CS319 Object Oriented Software Engineering Project

Design Report

Iteration 1

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: The usability must be kept stable in spite of the added functionalities. Therefore, all the new functionalities of the system should be properly instructed to avoid any confusion to the user. The gameplay has a big role in usability as well. It will have the proper GUI implementation to smoothly drag, drop, rotate and flip puzzle pieces. 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: The testing process will be ongoing not only after the coding is done, but also throughout the stage of implementation, in order to build a system that is as much bug-free as possible. Every boundary condition should be taken into account to prohibit any user from coming across errors, which can exterminate the quality of the experience very quickly.

Maintenance Criteria

Extendibility: Extendibility is very important especially for games since a game must be changing and adapting time to prevent itself from becoming obsolete. The system will be adjustable and adding new functions and classes will be relatively easy. Object oriented design has a key role to provide extendibility. Since there are lots of objects that are independent, no big changes in code will be needed to compose new functions to the system.

Readability: The system should be implemented as clearly as possible for any developer that examines the code to have the best understanding possible. 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 be quickly responding to any input from user. 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.

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.

1.3 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.

1.4 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].

2.Software Architecture

2.1. Subsystem Decomposition

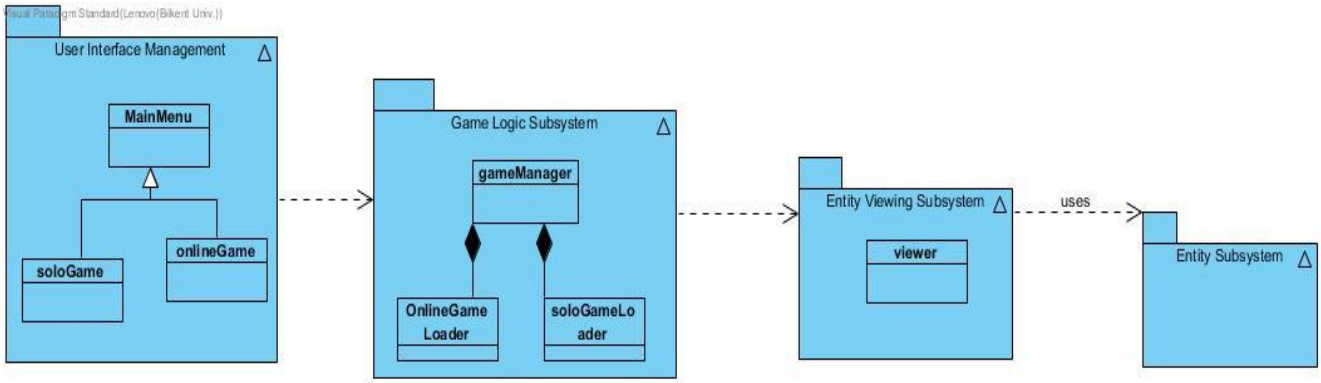


Figure-1 Subsystem Decomposition

Software Architecture
Overview

Here, we will be describing our methods of working as a team or an individual in order to accomplish the challenges we face in this project. We are building the software in parts that we call subsystems which will be explained further in following sections.

The Subsystems Decomposition

In this section we will explain how we divided our project into multiple subsections in order to reduce workload and untangle the problems group working may cause. For example, if we would divide the project class by class, it is inevitable that coupling problems will occur since the purpose of classes or objects in general is that them being compatible and ready to be called or used by any other part of the software it is designed for. By dividing the project in subsystems, we are making sure that every subsystem is a working group of code that can take inputs and outputs without causing any problems what so ever. This is because each subsystem is made by one person and this person constantly tests and analyses the code and makes sure everything is not only working perfectly by themselves, but also working perfectly by each other. The subsystems we decided were UI Subsystem, Game Logic

Subsystem, Entity Subsystem and Viewing Subsystem. In the following paragraph, we will explain these subsystems further.

UI Subsystem

As it can be understood from its name, UI Subsystem handles the interactable part of our software. The IQ Puzzler game we are building is actually not a very interaction based game yet in the implementations, there is a lot to do in this subsystem like the resizing operations or dragging and dropping objects or rotating objects depending on the players will. One other thing we are implementing with this subsystem is, the move counters and listeners which will count the moves player makes in order to complete the puzzle.

Game Logic Subsystem

This is the brain of the game. Every calculation or relatively dependent action will be handled in this subsystem. For example, the conditional states like winning the game or placing a piece into the puzzle will be determined here. Also the online applications or map creator mode will also be done using the Game Logic Subsystem.

This is the subsystem that will have the most traffic in terms of inputs and outputs in all subsystems because it both has to handle the inputs coming from the player and put them into calculation and process these actions before creating a suitable output and sending it back to the other Subsystems accordingly.

Entity Viewing Subsystem

This is the visual engine of the game. All the pieces and maps information will be decomposed in this subsystem and turned into visual objects in the window for the player to understand and play the game properly. The visuals will not be stored here yet the operations dependant on showing anything on screen will be handled in this subsection. This also includes the highscores, map previews, online map browsers, menu options and so on.

Entity Subsystem

This subsystem is like the storage unit of the software. All the map information will be stored here in a form of a compressed information that can be decompressed and turned into the real visual map anytime the player requests to play the game. Along with the maps, the puzzle pieces and their attributes are also stored here. If the player wishes to download and play another level made by someone else in the online custom map mode, the new map will be stored here as a downloaded custom map so the player does not have to download the same map information over and over again.

Unlike other subsystems, this subsystem almost never gets any input yet it always sends out outputs. Almost every single action in game requires a stream of information from this subsystem.

The points or high scores will be stored in this as well and will be sent to the server if the player achieved that score in an online game to compare his or her score with other players.

Not as significant as others but the hints we will be giving during the tutorial is also here.

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 SQL 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

Our game is mostly playable without an internet connection yet if the player wishes to play the maps other people created, the player has to connect to SQL where all these information is stored. Also for viewing and comparing high scores with other players, an SQL connection is needed as well.

In terms of security and privacy, we narrowed the type of information that can be sent or received to our servers to a minimal and all this information is sort of encrypted due to our compression techniques where only the game itself knows how to decompress the information and the SQL does not.

Also the player does not have to give out any information to the game, all the game requires is a nickname and a password. Also this is something optional for online players so there won't be any security or privacy problems anytime.

2.5. Boundary Conditions

Initialization

The game is started by clicking on the .exe file. There is no installation required since the game does not need any kind of extensions or additional files.

Termination

The player can terminate the game from almost any window he or she desires from the pause menu. If the player is already in the main menu they can just click the “exit game” button when they wish to stop the file.

If the player is in the map browser or level selection screen, they can simply click to go back and then exit as described above.

If a current game is being played at the moment of termination request, a pop-up window will come up to ask if the player really intends to end the game or no.

Error

If any kind of an error occurs, the game will show pop-ups without closing or minimizing the window since the puzzles are timed and this kind of obstacle may frustrate players if they are trying to achieve high scores.

Error messages can be ranged from a loading error or simply a file locating error which are sort of unsolvable so the best option is to restart the game which will be the only option the error pop-up offers.

For any other kind of errors like trying to play a puzzle level which the player did not have access to. For these kind of simple errors, the game will not offer anything for the player and will just show the error and take the player back to the previous screen without letting the player do the forbidden thing he or she was trying to do initially.

