Bilkent University

Department of Computer Engineering

CS319
Object Oriented Software Engineering
Project Design Report

Katamino v2.0

Group 2-F

Sena Er - 21502112

Zeynep Sonkaya - 21501981

Hüseyin Orkun Elmas - 21501364

Utku Görkem Ertürk - 21502497

Bartu Atabek - 21602229

Table of Contents

1. Introduction

1.1 Purpose of the system

Katamino v2.0 is a 2D brain training, puzzle game. It is designed in a way that lets the player work on increasing their problem solving abilities by continuously simulating it with puzzles and also give the player satisfaction to encourage the player to keep playing. Katamino v2.0 is based on the original Katamino game version, but we also add more functionalities to the game. Katamino v2.0 is a 2D desktop game with 3 different modes: multiplayer, single player classical and single player custom. Single player classical mode includes a level-based system, similar to the original Katamino. Katamino v2.0 also includes a custom mode to enable different board shapes for players. We also add multiplayer functionality to make users compete and make the game more interesting. Also, to encourage players, Katamino v2.0 includes score system in single player games, which players can also compete in the high score table.

Our design and our architecture will consist of subsystems to meet all functional requirements in our analysis and design goals that are explained in the upcoming section.

1.2 Design goals

We prefer designing the system before implementation, therefore we first establish our criteria and trade-offs for the design. These design goals will also guide our choices in the implementation process. Below, each of our design goals is detailed.

1.2.1 User Criteria

Ease of Learning

Our game will include a tutorial for any player who want to learn about the game. The design of the game graphics and controllers will be intuitive, the game will be easy to learn.
To further increase the ease of learning, we will include a tutorial in the start of first game of a user. This tutorial will cover interacting with the game. The tutorial will be interactive to increase the understanding of the player. After the tutorial, a new player will be able to play the game.

User - Friendliness

This objective is also relevant to our previous point, ease of learning. Katamino will have an user friendly interface design which will enable players to easily understand control mechanisms such as dragging or rotating a pentomino. In order to increase the user friendliness of the system, we tried to make the general game flow and the control

mechanisms similar to games that are commonly played in our game design. Thus, we aim to increase the ease of adapting to the game play. The game controls are heavily based on mouse actions with addition of a few key controls. For example, dragging is initiated first by clicking on the pentomino to select one, then, holding whilst moving the mouse which is moving the pentomino.During holding of pentomino via mouse click,rotating it with “A” to left,with “D” to left and flipping it with W and S is possible and chosen intentionally because of the common use of those keyboard controls in many games. These set of actions are basic and standard in the arcade genre of games

.

Another design choose in our game to achieve a high level of user friendliness is having a brief tutorial in our final version which teaches the players how to play when they first open the game.

Moreover, to increase the user friendliness, we received feedback from our target audience and improved our interface in this direction.

Performance

Performance in games have crucial value, in our design of the Katamino v2.0, players will not wait for a long time when they want to perform a specific action such as saving, loading user or moving pentominoes. The average response time will be less than 1 seconds on the test computer[1]. To achieve this, we will make use of better algorithms to rotate and flip the pentominoes. To increase the efficiency in runtime, we combined responsibilities in a couple of classes, We are using a design, where gui interactions are seeded to the model of the class. So that we keep momentarily game decisions done in GUI data on the controller instead of getting data from the model. That approach has increased our performance, because it eliminated data transfer between objects.

Portability

Playing the game across platforms such as MacOs, Linux or Windows is an important feature we want to have to increase the audience of our game. In order to achieve higher portability we use chose Java language and made use of the JVM.

Also in our source code, we did not make any assumptions on the operating system of the player to prevent any limitations.

1.2.2 Maintenance Criteria

Robustness

Players should play our game without bugs that locks down the system that disrupt the flow of the game. To achieve this goal, we will review the user’s interactions with the game through user interface that is designed for determining actions even though player does not perform the full action such as, dragging the pentominoes to exact locations on the game

[1] Test computer’s specifications are;
Intel Core i7-7500U CPU @2.70 GHz, 8GB RAM, Windows 10 x64.

board, if the player does not drag a pentomino to an exact cell on board, our system will estimate the desired location and place the pentomino there.

Moreover, our system will not allow players to put multiple pentominoes on top of each other and rotation of pentominoes that are on the edges of the system, since rotating them might lead to parts of the pentomino outside of the board. These are done to decrease potential bugs due to a more complex system.

Additionally, our design is up to handle exceptions in a safe manner without reflecting to the player.

Maintainability

Change is frequent in software engineering; therefore, we prioritized maintainability. Game's subsystems have a hierarchical structure so that new features can be added easily without any maintenance problem. Katamino v.2.0 designed to allow modification of the source code without having side effects. We will make use of the loose coupling principle, to make connected classes more independent and therefore prevent them getting affected by a alteration in the other one.

In order to increase maintainability of our system, we chose a 3-layer closed architecture to achieve maintainability. Since, each layer can only depend on the layer immediately below it, coupling is reduced between layers. Thus, a change in one layer will cause less change in other layers.

