QuadZillion

CS319 Iteration II Design Report

Bilkent University Department of Computer Engineering

Group 1G Supervisor: Eray Tüzün

Berk Güler
Enver Yiğitler
Kasım Sarp Ataş
Melike Arslan
Ufuk Bombar

Table of Contents

1 Introduction

1.1 Purpose of the System

QuadZillion is a 2-D solitaire puzzle game. The motivation behind developing a digital version of a game that already exists in the form of board game is to have an alternative with an appealing and user-friendly interface available on computers. Players will be provided with different modes of the game; each mode challenges the player on different aspects. QuadZillion encourages critical thinking and requires players to come up with creative ways to solve the given puzzles.

1.2 Design Goals

Design is one of the most important steps to create a game since what is designed will be the main focus of the game. In general, there are various number of highly desirable qualities, however they will be described in more detail under the following main criteria's.

1.1.1 End User Criteria

End user criteria is inferred from the application domain. It includes the user's point of view of using the system. Since our system is a game, the fact that whether the system supports the work of the user is not relevant. We expect the user to understand and be able to use the system very easily since it is a game system. The user will be provided with a selection of choices on every situation and immediate response or display is given back to the user. There is also an explanation on how to play the game, which will allow the user to get familiar with the game mechanics in the shortest time possible.

1.1.2 Performance Criteria

Performance criteria is inferred from the application domain. It includes space and time requirements. The system is responsive meaning that it is an interactive system letting the user interact with it and be provided with some results depending on the interactions. Memory space will be available for speed optimizations. The response to the user will be instant. When the user requests something, the system will provide an output instantly.

1.1.3 Functionality Criteria

Dependability criteria is another criterion inferred from the application domain. In this criteria the effort to minimize the system crashes and their results will be determined. To minimize the crashes, the availability of the system to the user is kept at the lowest level. There are no security risks or safety issues associated with the system environment because the game is a locally played game that does not require any database system or network connection.

1.1.4 Maintenance Criteria

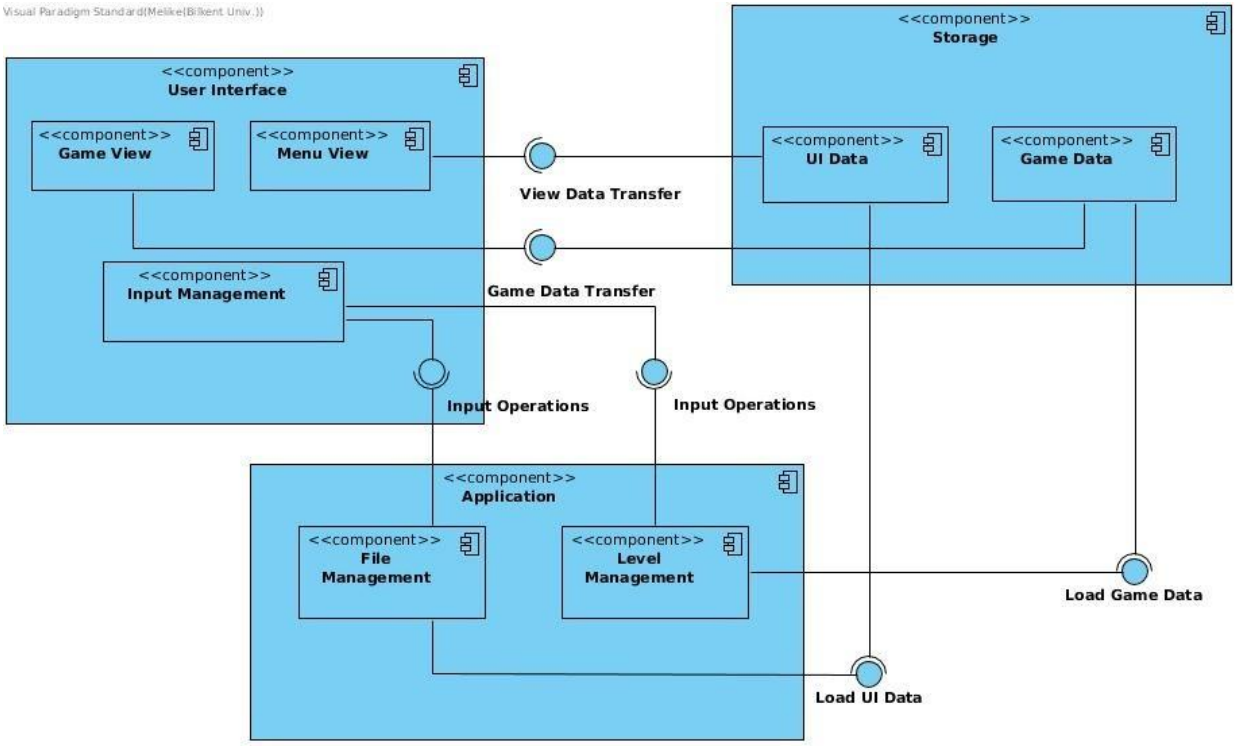
Maintenance criteria is dictated by the customer and the supplier. Maintenance is an important factor of a game after design and deployment. QuadZillion will not be using any servers or multiplayer options therefore we will not be focusing on updating the software with newly emerging technologies. However, we will release new patches of the game to solve the bugs which might emerge after the release. Another aspect for maintenance is the features we will add after the initial release. For example, other than the modes that are included, which are Vanilla, Extended and Puzzle modes, new game modes can be added to system. The system's hierarchical structure is is designed so that these features can be implemented easily without changing the general core of the game.

