CS 319 - Object Oriented Software Engineering

Project Design Report

Iteration 2

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
- 1.2 Design Goals
  - 1.2.1 Criteria
    - 1.2.1.1 User-Friendliness
    - 1.2.1.2 Performance
    - 1.2.1.3 Functionality
    - 1.2.1.4 Ease of Learning
    - 1.2.1.5 Ease of Use
  - 1.2.2 Trade-Offs
    - 1.2.2.1 Functionality vs. Usability
    - 1.2.2.2 Space vs. Speed
    - 1.2.2.3 Cost vs. Portability
    - 1.2.2.4 Delivery Time vs. Performance

2. Software Architecture
- 2.1 Subsystem Decomposition
- 2.2 Architectural Styles
- 2.3 Hardware / Software Mapping
- 2.4 Persistent Data Management
- 2.5 Access Control and Security
- 2.6 Boundary Conditions
  - 2.6.1 Initializing the Game
  - 2.6.2 Terminating the Game
  - 2.6.3 Error

3. Subsystem Services
- 3.1 UserInterface Subsystem
- 3.2 MenuManagement Subsystem
- 3.3 GameManagement Subsystem
- 3.4 FileManagement Subsystem
- 3.5 NetworkManagement Subsystem

4. Low-level Design
- 4.1 Object Design
- 4.2 Design Decisions – Design Pattern
  - 4.2.1 Façade Design Pattern
  - 4.2.2 Singleton Design Pattern
- 4.3 Packages
  - 4.3.1 Packages Introduced By Developers
    - 4.3.1.1 user_interface
    - 4.3.1.2 menu_management
    - 4.3.1.3 game_management
    - 4.3.1.4 file_management
    - 4.3.1.5 network_management

4.3.2 External Library Packages
4.3.2.1 java.awt
4.3.2.2 java.nio
4.3.2.3 java.util
4.3.2.4 java.io
4.3.2.5 java.swing
4.4 Class Interfaces
4.4.1 MainMenuView
4.4.2 VideoView
4.4.3 LeaderboardView
4.4.4 NumberOfPlayerView
4.4.5 GameModeScreen
4.4.6 TutorialScreen
4.4.7 PauseView
4.4.8 LevelScreen
4.4.9 GameFinishedScreen
4.4.10 GameFrame
4.4.11 SettingsView
4.4.12 MainMenuController
4.4.13 SettingsManager
4.4.14 GameManager
4.4.15 FileManager
4.4.16 GameEngine
4.4.17 SoundManager
4.4.18 NetworkManager
4.4.19 Board
4.4.20 GameObject
4.4.21 PentominoesSet
4.4.22 Message
4.4.23 Level
4.4.24 Pentomino
4.4.25 Theme
4.4.26 Account
4.4.27 Music
5. References

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. Also multiplayer will add competition factor to this challenging game.

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

1.2.1.1 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.

1.2.1.2 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.

1.2.1.3 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.

1.2.1.4 Ease of Learning:

Another goal of our design is to shorten the time required for new users to learn the game. In order to achieve this we come up with two main ideas. One is showing a tutorial video which covers basics of the game and the other one is letting the player play a sample level. In this sample level, solution is giving by the game itself and it tells player to where to put given pieces and how to put them like now press 'r' button to rotate the piece etc. This will allow user to grasp the mechanics and the logic of the game easily. Also, this way user can get into the game faster.

1.2.1.5 Ease of Use:

Board game version of Katamino is based on taking pieces and putting them on board. We adapted drag and drop method for user interactions with the game. Other option we considered was using arrow keys but we thought that drag and drop is more similar to real life experience of Katamino. It is also more instinctive. Even without watching tutorial videos or playing sample levels, players should play the game instinctively.

1.2.2 Trade-Offs:

1.2.2.1 Functionality vs. Usability

The method which is going to be used to play Katamino is so easy and understandable which is drag and drop method that is mostly used a lot of games. It is preferred mostly usability over functionality since the game is based on educational purposes. Tutorial screen is also implemented to play and understand the game easily.

1.2.2.2 Space vs. Speed

Katamino is going to use 30MB of space for single player. 30MB memory usage is not going to be disadvantaged when the speed of the game is being considered.

1.2.2.3 Cost vs. 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.

1.2.2.4 Delivery Time vs. Performance

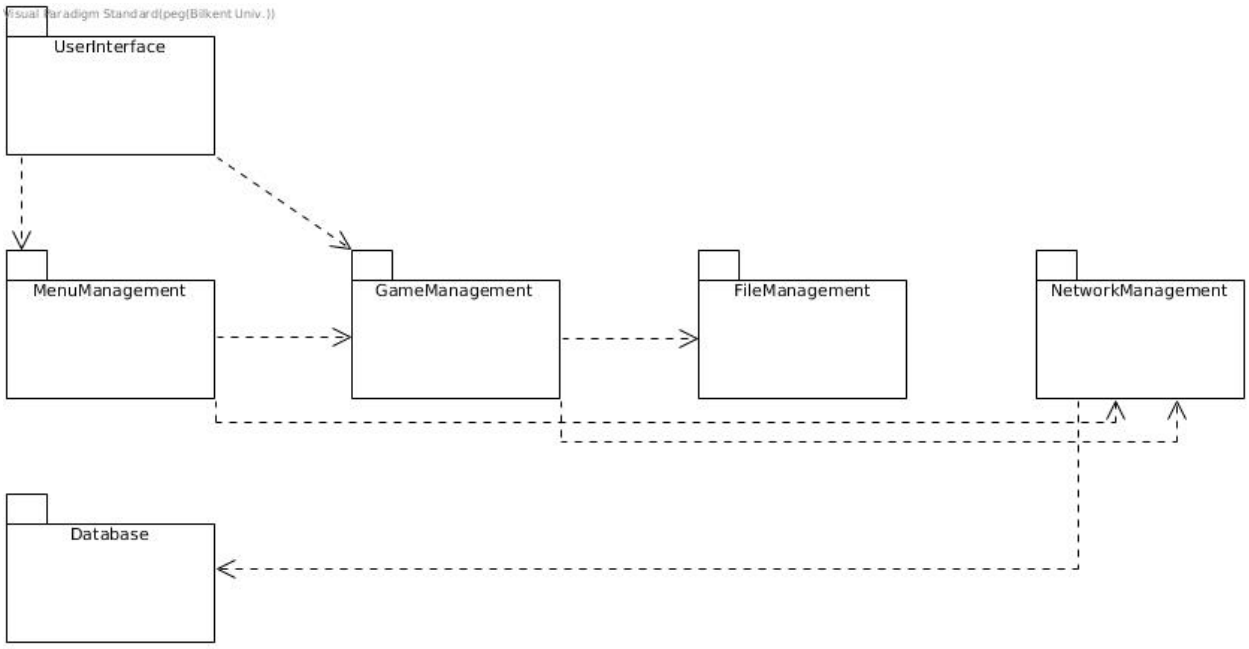
Katamino is implemented in Java. Java is preferred in this project since it is more functional and easier in terms of usage of controlling the memory rather than other programming languages such as C++, etc. Since there will not be any memory managing problems which causes to delay the delivery time.

2. Software Architecture

2.1 Subsystem Decomposition

In order to reduce the complexity of the project and let us work on different parts at the same time, we decided to divide the architecture of Katamino in smaller independent subsystems. While doing so, we tried to achieve high coherence by grouping classes with similar task into same subsystems. We ended up with five subsystems and one database which are highly coherent. On the coupling side of the architecture, we tried to achieve low coupling. We used 3 layer architectural style to help us organize, which subsystem should communicate with which subsystem. We accomplished low coupling to some extent. We can say that they are mostly independent from each other in the sense of change. However, they depend on each other to fulfill some of their tasks. We expect our subsystem decomposition to help us to trace errors easily. It also gives us opportunity of reusability since we can use one of our subsystems in a possible future project.

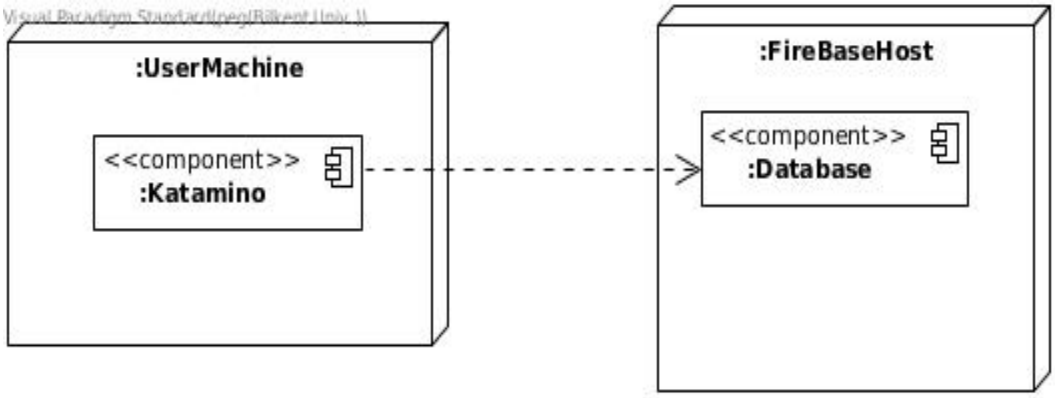
We also divided our system into three layers, the first layer which includes subsystems involving the user and the game interfaces, this layer is where user interacts with the system. This layer communicates with the second layer in order to update frames. Second layer is the application layer. It includes the subsystems that involve the application of the game. These subsystems are used to manage the game logic as well as maintaining the communication with the database which is the third layer of our subsystem decomposition. Subsystems in this layer are the main control units of our game. They include both manager classes and model classes. Third and the last layer include the storage layer in which our project only has the database regarding that. It is a Firebase Database and it is responsible for multiplayer games. The subsystems in each layer are connected to each other according to the logic of the game. The image below shows how we divided our system into subsystems.



