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 eveybody 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 diiferent 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. 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.
- **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. Usability is highly prioritized, this means keeping the functionality limited.

- **Security:** We aim to keep the game offline (the leaderboards are offline too) therefore we do not have any security concerns.

2. High Level Software Architecture

2.1. Subsystem Decomposition

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, Management and Model. These main packages directly refers View, Controller and Model packages in MVC(Model – View – Controller) system design pattern. We decided to use this pattern for an easy implementation stage.

Inside these packages there are smaller packages. We used opaque layering (as seen in Figure 1) to decompose our system as we believe in its understandability. These packages are determine based on their functionalities.

- User Interface package has Menu GUI and Game GUI packages as smaller packages. These packages provides a graphical interface for users.

- Management Package has three different smaller packages. These are Game Management, File Management and Board Management packages. The main goal of this package is make the game successfully work.
- Model Package has Board Components and Game Components. These packages are generally the objects of the game.

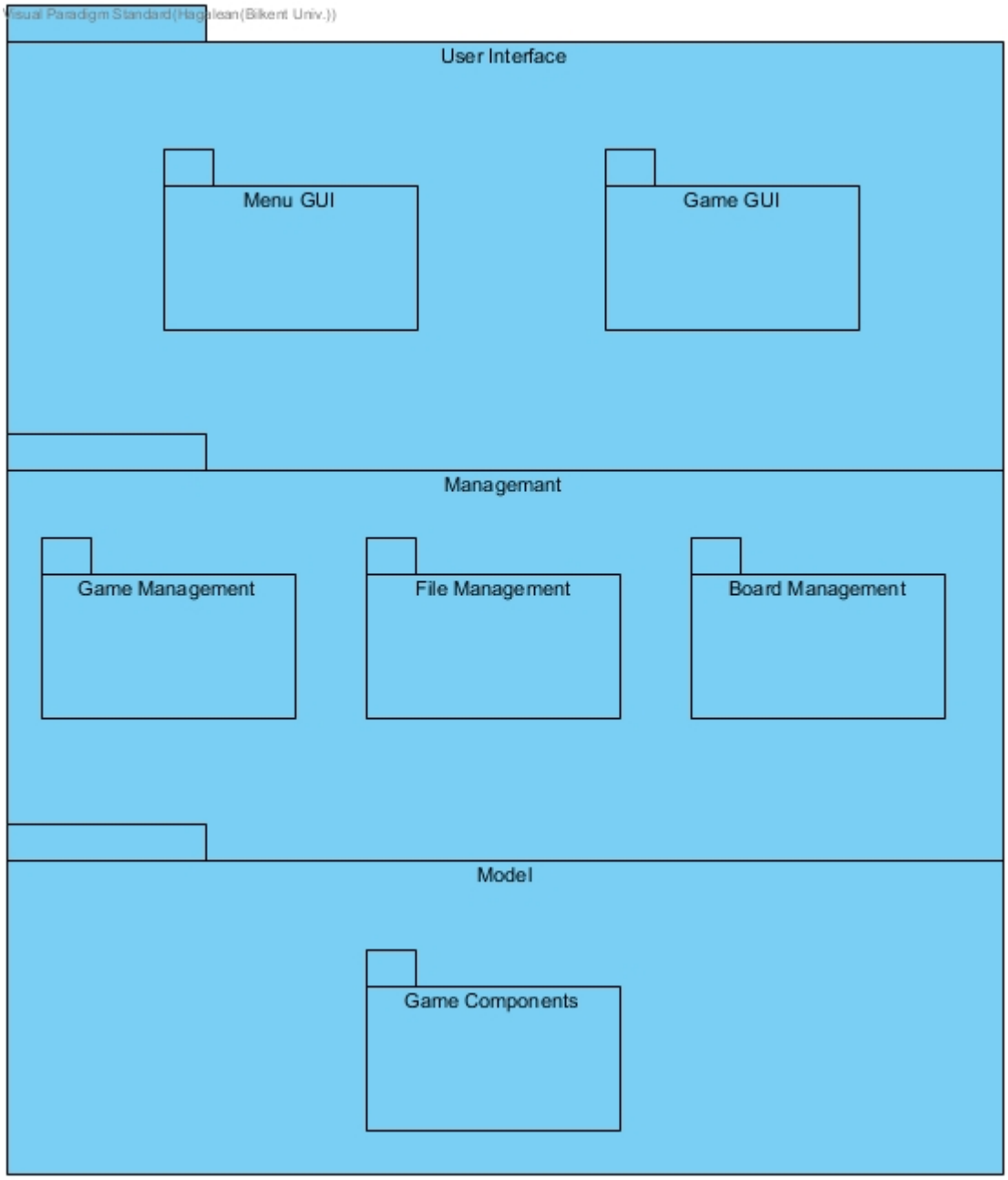


Figure 1: Opaque layering in Katamino.

