Bilkent University

Department of Computer Engineering

CS319 Term Project

Katamino

Design Report

Group 1H

Supervisor: Eray Tüzün
Group Members: Ege Özcan, Mustafa Bayraktar, Simge Tabak, Yağız Efe Mertol, Zeynep Gözel

Progress Report
Nov 8, 2018

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of Term Project course CS319

Contents

1. Introduction

1.1. Purpose of the System

Katamino is a 2D mind game that aims to get people thinking. It is designed so that everybody can understand and play the game by just using their common sense. It is different from the original game in a few aspects: our game has many different board shapes whereas the original game is limited to a rectangle, our game has hint and time systems implemented which the original game does not have. Katamino is planned to be a thought provoking, portable game with intuitive user experience.

1.2. Design Goals

- **Performance:** The system is going to be swift. Every user action should be quickly recognized by the system within 0.1 seconds. As the game is a simple one, we aim to keep it CPU friendly, meaning that it will not need great amounts of computing power. Speed will be our priority compared to space, as the game is a small one, we do not expect any problems related to space.
- **Portability:** We want our game to work on most common operating systems such as Windows, Linux, MacOS. Therefore, we will use Java.
- **Modifiability:** We want to have a modifiable system that allows us to add extra features in the future. We will divide the program into many logical classes and functions so that when we want to add new features to the game, we can do it easily. For example, even though we do not plan on implementing online functionality, we will keep it in mind while coding so that we can easily implement such functionality without making any changes to the core of the game.

- **Usability:** The system should be easy to understand and easy to use. A new player should not be discouraged by the complexity of the system. To achieve this, we plan to keep our interface very simple yet useful. However, this simplicity does not mean we will have a basic interface, we will have a simple yet elegant interface that will satisfy the users.
- **Security:** We aim to keep the game offline for its initial release, therefore everything related to the game will be stored locally. The leaderboards will be stored locally too, so people will be able to compete with each other's scores using the same computer and not with other people on other computers. Therefore, we do not have any security concerns.

2. High Level Software Architecture

2.1. Subsystem Decomposition

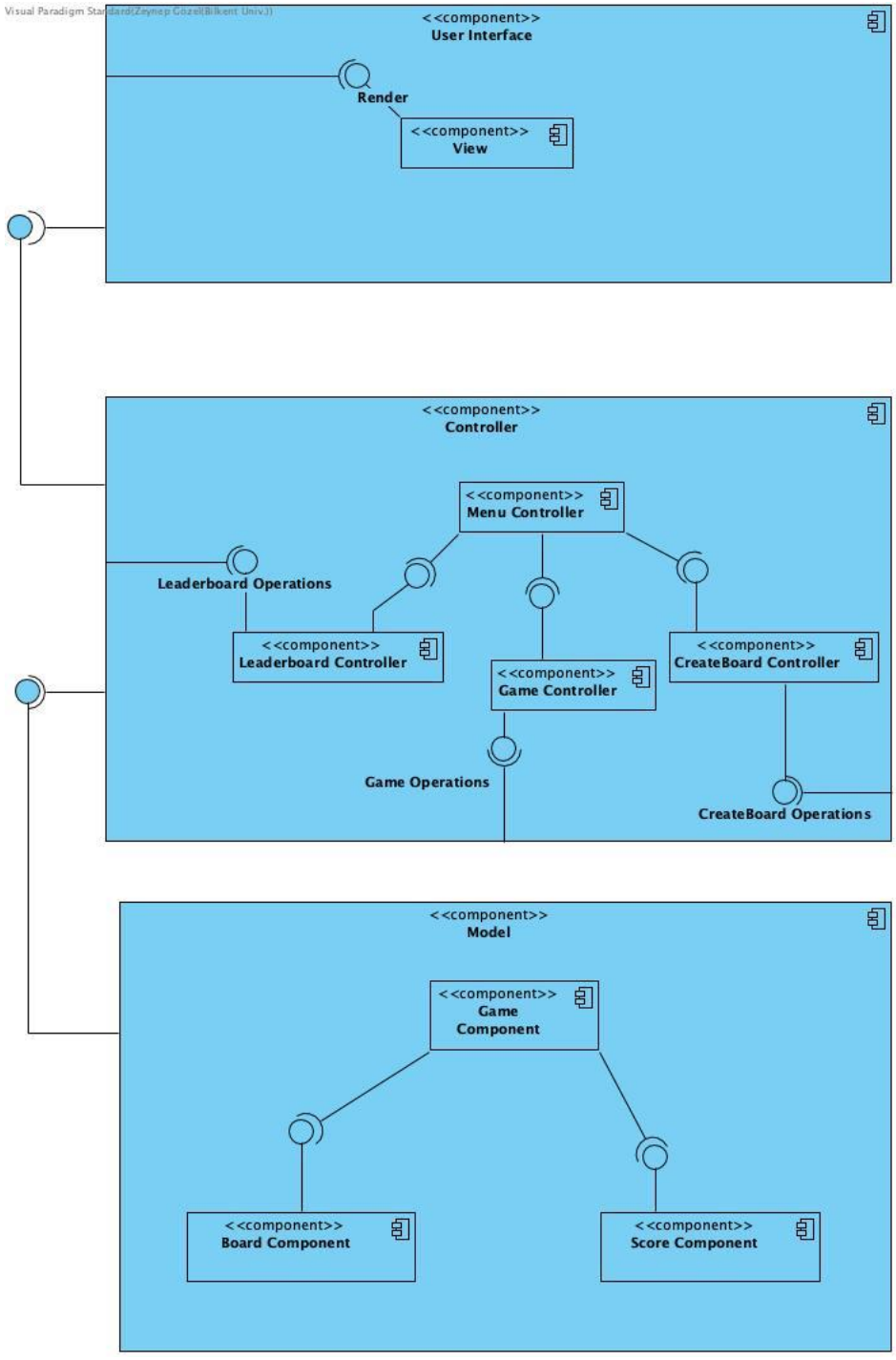


Figure 1: Deployment Diagram for Katamino

We decomposed our system into subsystems since it will be easier to see and control the components of our system. Owing to this decomposition, we will be able to modify our implementation when necessary.