1.1.5 Cost Criteria

Cost criteria is also dictated by the customer and the supplier. It includes the cost to develop, deploy and administer a system. For our system, this is not a major aspect for this system since it is a game system for a course term project.

2. High Level Software Architecture

2.1 Subsystem Decomposition



This section covers the division of systems into subsystems. This process of dividing systems into subsystems reduces the coupling between them and also a fundamental way to approach problems; divide and conquer.

The decomposition of the system began with differentiating and grouping the similar subsystems in the components. The procedure we use derived from the MCV (Model View Controller) design pattern. The division made in way that both MVC principles applied and coupling reduced.

We applied MCV system design pattern because; MVC design pattern is a simple to understand, efficient and adaptable design pattern. This pattern will be the best fit for our design.

2.2 Hardware/Software Mapping

Our system will be implemented using the Java programming language, and it will require JavaFX library to perform visualization and input tasks. In order to run our software "QuadZillion", the user is required to install Java Runtime Environment(JRE) version 11. In addition to that, our system will require an internal or external GPU (Graphics Processor Unit) in order to visualize the operation. Additionally, keyboard and/or mouse to invoke methods designed to handle input. Used services with their descriptions are given;

- **JavaFX:** We used this service to support Keyboard and Mouse as well as displaying the game in an optimized fashion using the internal or external GPU of the configuration.
- **Java:** Default packages of java is used for saving implementation time on trivial classes such as java.util package. Used sub-services include, data structures and functions for different purposes for example file read/write.

2.3 Persistent Data Management

QuadZillion only has local data storage. It does not have database or any other complex storage systems. Each mode of the game is stored differently. Vanilla mode of the game is hard-coded because the instances of the levels in vanilla mode such as pieces should not be modifiable. Extended mode levels are stored in a JSON file because the pieces can vary in shape so they are modifiable and the json is updated according to that. Puzzle mode levels are also stored in a JSON file. In the puzzle mode an image will also be stored within the same directory as the JSON file and the name of this image will be written in the JSON file. In addition, we will store sound data as an .mp3 file, images for icons and backgrounds as .jpeg or .png files, a tutorial video as .mp4 file. Moreover, for the layout of the screens FXML format is used and for the theme of the game .css format is used.

2.4 Access Control and Security

QuadZillion will not use any online database or network connection. The game can be simply installed and played locally on each computer without the requirement of internet

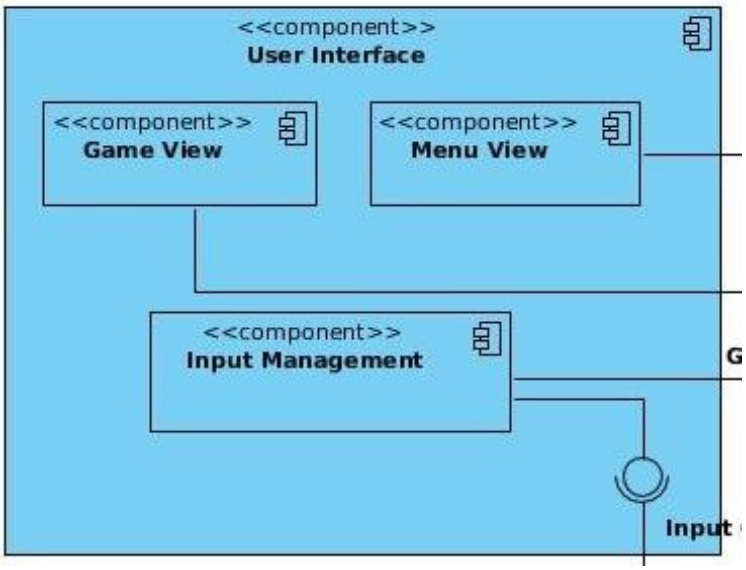
connection. Since there is no authentication system there will be no security issues for the user.

2.5 Boundary Conditions

QuadZillion will have an executable .jar file. Creating executable jar file makes the game easier to transfer to another device, therefore it becomes easier to share the game. QuadZillion can be terminated by clicking the “QUIT” button. In gameplay user can open pause menu by clicking the pause symbol. In order to close the pause menu in the game, user can press “Esc” button. If user presses the “Return to Main Menu” button, user will be redirected to the main menu of QuadZillion. User can disable the music by pressing corresponding buttons on the screen or the user can disable it from settings. User will be able to use the right-click on the mouse to rotate the pieces. User will able to use the mouse wheel in order to mirror the pieces horizontally. If game collapses before the level is finished and user returns to the main menu, the score of the player from the current level will be lost and cannot be reached in scoreboard.