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 .gif 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. 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.

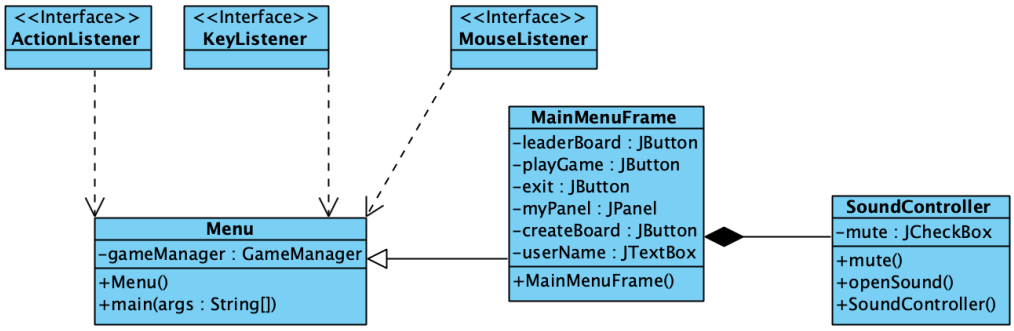
2.5. Boundary Conditions

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.

3. Subsystem Services

3.1. UserInterface Subsystem



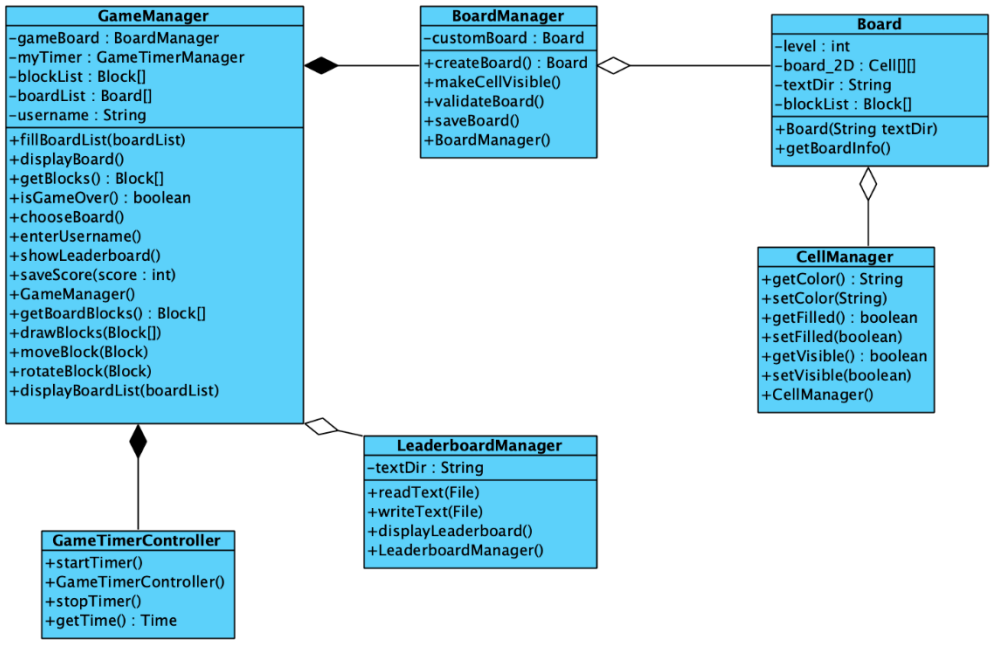
UserInterface Subsystem has two packages which are:

1. Menu GUI
2. Game GUI

Menu GUI provides a menu screen for the user when the game begins. There are three classes inside this package. Menu, MainMenuFrame, and SoundController classes are responsible for creating a page which has buttons, text boxes, and sound control area. Katamino does not have any other control system so the user can check a box to mute or unmute the game. In menu screen user is able to look for the leaderboard, play a new game, create a board, enter a username or exit from the game.

Game GUI provides cells, blocks and a board for the game. It has a ScreenManager class which is responsible for creating these items by getting their information from GameManager class. Game GUI involves all screens except the menu screen.