We plan to have 3 main packages in our program: User Interface, Controller and Model. These main packages directly refer View, Controller and Model packages in MVC (Model – View – Controller) system design pattern. We decided to use this pattern for an easy implementation stage as it is easy to make work division for an MVC system. This approach also helps us accomplish our modifiability goals, because when we want to add new features or change some functionality, we won't have to make big changes.

According to our design goal, we decided to use model view controller architecture. On our model layer, we have the components of Board and Score components both plugged into the socket of Game Component. Game component consists of a board that has its own functionalities and a Score component which contains a timer and username. Controller component consists of the Leaderboard controller game controller and createBoard Controller which are each directed by the menu controller due to navigation issues. Our view component is rendered to be shown on the screen.

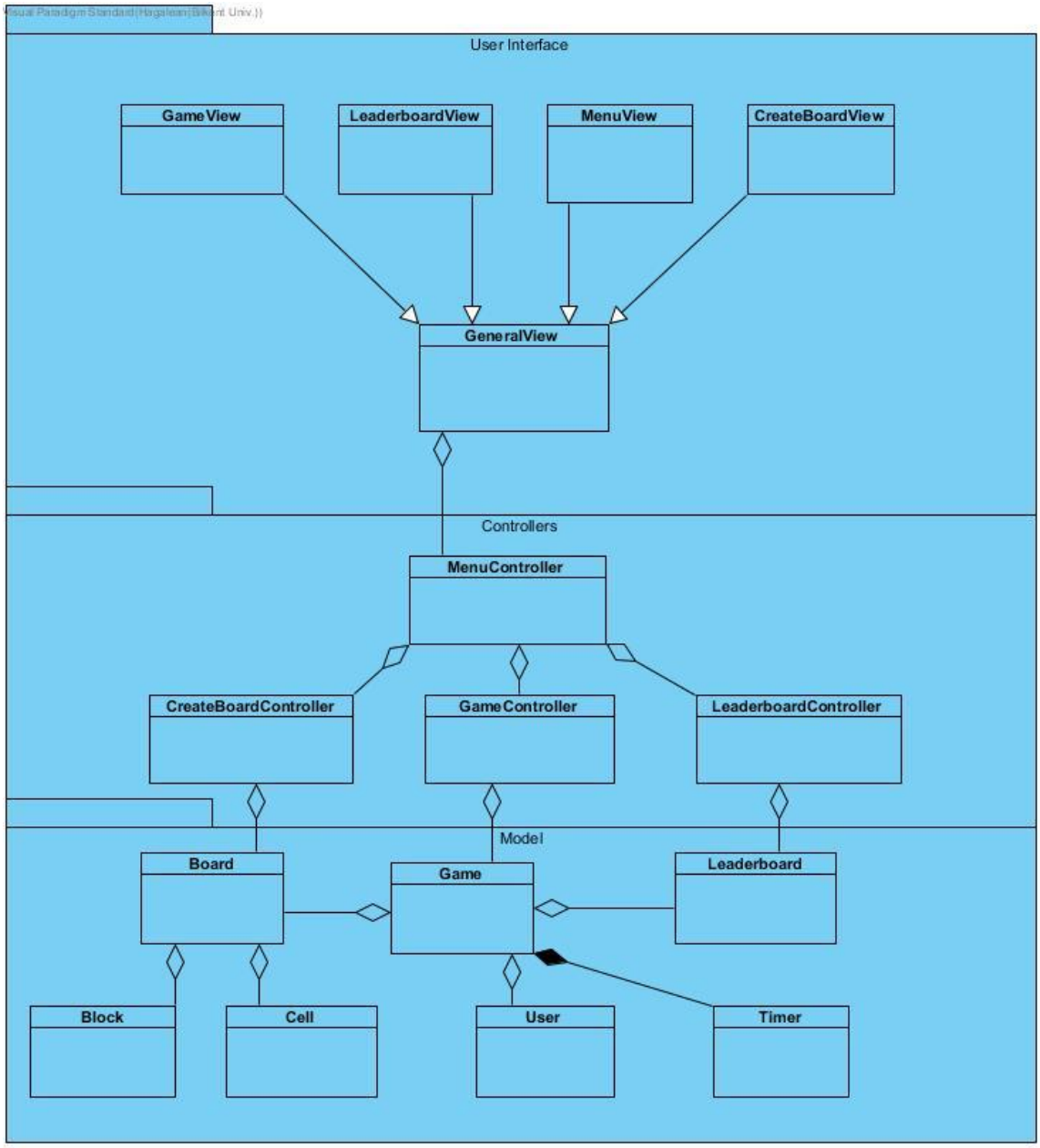


Figure 2: Opaque layering in Katamino.