2.2 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.

2.3 Hardware / Software Mapping



There are 2 types of hardware in terms of machines in Katamino. First of these two is the userMachine. It is the machine Katamino is played on. Operating system is not important as long as it has Java's Development Kit and Runtime Enviroment are installed. Other one is the database which is actually Google's Firebase Realtime Database. We decided to use Firebase because it is easy to learn and most importantly reliable which is important when multiplayer games are considered. Other than machines used in our game, there are also input devices being used in our game. They are basic components of each personal computer like mouse and keyboard. Therefore, the user will not need any specific and additional hardware installed to play the game.

2.4 Persistent Data Management

Katamino will have both file system and database to store data. The game options such as theme, sound, etc. will be stored in a formatted files in the local storage. Options is going to be set default at the beginning of game. Firebase will store users account related informations such as usernames, passwords, friends and so on. Database also stores world rankings and scores of each player registered their accounts to multiplayer Katamino. These ranking will be represented in a leaderboard with the data fetched from the database. Other purpose of the Firebase is creating multiplayer related data flow. In multiplayer games users need to see each other's board and it is only possible through a real time database which is why we choose to use database.

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

2.6.1 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.

2.6.2 Terminating the Game

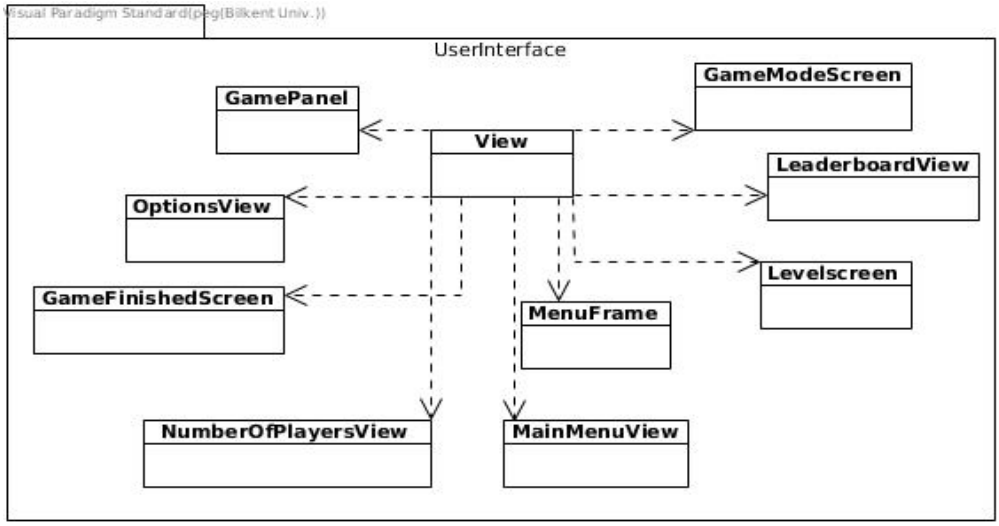
Terminating the game will be possible for the user anytime he/she wants. It is enough to click the quit button provided in the main menu. Also there will be a close button on top of each screen.

2.6.3 Error

There could be an error while getting the resource data. We will save the resource data and 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. We will save the resource data through database.

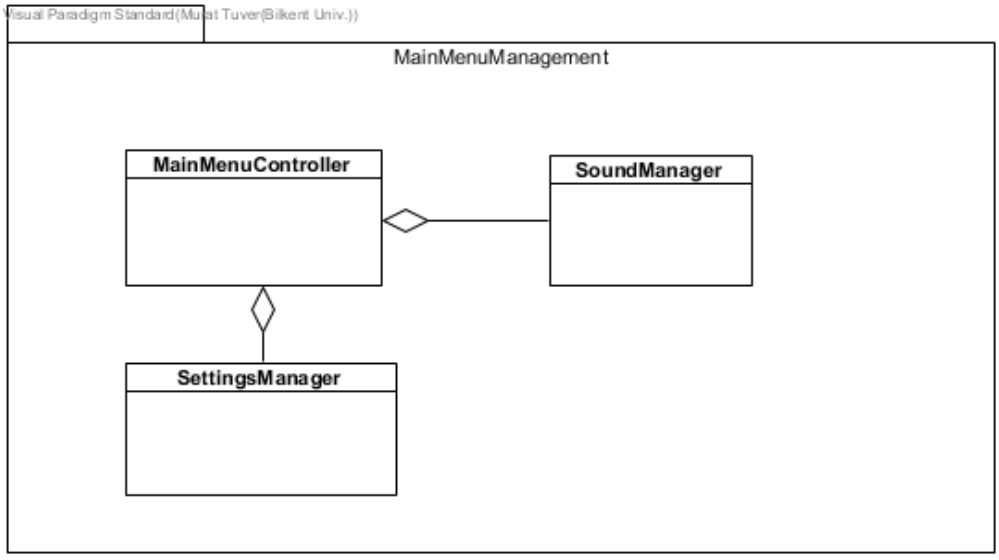
3. Subsystem Services

3.1 UserInterface Subsystem



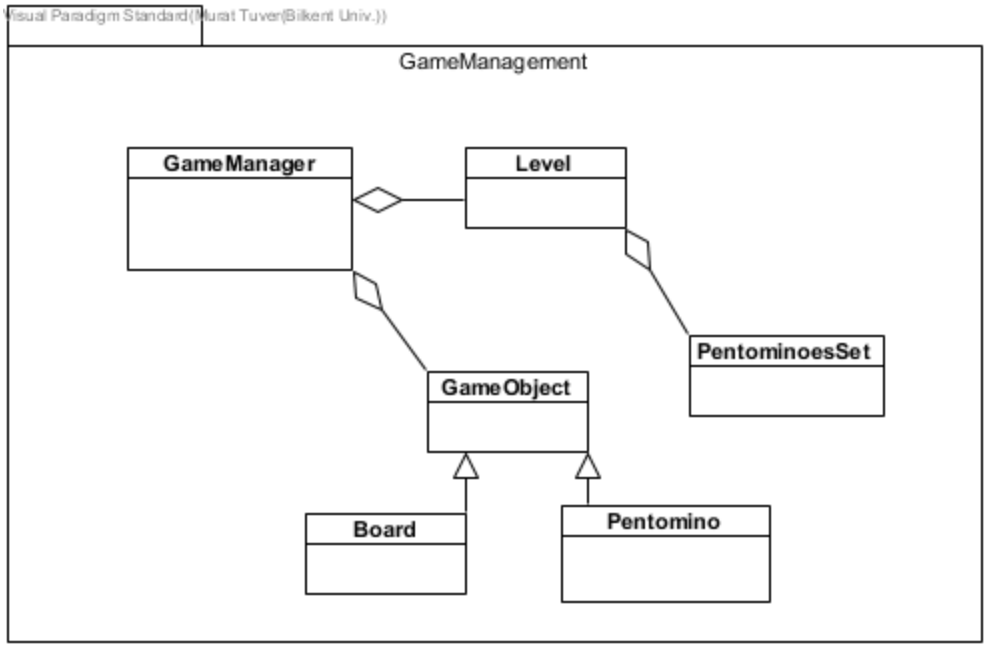
First subsystem of our system decomposition is the UserInterface subsystem. Purpose of this subsystem is to collect all the views in a package so that we can develop GUI part of our software while working on the game logic at the same time. This subsystem is the only subsystem in the first layer which is named presentation layer. It also gives us flexibility during the development so that we can change views at any point of our development without concerning about the logic. Moreover, it is easier to work on game themes when all the user interfaces are in a same subsystem.

3.2 MenuManagement Subsystem



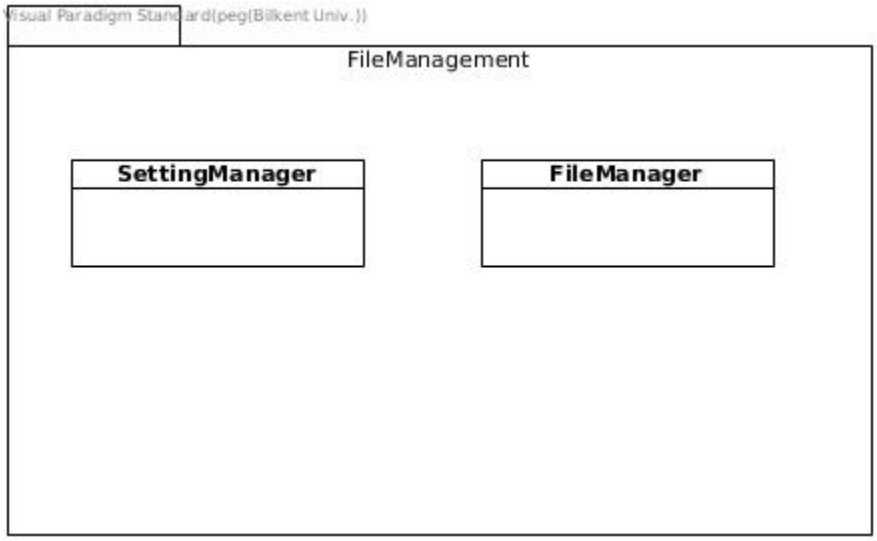
This subsystem is responsible from all the menu related actions like going to a menu from another one. The most important role of this subsystem is to determine what type of a game user wants to play by showing necessary screens to the user and generating the right type of game by communicating with the GameManagement subsystem. It is also in connection with the NetworkManagement subsystem in order to show leaderboard when prompted by the user.