3. Subsystem Services

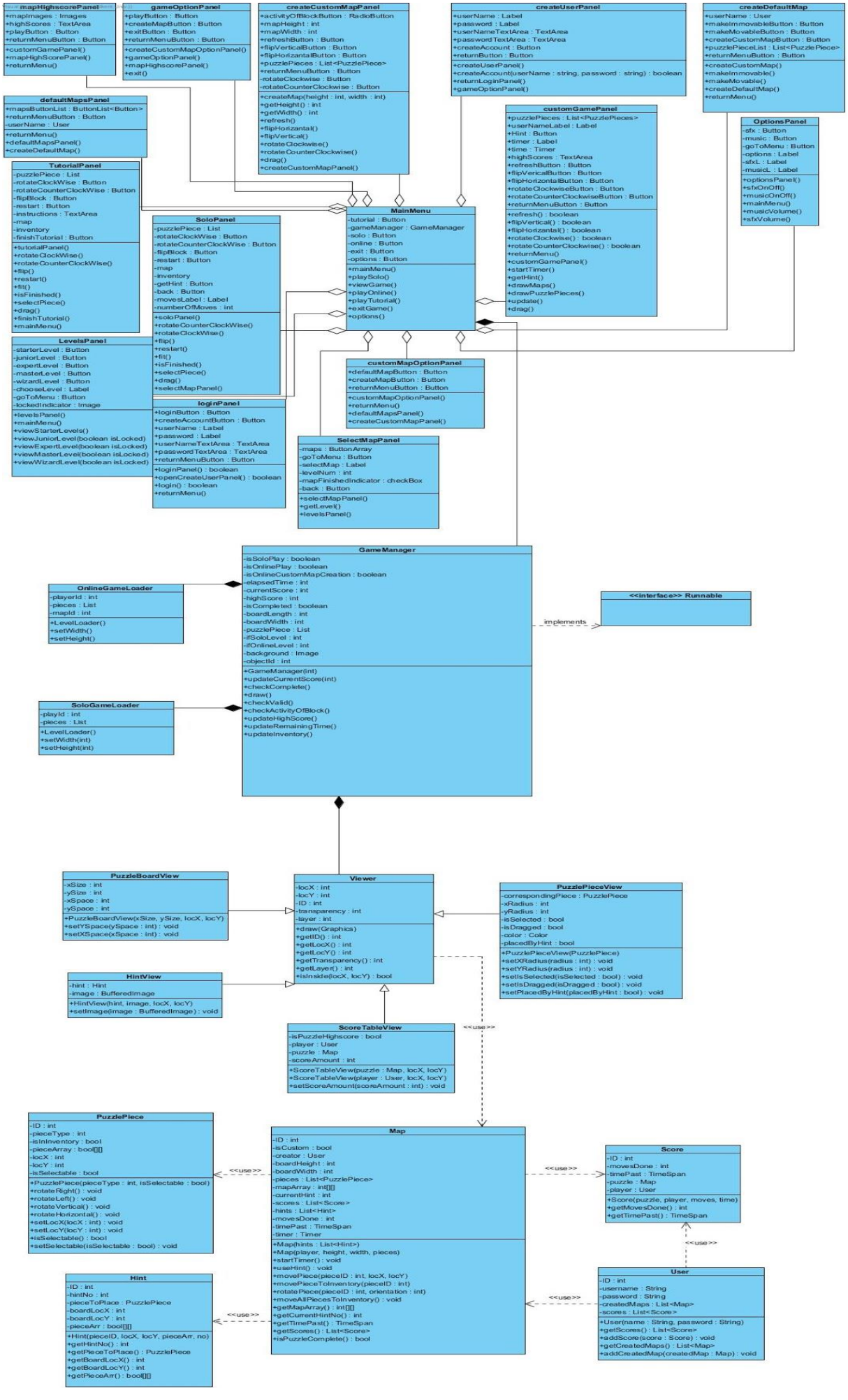

mapHighscorePanel
+mapImages : Images
+highScores : TextArea
+playButton : Button
+returnMenuButton : Button
+customGamePanel()
+mapHighScorePanel()
+returnMenu()
gameOptionPanel
+playButton : Button
+createMapButton : Button
+exitButton : Button
+returnMenuButton : Button
+createCustomMapOptionPanel()
+gameOptionPanel()
+mapHighscorePanel()
+exit()
createCustomMapPanel
+activityOfBlockButton : RadioButton
+mapHeight : int
+mapWidth : int
+refreshButton : Button
+flipVerticalButton : Button
+flipHorizantalButton : Button
+puzzlePieces : List<PuzzlePiece>
+returnMenuButton : Button
-rotateClockwise : Button
-rotateCounterClockwise : Button
+createMap(height : int, width : int)
+getHeight() : int
+getWidth() : int
+refresh()
+flipHorizantal()
+flipVertical()
+rotateClockwise()
+rotateCounterClockwise()
+drag()
+createCustomMapPanel()
createUserPanel
+userName : Label
+password : Label
+userNameTextArea : TextArea
+passwordTextArea : TextArea
+createAccount : Button
+returnButton : Button
+createUserPanel()
+createAccount(userName : string, password : string) : boolean
+returnLoginPanel()
+gameOptionPanel()
createDefaultMap
+userName : User
+makeImmovableButton : Button
+makeMovableButton : Button
+createCustomMapButton : Button
+puzzlePieceList : List<PuzzlePiece>
+returnMenuButton : Button
+createCustomMap()
+makeImmovable()
+makeMovable()
+createDefaultMap()
+returnMenu()
defaultMapsPanel
+mapsButtonList : ButtonList<Button>
+returnMenuButton : Button
-userName : User
+returnMenu()
+defaultMapsPanel()
+createDefaultMap()
customGamePanel
+puzzlePieces : List<PuzzlePieces>
+userNameLabel : Label
+Hint : Button
+timer : Label
+time : Timer
+highScores : TextArea
+refreshButton : Button
+flipVericalButton : Button
+flipHorizontalButton : Button
+rotateClockwiseButton : Button
+rotateCounterClockwiseButton : Button
+returnMenuButton : Button
+refresh() : boolean
+flipVertical() : boolean
+flipHorizantal() : boolean
+rotateClockwise() : boolean
+rotateCounterClockwise() : boolean
+returnMenu()
+customGamePanel()
+startTimer()
+getHint()
+drawMaps()
+drawPuzzlePieces()
+update()
+drag()
OptionsPanel
-sfx : Button
-music : Button
-goToMenu : Button
-options : Label
-sfxL : Label
-musicL : Label
+optionsPanel()
+sfxOnOff()
+musicOnOff()
+mainMenu()
+musicVolume()
+sfxVolume()
TutorialPanel
-puzzlePiece : List
-rotateClockWise : Button
-rotateCounterClockWise : Button
-flipBlock : Button
-restart : Button
-instructions : TextArea
-map
-inventory
-finishTutorial : Button
+tutorialPanel()
+rotateClockWise()
+rotateCounterClockWise()
+flip()
+restart()
+fit()
+isFinished()
+selectPiece()
+drag()
+finishTutorial()
+mainMenu()
SoloPanel
-puzzlePiece : List
-rotateClockWise : Button
-rotateCounterClockWise : Button
-flipBlock : Button
-restart : Button
-map
-inventory
-getHint : Button
-back : Button
-movesLabel : Label
-numberOfMoves : int
+soloPanel()
+rotateCounterClockWise()
+rotateClockWise()
+flip()
+restart()
+fit()
+isFinished()
+selectPiece()
+drag()
+selectMapPanel()
MainMenu
-tutorial : Button
-gameManager : GameManager
-solo : Button
-online : Button
-exit : Button
-options : Button
+mainMenu()
+playSolo()
+viewGame()
+playOnline()
+playTutorial()
+exitGame()
+options()
LevelsPanel
-starterLevel : Button
-juniorLevel : Button
-expertLevel : Button
-masterLevel : Button
-wizardLevel : Button
-chooseLevel : Label
-goToMenu : Button
-lockedIndicator : Image
+levelsPanel()
+mainMenu()
+viewStarterLevels()
+viewJuniorLevel(boolean isLocked)
+viewExpertLevel(boolean isLocked)
+viewMasterLevel(boolean isLocked)
+viewWizardLevel(boolean isLocked)
customMapOptionPanel
+defaultMapButton : Button
+createMapButton : Button
+returnMenuButton : Button
+customMapOptionPanel()
+returnMenu()
+defaultMapsPanel()
+createCustomMapPanel()
loginPanel
+loginButton : Button
+createAccountButton : Button
+userName : Label
+password : Label
+userNameTextArea : TextArea
+passwordTextArea : TextArea
+returnMenuButton : Button
+loginPanel() : boolean
+openCreateUserPanel() : boolean
+login() : boolean
+returnMenu()
SelectMapPanel
-maps : ButtonArray
-goToMenu : Button
-selectMap : Label
-levelNum : int
-mapFinishedIndicator : checkBox
-back : Button
+selectMapPanel()
+getLevel()
+levelsPanel()
GameManager
-isSoloPlay : boolean
-isOnlinePlay : boolean
-isOnlineCustomMapCreation : boolean
-elapsedTime : int
-currentScore : int
-highScore : int
-isCompleted : boolean
-boardLength : int
-boardWidth : int
-puzzlePiece : List
-ifSoloLevel : int
-ifOnlineLevel : int
-background : Image
-objectId : int
+GameManager(int)
+updateCurrentScore(int)
+checkComplete()
+draw()
+checkValid()
+checkActivityOfBlock()
+updateHighScore()
+updateRemainingTime()
+updateInventory()
OnlineGameLoader
-playerId : int
-pieces : List
-mapId : int
+LevelLoader()
+setWidth()
+setHeight()
<<Interface>> Runnable
implements
SoloGameLoader
-playId : int
-pieces : List
+LevelLoader()
+setWidth(int)
+setHeight(int)
PuzzleBoardView
-xSize : int
-ySize : int
-xSpace : int
-ySpace : int
+PuzzleBoardView(xSize, ySize, locX, locY)
+setYSpace(ySpace : int) : void
+setXSpace(xSpace : int) : void
Viewer
-locX : int
-locY : int
-ID : int
-transparency : int
-layer : int
+draw(Graphics)
+getID() : int
+getLocX() : int
+getLocY() : int
+getTransparency() : int
+getLayer() : int
+isInside(locX, locY) : bool
PuzzlePieceView
-correspondingPiece : PuzzlePiece
-xRadius : int
-yRadius : int
-isSelected : bool
-isDragged : bool
-color : Color
-placedByHint : bool
+PuzzlePieceView(PuzzlePiece)
+setXRadius(radius : int) : void
+setYRadius(radius : int) : void
+setIsSelected(isSelected : bool) : void
+setIsDragged(isDragged : bool) : void
+setPlacedByHint(placedByHint : bool) : void
HintView
-hint : Hint
-image : BufferedImage
+HintView(hint, image, locX, locY)
+setImage(image : BufferedImage) : void
ScoreTableView
-isPuzzleHighscore : bool
-player : User
-puzzle : Map
-scoreAmount : int
+ScoreTableView(puzzle : Map, locX, locY)
+ScoreTableView(player : User, locX, locY)
+setScoreAmount(scoreAmount : int) : void
<<use>>
PuzzlePiece
-ID : int
-pieceType : int
-isInInventory : bool
-pieceArray : bool[][]
-locX : int
-locY : int
-isSelectable : bool
+PuzzlePiece(pieceType : int, isSelectable : bool)
+rotateRight() : void
+rotateLeft() : void
+rotateVertical() : void
+rotateHorizontal() : void
+setLocX(locX : int) : void
+setLocY(locY : int) : void
+isSelectable() : bool
+setSelectable(isSelectable : bool) : void
Map
-ID : int
-isCustom : bool
-creator : User
-boardHeight : int
-boardWidth : int
-pieces : List<PuzzlePiece>
-mapArray : int[][]
-currentHint : int
-scores : List<Score>
-hints : List<Hint>
-movesDone : int
-timePast : TimeSpan
-timer : Timer
+Map(hints : List<Hint>)
+Map(player, height, width, pieces)
+startTimer() : void
+useHint() : void
+movePiece(pieceID : int, locX, locY)
+movePieceToInventory(pieceID : int)
+rotatePiece(pieceID : int, orientation : int)
+moveAllPiecesToInventory() : void
+getMapArray() : int[][]
+getCurrentHintNo() : int
+getTimePast() : TimeSpan
+getScores() : List<Score>
+isPuzzleComplete() : bool
Score
-ID : int
-movesDone : int
-timePast : TimeSpan
-puzzle : Map
-player : User
+Score(puzzle, player, moves, time)
+getMovesDone() : int
+getTimePast() : TimeSpan
Hint
-ID : int
-hintNo : int
-pieceToPlace : PuzzlePiece
-boardLocX : int
-boardLocY : int
-pieceArr : bool[][]
+Hint(pieceID, locX, locY, pieceArr, no)
+getHintNo() : int
+getPieceToPlace() : PuzzlePiece
+getBoardLocX() : int
+getBoardLocY() : int
+getPieceArr() : bool[][]
User
-ID : int
-username : String
-password : String
-createdMaps : List<Map>
-scores : List<Score>
+User(name : String, password : String)
+getScores() : List<Score>
+addScore(score : Score) : void
+getCreatedMaps() : List<Map>
+addCreatedMap(createdMap : Map) : void