We used opaque layering (as seen in Figure 2) to decompose our system as we believe in its understandability. These packages are determine based on their functionalities.

- User Interface package has GameView, LeaderboardView, MenuView and CreateBoardView classes, which all inherit GeneralView class. This package provides a graphical interface for users.

- Controller Package has four different classes. MenuController class has CreateBoardController, LeaderboardController, GameController classes. The main goal of this package is making the game successfully work.
- Model Package has Board, Game, Block, Cell, Timer, Leaderboard and User classes. These are generally the objects of the game. These models are managed by the Controller subsystem classes. Board is controlled by CreateBoard class, Game is controlled by GameController class and Leaderboard is controlled by LeaderboardController.

2.2. Hardware/Software Mapping

Katamino will be implemented for computers with Java programming language by using JavaFX. Therefore, the software requires Java Runtime Environment to be installed on the computer to be executed. A version of Java Runtime Environment that supports the JavaFX libraries will be enough. Katamino will be played by using mouse as input tool. Therefore, as a hardware requirement mouse is essential. A keyboard will be used to type name for the leaderboard. Therefore, keyboard is also a hardware requirement. The system requirements will be limited to be played on most of the computers. The storage issue will be handled on the hard drive in terms of text files. As we will be using text files, the speed of the hard drive will not be so important. The text files will contain board data and leaderboard. The software is offline, so no internet connection or database is needed to operate.

2.3. Persistent Data Management

The board data and leaderboard for the game Katamino will be kept in the hard drive as text files formatted with json. The live game data will not be stored on the hard drive because the main aspect of game is to beat time and the user should not close the game, think and keep going. The visuals will be stored in .jpg format and the sounds which will be used will be stored in a compressed sound format, included in the .jar file.

2.4. Access Control and Security

As our game is offline, there are no issues about a database or internet related issue to worry about. However, the live game data is not going to be stored in the hard drive to prevent manipulation. This data will be kept in the RAM making it both faster and more secure compared to being stored in the hard drive. Leaderboard is kept uniquely for the device that the game runs on. Therefore, there is no need for any real security management or access control.

2.5. Boundary Conditions

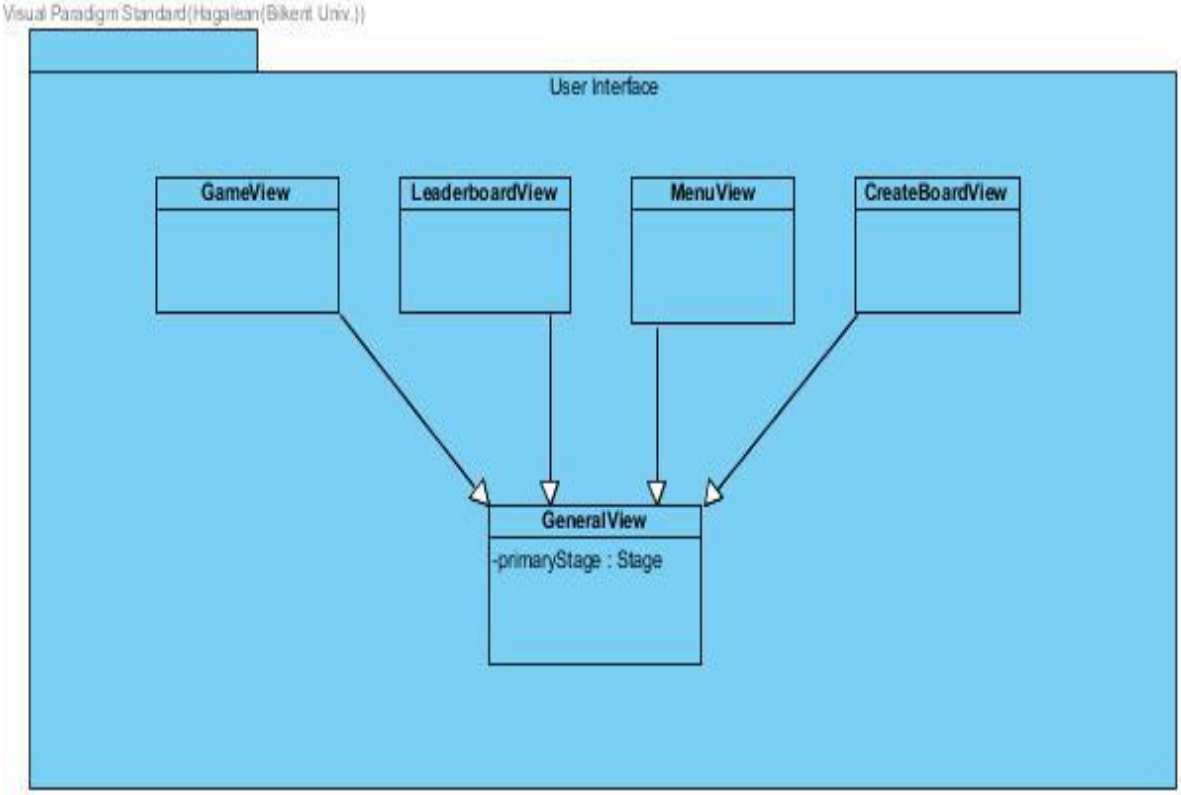
Katamino will run from an executable .jar file, therefore there will be no installation needed. The game will be highly portable, but the leaderboard will be unique to the computer. The jar copied will be enough to play the game.