3. Subsystem Services

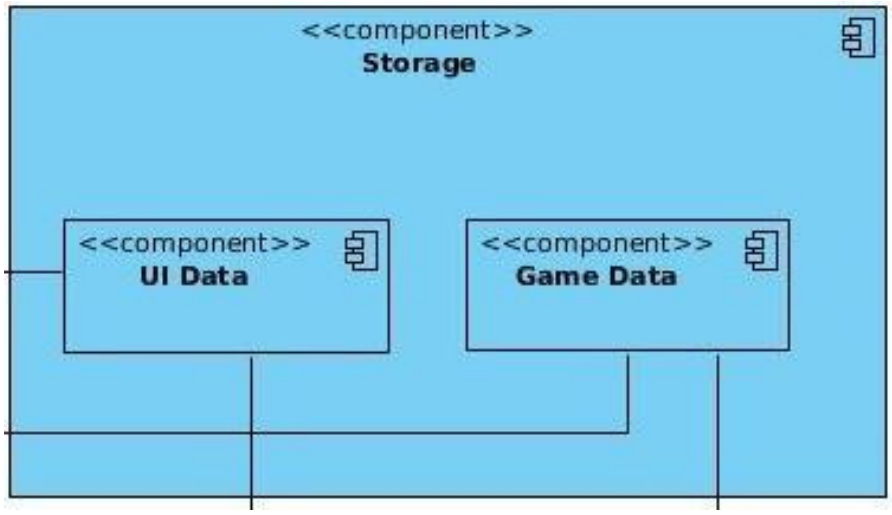
3.1 User Interface Subsystem



User Interface Subsystem is responsible for providing a graphical interface which can be displayed by monitor to the user. It contains three main subsystems which have divided for reducing coupling among classes. The following bullet points are given to explain their duties.

- **Game View:** This component is responsible for managing how the game is rendered to the user. It contains the classes that are named View objects in MVC design pattern. These View classes takes Model objects and displays them in the game view. The game view is only used when the player plays the game.
- **Input Management:** This component manages the input that are taken directly from the peripherals such as mouse and keyboard. It only contains controller objects for the sake of taking input. Additionally, input
- **Menu View:** This component manages all the controller, the scene classes and the interactions between those classes.

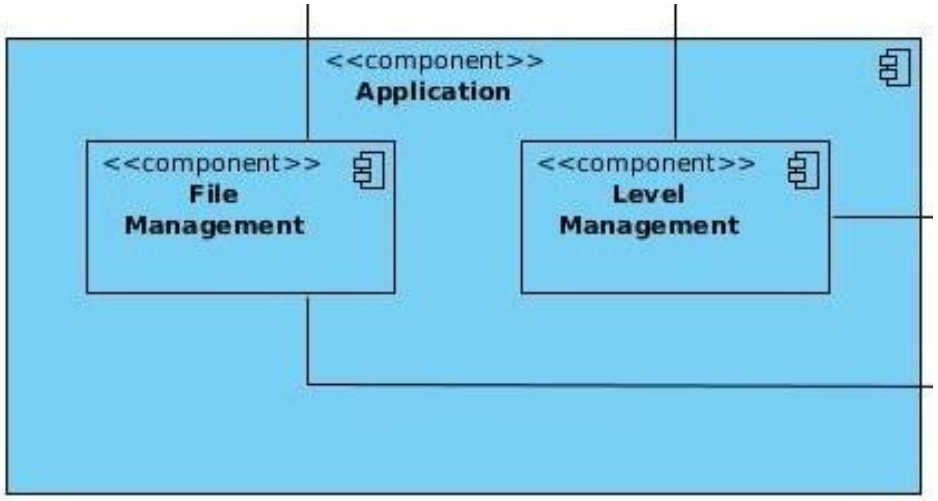
3.2 Storage Subsystem



Storage Subsystem is responsible for storing the UI Data and Game Data of the Game. These components are connected to the other subsystems with specific purposes. The following bullet points are given to explain each component of the subsystem:

- **UI Data:** This component is responsible for storing user-interface data attributes. These attributes include images or css files for themes. It is available for the Application Subsystem as Load UI Data and for the User Interface Subsystem as View Data Transfer.
- **Game Data:** This component is responsible for storing attributes within the game. These attributes include the pieces or the grids within a level. It is available for the Application Subsystem as Load Game Data and for the User Interface Subsystem as Game Data Transfer.