Figure-2 Class Diagram

Better Resolution Can Be Seen From:
https://github.com/csLover1337/Elon-s-musk/blob/master/Reports/design%20class%20diagram.jpg

3.1 User Interface Subsystem

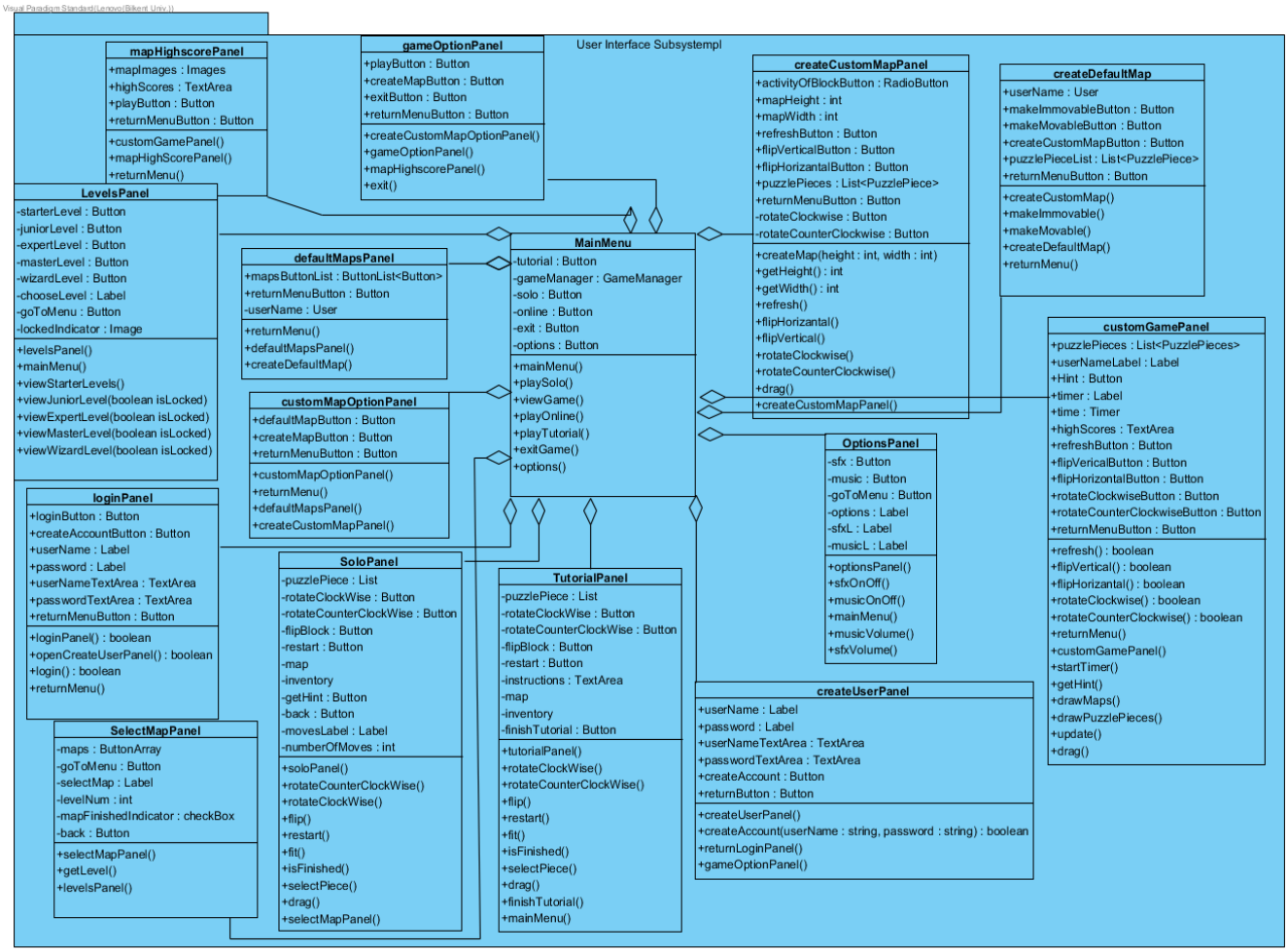
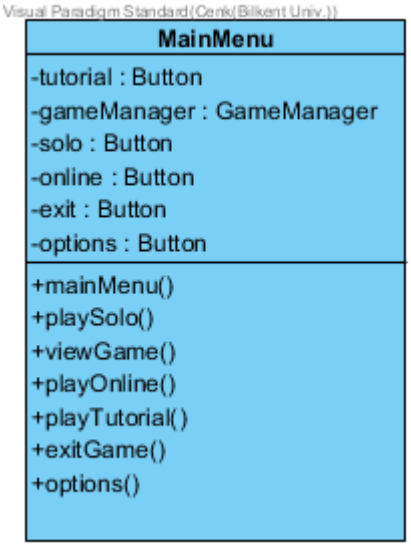