3.3 GameManagement Subsystem



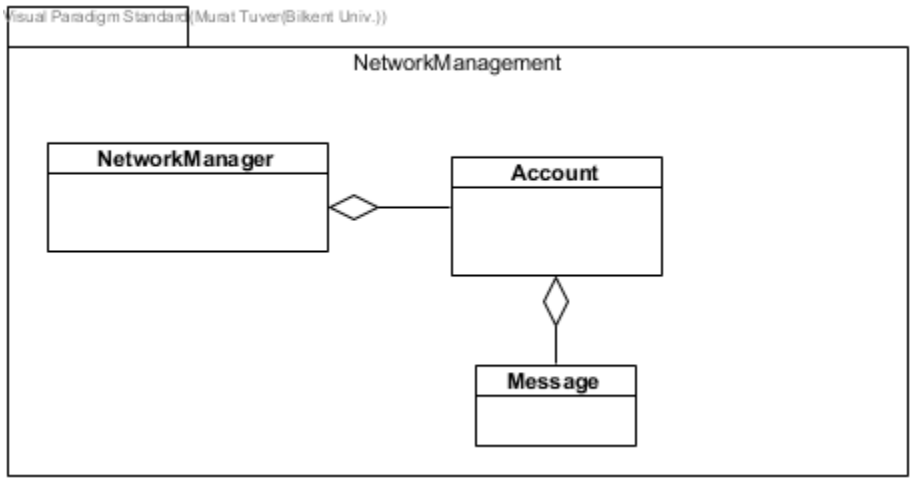
GameManagement Subsystem contains all the decision making mechanisms of gaming phase. Most of the model type classes of MVC design pattern is located in this subsystem. It also includes certain controller classes to regulate the game. It gives and takes data from NetworkManagement Subsystem in order to keep the sync of multiplayer games. Also it works with MenuManagement Subsystem to get back to the main menu after game is finished.

3.4 FileManagement Subsystem



FileManagement Subsystem is rather simple. It saves and loads settings of the user preferences, keeps track of which levels are unlocked and stores high scores of the arcade mode. It also supplies sound files and image files to the GameManagement subsystem. This way all of the file reading and writing operations are collected under one subsystem.

3.5 NetworkManagement Subsystem

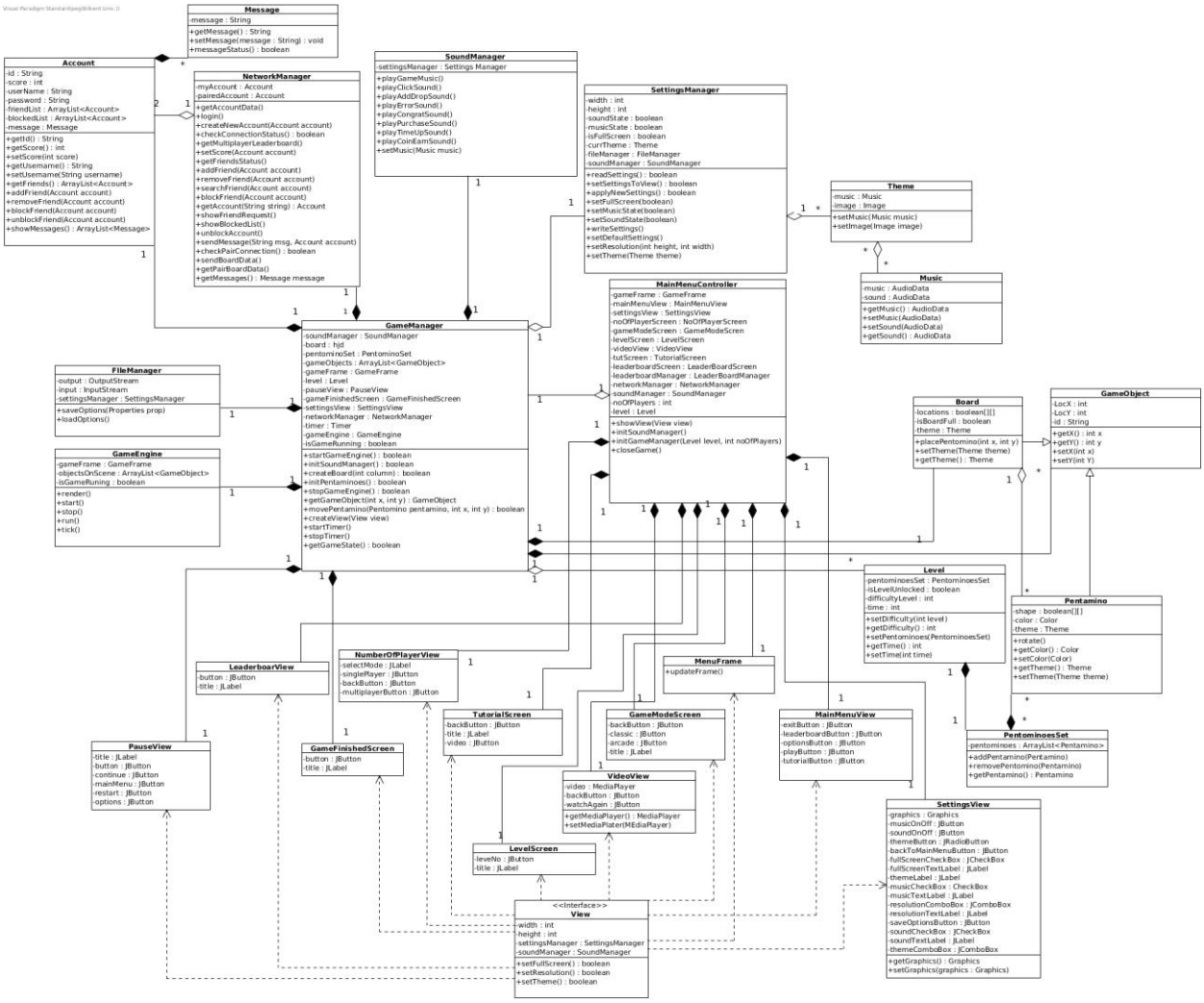


Main purpose of this subsystem is to communicate with our database. In the sake of Façade design pattern, instead of other components of our system, this subsystem fetches and updates data being stored in the database. It is in deep relation with GameManagement subsystem when a multiplayer game is being played. It constantly informs database about the changes user makes to his/her board and gets

data related to opponent’s board’s status. Other than in-game actions, it presents world rankings, multiplayer leaderboard, to the MenuManagement subsystem. It also allows user to send messages to each other, add each other as friends and other features that requires internet connection.

4. Low-level Design

4.1 Object Design



4.2 Design Decisions - Design Pattern

4.2.1 Facade Design Pattern

Since Facade design pattern is easy to apply and reduce the complexity of the subsystems for the clients, we decided to use as one of our design patterns. Facade design pattern is used to simplify the interactions between the subsystems. It is used for the GameManagement and NetworkManager subsystems in the application layer. For simplification, it is created GameManagement subsystem for the subclasses of it to make the interaction between GameManagement subclasses and NetworkManager. Secondly facade design pattern is used for MenuManagement, NetworkManagement and

Database subsystems. For the interactions between MenuManagement and Database, NetworkManagement subsystem is used to make the accessibility between subclasses easier.

4.2.2 Singleton Design Pattern

For some classes, it is needed to create just one instance of that class for the whole game. So singleton design pattern is preferred for these classes. Singleton design pattern is used for GameManager, GameEngine and SoundManager. Since these classes are initialized at the beginning of the game and there is not going to be another instance of these classes, singleton pattern is preferred for them. Creating more than one instance of these classes cause problems which effect to control game so singleton design pattern is used.

4.3 Packages

Packaging is done accordingly to our subsystem decomposition. This way our developers can work on different packages without worrying about interrupting other developers working on other topics.

4.3.1 Packages Introduced By Developers

4.3.1.1 user_interface

This package includes classes which either extend JFrame or JPanel. These classes represent graphical user interfaces to players so that they can interact with the system. These classes also have action listeners in order to get input from users devices. Some of these are simple screens used in menu in order to receive input from the player. On the other hand, classes like GameEngine are more complicated which constantly paint game objects present in the game.

4.3.1.2 menu_management

This package includes classes related to actions performed in the menu. It coordinates views in user_interface package that are related to menus.

4.3.1.3 game_management

This package will be used to connect the user with the game by controlling the whole game features like placing pentominoes on board, rotating pieces etc. Implementations of entity objects are also made in this package. Singleton design pattern is used in some controller classes like GameEngine and GameManager. Decision behind using such pattern comes from possible conflicts that may happen if more than one instance of these classes is instantiated. For example, two GameEngine instances would lead to over usage of computer resources like memory and cpu.

More importantly, instance of two GameManager classes would create conflicts in game logic.