3.3 Application Subsystem



Application Subsystem is responsible for the controls of the game. It includes two components called File Management and Level Management. These components are connected to the User Interface Subsystem. The following bullet points are given to explain each component of the subsystem in more detail:

- **File Management:** This component is responsible for loading files in the data storage. It's connected to Input Management Component in the User Interface Subsystem. It provides the User Interface with the file inputs and provokes the Input Management Component. The files can be different formats such as ".json", ".mp3", ".jpg", ".png", etc. It is also connected to the Storage Component. The UI Data Component provides the File

Management with the user interface data and File Management component is responsible for loading this data.

- **Level Management:** This component is responsible for managing the main core of the game. It contains the general rules of the game. For example, the implementation of what happens if the player moves a piece into a grid is inside this component. This component is connected to Input Management Component in the User Interface Subsystem. It provides the User Interface with the level inputs and provokes the Input Management Component. It is also connected to the Storage Component. The Game Data Component provides the Level Management with the game data and Level Management component is responsible for loading this data.

4. Low-Level Design

4.1 Object Design Trade-offs

4.1.1 Development time vs Performance

We preferred JavaFX mainly because JavaFX provides developers advantages, such as easily GUI management. Scene Builder is one of the advantages that is provided by JavaFX. These advantages reduce the development time but also reduces the performance. However, using JavaFx is more beneficial than it is unbeneficial.

4.1.2 Understandability vs Functionality

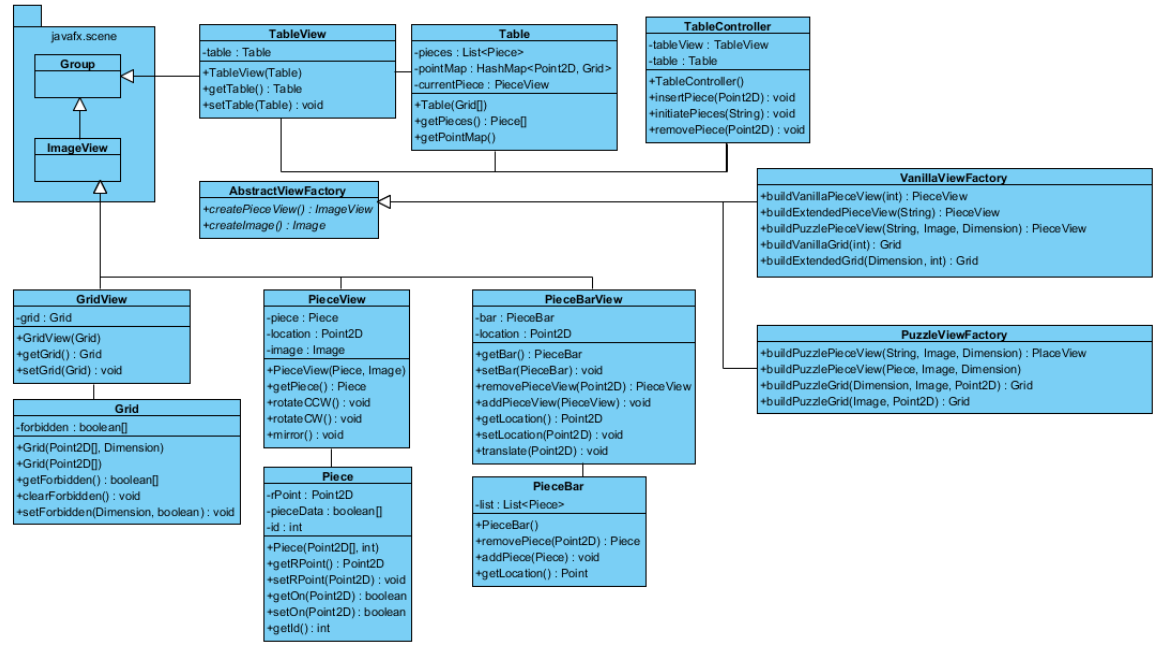
QuadZillion is a basic puzzle game that has no complex gameplay such as "power ups" nor "keyboard inputs" etc. Since the game is easy to understand, we added some new modes to get more of the player's attention. That decreases the understandability of the game but adds more interest to the game itself.