Figure-3 User Interface Subsystem

This subsystem consists of panels that the user can see when the game processes. The user can play the game in two different modes. One of them is solo mode and the other one is online mode. Since they have different characteristic features, panels will be showed in two separate subsections.

3.1.1 Solo Game User Interface

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.

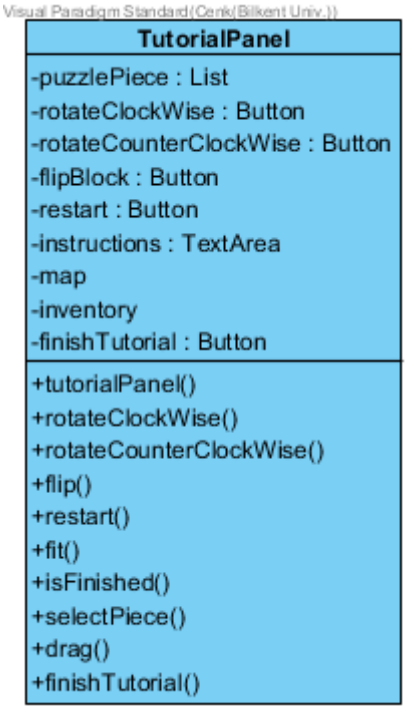
Boolean playOnline: This method open the online game panels when the button is pressed.

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



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

Map:

Inventory:

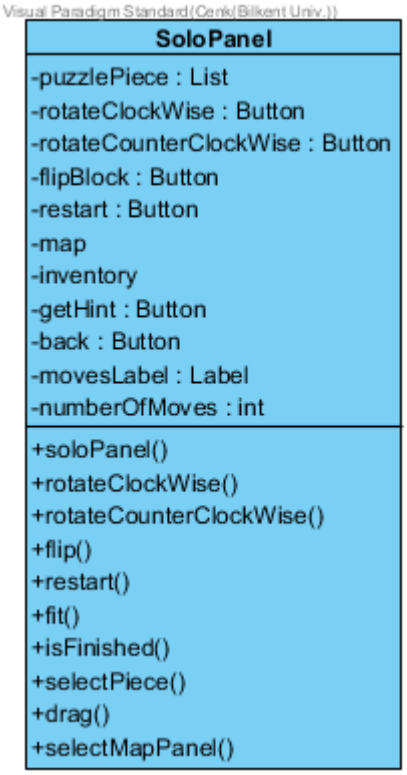
Button finishTutorial: Ends the tutorial mode and closes the panel.

Constructors:

tutorialPanel() : Initializes the tutorial panel.

Methods:

SoloPanel 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.

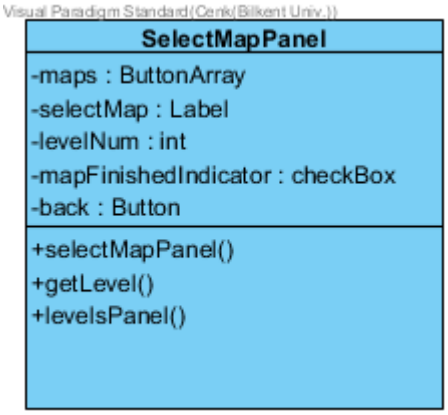
boolean fit: This method checks if piece that is chosen by the user and dragged to the map can fit in the desired place.

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



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:

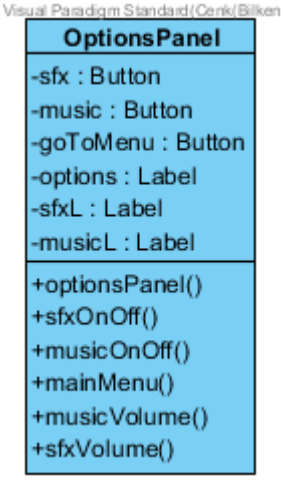
void selectMapPanel: This constructor initializes the select map panel.

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



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.

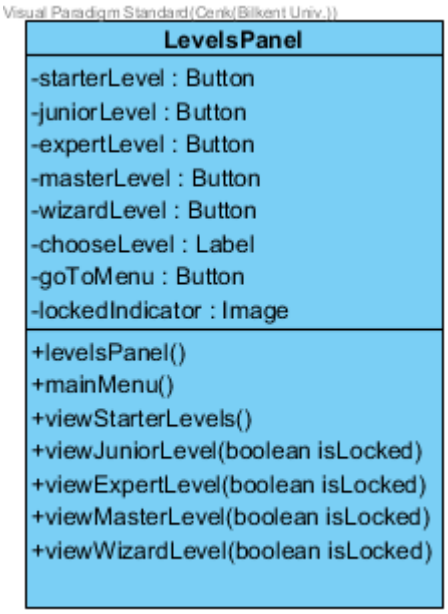
Label musicL: It is for the implication of which button is for music.

Constructors:

Void optionsPanel: This constructor initializes the options panel.

Methods:

LevelsPanel Class



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.

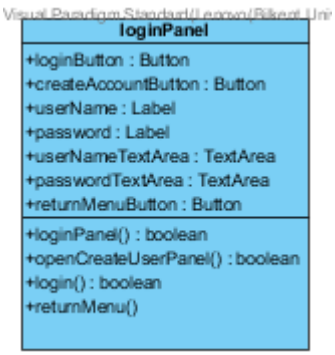
boolean viewExpertLevel: If the user presses the expert level button it will open the maps inside the expert level pack. Only if the pack not locked.

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.

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.

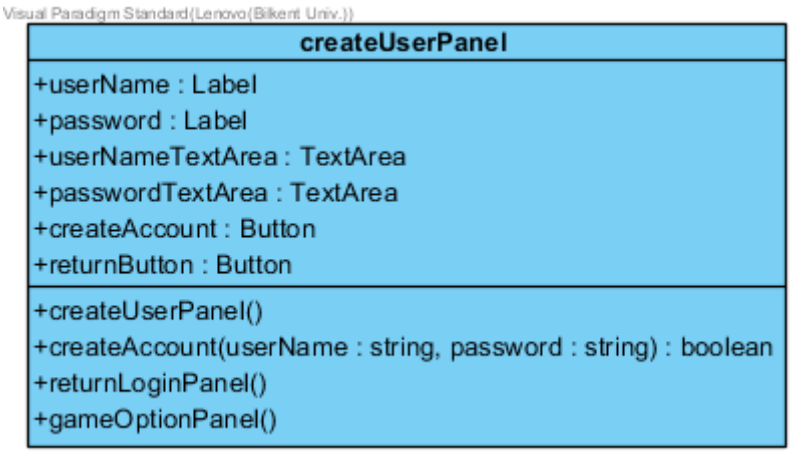
Methods:

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



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.