4.3.1.4 file_management

Function of the classes in this package is to read and write files from/to hard drive. These are required in order to store which settings user prefers, which levels are unlocked in single player game mode and so on.

4.3.1.5 network_management

Classes in these package are used in features that require connection to the database. Board states, user account informations, messages users send and receive are all regulated by this class.

4.3.2 External Library Packages

4.3.2.1 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.

4.3.2.2 java.nio

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

4.3.2.3 java.util

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

4.3.2.4 java.io

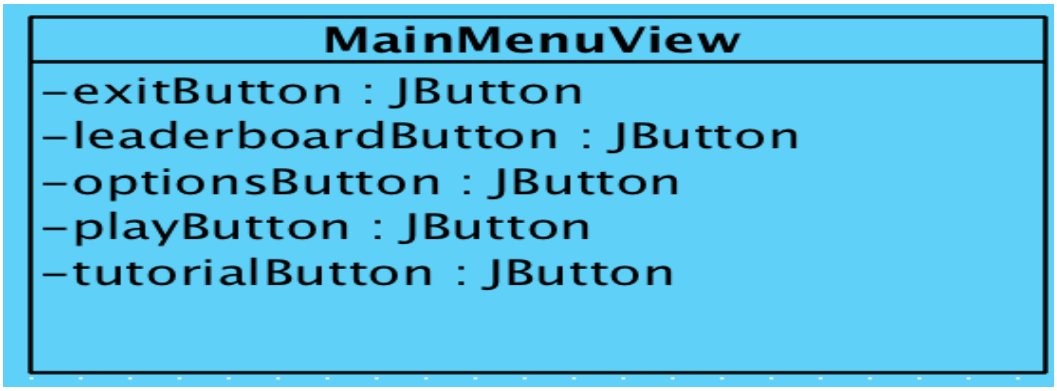
This package will be used for input and output through data streams and file system.

4.3.2.5 javax.swing

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

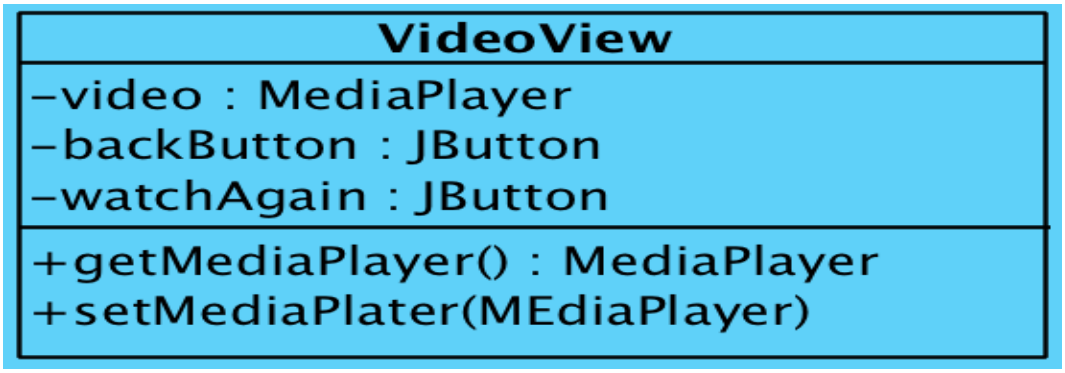
4.4 Class Interfaces

4.4.1 MainMenuView



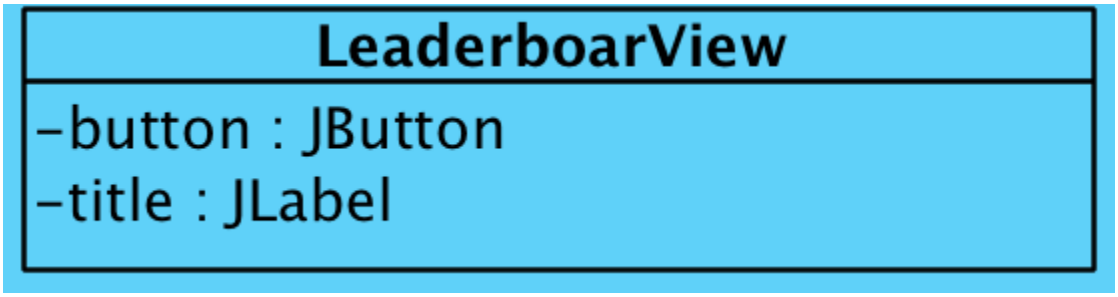
This class is the first thing that the user sees when they open the application. So they can choose either one of the buttons provided as the attributes of this class. The user will be able to exit the game by clicking on the exitButton, see the leaderboard by clicking on the leaderboardButton, go to 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.

4.4.2 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.

4.4.3 LeaderboardView

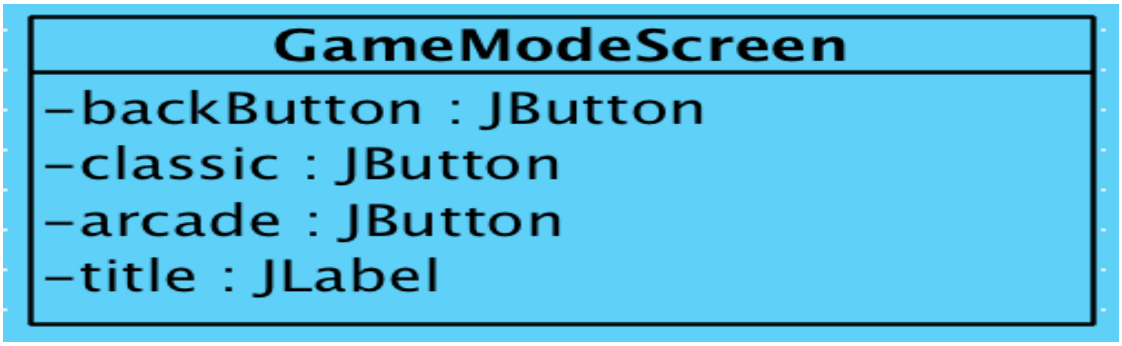


This class provides a window for the user when the leaderboardButton is pressed in the main menu. This window basically shows the leaderboards. The user will be provided with a backButton button of type JButton which will go back to the main menu. There will be a label of type JLabel as well.

4.4.4 NumberOfPlayersView

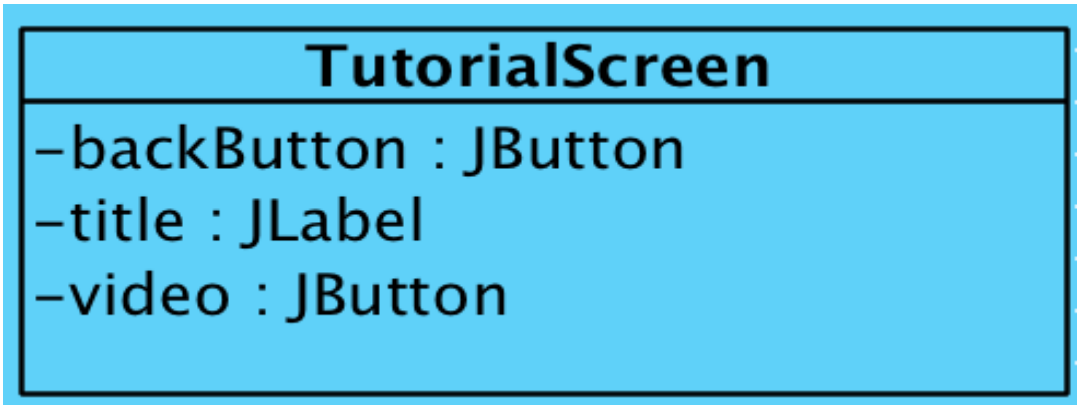
In this window the user will be provided with two button, singlePlayerButton and multiPlayerButton of type JButton to choose either a single or multiplayer mode. There will also be a backButton button provided in case of going back to the main menu. Additionally, there will a selectMode label of type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

4.4.5 GameModeScreen



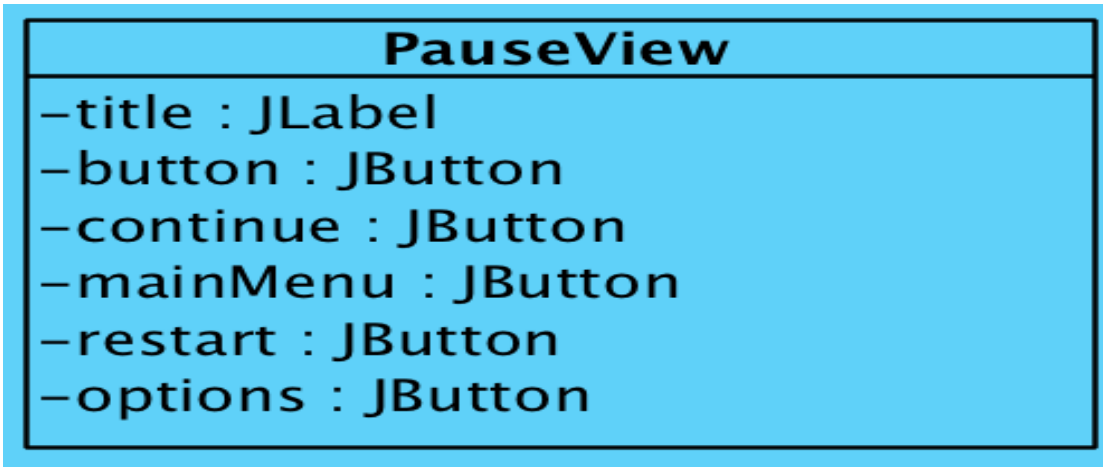
In this screen the player is provided three buttons of type JButton in order to choose the mode of the game, either classic or arcade. The third button, backButton will take the user to the main menu. We will use a label of JLabel for showing the title.