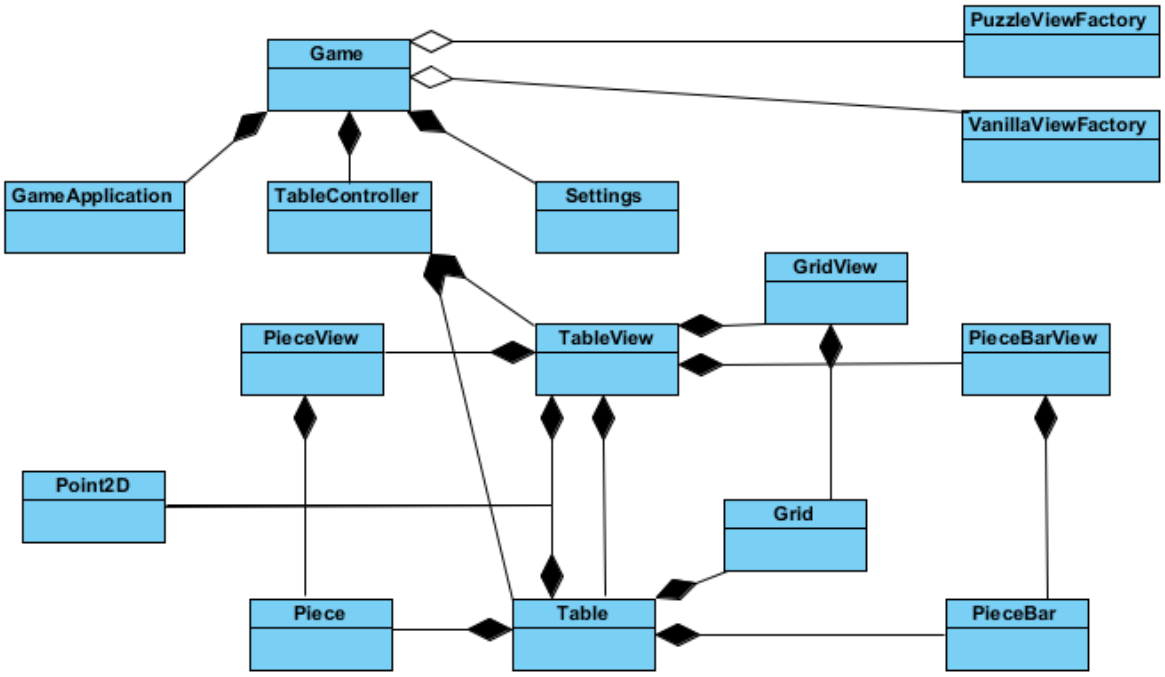
4.1.3 Simplicity vs Number of Options

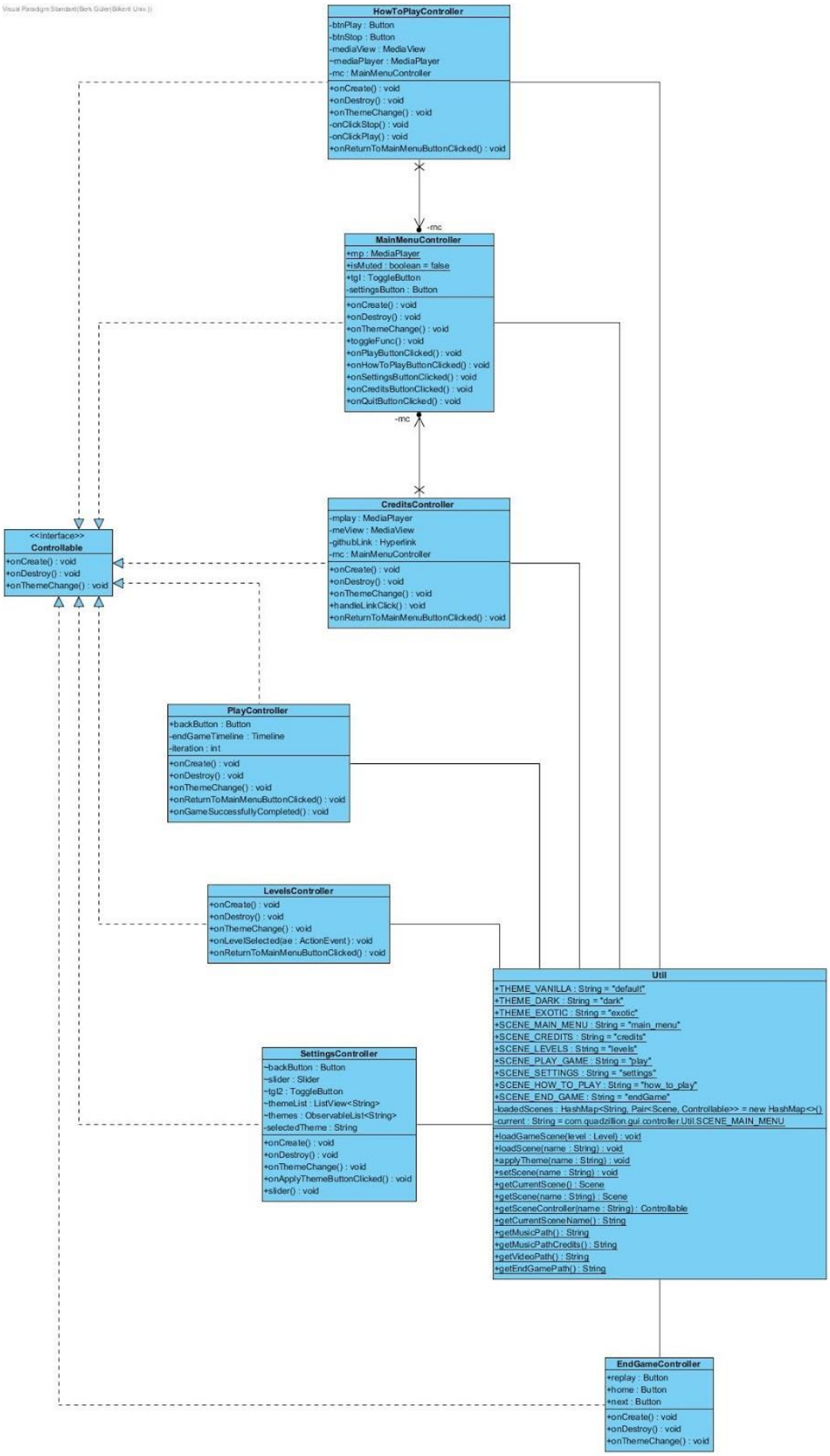
We acknowledged minimalism as our design principle in user interface as well as system design. Simplicity and clear design is the goal we want to achieve in this program.