3.2. Management Subsystem



Management Subsystem has three packages which are:

1. Game Management

Game Management is responsible for controlling the game by providing board and time information for the system. GameManager class keeps the time by using GameTimer and GameTimerController classes. When the user starts a new game and selects a board, GameManager 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.

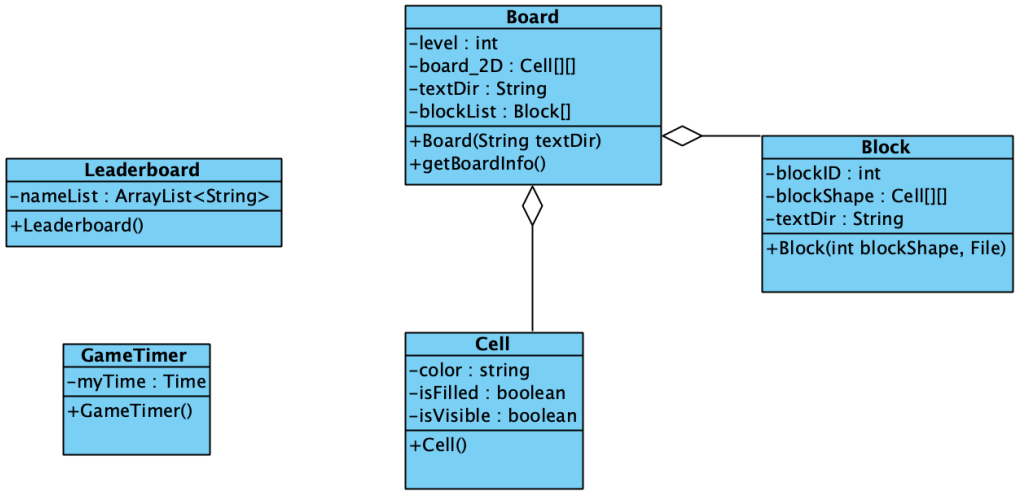
2. File Management

File Management is keystone of Katamino. Board and leaderboard information are kept in text files. LeaderboardManager class is responsible for reading and writing the data for leaderboard of the game. Board class keeps the board information inside a text file and when user wants to create a new board and saves it, BoardManager class writes new data to this text file.

3. Board Management

Board Management is responsible for managing the custom boards and the premade boards. BoardManager 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 making specific cells visible. Then, the user tries to fill the newly created board with blocks. After this step, when the user specifies that the block placement is finished, BoardManager will validate if all the visible cells are filled with blocks. 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 GameManager.

3.3. Model Subsystem



Model Subsystem has a component called Game Component. It involves different types of game objects such as blocks, board, cells and a game timer. The game components listed are going to be used for the gameplay and are going to be instantiated and used when they are needed. The Model Subsystem is the Model section of the MVC and contains the boundary objects of the systems.

