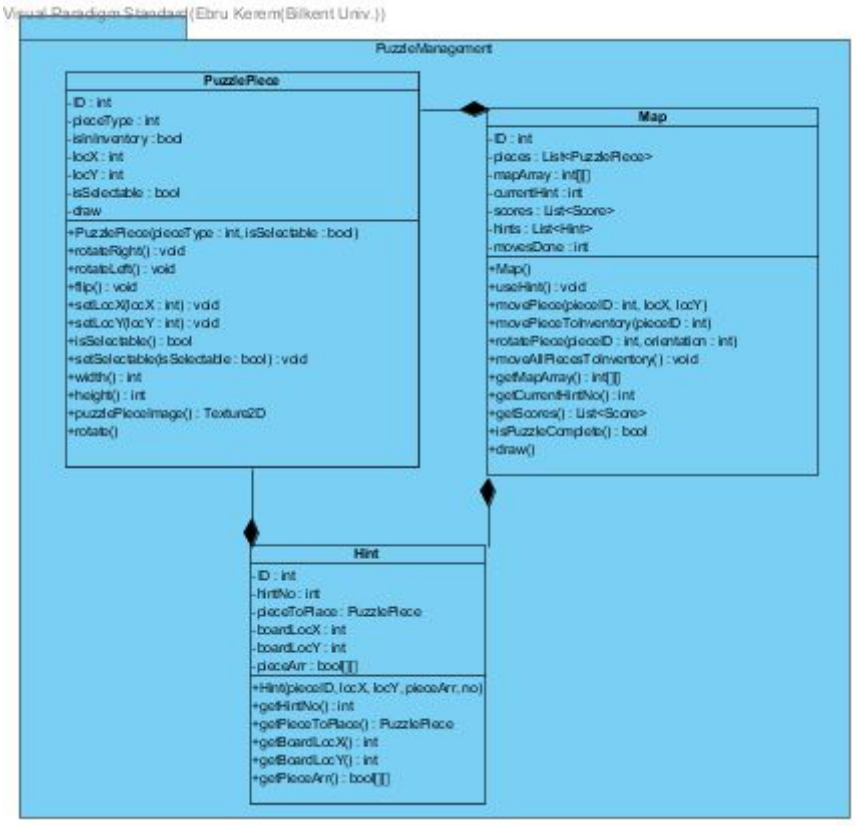

Figure 25: Puzzle Management Package

Puzzle Piece Class

Puzzle piece class hold the properties of the pieces which will be used in the game and the class draws them according to the given datas.

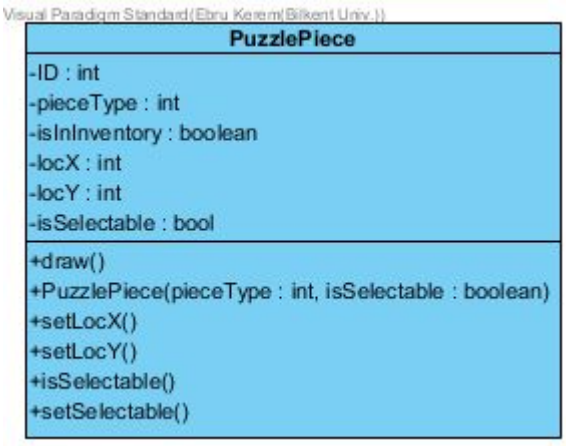

Figure 26: Puzzle Piece class

Attributes:

int pieceType: There are 12 types of puzzle piece in the game. This attribute decides to take one of them.

bool isInInventory: This attribute decides that whether the puzzle piece is in inventory or not.

int locX: This attribute gives the x location of the puzzle piece.

int locY: This attribute gives the y location of the puzzle piece.

Constructors:

PuzzlePiece(int pieceType, bool isSelectable): Initializes the puzzle piece with the given properties.

