CS 319 - Object Oriented Software Engineering
Project Design Report 1

Katamino PC Game

Group 3C

Murat Tüver, 21602388

Sera Fırıncıoğlu, 21401803

Berk Yıldız, 21502040

Pegah Soltani, 21500559

Can Savcı, 21300803

1. Introduction

1.1 Purpose of The System

Katamino is a strategy and puzzle game which is used by the all ages. It is implemented differently to entertain its users more. There are two modes and many levels to make the Katamino more challenging. The purpose of the game is to fill the board with given pentominoes correctly to pass the next levels. It is planned to make Katamino user-friendly and run with high performance for its users.

1.2 Design Goals

One of the significant phase of the creating a game is designing. There are many specifications which are needed to be concerned in order to offer the game to its users in a better domain. Non-functional requirements of this project are going to be explained more in the design goals part.

1.2.1 Criteria

User - Friendliness:

Katamino is going to be design by concerning the mainly easy-to-use and entertaining features for its users. Since the game is mostly strategic and required to think about the movements or steps, user interface is going to be designed more basic. Pentominoes which are the object is going to be used mostly by the users are going to be designed similar to the real life shapes. There will be tutorial level for the users which are not familiar with the game before. Users can move the pentominoes just by dragging and dropping them to the board.

Performance:

The response which is going to come from the game engine is going to be less than 5 ms to make the game more efficient and fast for the users. It is going to be aimed that the memory usage will be less which it is going to be planned not more than 128 MB for the operation of the game since there is not many objects. In this way interaction between the game and players will be more fast and useful in terms of performance of the game.

Functionality:

Since the Katamino is a game based on strategy and focused on thinking mostly, there is not going to be many features to change the main structure of the game. In the case of advancing the game more, since the codes are well-commented, adding new features to the game will be easy for the developers. Datas for the leadership board and levels of the game are going to be saved in synchronized way in order to not to lose current datas of the game in the upcoming access for unexpected cases.

Trade-Offs:

Functionality and Usability

Since Katamino is played by the mostly child users, the user interface of the game will be easy and entertaining for the usage and management. There will not be a lot of functions in terms of usage. It will be designed levels to make the game more challenging but still usage will be easy.

Memory and Performance

The size of the game will be 30 MB which is for just one player. It is going to be updated with multiplayer version of the game. It will not be used a lot of objects for Katamino since this is a basic game. So memory usage of the time will not affect the performance of the game a lot. Our game will be designed for the desktops. For the laptop users, there might be needed more energy.

Cost and Portability

Since our application has educational purposes we do not intend to charge any of our customers. Therefore in sense of cost, our project will be cost free. Speaking of portability, any user that owns a computer can easily play Katamino on their desktops. Since we preferred to use Java as programming language, our game can run in any operating system which contains the JDK.

2. Software Architecture

2.1 Overview

In order to give more detailed information about the project, we decided to divide the architecture of Katamino in smaller independent subsystems. This action will not only reduce the complexity of our project it will also help us see smaller details and get rid of redundancies if needed.

2.2 Subsystem Decomposition

For understanding the purpose of each of the packages and classes designed by us, we decomposed each package one by one to their classes and explained the purpose of each attribute and operation. In this way we can understand our design pattern as well as discovering new potential requirements both functional and non-functional.

2.3 Architectural Styles

Our architectural style is based on the hierarchy of packages and classes. The whole game consists of three general packages. One handles the contents that should be showed on the screen (View package). Other package handles the connection between the user and the game (Controller package). Lastly we designed a package called the Model package which contains all the graphical objects that will be used to implement the game.

2.4 Hardware / Software Mapping

We will use Java programming language for implementing our game and use the packages accordingly. Speaking of the hardware requirements, one with a simple keyboard, mouse and a monitor will be able to enjoy the game. Katamino is a game that is thoroughly compatible with keyboard and mouse use. Therefore, the user will not need any specific and additional hardware installed to play the game.

2.5 Access Control and Security

In the sense of access control, anyone who has the application on their desktop would be able to access the game easily without having to create an account. However, the procedure differs when the player chooses to play the game in multiplayer mode. In that case user must have an account to sign in. For creating an account, the user should provide the program with basic information such as their name, surname, their favorite nickname and a password. In multiplayer mode, the user will connect to the network so he/she can play with their friends online, send messages and add/remove/block each other. Speaking of security, we are doing a research on how to keep our network secure and it will be one of the extra features that we are willing to provide. However, the

information of our users will not be shared anywhere since we are planning to use firebase which is provided by google so our main security concerns are already covered by Google.

2.6 Boundary Conditions

Initializing the Game

Initializing this game will not require any additional program or a special skill by the user. The game will come with a .jar file which is executable.