There will be an exit game option on the menu and in any other way the game is closed while playing, the live data will not be saved as it is not really needed. In a potential crash of computer hardware or software, the live game data will be lost. Therefore, our game will not store game data after any kind of game termination. We saw that storing or saving game data would increase the space taken up by our program while slowing down the

game. Therefore, we decided it would not be a meaningful decision to implement such functionality in our game.

3. Subsystem Services

3.1. User Interface Subsystem



User Interface Subsystem has 4 classes which are:

1. MenuView

   MenuView provides a menu screen for the user when the game begins. There are three classes inside this package. This class is responsible for displaying a page which has buttons, text boxes, and sound control area. In the menu screen the user can choose to display the leaderboard, play a new game, create a board or exit the game.

2. GameView

   GameView provides cells, blocks and a board for the game. These elements are created by getting their information from GameManager class. GameView is what the user sees while playing the game.

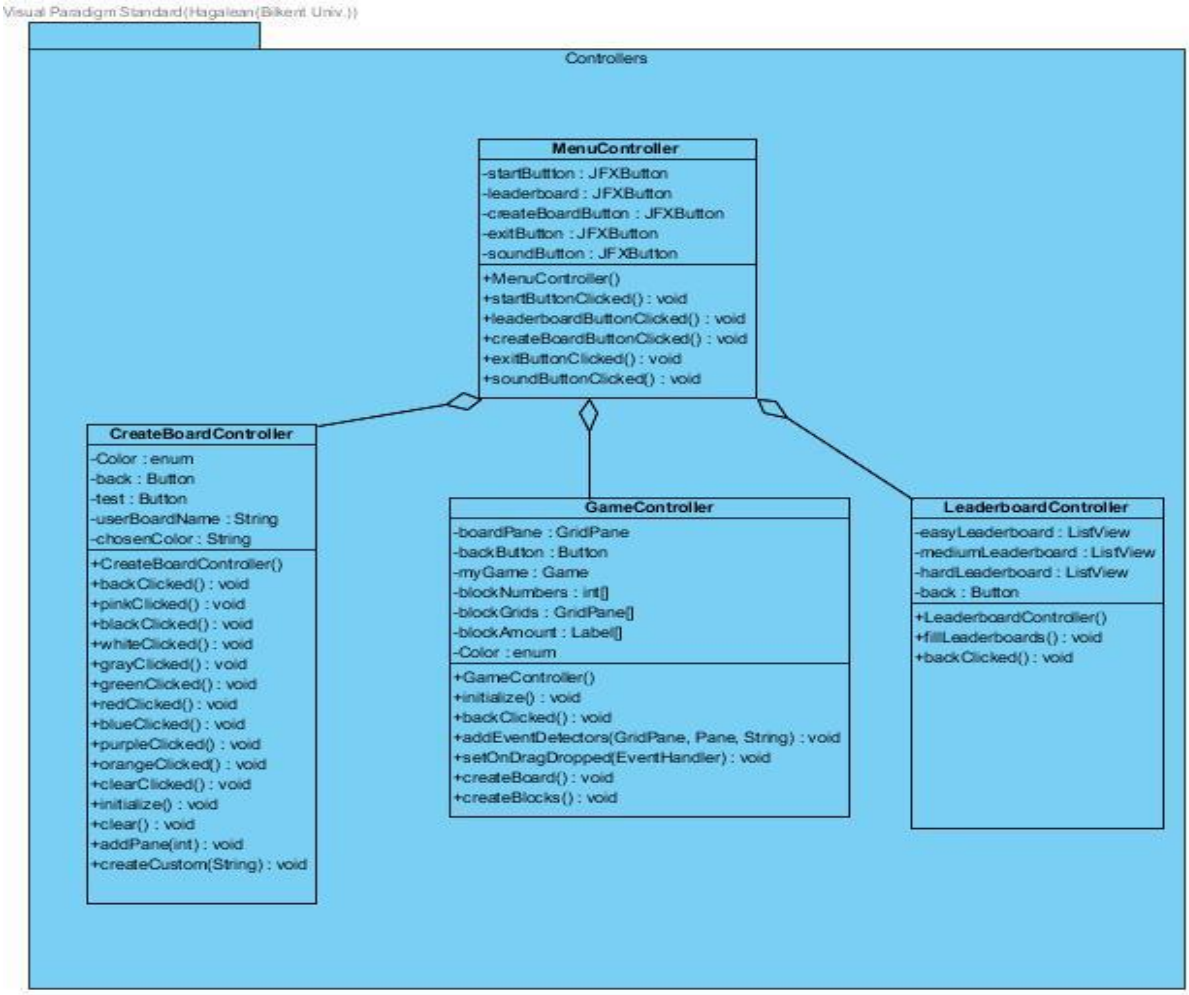
3. LeaderboardView

   LeaderboardView is a simple UI class that is responsible for formatting the leaderboard screen.

4. CreateBoardView

   CreateBoardView is the screen shown to the user during custom board creation. This screen contains an empty board that consists of fillable cells, a color selection tool and a block list for testing the board.

3.2. Controller Subsystem



1- Menu Controller

MenuController is responsible for directing the user to different game modes. By using buttonClicked methods assigned to buttons, different game modes are opened.