4.4.6 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.

4.4.7 PauseView



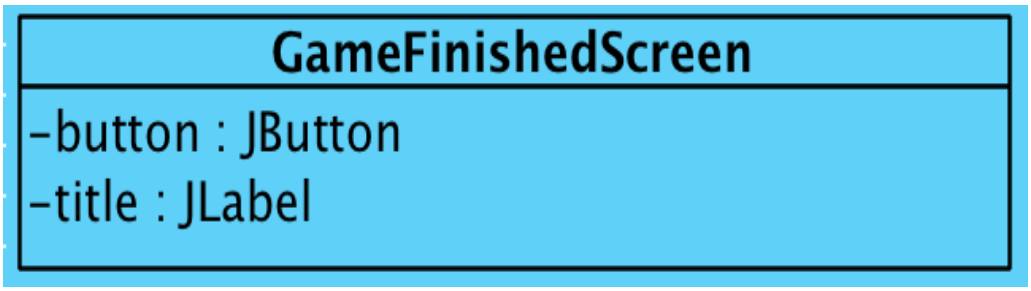
This screen will be displayed when the user pauses the game at anytime. There will be a label of type JLabel for showing the title. Additionally there will be buttons of type JButton continue, for continuing the game from where it was paused. A restartButton for restarting the game, a optionsButton button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

to the options screen and finally a mainMenuButton button for going to the main menu.

4.4.8 LevelScreen

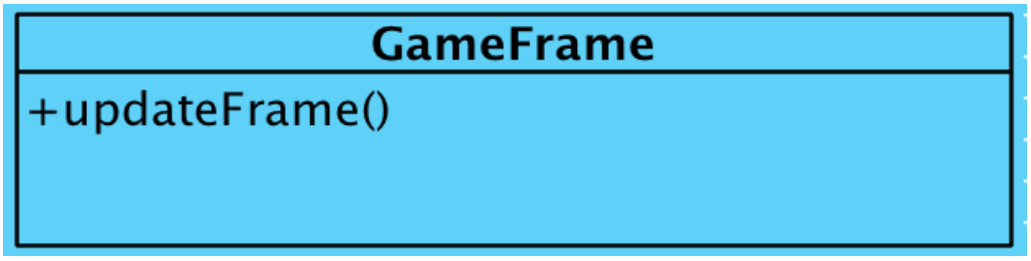
This screen will display the level number with a label of type JLabel along with other buttons indicating each level of type JButton.

4.4.9 GameFinishedScreen



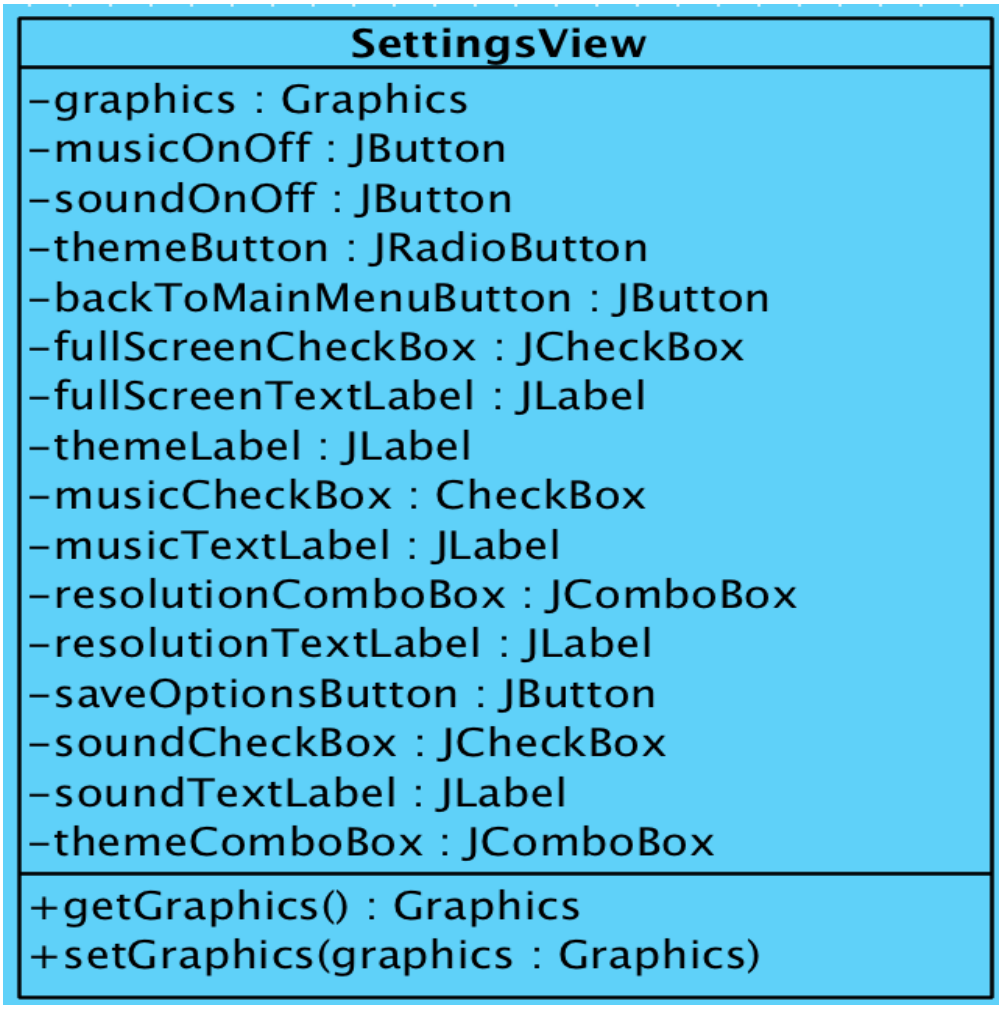
This screen will be displayed once the user has finished the whole game. A title of JLabel including a “Congrats!” message will pop to the screen. There will also be a mainMenuButton of JButton type to redirect the user to the starting point again.

4.4.10 GameFrame



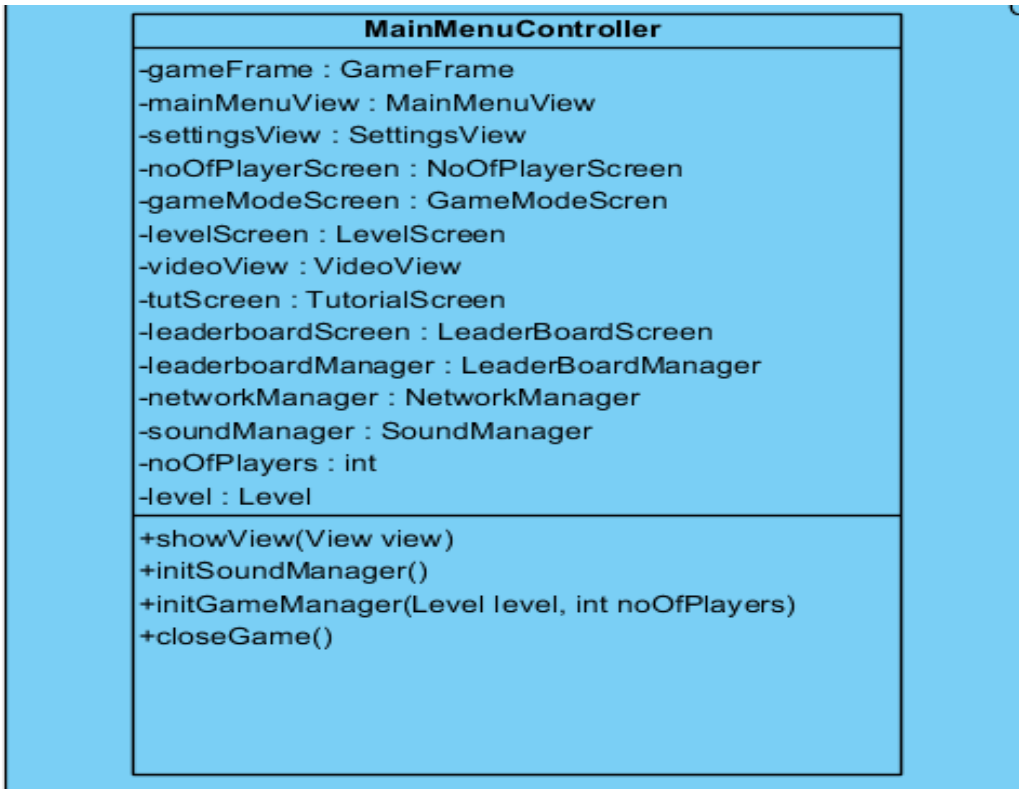
It will be the main frame and each screen will be implemented inside of it. The frame will be updated by the updateFrame() method of this class.

4.4.11 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.