Terminating the Game

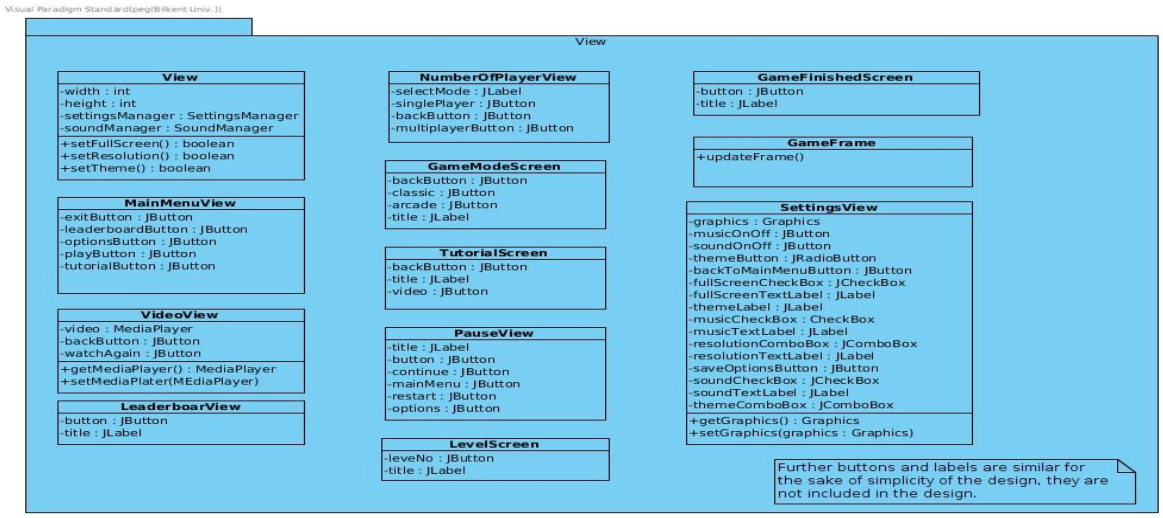
Terminating the game will be possible for the user anytime he/she wants. It is enough to click the quit button provided in the main menu. Also there will be a close button on top of each screen.

Error

There could be an error while getting the resource data. We will provide a function checking whether all resources are there or not. In that case, the user will be provided by a message box apologizing for the inconvenience and asking the user to re-run the program. Another error that could happen (which is more probable) will be in the multiplayer which will be related to connection to the database and the Internet. We will provide the user windows with messages to guide them.

3. Subsystem Services

3.1 User Interface



3.1.1 MainMenuView

This class is the first thing that the user sees when they open the application. So they can choose either one of the buttons provided as the attributes of this class. The user will be able to exit the game by clicking on the exitButton, see the leaderboard by clicking on the leaderboardButton, go to options page by clicking on the optionsButton, click the tutorialButton in order to see the basics of the game or directly play the game by clicking on the playButton. All the buttons provided in this class are from type JButton.

3.1.2 VideoView

This window opens when the user chooses to watch a video as a tutorial from the tutorial segment. The user will be able to watch the video that is provided by the set method that sets a video from MediaPlayer library provided by Java that solves a simple example of the puzzle. He/She can either click on the backButton to go back to the main menu window again, or rewatch the video by clicking on the watchAgain button.

3.1.3 LeaderboardView

This class provides a window for the user when the leaderboardButton is pressed in the main menu. This window basically shows the leaderboards. The user will be provided with a backButton button of type JButton which will go back to the main menu. There will be a label of type JLabel as well.

3.1.4 NumberOfPlayersView

In this window the user will be provided with two button, singlePlayerButton and multiPlayerButton of type JButton to choose either a single or multiplayer mode. There will also be a backButton button provided in case of going back to the main menu. Additionally, there will a selectMode label of type JLabel.

3.1.5 GameModeScreen

In this screen the player is provided three buttons of type JButton in order to choose the mode of the game, either classic or arcade. The third button, backButton will take the user to the main menu. We will use a label of JLabel for showing the title.

3.1.6 TutorialScreen

This screen will be displayed when the user clicks on the tutorialButton in the main menu. Tutorial screen will have a label of JLabel for displaying the title as well as a backButton for turning back. We will also provide the user with videoButton and link it to the VideoView class.

3.1.7 PauseView

This screen will be displayed when the user pauses the game at anytime. There will be a label of type JLabel for showing the title. Additionally there will be buttons of type JButton continue, for continuing the game from where it was paused. A restartButton for restarting the game, a optionsButton button for going to the options screen and finally a mainMenuButton button for going to the main menu.

3.1.8 LevelScreen

This screen will display the level number with a label of type JLabel along with other buttons indicating each level of type JButton.

3.1.9 GameFinishedScreen

This screen will be displayed once the user has finished the whole game. A title of JLabel including a “Congrats!” message will pop to the screen. There will also be a mainMenuButton of JButton type to redirect the user to the starting point again.