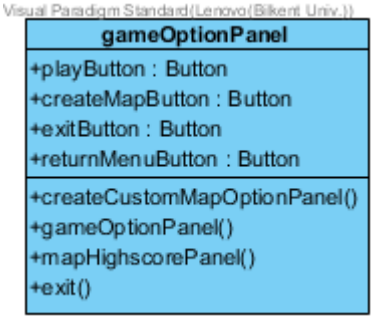
Methods:

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.

gameOptionPanel Class



Attributes:

Button playButton: 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 exitButton: This attribute is a button. When the user press this button, he/she can exit the game.

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

Constructors:

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

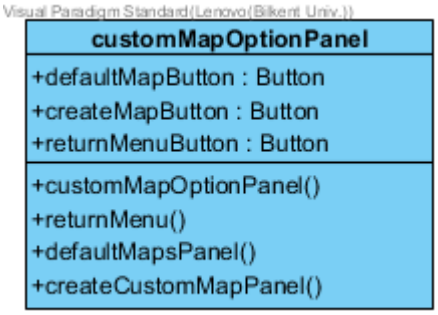
Methods:

boolean createCustomMapOptionPanel: This method is used to lead the user to the panel which shows the custom map creation options.

void mapHighscorePanel: This method is used to lead the user to the panel which shows the maps and maps' highscores.

void exit: This method is used to exit the game.

customMapOptionPanel Class



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.

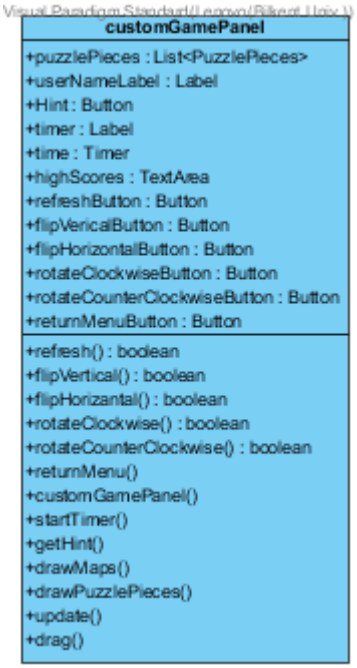
Methods:

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



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.

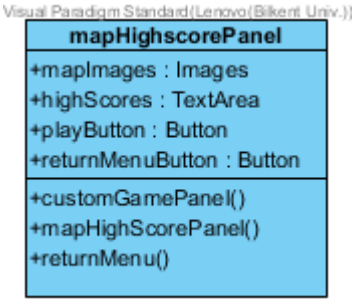
void drawMaps: This method initializes the map and draw it on the screen.

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



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:

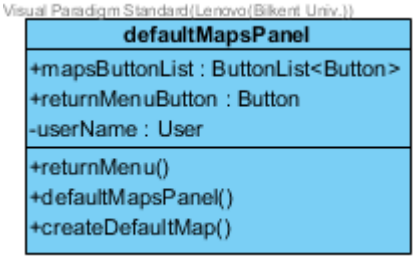
void mapHighScorePanel: Initializes the mapHighScorePanel: maps, buttons, high score tables.

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



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:

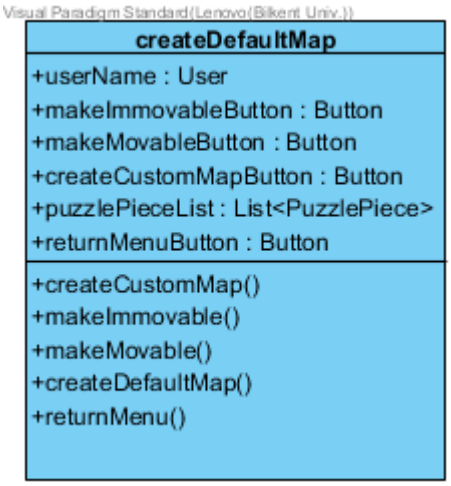
void defaultMapsPanel: Initializes the defaultMaosPanel: maps, buttons.

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



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:

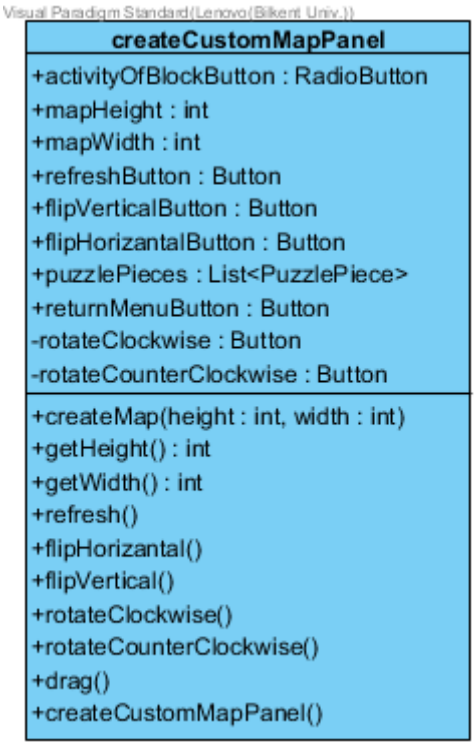
void returnMenu: When the user wants to return previous page, this method calls the previous panel.

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



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.

3.2 Entity Viewing Subsystem

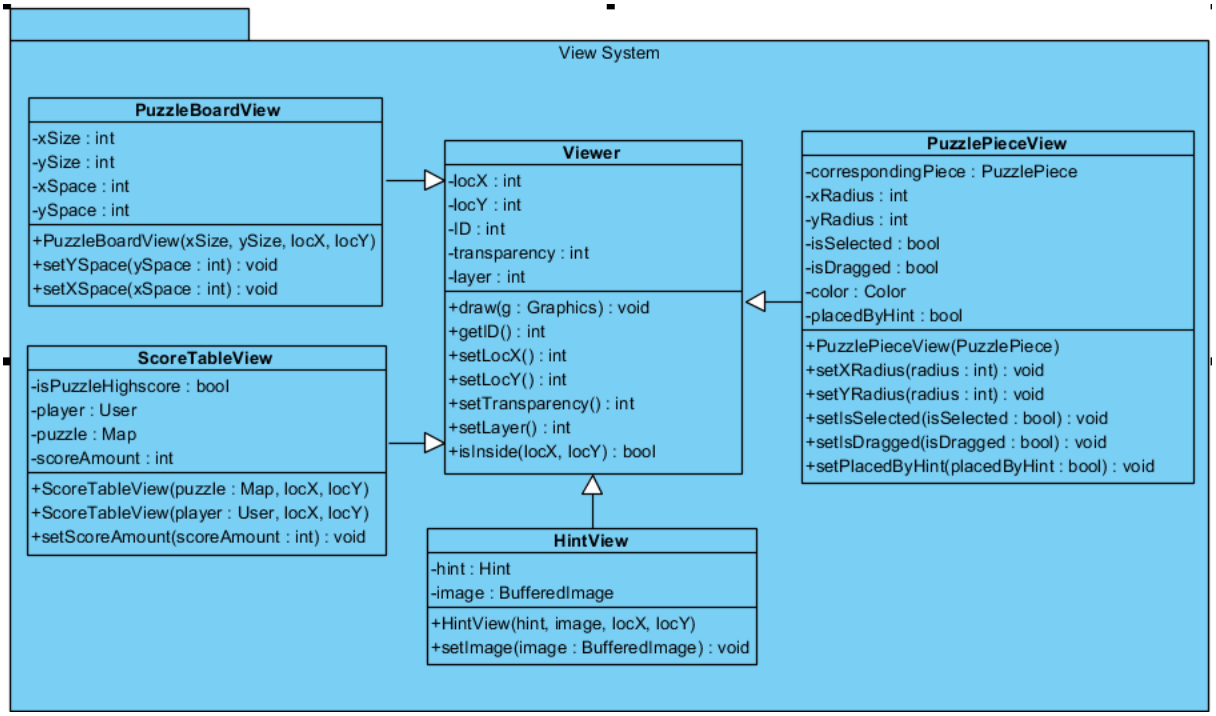
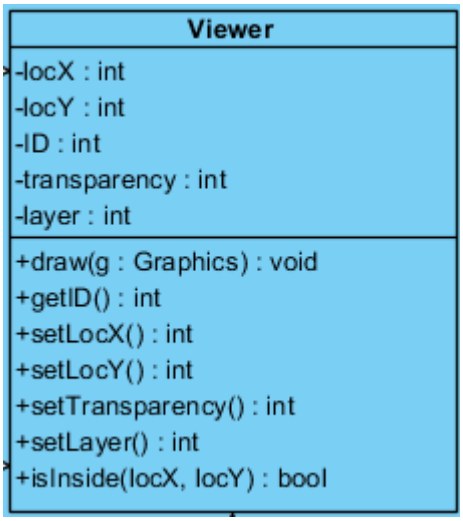


Figure- 4 Entity Viewing Subsystem Diagram

This system consists of viewer objects of the game. Game logic subsystem will determine how to view game objects in the game panels by using controllers. This system has a general abstract class called “Viewer” which every other view classes inherit from.