Methods:

void rotate(int orientation): This method rotates the selected puzzle piece with given orientation.

void setLocX(int X): This method sets the puzzle's x location with given location.

void setLocY(int Y): This method sets the puzzle's y location with given location.

bool isSelectable(): This method shows the availability of the puzzle piece.

void setSelectable(bool selectable): This method changes the availability of the puzzle piece.

Map Class

This class holds the map of the game and puzzle pieces which this map has and handles to draw them.

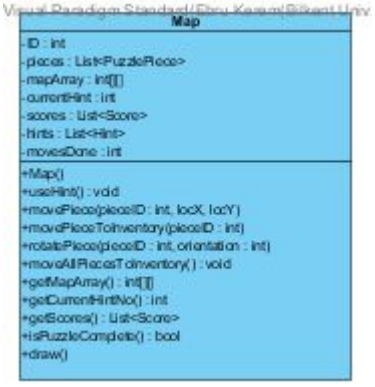

Figure 27: Map class

Attributes:

List<PuzzlePiece> pieces: This attribute holds the puzzle objects as an array. Thus, the map can use them.

int[][] mapArray: This attribute helps to decide whether the game finished or not.

List<Score> scores: This attribute holds the scores as an array. Thus, the map can use them.

List<Hint> hints: This attribute holds the hints as an array. Thus, the map can use them.

int movesDone: This attribute holds the number of move.

Constructors:

Map(): This is a constructor to initialize the map.

Methods:

void useHint(): When the user wants to use hint, this method enables to use the hint.

void movePiece(): This method helps to move the selected puzzle piece.

void draw(): This method draws the map.

boolean isPuzzleComplete(): This method checks if the puzzle is complete or not.

Hint Class

Hint class hold the properties of the pieces which will be used in the hints.

Figure 28: Hint class

Attributes:

int hintNo: This attribute holds the number of the hint.

PuzzlePiece pieceToPlace: This attribute holds the puzzle piece which will be given as a hint.

int boardLocX: This attribute holds the location of the hint.

int boardLocY: This attribute holds the location of the hint.

Constructors:

Hint(): This is a constructor to initialize the hint class.

Methods:

int getHintNo(): This method gets the number of the current hint which will be used.

PuzzlePiece getPieceToPlace(): This method gets the current puzzle piece which will be used as a hint.

int getBoardLocX(): This method gets the current hint's location.

int getBoardLocY(): This method gets the current hint's location.

3.1.6. Custom Puzzle Management Package

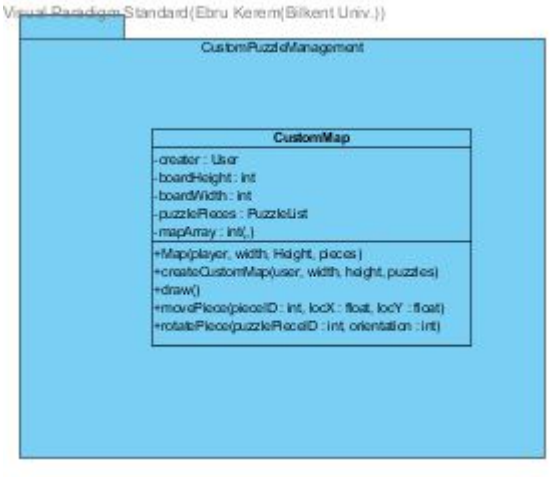

Figure 29: Custom Puzzle Management Package

Custom Map Class

This class holds the custom map of the game, puzzle pieces which this map has, the user which belongs the map. Moreover, this class handles to draw them.

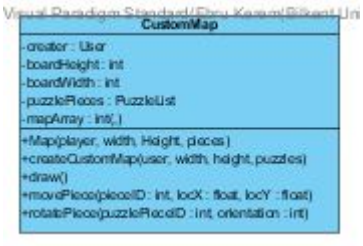

Figure 30: Custom Map Class

Attributes:

User creater: This attribute holds the user of the online game.

int boardHeight: This attribute holds the size of the game board since it can be changed in the custom maps.

int boardWidth: This attribute holds the size of the game board since it can be changed in the custom maps.