2- Game Controller

GameController is responsible for controlling the game by providing board and time information for the system. GameController class keeps the time by using GameTimer class. When the user starts a new game and selects a board, GameController class provides the blocks needed for that board and controls the game if it is ended or not. At the end of the game, it sends the time score to the leaderboard if user achieved to be in the list.

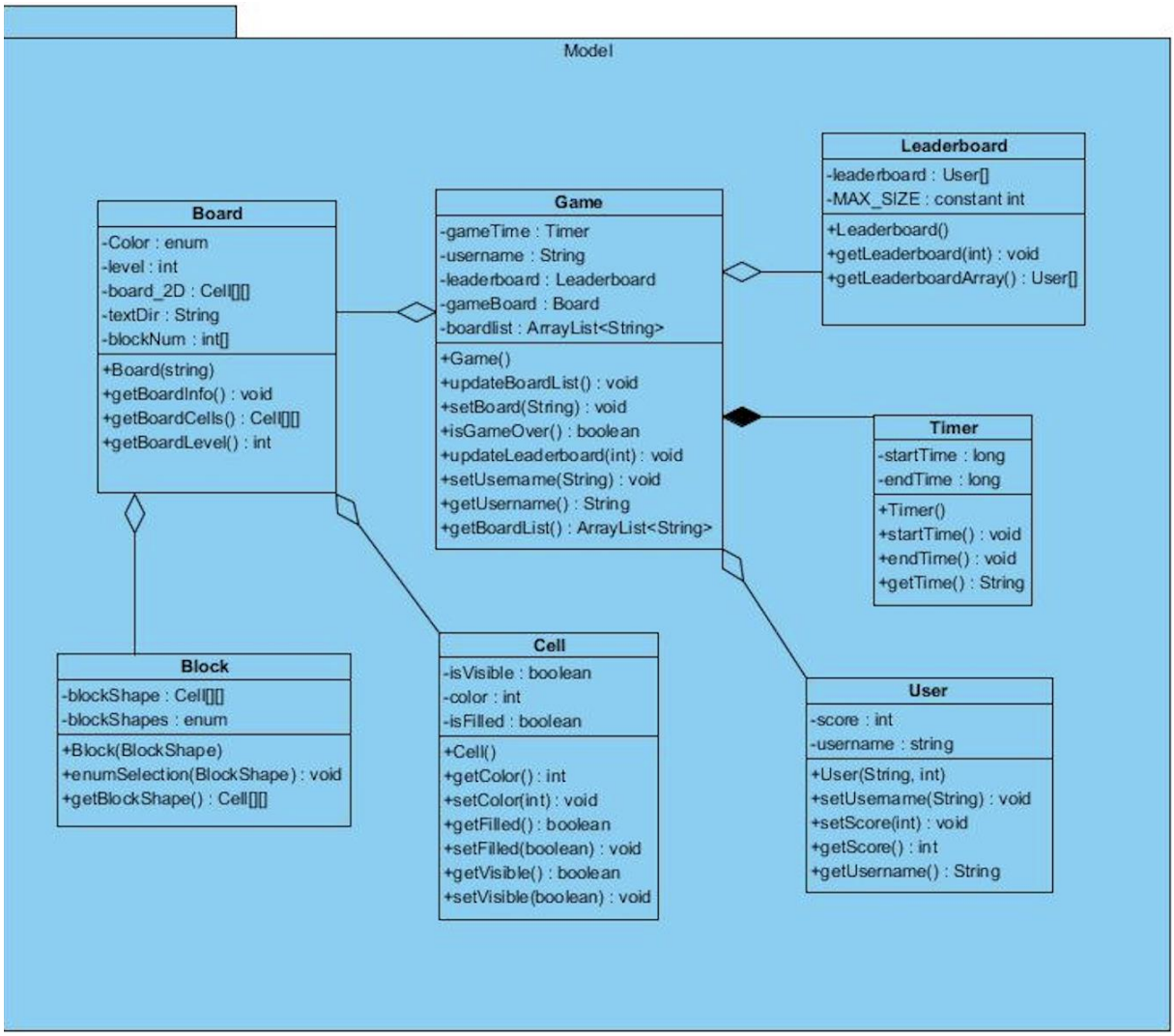
3- Leaderboard Controller

The Leaderboard controller class reads the leaderboard from a txt file for a specific hardness of game and displays usernames and their scores on a ListView.

4- Create Board Controller

CreateBoardController class is responsible for the creation of custom boards. Creation of custom boards involves many steps. First, the user starts to create a board by painting specific cells on the board grid. Then, the user specifies the maximum amount of each block, that can be used to fill the board. After this step, when the user specifies that the block placement is finished, the user will enter a name for the custom board and then board manager will save the custom board along the premade ones. At the start of gameplay, the user will choose a board from the premade boards and the board will be passed to the GameController.

3.3. Model Subsystem



Model Subsystem has seven classes which are:

1- Cell

This class is used to create boards and blocks. Cell objects can be filled and visible. Each cell object has a color which is set when isFilled is true.

2- Block

Block is used to fill cells in the board. Each block is composed of 3x3 cell array. This class has an enum called blockShapes to keep 10 different shapes for blocks.

3- Timer

Timer class is used to keep time during the game. getTime() function uses startTime() and endTime() functions to calculate the elapsed time after the game begins.

4- User

User class keeps score and name of the user. Username is set at the beginning of the game and score of the user is set based on the elapsed time when the game ends.