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

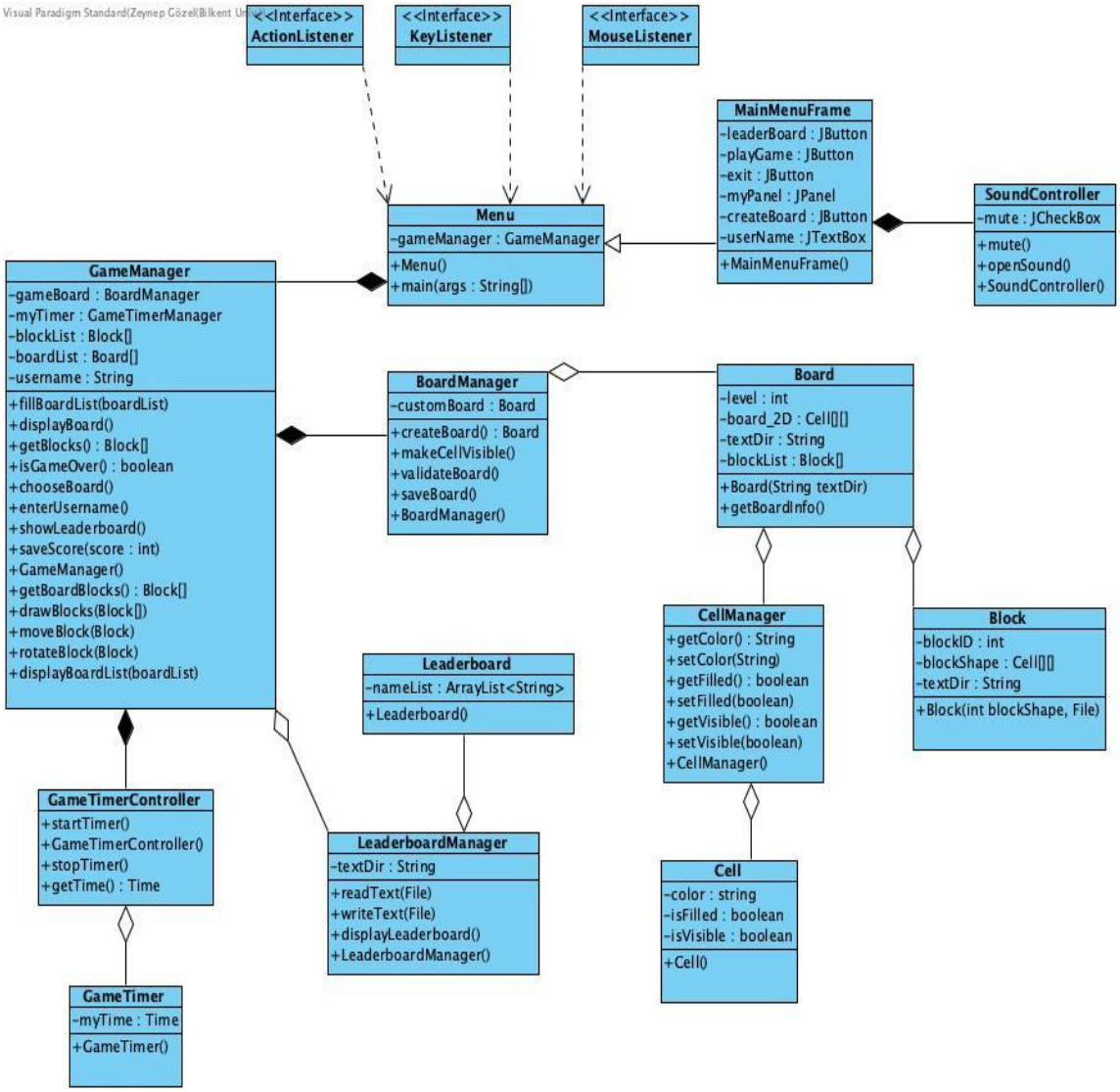
Abstraction was the main approach during our design, it allowed our design to be much more organized and therefore more understandable(for developers). This simplification comes at a cost though, our abstractions will probably result in redundant memory usage compared to a functional decomposition based system. But as of late 2018 memory is abundant and we don't expect our system to require great amounts of memory therefore memory was not a high priority for us. 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 which we had no problem with.

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 are able to 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. Menu GUI Package

This package includes all parts of the menu section such as classes and methods for representation of the menu page.

4.3.1.2. Game GUI Pacakge

This package includes all parts of the game section such as classes for representation of the game board and blocks.

4.3.1.3. Game Management Package

This package includes classes for managing the game. The control of the is provided by this package.

4.3.1.4. File Management Package

This package includes classes for getting information from storage and managing them.

4.3.1.5. Board Management Package

This package includes classes for managing the board of the game.

4.3.1.6. Game Component Package

This package 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 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. GameManager Class

Attributes:

- **private Board gameBaord:** This attribute provides a playable board according to user's selection.
- **private GameTimer myTimer:** A timer object to keep duration of the game for leaderboard.
- **private Block [] blockList:** An array for keeping blocks. Initially, blockList is full of all blocks used in all boards.
- **private Board [] boardList:** An array for keeping boards. Initially, boardList is full of all boards, including the boards that the user creates.
- **private String username:** is assigned to the username input of the player.

Constructor:

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

Methods:

- **private Board[] fillBoardList():** A method for filling boardList array with all boards that are in our json file.
- **private void displayBoardList(boardList : Board[]):** A method for displaying all boards in boardList array.
- **private void displayBoard():** A method for displaying the selected board to the user.
- **private Block [] getBlocks():** A method for returning all blocks.
- **private boolean isGameOver():** This method checks whether the game is over or not.
- **private void chooseBoard():** Player chooses a board from the displayed board list.
- **private void enterUsername():** A method to get username from the player.
- **private void showLeaderboard():** Displays the leaderboard.
- **private void saveScore(int score):** This method saves the time of the player for leaderboard.
- **private Block[] getBoardBlocks():** This method gets the blocks of gameBoard.
- **private void drawBlocks(Block []):** This method gets needed blocks from getBoardBlocks method and displays it to the user.
- **private void moveBlock(Block):** By using MouseListener interface, user can drag and drop blocks by this method.
- **private void rotateBlock(Block):** User can rotate a block to change its direction by MouseListener.