3.1.10 GameFrame

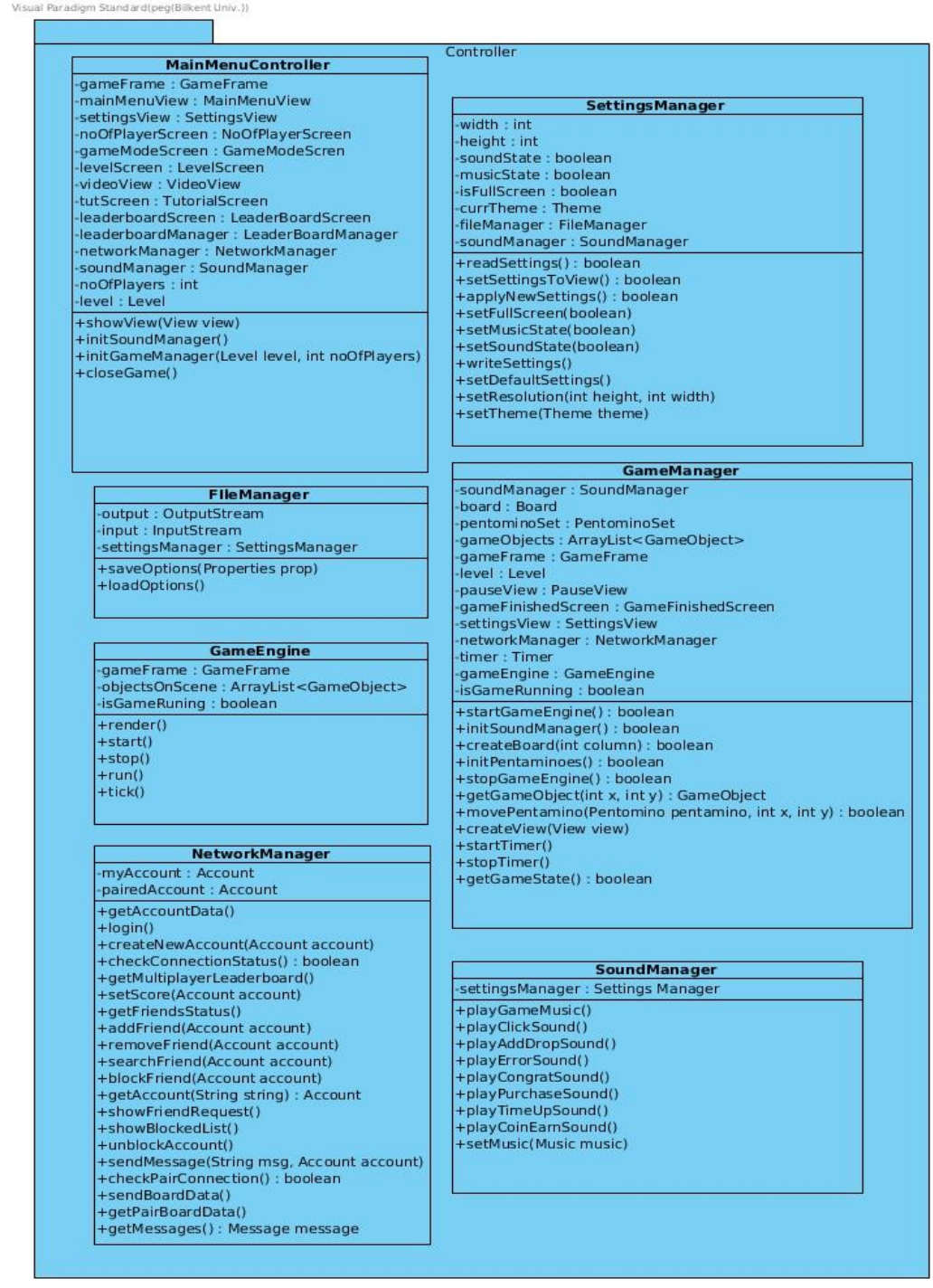
It will be the main frame and each screen will be implemented inside of it. The frame will be updated by the updateFrame() method of this class.

3.1.11 SettingsView

There will be an instance of Graphics to handle the graphics of the screen. Also we will provide several attributes such as: musicOnOff, for controlling the music, soundOnOff, for controlling the sounds, themeButton of type JRadioButton for choosing the desired theme, backToTheMainMenuButton for going back to the main menu, fullScreenCheckBox of type JCheckBox for controlling the screen scales, musicTextLabel which is basically a label of type JLabel, resolutionComboBox of type JComboBox for controlling the resolution, resolutionTextLabel of type JLabel, saveOptionsButton of type JButton for saving the set options by the user, soundCheckBox of type JCheckBox which checks whether the sound is on or off and finally soundTextLabel of type JLabel showing the sound status. We also provided

methods for getting the graphics getGraphics() which returns the graphics and a setGraphics(Graphics) to set each desired graphic.