4.2 Final Object Design

Below the full class diagram is given, we have both class and object diagrams for our core package.







Detailed version can be found here: https://hizliresim.com/7a0WkP

4.3 Packages

During our implementation of the game we decided to divide the classes are related to each other into packages. These packages are mainly divided into two.

4.3.1 GUI Package

- **Controller Package:** This package includes all the controller classes that are created by JavaFX.
- **Resources Package:** This package includes all the external resources used inside the application; Images, sounds, CSS files for defining the visuals of the scene and FXML Files for layout and components of the scene.

4.3.2 External Packages

4.3.2.1 Package javafx.stage

This package provides the fundamental classes to display the rendered scene.

4.3.2.2 Package javafx.scene

This package provides set base of classes for rendering a scene.

4.3.2.3 Package javafx.fxml

This package provides the necessary classes to create and load a scene from an FXML file.

4.3.2.4 Package javafx.scene.media

This package provides the necessary classes for playing music/video inside the scenes.

4.3.2.5 Package java.util

This package provides the classes for the data structures used in the application such as HashMap and ArrayList.

4.3.2.6 Package javafx.geometry

This package provides the Points2D class necessary to hold the coordinates of the pieces and the boards.

4.3.2.7 Package javafx.scene.shape

This package provides the Circle and Rectangle classes which are necessary to draw the board and pieces.

4.4 Class Interfaces

4.4.1 GameApplication Class

Attributes:

private static GameApplication instance: This attribute is used by a getter method in the Game class.

private static Stage primaryStage: This attribute is used as the main stage of JavaFX.

Methods:

public void start(Stage stage): This method is called when the application starts. Initializes the stages and loads the scenes into the stage. Sets the title and the themes of the main scene.

4.4.2 MainMenuController Class

Attributes:

public static MediaPlayer mediaPlayer: This attribute will be used to create a MediaPlayer instance to play a music in the main menu.

private ToggleButton toggle: This attribute will be used the toggle the mute button on/off.

private Button playButton: This attribute will be used to open the game play scene.

private Button howToPlayButton: This attribute will be used to open the How to Play scene.

private Button creditsButton: This attribute will be used to open the credits scene.

private Button settingsButton: This attribute will be used to open the settings scene.

private Button quitButton: This attribute will be used to quit the game.

Methods:

public void onCreate(): MainMenuController class implements the Controllable interface. This method is called when the scene is loaded. It behaves similarly to a constructor. Initializes the mediaPlayer and all the button attributes.

private void onHowToPlayButtonClicked(): This method is used to load the how to play scene when the play button is clicked.

private void onSettingsButtonClicked(): This method is used to load the settings scene when the play button is clicked.

private void onCreditsButtonClicked(): This method is used to load the credits scene when the play button is clicked.

private void onQuitClicked(): This method is used to quit the game.

private void onToggleClicked(): This method is used to mute the background music.

4.4.3 SettingsController Class

Attributes:

private Button backButton: This attribute is used to go back to the main menu from the settings.

private ToggleButton toggle: This attribute is used to toggle the music on and off.

private Slider slider: This attribute is used to change the music level from a slider.

private ListView<String> themeList: This attribute is used to hold the names of themes.

private ObservableList<String> themes: This attribute holds the names of the themes and responsible for presenting the names to the user.

private String selectedTheme: This attribute is set to a theme by ObservableList.

Methods:

public void onCreate(): SettingsController class implements the Controllable interface. This method is called when the scene is loaded. It behaves similarly to a constructor. Initializes all the button attributes.

public void muteToggle(): This method is used to toggle the mute on and off.

private void onApplyThemeButtonChanged(): This method is used to apply the theme changes when the button is clicked.

private void slider(): This method is used to change the music volume using the slider.

private void onBackButton(): This method is used to return to main menu when the back button is clicked.