5- Leaderboard

Leaderboard class is used to keep the users whose scores are in top ten. MAX_SIZE is an attribute of leaderboard class to initialize the size of the leaderboard array which is 10.

6- Board

Board class is composed of 20x20 cell objects. Each board object has a text directory which keeps the text files for shape and level of the board, color for each cell and number of blocks used to fill the board.

7- Game

Game has board, user, timer and leaderboard and boardlist objects. It has methods to update the leaderboard, to set the board to the one the user chose, to check whether the game is over, and to encolour the cells of the board when a block is placed on them.

4. Low-Level Design

4.1. Object Design Trade-Offs

4.1.1. Understandability vs. Functionality

We decided to keep the game simple enough to reach out to a wide audience that includes kids. Therefore, we aimed to make the game as straightforward as possible. So even though we provide some extra functionality compared to the original game, we limited these functionalities so that the game is not confusing for anyone.

4.1.2. Memory vs. Maintainability

Abstraction was the main approach during our design, it allowed our design to be much more organized and therefore more understandable (for developers). This simplification comes at a cost though, our abstractions will probably result in redundant memory usage compared to a functional decomposition-based system.

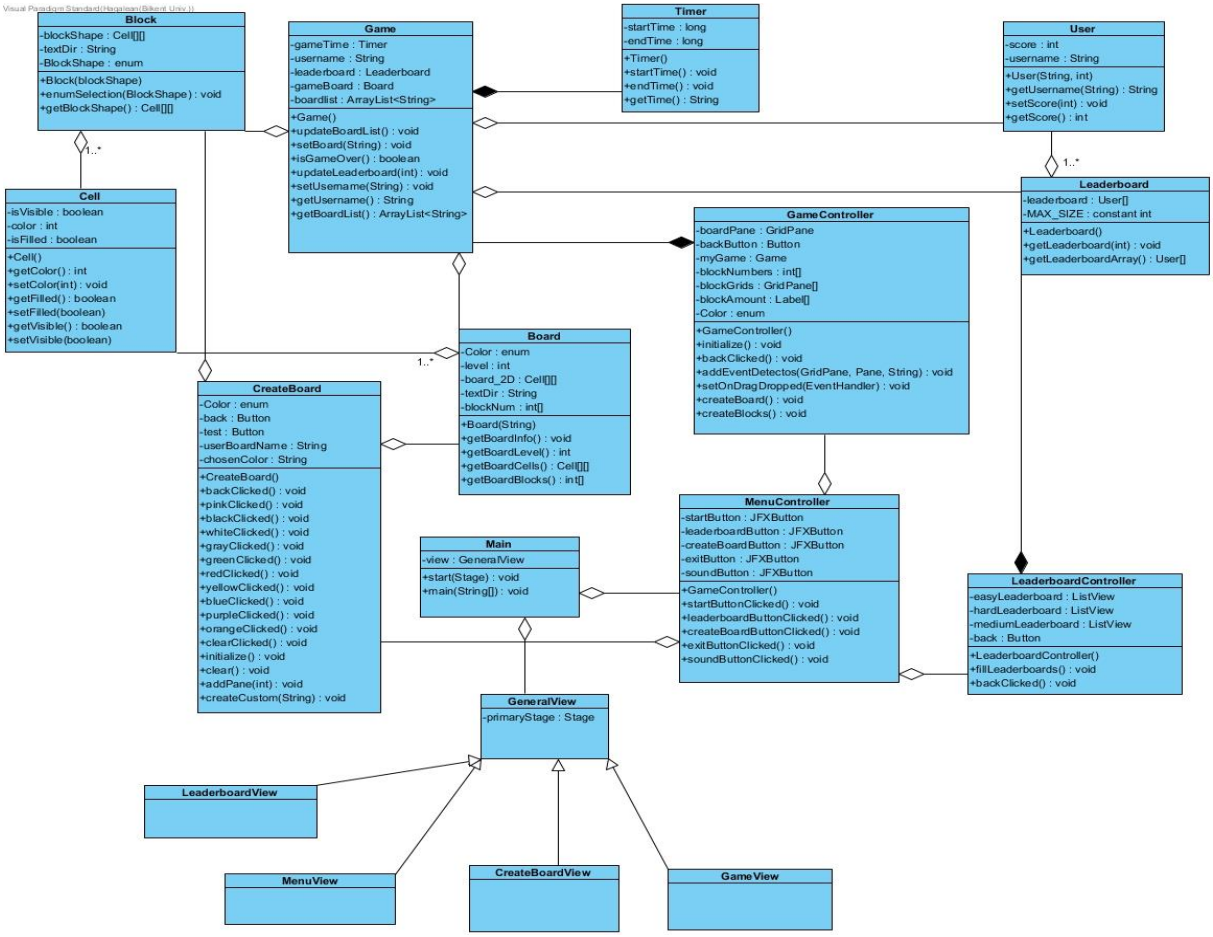
On the other hand, maintainability was a priority for us. We wanted our program to be easy to maintain; we aimed to design a system that could evolve with technology and user needs. To make these possible we had to make our design easy to follow, this meant sacrificing negligible amounts of memory.