4.4.12 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

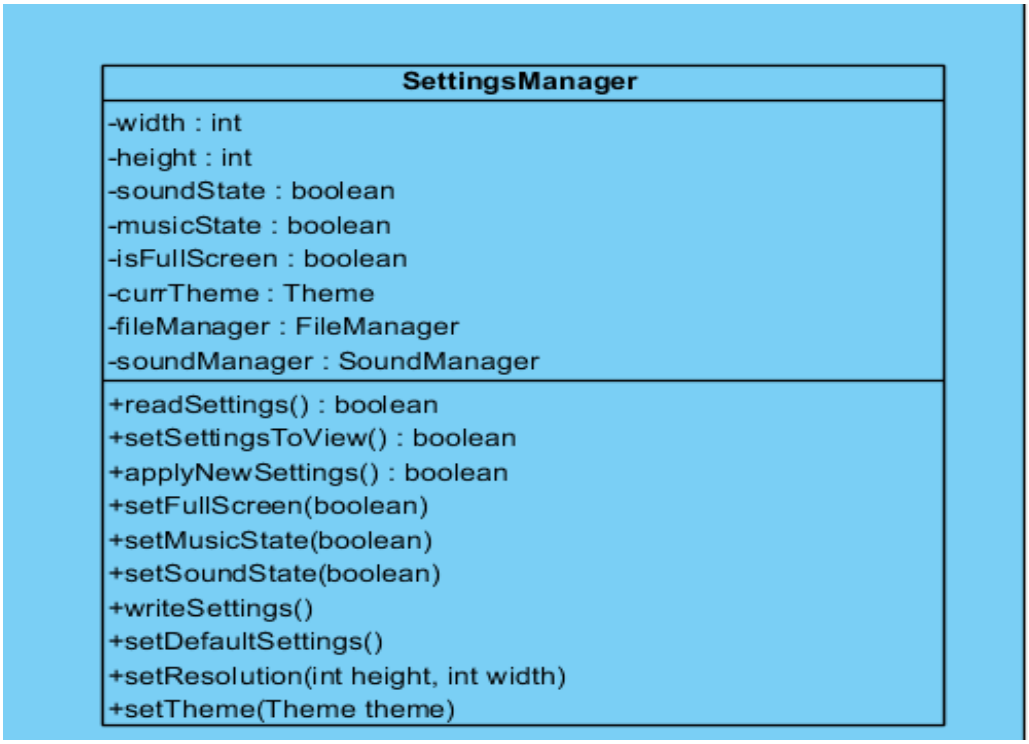
level. In this method, GameEngine is initialized in order to render frames. Also GameManager is created to control the game and its flow.

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.

4.4.13 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

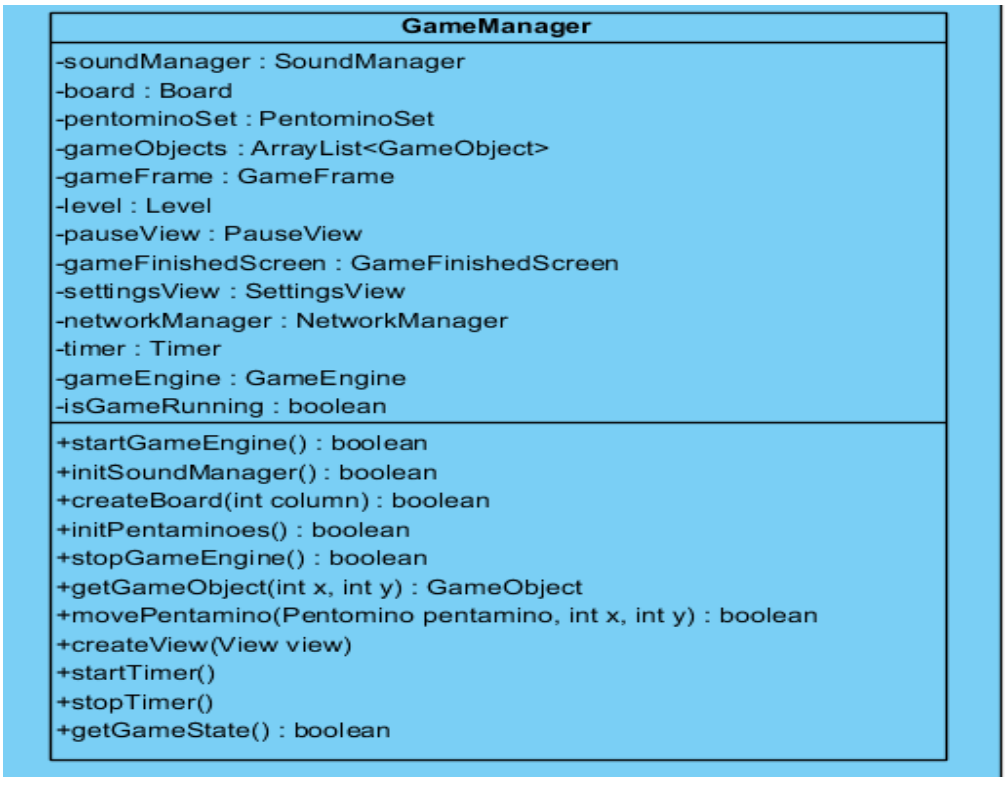
public void setSoundState(boolean soundState): Turns sounds on/off by changing the variable soundState

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.

4.4.14 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.

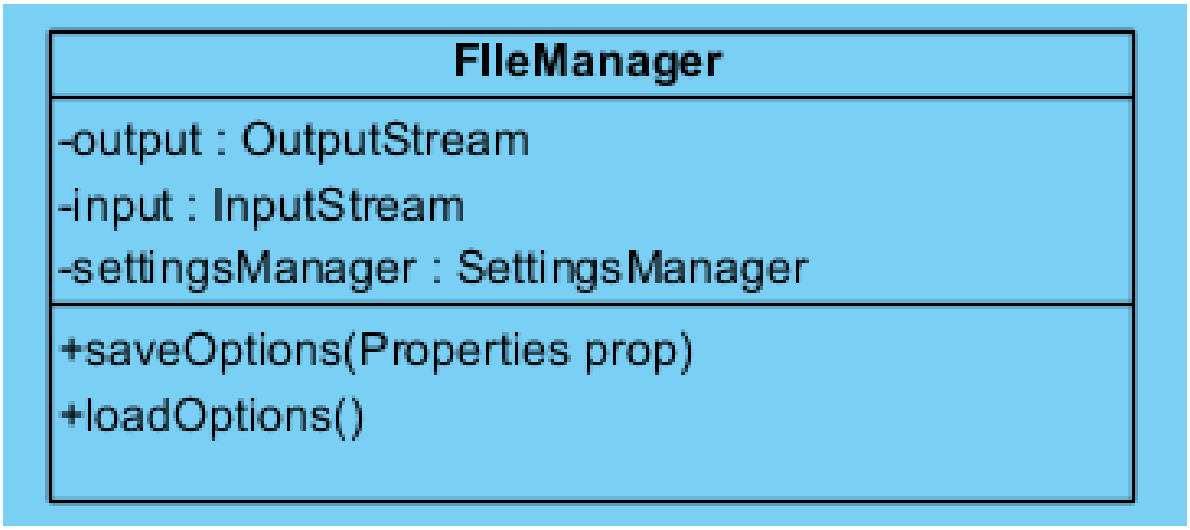
public void stopTimer(): Stops the timer.

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.

4.4.15 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:

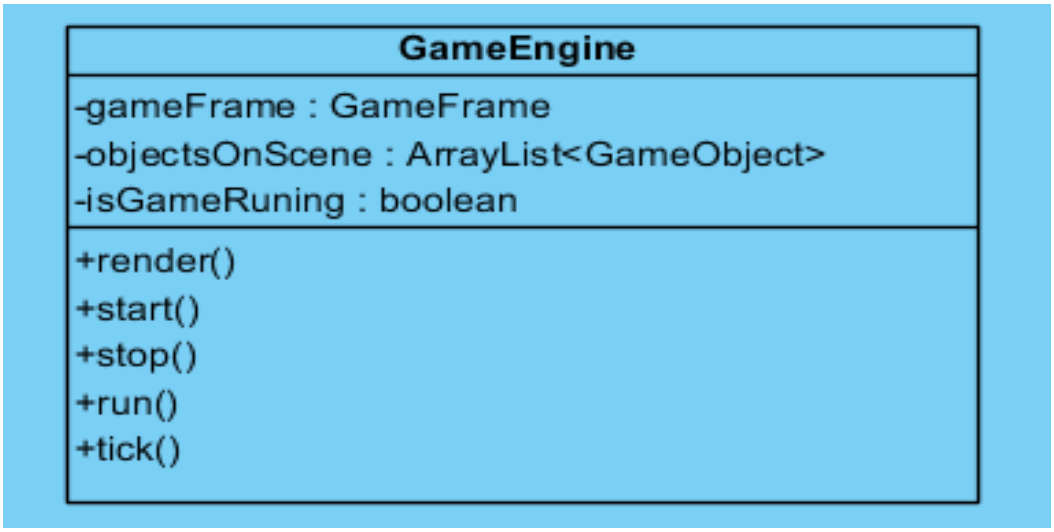
public saveOptions(Properties prop): It takes a properties file as a paramater with contains the user prefered game settings.

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.

4.4.16 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.