4.4.4 Util Class

Attributes:

private static final HashMap<String, Pair<Scene, Controllable>> loadedScenes: This attribute is used to change scenes and themes.

public static ArrayList<Level> levels: This attribute holds the premade levels.

static GamePane gamePane: This attribute is used as a container for every other component in the game.

Methods:

public static void setScene(String name): This method is used to change scenes with respect to clicked buttons. This method is called from the controllers.

public static void applyTheme(String themeName): This method is used to change the theme of the scene via changing the CSS file dynamically.

public static void loadGameScene(ArrayList<Level> levels): This method is used to load the play game scene.

public static void addGamePanel(): This method is used to initialize the custom GamePane instance, which extends the JavaFX Pane class.

public static Scene getCurrentScene(): This method is used to return the current scene that is being displayed on the screen.

public static Controllable getSceneController(): This method is used to return the controller instance of a scene.

public static void removeChildren(): This method is used to remove the contents of the game play scene before loading other levels.

public static String getVideoPath(): This method is used to return the path of the video file to be played in the How to Play scene as a String.

public static String getMusicPath(): This method is used to return the path of the music file as a String.

4.4.5 PlayController Class

Attributes:

private Button backButton: This attribute is used to return to main menu from the Play Game scene.

private Timeline endGameTimeLine: This attribute is used to create a new Timeline for the blur effect after the game is completed.

private int iteration: This attribute is used for setting the quality of the blurred screen after the game is completed

Methods:

private void onReturnToMainMenuClicked(): This method is used to go back to the Main Menu scene.

private void onGameSuccessfullyCompleted(): This method is to create the blur effect after the game is completed.

4.4.6 HowToPlayController Class

Attributes:

private MediaPlayer mediaPlayer: This attribute is to create a MediaPlayer instance to play a video file.

private MediaView mediaView: This attribute is to create MediaView instance to be used with the MediaPlayer to play a video file.

private Button backButton: This attribute is used to return to main menu.

private Button playButton: This attribute is used to start the video file.

private Button stopButton: This attribute is used to stop the video file.

Methods:

public void onCreate(): HowToPlayController class implements the Controllable interface. This method is called when the scene is loaded. It behaves similarly to a constructor. Initializes all the button and media player attributes.

private void onClickStop(): This method is used to stop the video file.

private void onClickPlay(): This method is used to play the video file.

private void onReturnToMainMenuClicked(): This method is used to return to the main menu.

4.4.7 EndGameController Class

Attributes:

private Button replay: This attribute is used to replay the same level.

private Button home: This attribute is used to return to levels home scene.

private Button next: This attribute is used to switch to the next level.

Methods:

public void onCreate(): EndGameController class implements the Controllable interface. This method is called when the scene is loaded. It behaves similarly to a constructor. Initializes all the buttons.

4.4.8 TableView Class

Attributes:

private Table table: Instance of the model.

Methods:

public Table getTable(): This method returns the table instance.

public void setTable(Table table): This method sets the table instance.

Constructor:

public TableView(Table table): Constructor to initialize the table instance.

4.4.9 Table Class

Attributes:

private ArrayList<Piece> pieces: This attribute is container for piece objects.

private HashMap<Point2D, Grid> pointMap: This attribute takes locations as parameter and returns which grid it belongs to.

private PieceView currentPiece:

4.4.10 GridView Class

Attributes:

private Grid grid: This attribute is used for initializing a grid instance.

Constructor:

public GridView(Grid): Initiates the instance of the grid.

Methods:

public Grid getGrid(): This method returns the grid instance.

public void setGrid(): This method sets the grid instance.

4.4.11 TableController Class

Attributes:

private TableView tableView: This attribute is used to hold the table view which will be controlled in this class.

private Table table: This attribute is used to hold the table which will be accessed in this class.

Constructor:

public TableController(): This constructor initializes the attributes above.

Methods:

public void insertPiece(Point2D point2D):

public initiatePieces(String json): This method is used to initiate pieces based on the data taken from JSON file.

public removePiece(Point2D point2D): This method removes the piece which points given as paramater from the board.

4.4.12 PieceView Class

Attributes:

private Piece piece: This attribute is used for piece.

private Point2D location: This attribute is used for the piece locations.

private Image image: This attribute is used for the view of the piece.

Methods:

public Piece getPiece():

public void rotateCCW(): This method is used to rotate the piece counter clockwise.

public void rotateCW(): This method is used to rotate the piece clockwise.

public void mirror(): This method is used to mirror the piece.

Constructor:

public PieceView(Piece piece): This constructor initializes the piece and other attributes.

4.4.13 Piece Class

Attributes:

private Poin2D rPoint: This attribute holds the value of the location of the piece relative to board.

private boolean[] pieceData: This attribute stores information about the shape of the piece.

private int id: This attribute stores the special identity of the piece.