4.1.3. Development Time vs. User Experience

Our aim is to cater to a wide audience and to achieve that we need to prioritize understandability; the user experience and the user interface plays a crucial role in the understandability of a system. Therefore, our goal while designing this program was to create an interface that was pleasing to look at and easy to navigate. Many interface

libraries can power systems that are easy to navigate but visuals are also key elements of the user experience. We did not want to subject our users to robotic interfaces, we wanted something that would be pleasing and comforting to look at. Therefore, we never considered libraries like SWING and decided to use JAVAFX as our interface library. We believe that JAVAFX will allow us to create the elegant experience we desire. This elegance comes at a cost of course, it will take longer to develop a program using JAVAFX, fine tuning the experience to make it as smooth as possible will take even more time.

4.2. Final Object Design



4.3. Packages

4.3.1. Internal Packages

4.3.1.1. User Interface Package

User Interface package corresponds to the "view" part of our MVC design. It contains different views for the main parts shown to the user which are game, leaderboard, menu and create board parts of the game. They inherit a general view.

4.3.1.2. Controller Package

This subsystem corresponds to the "controller" part of our MVC design. It controls the model classes depending on the flow of the game.

4.3.1.3. Model Package

Model package contains the models of the classes, which corresponds to the "model" part of our MVC design. It has the components of the game such as blocks, boards, cells and timer.

4.3.2. External Packages

4.3.2.1. Java.Util

We will use this library package which includes a lot of important classes. We will use some Arraylists at some point to collect names. Also, we need a timer to keep

duration of the game. This package will provide Timer class. EventListener is another essential class for us. Game will get inputs from user mostly by this class.

4.3.2.2. JavaFX

"JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms."[1] JavaFX will provide all graphical user interface parts of the game. The visual pages such as menu, leaderboard, game will use classes in JavaFX library package. We will use these classes for inputs from the user, layouts, images, events and animations.

4.4. Class Interfaces

4.4.1. MenuController Class

Attributes:

- **private JFXButton startButton:** A button for starting the game.
- **private JFXButton leaderboard:** A button for displaying leaderboard.
- **private JFXButton createBoardButton:** A button for board creation part of the game.
- **private JFXButton exitButton:** A button for exit.
- **private JFXButton soundButton:** A button for turning on and turning off the sound.

[1] "Release: JavaFX 2.2.21." *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, 14 Mar. 2013, docs.oracle.com/javafx/2/overview/jfxpub-overview.htm.

Constructor:

- **MenuController ():** A default constructor to initialize the MenuController class to handle user input.

Methods:

- **private void startButtonClicked ():** When the start button is clicked, game initializes and starts.
- **private void leaderboardButtonClicked ():** When the leaderboard button is clicked, program displays the leaderboard page.
- **private void createBoardButtonClicked ():** When the create board button is clicked, program displays the board creation page.
- **private void exitButtonClicked ():** When the exit button is clicked, program stops.
- **private void soundButtonClicked ():** By clicking the sound button user can mute or unmute the sound.

4.4.2. CreateBoardController Class

Attributes:

- **private enum Color:** There are 10 different color for painting the cells.
- **private Button back:** Button for going back to the menu.
- **private Button save:** Button for saving the board created by the user.
- **private String userBoardname:** Name of the board created by the user.
- **private String chosenColor:** Selected color by the user to paint the cell.

Constructor:

- **CreateBoardController ():** A default constructor to initialize the CreateBoard class to handle user input while creating a board and saving the process.

Methods:

- **private void backClicked ():** When the back button is clicked, program goes to the menu.
- **private void pinkClicked ():** When the pink color is clicked, selected color to paint the cell becomes pink.
- **private void blackClicked ():** When the black color is clicked, selected color to paint the cell becomes black.
- **private void whiteClicked ():** When the white color is clicked, selected color to paint the cell becomes white.
- **private void grayClicked ():** When the gray color is clicked, selected color to paint the cell becomes gray.
- **private void greenClicked ():** When the green color is clicked, selected color to paint the cell becomes green.
- **private void redClicked ():** When the red color is clicked, selected color to paint the cell becomes red.
- **private void blueClicked ():** When the blue color is clicked, selected color to paint the cell becomes blue.
- **private void purpleClicked ():** When the purple color is clicked, selected color to paint the cell becomes purple.
- **private void orangeClicked ():** When the orange color is clicked, selected color to paint the cell becomes orange.

- **private void clearClicked ():** When the user clicks on clear button, clear () method is called.
- **private void initializes ():** Initializes the whole createBoard page with all components.
- **private void clear ():** This method is called when the user clicks on the clear button. It clears the painted board.
- **private void addPane (int, int):** This method adds a pane to given coordinates.
- **private void createCustom (String):** This method creates the necessary text file to save the board information.

4.4.3. GameController Class

Attributes:

- **private GridPane boardPane:** This is the grid pane of the board
- **private Button backButton:** Button for going back to the menu.
- **private Game myGame:** A game object for controller.
- **private int [] blockNumbers:** An array for keeping number of blocks for the selected board.
- **private GridPane [] blockGrids:** Grid pane array for blocks.
- **private Label [] blockAmount:** Label array to show the remaining amounts of each block.
- **private enum Color:** Color enum for painting the board.

Constructor:

- **GameController ():** A default constructor to initialize the class so that the user can play the game and the input is processed.

Methods:

- **private void initializes ():** Initializes the whole Game page with all components.
- **private void backClicked ():** When the back button is clicked, program goes to the menu.
- **private void addEventDetectors (GridPane, Pane, String):** This method adds event detectors to both the blocks and the grid cells.
- **private void createBoard ():** This method creates board.
- **private void createBlocks ():** This method creates blocks.
- **public boolean isGameOver ():** This method checks if the game is over on each block inserted.