3.2 Game Management



3.2.1 MainMenuController

Attributes:

private GameFrame gameFrame: This attribute stores a reference to a singleton GameFrame class in order to set necessary views to the frame.

private MainMenuView mainMenuView: This is the main screen of the menu where player navigates through game features. As soon as Katamino launched, this attribute is initialised and set to the gameFrame. It is also set to the gameFrame whenever user desires to go back to the main menu.

private SettingsView settingsView: Reference for the SettingsView class. It is initialized when user goes to the settings section and set to gameFrame.

private NoOfPlayerScreen noOfPlayerScreen: Reference to class which determines the type of the game whether it is single player or multiplayer. It screen is set to the gameFrame when user click the play button on the main menu.

private GameModeScreen gameModeScreen: Reference to a GameModeScreen instance. This screen is set to the GameFrame in order to decide the game mode user wants to play. It is initialised and set to the gameFrame when user selects the single player option.

private LevelScreen levelScreen: Reference for the LevelScreen class. It is represented to the user when user needs to select the level he/she wants to play.

private TutorialScreen tutScreen: Reference to generate tutorial screen from the main menu. It is launched when user enters to the tutorial section through main menu.

private VideoView videoView: This attributes holds a reference to a VideoView class in order to play a video to user in the tutorial section.

private LeaderboardScreen leaderboardScreen: Reference of the LeaderboardScreen class. This screen is set to the gameFrame when user wants to see the leaderboard.

private LeaderboardManager leaderboardManager: A controller class for getting data needed to show on leaderboardScreen.

private NetworkManager networkManager: Reference for the NetworkManager class it is used for multiplayer section in order to communicate with the database.

private SoundManager soundManager: Reference for the soundManager in order to play music in the main menu.

private int numberOfPlayers: Holds the number of players in order instantiate the proper game. (singleplayer or multiplayer)

private Level level: Level number which user wants to play. This data is taken from the user through a generated NoOfPlayerScreen.

Methods:

public void showView(View view): Initializes a view and sets it to the gameFrame. Parameter view can be MainMenuView, SettingsView, NoOfPlayerScreen, GameModeScreen, LevelScreen, VideoView, TutorialScreen or a LeaderboardScreen since View is a interface.

public void initSoundManager(): Initializes the SoundManager which is responsible for playing music and other sounds such as clicking sounds.

public void initGameManager(Level level, int NoOfPlayers): This method creates the actual game stage by using the attributes numberOfPlayers and level. In this method, GameEngine is initialized in order to render frames. Also GameManager is created to control the game and its flow.

public void closeGame(): This method terminates the software. It consists of few steps, first it dereferences the views then it manager classes are dereferenced. Finally, gameFrame is dereferenced which is essentially the view user sees.

Constructors:

private MainMenuController(): Default constructor for the MainMenuController class. It is called when the software first launched. It is private because this class is singleton.

3.2.2 SettingsManager

Attributes:

private int width: Holds the width of the frame. Its value determined through the settings tab of the main menu.

private int height: Holds the height information about the view classes. Value can be adjusted from the SettingsView.

private boolean soundState: Boolean variable that represents whether the game sounds is on/off.

private boolean musicState: Variable that indicates whether game should play music or not.

private boolean isFullScreen: Shows whether gameFrame is maximized to the full screen or not.

private Theme currTheme: Holds which theme is currently being used.

private FileManager fileManager: This reference is used for fetching settings from local properties file. It also updates the same properties file any in any changes made.

private SoundManager soundManager: This reference is used for informing the singleton SoundManager about the changes made to the settings.

Methods:

public boolean readSettings(): Fetches the previously set settings from the local properties file by using the fileManager instance. Returns true if there was no trouble while fetching the data and returns false when it encounters with a problem such as file being deleted or not having permission to reach to the file.

public void setSettingsToView(): Called when user saves new settings. It immediately updates the properties of view class being used and it calls a repaint in order to display new changes to the user.

public void applyNewSettings(): Calls set methods for each variable in order to update them with the new values determined by the user.

public void setFullScreen(boolean toFullScreen): Changes the value of the variable isFullScreen with the value received from the SettingsView.

public boolean writeSettings(): Saves changes made to the settings to the local properties file by using the fileManager. Returns whether task is achieved without a problem or not.

public void setDefaultSettings(): Changes the settings to the predetermined default values.

public void setResolution(int width, int height): Values of variables width and height are changed with respect to the new value obtained from the SettingsView.

public void setMusicState(boolean musicState): Turns music on/off by changing the variable musicState

public void setSoundState(boolean soundState): Turns sounds on/off by changing the variable soundState

public void setTheme(Theme themeToSet): Changes the theme being used with the parameter.