Methods:

public Point2D getRPoint(): This method is used to return the location of the piece relative to the board.

public void setRPoint(Point2D point2D): This method sets the location of the piece relative to the board.

public boolean getOn(PointsD point2D): This method checks whether a piece exists on the board with given relative coordinates.

public boolean setOn(Point2D points2D): This method returns if the insertion of a piece was successful or not.

public int getID(): This method returns the id of the piece.

Constructor:

public Piece(Point2D[] point2D, int id): The constructor initializes the attributes of the Piece class.

4.4.14 PieceBarView Class

Attributes:

private PieceBar bar: This attribute holds the model of the PieceBar.

private Point2D location: This attribute stores the location of the pieces on the piece bar, which is the default positions of the pieces.

Methods:

public PieceBar getBar(): This method returns the PieceBar instance inside the class.

public void setBar(PieceBar bar): This method is used to set the PieceBar instance inside the class.

public PieceView removePieceView(Point2D point2D): This method removes the piece from the bar and updates it.

public void addPieceView(PieceView pieceView): This method adds a piece to the piece bar.

public Point2D getLocation(): This method returns the locations of the piece on the piece bar.

public void setLocation(Point2D point2D): This method sets the location of the piece on the piece bar.

public void translate(Point2D point2D): This method updates the position of the piece.

4.4.15 PieceBar Class

Attribute:

private List<Piece> list: This attribute holds the created pieces to be accessed later.

Constructor:

public PieceBar(): This constructor initializes the pieces inside the piece bar.

Methods:

public Piece removePiece(Point2D point2D): This method removes the piece from the piece bar and returns the piece.

4.4.16 TableController Class

Attributes:

private TableView tableView: This attribute is an instance TableView.

private Table table: This attribute is an instance of a Table model.

Methods:

public void insertPiece(Point2D point2D): This method sets the piece on the board with respect to given relative coordinate parameters.

public void initiatePieces(String JSON): This method initiates the pieces based on the JSON file.

public void removePieces(Point2D point2D): This method removes the piece from the board with given relative coordinates.

Constructor:

public TableController(): This constructor initializes the tableView and table attributes.

4.4.17 AbstractViewFactory Class

Methods:

public ImageView createPieceView(): This is an abstract method to create a piece view.

public Image createImage(): This is an abstract method to create an Image.

4.4.18 VanillaViewFactory Class

Methods:

public PieceView buildVanillaPieceView(int id):

public PieceView buildExtendedPiece(String JSON): This method creates a PieceView based on the JSON file.

public PieceView buildPuzzlePieceView(String JSON, Image image): This method creates a PieceView for the puzzle version based on the JSON file and the image.

public Grid buildVanillaGrid(int id): This method creates and returns a grid for the vanilla mode. Default dimensions is 4x4.

public Grid buildExtendedGrid(int id, Dimension dimension): This method creates and returns an extended mode grid based on the dimensions of the board.

4.4.19 PuzzleViewFactory Class

Methods:

public PieceView buildPuzzlePieceView(String JSON, Image image, Dimension dimension): This method

public PieceView buildPuzzlePieceView(Piece piece, Image image, Dimension Dimension():

public Grid buildPuzzleGrid(Dimension dimension, Image image, Point2D point2D): This method creates and returns a piece view for the puzzle mode.

public Grid buildPuzzleGrid(Image image, Point2D point2D): This method

4.5 Design Patterns

4.5.1 Model View Controller Pattern

In order to separate the user interface, data and controller from each other. This way the code organized into classes without overlapping duties. Dividing the application logic into three parts also allows for faster development process. Modifications in view part does not affect the other parts so it is more convenient to test changes. MVC pattern is suitable for interactive programs like games and this is why this pattern is adopted.

4.5.2 Factory Pattern

Factory pattern is used in the application to create objects with a multiple subclass. This prevents the cope duplication. In our application there are classes similar to each other but with differences. Using this pattern makes the code simple to understand and organized.

5 Improvement Summary

While developing any sort of program or game, one of the most common aspect is the improvements while the program or game is designed. Our project design report improved since the last iteration. We correctly draw the diagram for the subsystem decomposition and explained each component in the subsystem unlike the last iteration report. We improved our object design and class interfaces and these are also written in more detail than the previous one.

Not only did our report improve, but also our game improved as well. We polished our graphical user interface (GUI), also came up with new game modes as well. These modes are extended mode and puzzle mode. We also added some new features but we changed a few things as well. Now our game does not require a keyboard for the piece movements anymore, instead of a keyboard, we rotate or mirror the pieces with mouse clicks. In this report we learned the importance of good organization of reports and understandability of the UML diagrams.

6 Glossary

- **OOP:** Object Oriented Programming
- **JVM:** Java Virtual Machine
- **JavaFX:** Built-in GUI Library of Java
- **MVC:** Model View Controller Pattern
- **JRE:** Java Runtime Environment

7 References

- JavaFX Documentation: https://openjfx.io/javadoc/11/
- Quadrillion Original Game Web Page: https://www.smartgames.eu/uk/one-player-games/quadrillion