4.4.2. Cell Class

Attributes:

- **private String 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 the class.

4.4.3. CellManager Class

Constructor:

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

Methods:

- **private String getColor():** Returns the color of the cell.
- **private void setColor(String):** Sets the color of the cell by taking a string parameter.
- **private boolean getFilled():** Return true if isFilled is true.
- **private void setFilled(boolean):** Sets isFilled to the value of the boolean parameter.
- **private boolean getVisible():** Returns true if isVisible is true.
- **private void setVisible(boolean):** Sets isVisible to the value of the boolean parameter.

4.4.4. 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 Block[] blockList:** Array that keeps the blocks used to fill the board.

Constructor:

- **Board(File):** A default constructor to initialize the class that takes a file as a parameter.

Methods:

- **private String getBoardInfo():** Method for returning boardInfo.

4.4.5. BoardManager Class

Attributes:

- **private Board customBoard:** Keeps a Board object to manage it.

Constructor:

- **BoardManager():** A default constructor that uses create board method to create an empty board.

Methods:

- **private Board createBoard():** This method creates an empty board with fixed dimensions and invisible cells.

- **private void makeCellVisible():** This method makes the chosen cells visible to the player when creating a board. MouseListener is used to select the cell to be made visible in the empty board.
- **private void validateBoard():** This method makes the unfillable cells of the board inaccessible to the user, so that only the specific shape can be filled, not the whole rectangle.
- **private void saveBoard():** This method saves the board created by the user to the boardList.

4.4.6. Block Class

Attributes:

- **private int blockID:** 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.
- **private String textDir:** Keeps the directory of the text file that keeps the shapes of the blocks.

Constructor:

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

4.4.7. LeaderBoard Class

Attributes:

- **private ArrayList<String> nameList:** Array that keeps the names of the users.

Constructor:

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

4.4.8. LeaderBoardManager Class

Attributes:

- **private String textDir:** Keeps the directory of the text file that keeps the user names and their scores.

Constructor:

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

Methods:

- **private void readText(File):** This method reads the contents of the File that keeps leaderboard information and saves it to nameList.
- **private void writeText(File):** This method updates the File by adding the new user name and score if the score is high enough to get into the top ten list.
- **private void displayLeaderboard():** This method prints the leaderboard info kept in name list.

4.4.9. GameTimer Class

Attributes:

- **private Time myTime:** Time object to track the elapsed time.

Constructor:

- **GameTimer():** A default constructor to initialize myTimer as null.

4.4.10. GameTimerController Class

Attributes:

- **private GameTimer myTime:** Holds a GameTimer object.

Constructor:

- **GameTimerController():** A default constructor to initialize myTime.

Methods:

- **private void startTimer():** This method starts myTime.
- **private void stopTimer(File):** This method stops myTime.
- **private Time getTime():** This method returns myTime object.

4.4.11. SoundController Class

Attributes:

- **private JCheckBox mute:** JCheckBox that indicates whether the sound is open or not.

Constructor:

- **SoundController():** A default constructor that initializes the sound as open.

Methods:

- **private void mute():** Mutes sound when mute attribute is selected.
- **private void openSound():** Opens sound when mute attribute is not selected.

4.4.12. MainMenuFrame Class

Attributes:

- **private JButton leaderboard:** Button to see leaderboard.

- **private JButton playGame:** Button to start a new game.
- **private JButton exit:** Button to exit game.
- **private JButton createBoard:** Button to create a new board.
- **private JTextBox:** Text box that takes username input.
- **private JPanel myPanel:** Panel to hold buttons and text box.

Constructor:

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

4.4.13. Menu Class

Attributes:

- **private GameManager gameManager:** Holds a GameManager object to play the game, or to show leaderboard or to create a newBoard.

Constructor:

- **Menu():** A default constructor that initializes the sound as open.

Methods:

- **public void main(args: String[]):** Main method to take actions based on ActionListener, KeyListener and MouseListener.

5. 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.

2. “Package Java.util.” *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, 23 June 2018, docs.oracle.com/javase/7/docs/api/java/util/package-summary.html.

3. “Lesson: Object-Oriented Programming Concepts.” *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, docs.oracle.com/javase/tutorial/java/concepts/index.html.