Constructors:

private SettingsManager(): Default constructor for the SettingsManager class. It will be private in order to follow singleton design which will avoid having multiple SettingsManager instances. This way there won't be any conflicts.

3.2.3 GameManager

Attributes:

private SoundManager soundManager: Instance of the SoundManager. It is required in order to play music and sounds in the game.

private Board board: An instance of a board object. It this where pentomino objects will be placed.

private PentominoSet pentominoSet: This attribute holds a unique set of pentominoes for each level. These sets are predefined by us according to the game catalog of the original board game. With set will be used is determined when the constructor of this class is called.

private ArrayList<GameObject> gameObjects: This arraylist holds references of the game objects generated throughout the game. It is useful for rendering these objects.

private GameFrame gameFrame: This reference is required in order to render the frame and get inputs from the user through mouse inputs.

private Level level: Demonstrates the level being played for that game.

private PauseView pauseView: This attribute is initialized whenever user pauses the game and it is set to the gameFrame.

private GameFinishedScreen gameFinishedScreen: Screen for end of the game. It is shown to the user when level is finished.

private SettingsView settingsView: View that is generated when user desires to go to the settings sections through pause menu.

private NetworkManager networkManager: NetworkManager instance for multiplayer games.

private Timer timer: This attribute is used for keeping track of the time in arcade mode.

private GameEngine gameEngine: Instance of the gameEngine for rendering objects.

private boolean isGameRunning: Represents whether the game is still being played or finished.

Methods:

public boolean startGameEngine(): Initializes and starts the gameEngine. Returns false if any error encountered.

public boolean initSoundManager(): Creates soundManager in order to play music and the sound of the game.

public void createBoard(int column): Creates a board with respect to the level of the game.

public void initPentominoes(): Creates pentominoes that will be used in the level generated. Which pentominoes to generate is decided by the help of the PentominoSet of that level.

public boolean stopGameEngine(): Stop the gameEngine. Returns true false if any error encountered.

public GameObject getGameObject(int x, int y): Returns the gameObject in coordinate given in the parameters.

public boolean movePentomino(Pentamino pentamino, int x , int y): Moves the pentomino in the parameter to the given coordinate. Returns true if it is successfully moved to the given position (position on the board is available).

public void startTimer(): Starts timer.

public void stopTimer(): Stops the timer.

Public boolean getGameState(): Returns a boolean indicating whether game is running or not.

Constructors:

Public GameManager(Level level, SettingsManager): Default constructor that generates the game with the given level and the settings.

3.2.4 FileManager

Attributes:

private OutputStream output: Attribute for writing to a local file.

private InputStream input: Attribute for reading data from a local properties.

private SettingsManager settingsManager: Reference of the settingsManager in order to set/get data to/from local properties file.

Methods:

public saveOptions(Properties prop): It takes a properties file as a paramater with contains the user prefered game settings.

public loadOptions(): This method reads local properties file and changes attributes in the SettingsManager class.

Constructors:

public FileManager(): Constructs a FileManager for writing/reading operations.

3.2.5 GameEngine

Attributes:

private GameFrame gameFrame: This reference is necessary in order to render the objects in the game.

private ArrayList<GameObject> objectsOnScene: This ArrayList holds references for all the GameObjects instantiated in the game. These references required for rendering.

private boolean isGameRunning: Value of this attribute is taken from GameManager class and denotes whether game is still running or not. It is required in order to stop the main loop of the game.

Methods:

private void render(): Goes through every item in the objectsOnScene and calls render method of each game object.

private void run(): This method contains the main loop of the game. It constantly calls render and tick methods while isGameRunning is true. It also counts the number of frames per second in order to get feedback about the performance.

private void start(): Initializes and starts threads afterwards, run method is called.

private void stop(): Stops threads to avoid unnecessary performance usage.

private void tick(): It method it called in the main loop of the game. It allows gameobjects to update their attribute. It is called before the render method in order to avoid rendering images before changing the attributes which would result in gaming lag.

Constructor:

Private GameEngine(): A private constructor for following singleton design pattern. By doing so, we have eliminated the chance of having multiple instances which lead to huge conflicts like calling render and tick more than required.

3.2.6 SoundManager

Attributes:

private SettingsManager settingsManager: This reference is needed in order to decide whether SoundManager should play sounds and music.

Methods:

public void playGameMusic(): Plays music for the game

public void playClickSound(): Generates clicking sound.

public void playAddDropSound(): Generates drag and drop sounds.

public void playErrorSound(): Generates sound for invalid actions.

public void playCongratSound(): Play the sound for finishing the level.

public void playPurchaseSound(): Plays sound effects for purchasing items.

public void playTimeUpSound(): Plays sound indicating time is up and game is finished for arcade mode.

public void playCoinEarnedSound(): Generates sound for earning coins.

public void setMusic(Music music): Changes the music currently being played.