4.4.4. LeaderboardController Class

Attributes:

- **private ListView easyLeaderboard:** A ListView object for the easy leaderboard to display.
- **private ListView mediumLeaderboard:** A ListView object for the medium leaderboard to display.
- **private ListView hardLeaderboard:** A ListView object for the hard leaderboard to display.
- **private Button back:** Button for going back to the menu.

Constructor:

- **LeaderboardController ():** A default constructor to initialize the LeaderboardController class to fill the leaderboards.

Methods:

- **private void fillLeaderboards ():** This method calls three different leaderboard objects for each level and then it fills the ListView object with leaderboard objects' User objects.
- **private void backClicked ():** When the back button is clicked, program goes to the menu.

4.4.5. Cell Class

Attributes:

- **private int color:** Identifies the color of the cell.
- **private boolean isFilled:** Shows if a cell is occupied by a block or not.
- **private boolean isVisible:** is true if the cell is a part of the fillable board.

Constructor:

- **Cell ():** A default constructor to initialize a cell to be placed on the grid.

Methods:

- **public int getColor ():** Gets the color of the cell.
- **public void setColor(int):** Sets the color of the cell with given int value.
- **public boolean getFilled ():** Gets if the cell is filled or not.
- **public void setFilled(boolean):** Sets the cell filled or not with the given boolean value.
- **public boolean getVisible ():** Gets if the cell is visible or not.

- **public void setVisible(boolean):** Sets the cell visible or not with the given boolean value.

4.4.6. Board Class

Attributes:

- **private int level:** Identifies the hardness level of the board
- **private Cell [] [] board_2D:** A 2D array that keeps the cells that create a complete board.
- **private String textDir:** Keeps the directory of the text file that keeps the user names and their scores.
- **private int [] blockNum:** Array that keeps the blocks used to fill the board.
- **private enum Color:** Color enum for painting the board.

Constructor:

- **Board(string):** A default constructor to initialize the board with the given text file.

Methods:

- **public void getBoardInfo ():** Method for reading the information of the board from text file.
- **public Cell [] [] getBoardCells ():** returns boardCells' 2-D array
- **public int getBoardLevel ():** returns level of the board.

4.4.7. Block Class

Attributes:

- **private enum blockShape:** Each block shape is identified with different integers. This integer holds the identification number of a block.
- **private Cell [] [] blockShape:** A 2D array that keeps the cells that create a block.

Constructor:

- **Block (int):** Initializes the Block object with the specified blockShape.

Methods:

- **public void enumSelection (BlockShape):** It defines ten different block shapes.
- **private Cell [] [] getBlockShape ():** returns shape of the block by 2D cell array.

4.4.8. LeaderBoard Class

Attributes:

- **private User [] leaderBoard:** Array that keeps the User objects.

Constructor:

- **Leaderboard ():** A default constructor to initialize a Leaderboard.

Methods:

- **public void getLeaderboard(int):** Gets leaderboard with the given level from the specified textfile.
- **private User [] getLeaderboardArray ():** Returns the leaderboard array.

4.4.9. Timer Class

Attributes:

- **private long startTime:** Starting time.

- **private long endTime**: End time.

Constructor:

- **Timer ():** A default constructor to initialize the Timer.

Methods:

- **public void startTime ():** takes the current time and put it to startTime.
- **public void endTime ():** takes the current time and put it to endTime.
- **public String getTime ():** Gets elapsed time by subtracting start time from end time and then it formats that time as minutes and seconds.

4.4.10. User Class

Attributes:

- **private int score:** Score of the user.
- **private String username:** Name of the user.

Constructor:

- **User (String, int):** A default constructor to initialize the class with name and score parameters.

Methods:

- **public String getUsername ():** Gets the name of the user.
- **public void setUsername (String):** Sets the name of the user.
- **public void setScore(int):** Sets the score of the user.
- **public int getScore ():** Gets the score of the user.

4.4.11. Game Class

Attributes:

- **private Timer gameTimer:** Holds a GameTimer object.
- **private String username:** Name of the user.
- **private Board gameBoard:** Board of the game.
- **private ArrayList<String> boardList:** Board array list for user selection.

Constructor:

- **Game ():** A default constructor to initialize a Game.

Methods:

- **public void updateBoardList ():** Updates the boardList array.
- **public void setBoard (String):** Sets the selected board with given board name.
- **public boolean isGameOver ():** This method checks if the game is over or not.
- **public void updateLeaderboard(int):** This method updates the leaderboard with given score.
- **public void setUsername (String):** This method sets the username with the given String parameter.
- **public String getUsername ():** Returns the username.
- **public ArrayList<String> getBoardList ():** Returns the updated ArrayList.

5. Improvement Summary

- We updated our design report according to the written feedback we received. We clarified ambiguous parts of our report.
- We decided on numeric, quantitative values for our design goals so that they were measurable.
- We added a deployment diagram that shows the interactions of our subsystems.

- We removed some unnecessary classes like SoundController.

- We updated our opaque layering diagram to better reflect our current code. We also added arrows to our opaque layering diagram to better represent the relationships between parts of our system.

- We added a deployment diagram to our subsystem decomposition to increase understandability.

6. Glossary & References

1. "Release: JavaFX 2.2.21." *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, 14 Mar. 2013, docs.oracle.com/javafx/2/overview/jfxpub-overview.htm.