List<PuzzlePiece> puzzlePieces: This attribute holds the puzzle objects as an array. Thus, the map can use them.

int(,) mapArray: This attribute holds an array to control the map's condition in the process of the game.

Constructors:

CustomMap(player, width, height, puzzles): This is a constructor to initialize the custom map class with given properties.

Methods:

Map createCustomMap(player, width, height, puzzles): This method assigns the objects of the custom puzzle to the selected one.

void draw(): This method helps to draw the custom map.

movePiece(pieceID, locX, locY): This method enables to move the pieces.

rotatePiece(pieceID, orientation): This method enables to rotate the pieces.

4. Low-level Design

4.1 Object Design Trade-offs

4.1.1 Rapid Development vs Usability

Due to limited time granted for the development of the system, ease of interaction with the game is not going be a priority. The design of the puzzle pieces will not allow users to easily make progress on a puzzle like they would with the game's board counterpart. Nevertheless usability will be an important concern, a player shall be able to learn how to perform basic actions (for example moving or rotating a puzzle piece) after completing the tutorial.

4.1.2 Functionality vs Simplicity

The ability to create, share and play custom maps establishes high functionality to the system. However, the implementation of custom maps compromises the simplicity of the system. All objects of the system are required to be compatible with both solo maps and custom maps.

4.2 Object Design Patterns

4.2.1 Proxy classes

The user and map classes will have their proxy counterparts used in online version of the game. User objects include password, which should not be shared between players in online. Since users will need to see some of the information about other users (for example their username, scores, created maps) there shall be a proxy class of the user class. Similarly, browsing custom maps will require some but not all information about the custom maps in the database. Hence, the proxy class of the custom map class will be used for browsing custom maps.

4.2.2 Façade class

The GameManager class is the façade class for the business layer of the system. Every data transfer from and into the business layer is made by interacting with the GameManager object.

5. Improvement Summary

In the second iteration there are some core changes in the project like the addition of MonoGame framework. Besides the sequential changes from the use of MonoGame framework, there are some changes specific to the system design report that can be listed as

- Changes prompted by first iteration feedback, like moving the references part to the end of the document.
- The general structure of the report was influenced by sample reports in the course web page, this is changed to the sample outline in the project description document
- The 4th section is added, object design explanation is extended
- Quantifiable notations are added to the nonfunctional requirements
- Subsystem decomposition is changed; the subsystems are defined in a different way and are represented in a 3 layer architectural style.
- Access control and security (2.4) section and Boundary Conditions (2.5) are redefined

6. Glossary & references

6.1 Definitions, Acronyms and Abbreviations

GUI: Acronym of “graphical user interface”. GUI has the elements of any graphical properties of a system.

Boundary Condition: Conditions that make the system inputs in bounds of their maximum and minimum limits. Boundary conditions must be adjusted and determined to avoid “out of bounds” errors.

Laggy: User inputs in a laggy system are responded later than the desired time. Poorly implemented code or slow internet might cause a laggy system.

Bytecode converter: Bytecode converter in Java compiler’s converts the written java code into bytecode, this code can run in any kind of machine that has Java Runtime Environment downloaded.

FPS: Acronym of “frames per second”. Fps is the number of consecutive images the system displays in a second.

6.2 References

[1]Bruegge, B. and Dutoit, A. (2014). *Object-oriented software engineering*. 3rd ed. Harlow: Pearson.

[2]Smartgames.eu. (2018). *IQ Puzzler Pro - SmartGames*. [online] Available at: https://www.smartgames.eu/uk/one-player-games/iq-puzzler-pro [Accessed 8 Nov. 2018].

[3]EDUCBA. (2018). *Java vs C# - 8 Most Important Differences You Should Know*. [online] Available at: https://www.educba.com/java-vs-c-sharp/ [Accessed 8 Nov. 2018].