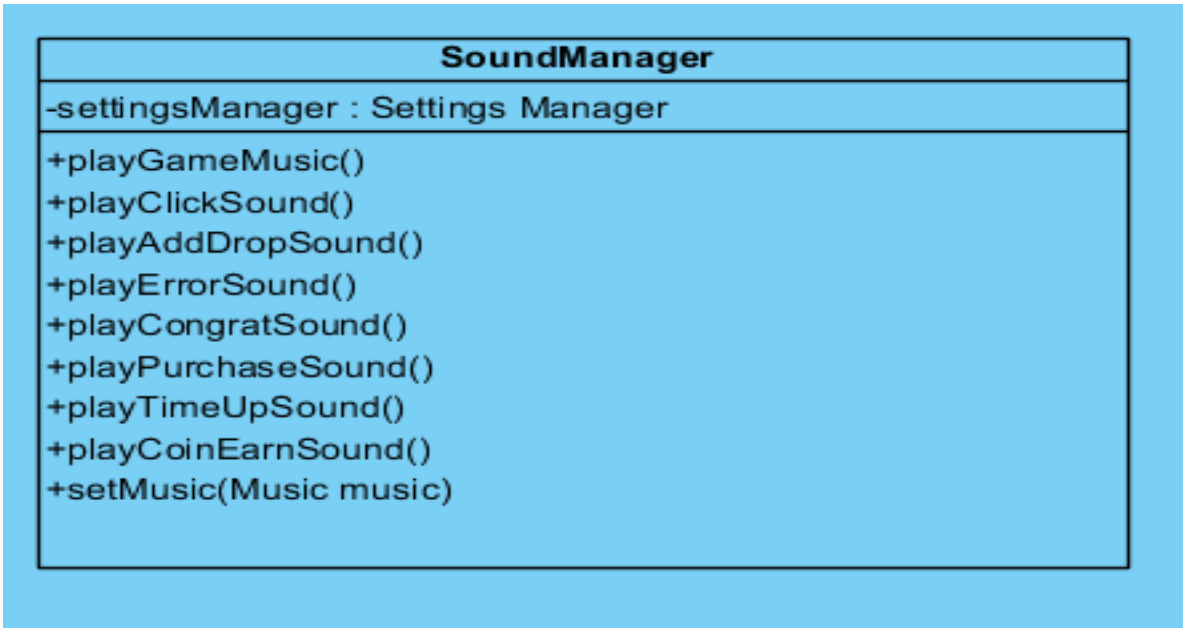
private void stop(): Stops threads to avoid unnecessary performance usage.

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.

4.4.17 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.

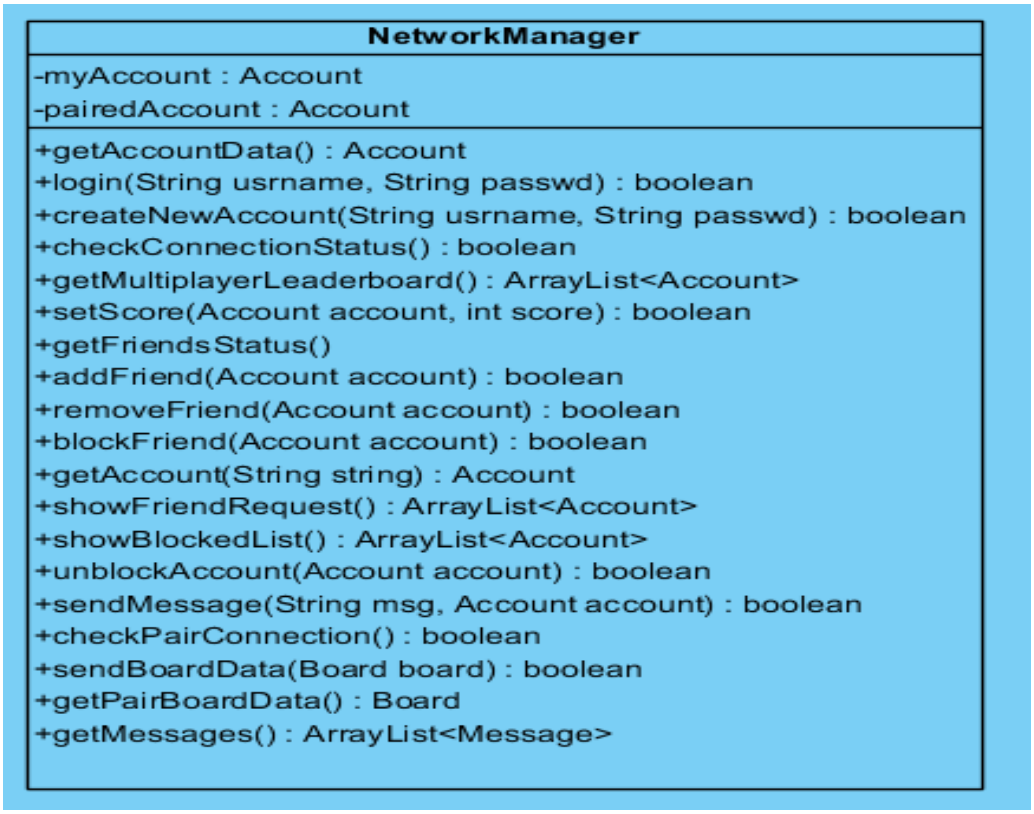
public void playCoinEarnedSound(): Generates sound for earning coins.

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

Constructors:

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

4.4.18 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 boolean createNewAccount(String usrname, String passwd): Adds account in the parameter to the database.

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

public ArrayList<Account> getMultiplayerLeaderboard(): Takes the data related to the multiplayer leaderboard from the database and represents it in ArrayList in order.

public boolean 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.

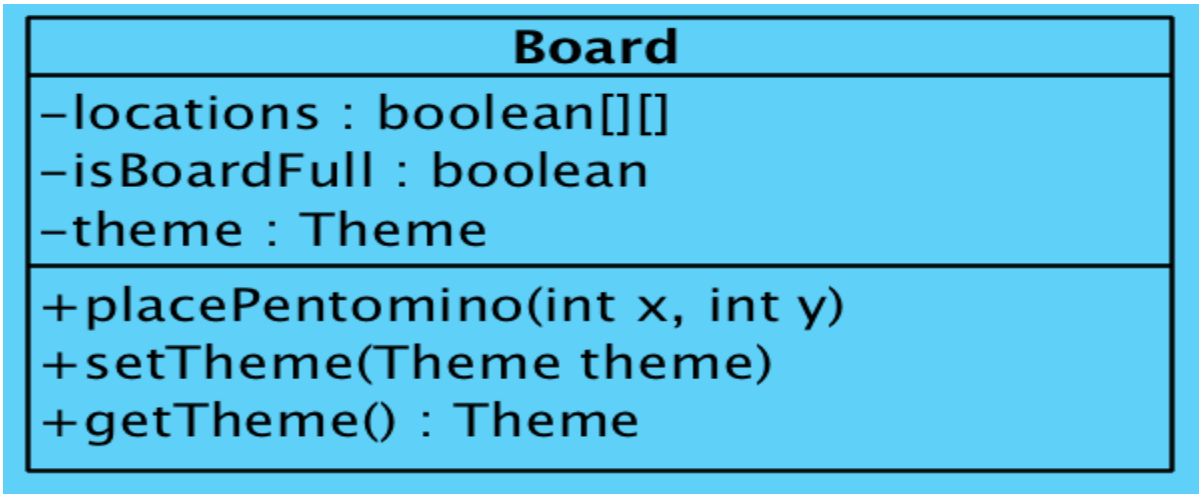
public Board getPairBoardData(): Gets the opponents board status from the database.

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

Contructors:

public NetworkManager(): Default constructor.

4.4.19 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.

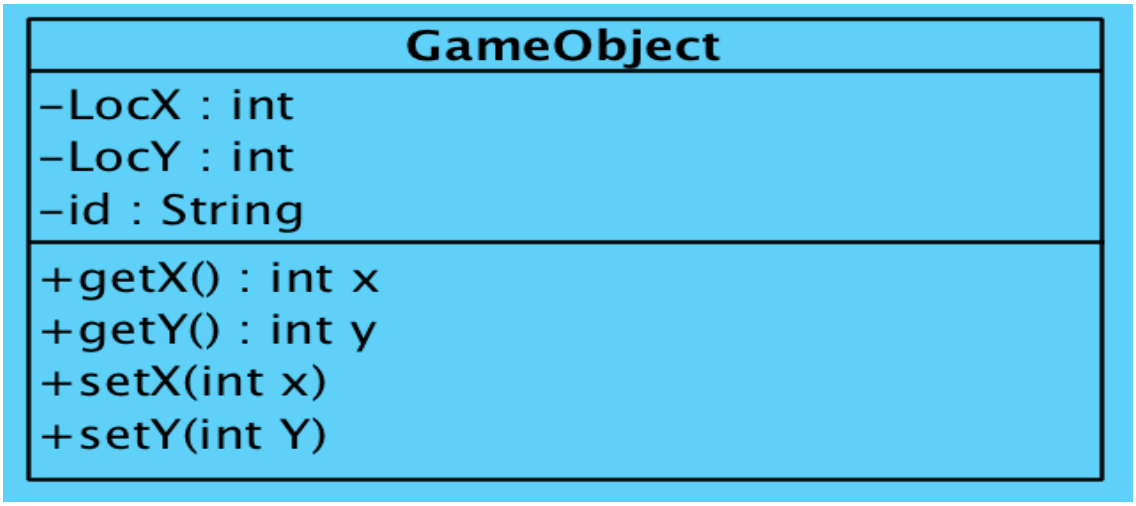
Methods:

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.