Viewer Class



Attributes:

int locX : The object's X coordinate location of its leftmost point.

int locY : The object's Y coordinate location of its topmost point.

The locX and locY are always given by the controllers from the game logic subsystem, as the panel sizes, hence locations, are subject to change.

int ID : The unique identification number of the object.

int transparency : The transparency of the object.

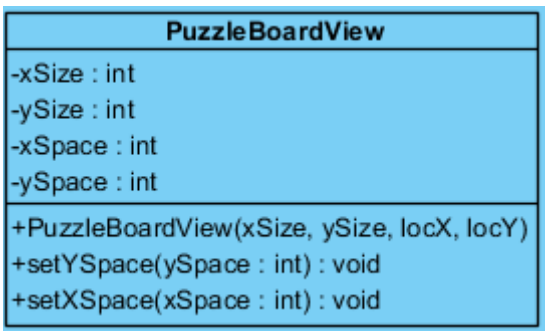
int layer : The presentation layer of the visualisation in the panel

Methods:

void draw(Graphics g) : This is an abstract method to draw the objects in the panel.

bool isInside(locX, locY) : This is an abstract method to check if a given location is inside the object's visual.

PuzzleBoardView Class



Attributes:

int xSize : The number of horizontal slots in the board. It is 11 for solo puzzles.

int ySize : The number of vertical slots in the board. It is 5 for solo puzzles.

int xSpace : The horizontal length of a slot in the board.

int ySpace : The vertical length of a slot in the board.

(xSize)*(xSpace)/(“horizontal length of panel”) will be a constant value. It will be the same for the vertical coordinate too.

int layer : The layer of this class will be lowest after panel itself.

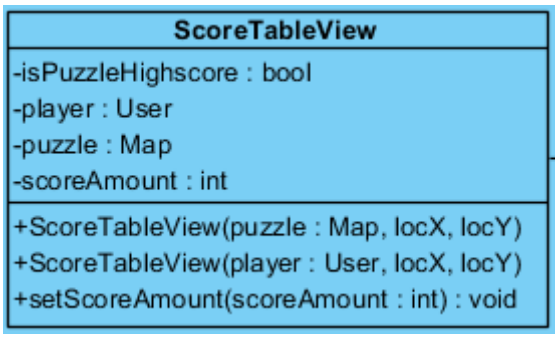
Constructor:

PuzzleBoardView(xSize,ySize,locX,locY) : Simply prepares object with given values.

Methods:

draw(Graphics g) : This class will simply prepare a grid in the panel to create the board lines. It may also draw more (perhaps highly transparent ellipses) depending on how it will look with the game panel.

ScoreTableView Class



Attributes:

bool isPuzzleHighscore : True if its created for a puzzle, false if its created for an user.

User player : Points to the player the object is created for.

Map puzzle : Points to the puzzle the object is created for.

int scoreAmount : The amount of scores to display in the table.

Constructors:

ScoreTableView(puzzle, locX, locY) : This constructor is used when highscores for a puzzle is to be displayed.

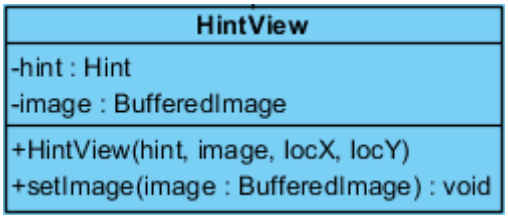
ScoreTableView(player, locX, locY) : This constructor is used when highscores of an user is to be displayed.

Methods:

void draw(Graphics g) : Draws a table and puts scores in a String format.

void setScoreAmount(scoreAmount) : Open to be changed by options.

HintView Class



Attributes:

Hint hint : The Hint object corresponded to its viewer.

BufferedImage image : The image to display hint.

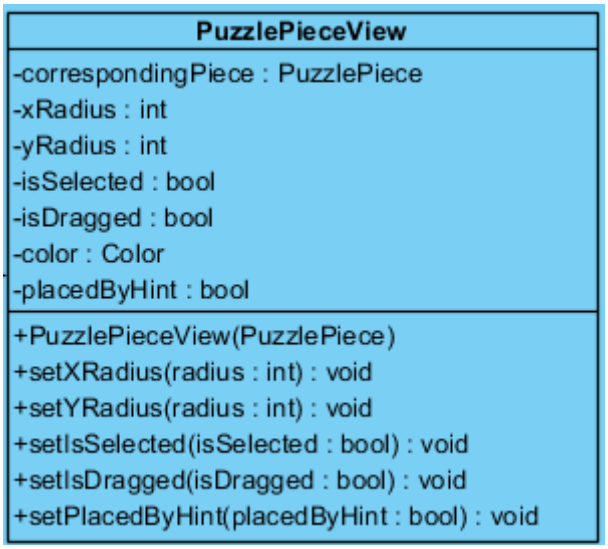
Constructor:

HintView(hint, image ,locX , locY) : Simple constructor.

Methods:

void draw(Graphics g) : Draws the hint on the panel.

PuzzlePieceView Class



Attributes:

PuzzlePiece correspondingPiece : Pointer to the corresponding puzzle piece.

int xRadius : The horizontal radius of the ellipses.

int yRadius : The vertical radius of the ellipses.

The radius values are related to the board's slot lengths. The radii are equal when the board is 5x11, which is the size of the solo board.

bool isSelected : When the piece is selected its view gets highlighted.

bool isDragged : While the piece is getting dragged its transparency increases.

Color color : Determined by the type of the puzzle piece.

bool placedByHint : The piece has a special visualization when placed by a hint

Constructor:

PuzzlePieceView(PuzzlePiece) : Initialized by a pointer to the piece to view. Initialized location is the inventory.

Methods:

void draw(Graphics g) : Draws ellipses and rectangles. Uses pointed piece's filling array with its own radii and location values to draw accordingly.

bool isInside(locX, locY) : Determines if a location is inside the drawn piece.