Constructors:

private SoundManager(): Singleton design pattern is followed for this constructor.

3.2.7 NetworkManager

Attributes:

private Account myAccount: Holds informations about the users profile.

private Account pairedAccount: Holds the information about the player currently being played against.

Methods:

public Account getAccountData(): Returns the users profile informations.

public boolean login(String usrname, String passwd): Method for verifying the user.

public Account createNewAccount(Account account): Adds account in the parameter to the database.

public boolean checkConnectionStatus(): Checks whether internet connection is suitable for a multiplayer game.

public void getMultiplayerLeaderboard(): Takes the data related to the multiplayer leaderboard from the database.

public void setScore(Account account, int score): Updates users high score in the multiplayer leaderboard which is stored in database.

public void getFriendsStatus(): Updates friends status by fetching data from the database.

public boolean addFriend(Account account): Adds account in the parameter to friend list in the database. Returns true if request is sent successfully.

public boolean removeFriend(Account account): Removes the friend from the user's friend list.

public Account getAccount(String name): Return the account searched with the name.

public boolean blockFriend(Account account): Adds given account to the block list which prevents account in the parameter from reaching to the user.

public ArrayList<Account> showFriendRequest(): Lists the friend request received.

public ArrayList<Account> showBlockedList(): Returns list of accounts blocked.

public boolean unblockAccount(Account account): Removed the block from the account in the parameter.

public boolean sendMessage(String msg, Account account): Sends a message to the given account.

public boolean checkPairConnection(): Checks whether user is connected to her/his opponent through internet.

public boolean sendBoardData(Board board): Sends the status of players board to the other player.

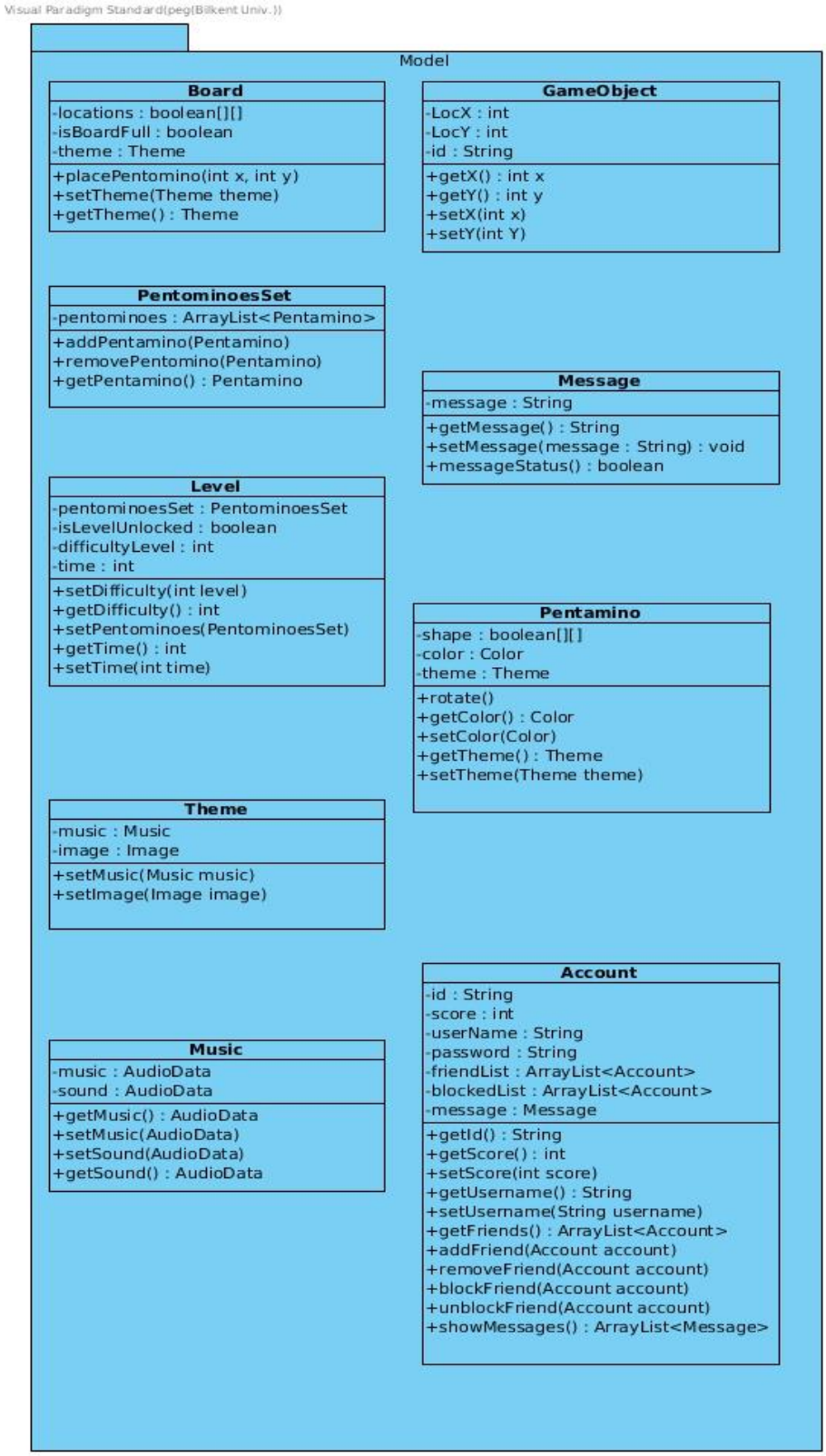
public Board getPairBoardData(): Gets the opponents board status from the database.