4.4.20 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:

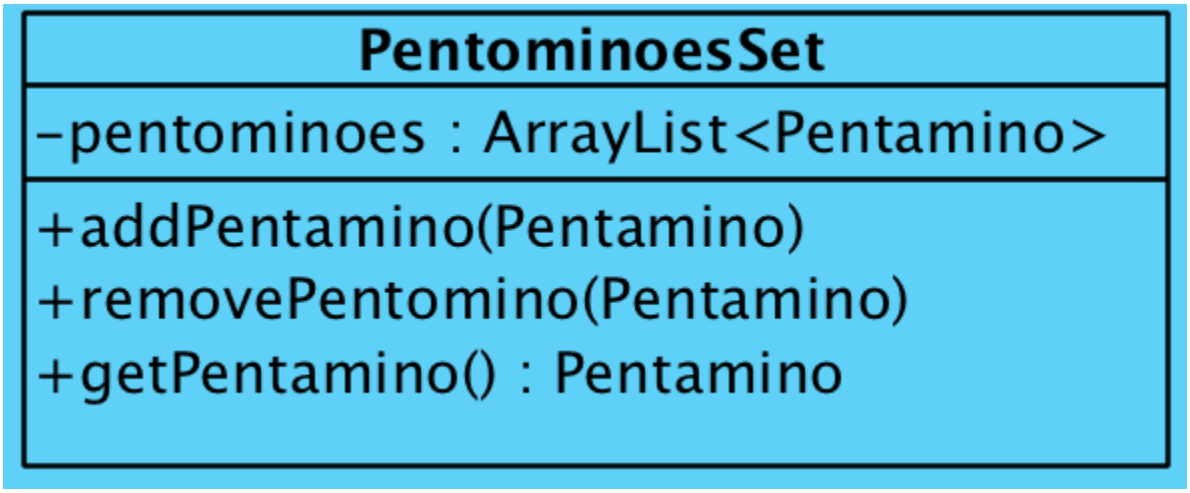
public int getX(): This method returns the x-axis coordinate of a pentomino.

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.

4.4.21 PentominoesSet Class



Attributes:

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

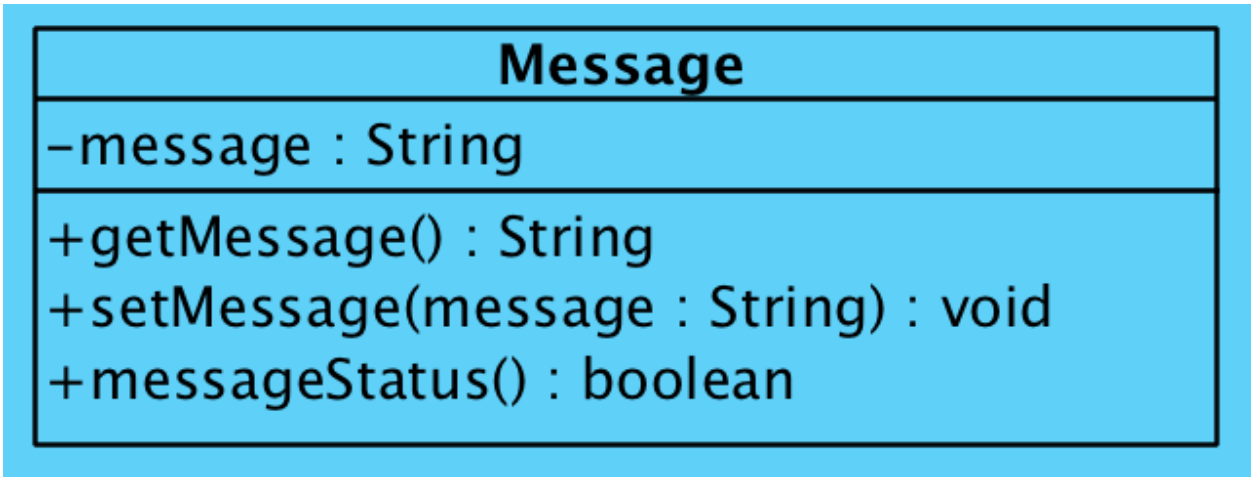
Methods:

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.

4.4.22 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.

4.4.23 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.

4.4.24 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.

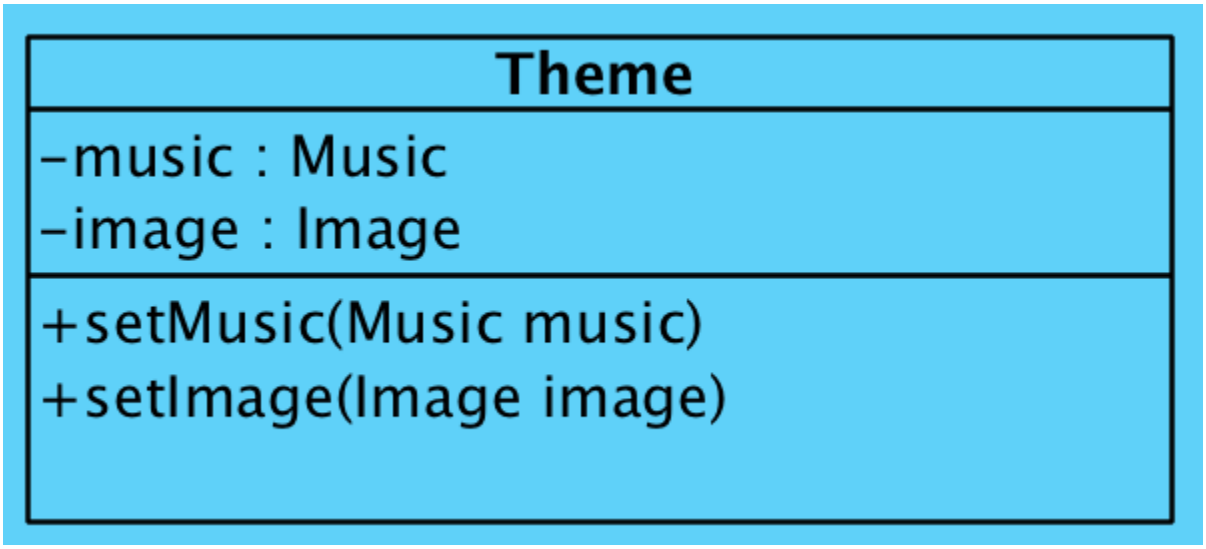
public Color getColor(): This method returns the color of a pentomino.

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.

4.4.25 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.

4.4.26 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.

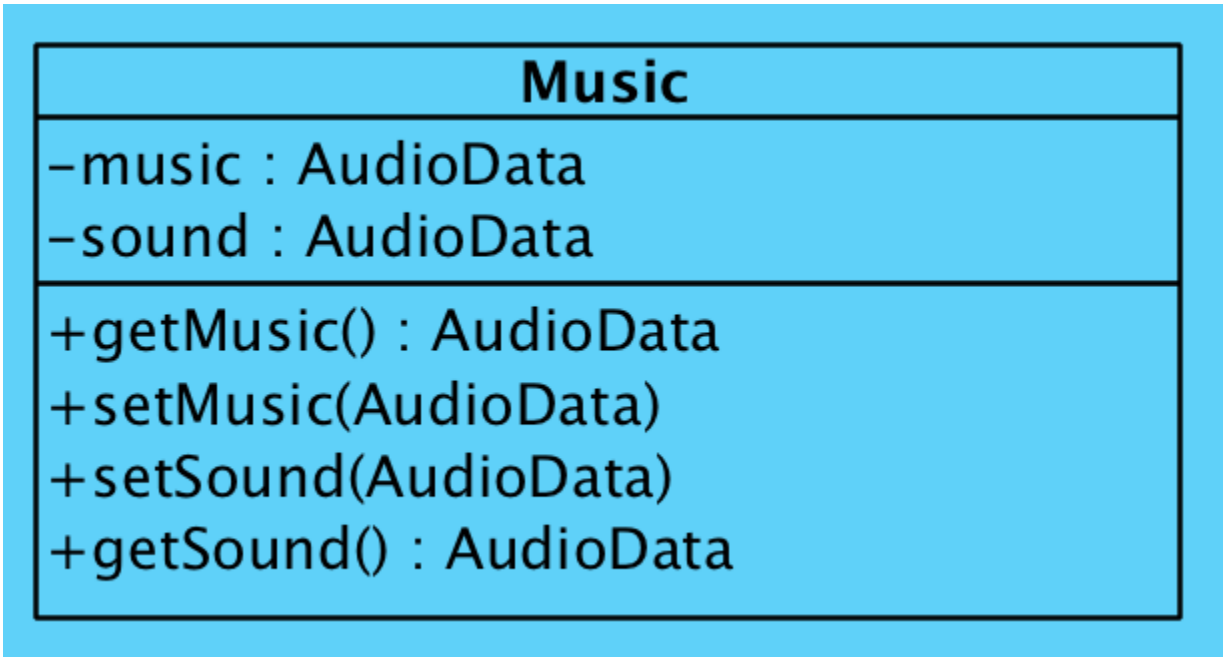
public void removeFriend (Account account): This method removes a user from the friendlist of an account.

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.

4.4.27 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.

5. References

[1] “Module Java.desktop.” , docs.oracle.com/javase/9/docs/api/java.desktop-summary.html.