3.3 Game Logic Subsystem

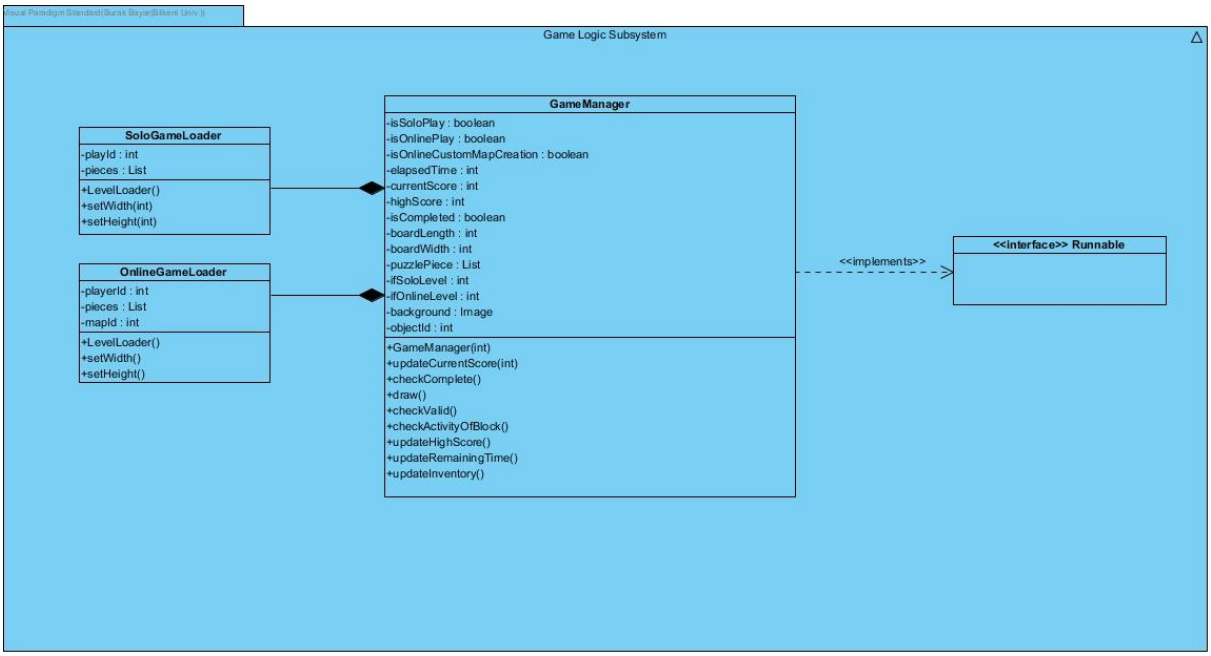


Figure- 5 Game Logic Subsystem Diagram

The diagram shows the composition of Game Logic Subsystem. This subsystem is responsible for loading all related objects and using them. Once this subsystem called it also means that game is initialized. All user interface subsystems are linked to this logic system and all components that are related with the game is initialized. Also, game Objects(like Pieces List) will be updated via this subsystem.

GameManager Class:

In this subsystem, there will be a list which is called puzzlePiece and it will keep pieces to be used in our game and all of these will be added to this list efficiently.

While game is continuing, this class will be in loop by drawing and updating all required methods continuously.

Attributes:

- **isSoloPlay:** This attribute will be used to decide whether it is a solo game or not.
- **isOnlinePlay:** This attribute will be used to decide whether user selected online game or not.
- **isOnlineCustomMapCreation:** This will be keeping data of user selection is towards creating a custom map or not.
- **elapsedTime:** Keeps track of time in terms of seconds and counts it. It will be kept as integer.
- **currentScore:** For both online and solo games score will be stored as integer. According to time elapsed, score will be counted and for online game it will be placed and sorted to highscores.
- **highScore:** For just online game score will be stored as integer and according to time elapsed, score will be counted and will be placed and sorted to highscores table.

- **isCompleted:** This is a boolean attribute and once the player wishes to complete finishing puzzle solving or creating a custom map, this will check whether all areas on board are filled or not.
- **boardLength:** This attribute will keep data of length for a specific level and initialize board accordingly.
- **boardWidth:** This attribute will keep data of width for a specific level and initialize board accordingly.
- **puzzlePiece:** This is a Lisk attribute and all puzzle pieces will be kept in that list and when they are required, this link will be used to get.
- **ifSoloLevel:** User Interface for solo and Online will differ and according to this boolean attribute, which panel will be used will be decided.
- **ifOnlineLevel:** User Interface for Solo and Online will differ and according to this boolean attribute, which panel will be used will be decided.
- **background:** This attribute keeps the background as image.
- **objectId:** This attribute represents the unique identification number of the piece object.

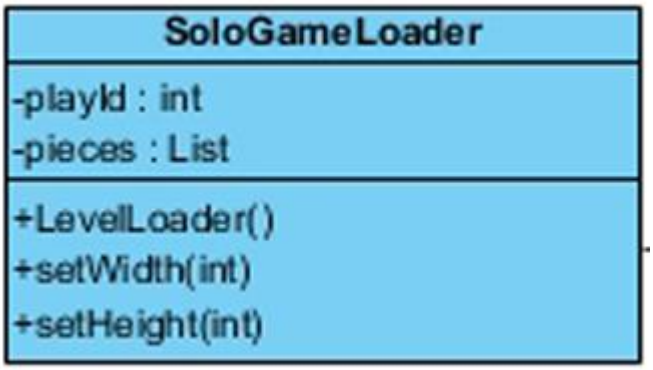
Constructor:

- **GameManager:** This constructor initialize a game according to given parameter which is kind of an id in integer form and according to game's level all related objects will be taken. For example, puzzle piece objects, map properties etc will be grabbed.

Methods:

- **void GameManager(int):** This method is for security reasons, if any player wants to reach all datas in a short time, it will close the program and give an error.
- **void updateCurrentScore(int):** This method updates score according to time.
- **boolean checkComplete():** This method checks whether a board is full-filled or not and returns a boolean.
- **void draw():** This method draws the map and all the other user-interface stuff.

- **boolean checkValid():** This method will be checking if the puzzle piece can be located to a location or not and returns a boolean according to that.
- **boolean checkActivityOfBlock():** While some of blocks will be turned off and no piece can be located, this method will be providing the information by returning boolean about whether this block can be located or not.
- **void updateRemainingTime():** This method is for giving extra points and in some time if the user finishes the puzzle, then remaining time will give him a extra score. UpdateRemainingTime will be counting this time and decreasing it each second.
- **void updateInventory():** This method will update inventory and once the piece is located from inventory to the board, it will delete this item from inventory. In other case, if the user locates a piece from board to inventory, then it will add this piece to inventory list.



SoloGameLoader Class

This class is responsible for Solo Game playing and it will help Game Manager to initialize game for online gaming. Game will be initialized according to pieces that will be used in game and mapId that stores the id of map and it will be used to identify map quantities.

Attributes:

- **playId:** It keeps data as integer and it is a unique number in database for map properties.

- **pieces:** It is a list that will be used while playing our game.

Constructor:

- **SoloGameLoader(playerId,pieces,mapId):** Constructor will be helping to initialize game according to pieces that will be used and created map's id that stores the data.

Methods:

- **LevelLoader:** This method will load the map and get map's properties.
- **setWidth:** By default, width of the board is 11 but if future update is needed and width will not be constant in solo game, it will help us(developer) to change.
- **setHeight:** By default, length of the board is 5 but if future update is needed and width will not be constant in solo game, it will help us(developer) to change.

OnlineGameLoaderClass

This class is responsible for Online Game playing and it will help Game Manager to initialize game for online gaming. Game will be initialized according to player's database id and pieces that will be used in game and mapId that stores the id of map and it will be used to identify map quantities.

Attributes:

- **playerId:** Every account has id in database as integer.
- **pieces:** It is a list that will be used while playing our game.

- **mapId:** It keeps data as integer and it is a unique number in database for map properties.