public ArrayList<Message> getMessages(): Takes the messages received from the database.

Contructors:

public NetworkManager(): Default constructor.

3.3 Model


Visual Paradigm Standard(peg(Bilkent Univ.))
Model
Board
-locations : boolean[][]
-isBoardFull : boolean
-theme : Theme
+placePentomino(int x, int y)
+setTheme(Theme theme)
+getTheme() : Theme
GameObject
-LocX : int
-LocY : int
-id : String
+getX() : int x
+getY() : int y
+setX(int x)
+setY(int Y)
PentominoesSet
-pentominoes : ArrayList<Pentamino>
+addPentamino(Pentamino)
+removePentomino(Pentamino)
+getPentamino() : Pentamino
Message
-message : String
+getMessage() : String
+setMessage(message : String) : void
+messageStatus() : boolean
Level
-pentominoesSet : PentominoesSet
-isLevelUnlocked : boolean
-difficultyLevel : int
-time : int
+setDifficulty(int level)
+getDifficulty() : int
+setPentominoes(PentominoesSet)
+getTime() : int
+setTime(int time)
Pentamino
-shape : boolean[][]
-color : Color
-theme : Theme
+rotate()
+getColor() : Color
+setColor(Color)
+getTheme() : Theme
+setTheme(Theme theme)
Theme
-music : Music
-image : Image
+setMusic(Music music)
+setImage(Image image)
Account
-id : String
-score : int
-userName : String
-password : String
-friendList : ArrayList<Account>
-blockedList : ArrayList<Account>
-message : Message
+getId() : String
+getScore() : int
+setScore(int score)
+getUsername() : String
+setUsername(String username)
+getFriends() : ArrayList<Account>
+addFriend(Account account)
+removeFriend(Account account)
+blockFriend(Account account)
+unblockFriend(Account account)
+showMessages() : ArrayList<Message>
Music
-music : AudioData
-sound : AudioData
+getMusic() : AudioData
+setMusic(AudioData)
+setSound(AudioData)
+getSound() : AudioData

3.3.1 Board Class

Attributes:

private boolean[][] locations: This attribute holds the availability of the squares on the board. For example if locations[1][2] is false, this means the square which is located in the coordinates (1,2) is full and player is not allowed to put any item on that square.

private boolean isBoardFull: This attribute is needed for understanding the fullness of the game board. If it is true, this means that there are no empty spaces remained and the game is finished.

private Theme theme: This attribute is used for holding the theme of the board. According the theme, the background pattern of the gameboard changes.

Methods:

public void placePentomino(int x, int y): This method places a pentomino on the board according to given coordinates.

public void setTheme(Theme theme): This method sets the theme of the board according to given theme. It changes the background pattern of the gameboard.

public Theme getTheme(): This method returns the theme of the gameboard.

3.4.1 GameObject Class

Attributes:

private int locX: This attribute is the place of a pentomino on x-axis.

private int locY: This attribute is the place of a pentomino on y-axis.

private String id: This attribute is the unique id of a pentomino.

Methods:

public int getX(): This method returns the x-axis coordinate of a pentomino.

public int getY(): This method returns the y-axis coordinate of a pentomino.

public void setX(int x): This method sets the x-axis coordinate of a pentomino.

public void setY(): This method sets the y-axis coordinate of a pentomino.

3.4.2 Pentominoes Set Class

Attributes:

private ArrayList<Pentomino> pentominoes: This attribute holds the list of some pentominoes which are defined in the game.

Methods:

public void addPentomino (Pentomino pentomino): This method adds a defined pentomino into the list.

public void removePentomino (Pentomino pentomino): This method removes the given pentomino which is given as parameter from the list.

public Pentomino getPentomino(): This method returns a pentomino from the list.