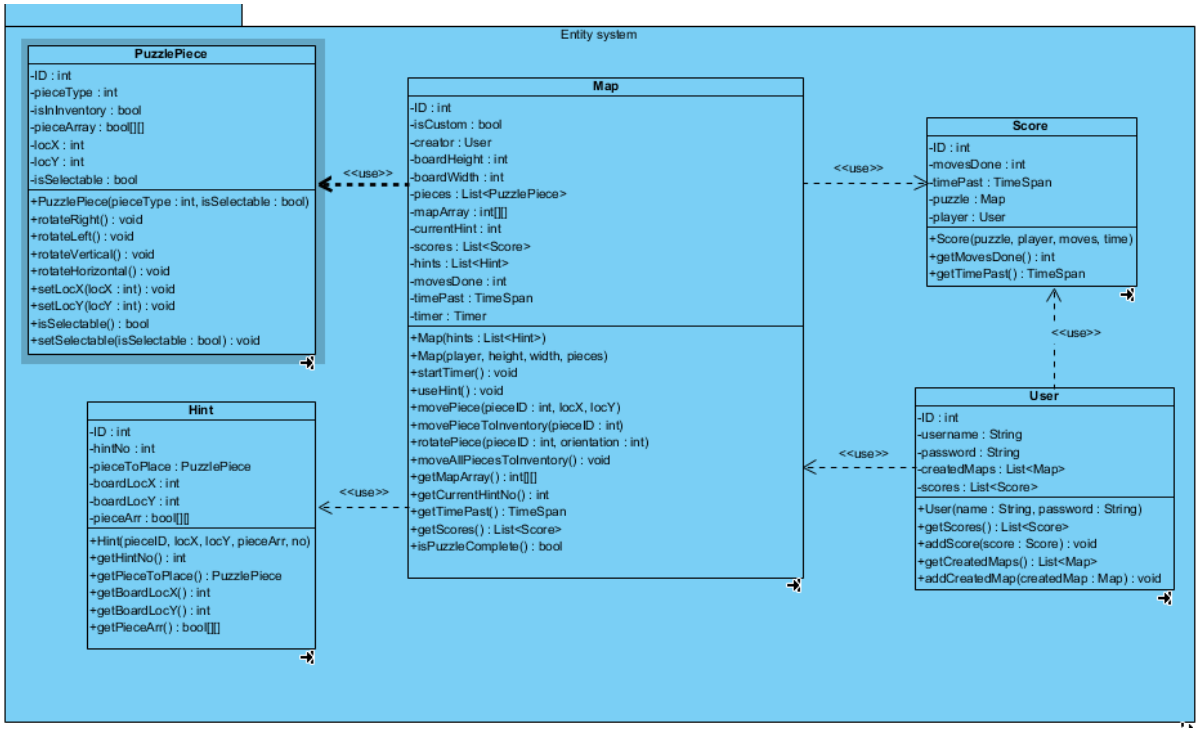
Constructor:

- **OnlineGameLoader(playerId,pieces,mapId):** Constructor will be helping to initialize game according to player's id, pieces that will be used and created map's id that stores the data.

Methods:

- **LevelLoader:** This method will load the map and get map's properties.
- **setWidth:** By default, width of the board is 11 but if player wants to create a custom map, then user can change width and this method is responsible for this action.
- **setHeight:** By default, length of the board is 5 but if player wants to create a custom map, then user can change height and this method is responsible for this action.

3.4 Entity Subsystem



This is the core system of the game, where attributes of what is inside the game is defined. Get and set methods are used for the most part except Map class. Only exceptions are puzzle piece class' orientation changing methods.

Map Class

Attributes:

bool isCustom : True when the puzzle is custom made.

User creator : Null for solo puzzles. Holds a pointer to the user created.

int boardHeight : Amount of vertical slots in the board.

int boardWidth : Amount of horizontal slots in the board.

List<PuzzlePiece> pieces : The list of puzzle pieces used in the board. For solo puzzles there are 12 pieces of types 1-12.

int[][] mapArray : This is a two dimensional array of piece ID's which represent the state of the board and pieces.

int currentHint : Initialized as 0, maximum value is equal to the amount of hints can be used

List<Score> scores : List of scores done in the puzzle. Solo puzzles can have max 1 score

List<Hint> hints : List of hints of the puzzle. Custom puzzles are currently not planned to have any.

int movesDone : Amount of moves done in the puzzle since starting.

Timer timer : Starts holding time when the puzzle starts.

TimeSpan timePast : Time spent solving the puzzle.

Constructors:

Map(List<Hint> hints) : This is the constructor used to initialize solo puzzles.

Map(User player, int height, int width, List<PuzzlePiece> pieces) : This is constructor used to initialize custom puzzles.

Methods:

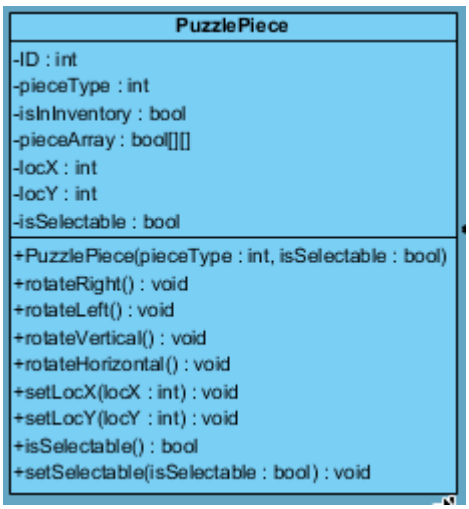
void useHint() : Activates the next hint, only usable in solo puzzles. Places a piece to its right place and makes it unselectable.

void movePiece(int pieceID, int locX, int locY) : Moves the piece with the given ID to the given location. May fail if its not possible to do so.

void movePieceToInventory(int pieceID) : Moves the piece with the given ID to the inventory.

void rotatePiece(int pieceID, int orientation) : The integer type orientation value is the action which corresponds to one of the rotate methods of puzzle piece.

PuzzlePiece Class



Attributes:

int pieceType : This value is determined by constructer parameters. There will be a set number of predetermined pieces that has a unique initial pieceArray. Some different piece types may create the same pieceArray after some rotations.

bool[][] pieceArray : This two dimensional array holds how the piece fills the slots. Works in the same way as mapArray attribute of Map class.

int locX : This value represents pieceArray's first column's location over the board.

int locY : This value represents pieceArray's first row's location over the board.

bool isSelectable : The selectable pieces can be moved over the board

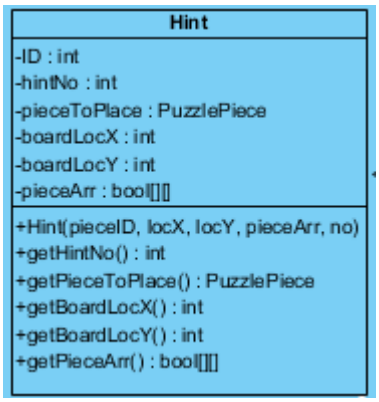
Constructors:

PuzzlePiece(int pieceType, bool isSelectable) : Constructs the puzzle piece

Methods:

4 rotate functions : These are simple mathematical algorithms to adjust pieceArray.

Hint Class



Attributes:

int hintNo : The order of hint in the corresponding puzzle

PuzzlePiece pieceToPlace : Pointer to the piece which useHint() method from Map class is going to place.

int boardLocX : Correct X location of the piece

int boardLocY : Correct Y location of the piece

bool[][] pieceArr : Correct form of the piece

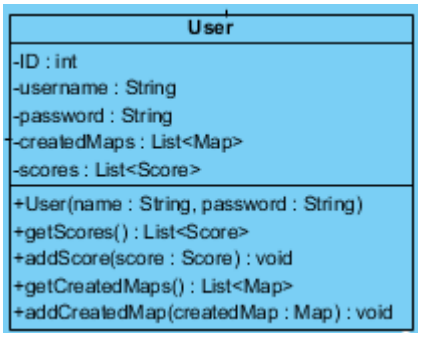
Constructors:

Hint(int pieceID, int locX, int locY, bool[][] pieceArr, int no) : Only used by solo puzzle generators.

Methods:

The methods will be used when useHint() function of the correlated puzzle is called.

User Class



Attributes:

String username : Player chosen String. Is family friendly.

String password : Player chosen String. Is not family friendly for most cases.

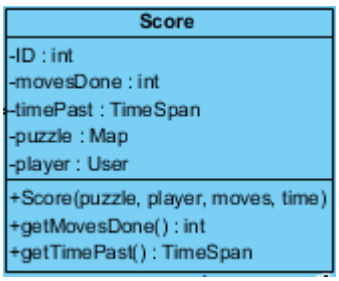
List<Map> createdMaps : A list of maps the player has made.

List<Score> scores : A list of scores the player has scored.

Constructors:

User(String name, String password) : A simple constructor for players.

Score Class



Attributes:

int movesDone : Number of moves done in the process of completing a puzzle.

TimeSpan timePast : Length of time passed in the process of completing a puzzle.

Map puzzle : Pointer to the corresponding puzzle

User player : Pointer to the corresponding user

Constructors:

Score(Map puzzle, User player, int moves, TimeSpan time) : Is called after finishing a puzzle