3.4.3 Message Class

Attributes:

private String message: This attribute is the message that player sends to an another player.

Methods:

public String getMessage(): This method returns the message that is written.

public void setMessage(String message): This method sets the written message.

Public boolean messageStatus(): This method returns true if the message is sent successfully and returns false if the sending is failed.

3.4.4 Level Class

Attributes:

private PentominoesSet pentominoesSet: This attribute holds the list of pentominoes which is given to the player according to his/her level.

private boolean isLevelUnlocked: This attribute is used for understanding the availability of a level for the player.

private int difficultyLevel: This attribute states the levels.

private int time: This attribute is the given time to player for finishing a particular puzzle in arcade mode.

Methods:

public void setDifficulty (int level): This method sets the level.

public int getDifficulty(): This method returns the level.

public void setPentominoes (PentominoesSet pentominoesSet): This method sets the pentominoes which will be given to player in the levels.

public int getTime(): This method gets the time which is given to player for finishing a particular puzzle in arcade mode.

public int setTime (int time): This method sets the time which is given to player for finishing a particular puzzle. Different time is given to player in each level according to level's difficulty.

3.4.5 Pentomino Class

Attributes:

private boolean[][] shape: This attribute holds the shape information of a pentomino. For example if shape[0][0] and shape [1][0] is true, it means that we have a shape which consists of two collateral squares.

private Color color: This attribute is the color of a pentomino.

private Theme theme: This attribute is the theme of a pentomino. According to theme, the color on the pentomino changes.

Methods:

public void rotate(): This method rotates a pentomino.

public Color getColor(): This method returns the color of a pentomino.

public void setColor(Color color): This method sets color to a pentomino.

public Theme getTheme(): This method gets the theme of a pentomino.

public void setTheme (Theme theme): This method sets a theme to a pentomino.

3.4.6 Theme Class

Attributes:

private Music music: This attribute is the music that plays on background for the particular theme.

private Image image: This attribute is the image that used for particular theme.

Methods:

private void setMusic (Music music): This method sets the background music to the theme.

private void setImage(Image image): This method sets image to the theme.

3.4.7 Account Class

Attributes:

private String id: This attribute is the unique Id of the accounts.

private int score: This attribute is the score of an account that the user collected.

private String userName: Unique username which is defined by the user.

private String password: Account password for log in which is defined by user.

private ArrayList<Account> friendList: This attribute holds the friends of the user which consists of other users of the game.

private ArrayList<Account> blockedList: This attribute holds the users which are blocked by the account owner.

private Message message: This attribute is the message that the user sends or receives.

Methods:

public String getId(): This method returns the unique Id of an account.

public int getScore(): This method returns the collected score of an user.

public void setScore (int score): This method sets the score of a user.

public String getUsername(): This method returns the defined unique username of the account.

public void setUsername (String username): This method sets the unique username of the account.

public ArrayList<Account> getFriends(): This method returns the friendlist of an user.

public void addFriend (Account account): This method adds another user as friend to the friendlist of an account.

public void removeFriend (Account account): This method removes a user from the friendlist of an account.

public void blockFriend (Account account): This method blocks the user which is selected by account owner.

public void unblockFriend (Account account): This method removes the block of a blocked user.

public ArrayList<Message> showMessage(): This method displays the messages of the account.

3.4.8 Music Class

Attributes:

private AudioData music: This attribute is the music that plays on background.

private AudioData sound: This attribute is the sounds of the game.

Methods:

public AudioData getMusic(): This method returns the music that plays on background.

public void setMusic (AudioData music): This method sets the music which will play on background.

public void setSound (AudioData sound): This method sets sounds of the game.

public AudioData getSound(): This method returns the particular sound.

4. Low Level Design

4.1 Packages

In Katamino, it is used two kind of packages. One of the package is defined by the developers and second package is external package which is offered by Java called JavaFX.

4.1.1 Packages Introduced By Developers

4.1.1.1 Model

This package will be used for designing graphical objects such as board and pentominoes and their features.

4.1.1.2 Controller

This package will be used to connect the user with the game by controlling the whole game features. Singleton design pattern is used in some constructors to prevent conflicts as well.

4.1.1.3 View

This package will be used to display the screens of required classes such as MainMenu, Settings, Tutorial, GameFrame.

4.2.2 External Library Packages

4.2.2.1 java.awt

This package will be used for creating user interface and painting graphics and images which are such as keyEventDispatcher, paint and shape interfaces.

4.2.2.2 java.nio

This package will be used for buffers which are containers for the data.

4.2.2.3 java.util

This package will be used for frameworks, time facilities, array list and serialization.

4.2.2.4 java.io

This package will be used for input and output through data streams and file system.

4.2.2.5 javax.swing

This package will be used for lightweight components such as menu, menuElements, windows, desktopManager which work the same on all platforms.