We also use the MVC design pattern to increase maintainability in our system. This decreases the coupling between classes and distributes the responsibilities to different classes. Therefore, modifications on one class has reduced impact.

Reuse of Components

Our components are designed to increase reuse of components. We created components that are used in multiple places and decreased the code duplication. For example we have a game board structure where we can use for every mode of the game: namely, single custom, single classic and multiplayer mode. Thus we do not have to design different game boards for each type of the game,our components are reusable.

Furthermore, we have draggable cells which can be pentomino cells, empty cells and blocked cells. So we don't need to rewrite same piece of code for different types of cells.
Moreover, we reuse code for game controllers through inheritance in the system.

1.2.2 Trade-offs

Functionality vs. Usability

One of the main target audience of Katamino v.2.0 is children, considering the preferences of target audience, we focus on ease of usability in the cost of functionality. To increase usability we prefer simpler interfaces and interactions between the user and the system.

Cost vs. Robustness

Katamino is a software that cost only developers time and robustness of the game is highly important for players. We prefer robustness over cost of our time to give the user better experience with the product.
In order to make a more robust software, we made regular tests of our game and corrected bugs. We also had other people play our game to find the bugs we might missed.

Efficiency vs. Portability

Portability is more important than efficiency in our design to increase portability of Katamino v2.0, we chose to use Java and use the advantages of JVM being runnable on many different operating systems.

Cost vs. Reusability

We want our game and its parts to be reusable in other systems. To increase reusability we use MVC pattern, which separates the responsibilities of classes. Which can be used more easily if needed in other systems.

In order to increase reusability, we prefered low coupling between subsystems, so that classes can be reused in our system and other systems, without the need to make many changes.

1.3 Definitions

Java Virtual Machine (JVM): An abstract computing machine for running a Java program

Model View Controller (MVC)**:** Separation of model, view and controller. Design pattern of Katamino. Model and controllers are explicitly defined in our project. Views are handled by .fxml files so that in this report there is no view classes.

Graphic User Interface (GUI): Form of Katamino user interface that allows users to interact with electronic devices through graphical icons and visual indicators

Java Development Kit (JDK): The full-featured SDK for Java.

Java Runtime Environment (JRE): A package of everything necessary to run a compiled Java program, including the Java Virtual Machine (JVM), the Java Class Library, and other infrastructures.

JavaFX: A software platform for creating and delivering desktop applications, as well as rich Internet applications that can run across a wide variety of devices. JavaFX make us to use .fxml files to manage view.

2. High-level software architecture

2.1 Subsystem decomposition

In this section, we will decompose Katamino system into subsystems, and group subsystems into layers to increase flexibility and maintainability of our design. We plan to increase the extendibility of our system by using layers of subsystems for later modifications. We first give the explanation of our subsystem decomposition and then provide the component diagram that represents our subsystem decomposition.

Our design consists of 3 layers, namely; GUI layer, Game Layer and Data Layer. We use a closed architecture model; in which, each layer can only access the layer immediately below it. We chose a 3 layer architecture to maximize maintainability. 3 layer architecture provides us low coupling between layers, thus when one component in one layer needs to be changed, the effect of this change is kept at a minimum on the whole system. Furthemore, We use three layer style architecture as presentation, application and data, to closely follow MVC pattern and increase flexibility and maintainability of our code.

In GUI Layer is responsible for all GUI tasks, such as displaying the game, displaying menus and taking inputs from the user. GUI Layer has two subsystems: Game GUI Manager and Menu GUI Manager. Game GUI Manager is responsible for managing graphical user interface in the game such as displaying board, pentominoes and movements; Menu GUI Manager is responsible for menu interfaces such as main menu, single player classical menu and other possible decisions that can be taken during interaction with user. In some cases, the GUI manager will propagate these inputs to the Game Layer, specifically to the Game manager subsystem. GUI manager is the top layer of our architecture, no subsystem depends on the GUI Manager layer.

In Game Logic Layer, the movements of pentominoes, their places on the board and the user input for the menus are processed by corresponding controllers. Game Manager subsystem, is responsible for controlling movements of pentominoes when called by the gui layer and updates the Game Objects Manager subsystem. Menu Manager subsystem is the collection of menu controllers, it is also called by the GUI layer, and controls the input from the GUI layer and communicates with the data layer. The game object management is responsible for managing game related data such as player,game boards and stopwatch. This subsystem is used by Game Manager subsystem, whenever there is a change in the data such as change of player data or change of board data. This will be an inner representation of the data in the game, Game Objects Subsystem will use Data Layer to save and load data when necessary.

In the Data Layer subsystem’s purpose is to manage all of the game and player data to be saved and read from save files for reloading latest game of a play and access highscores. We also use the Data Layer to load boards. Board files are used to accomplish load predetermined board qualities functionality. We do not allow to write data for modifying game

boards for reliability purposes. Data Layer is the bottom layer in our 3 layer architecture, it does not depend on any other layer.

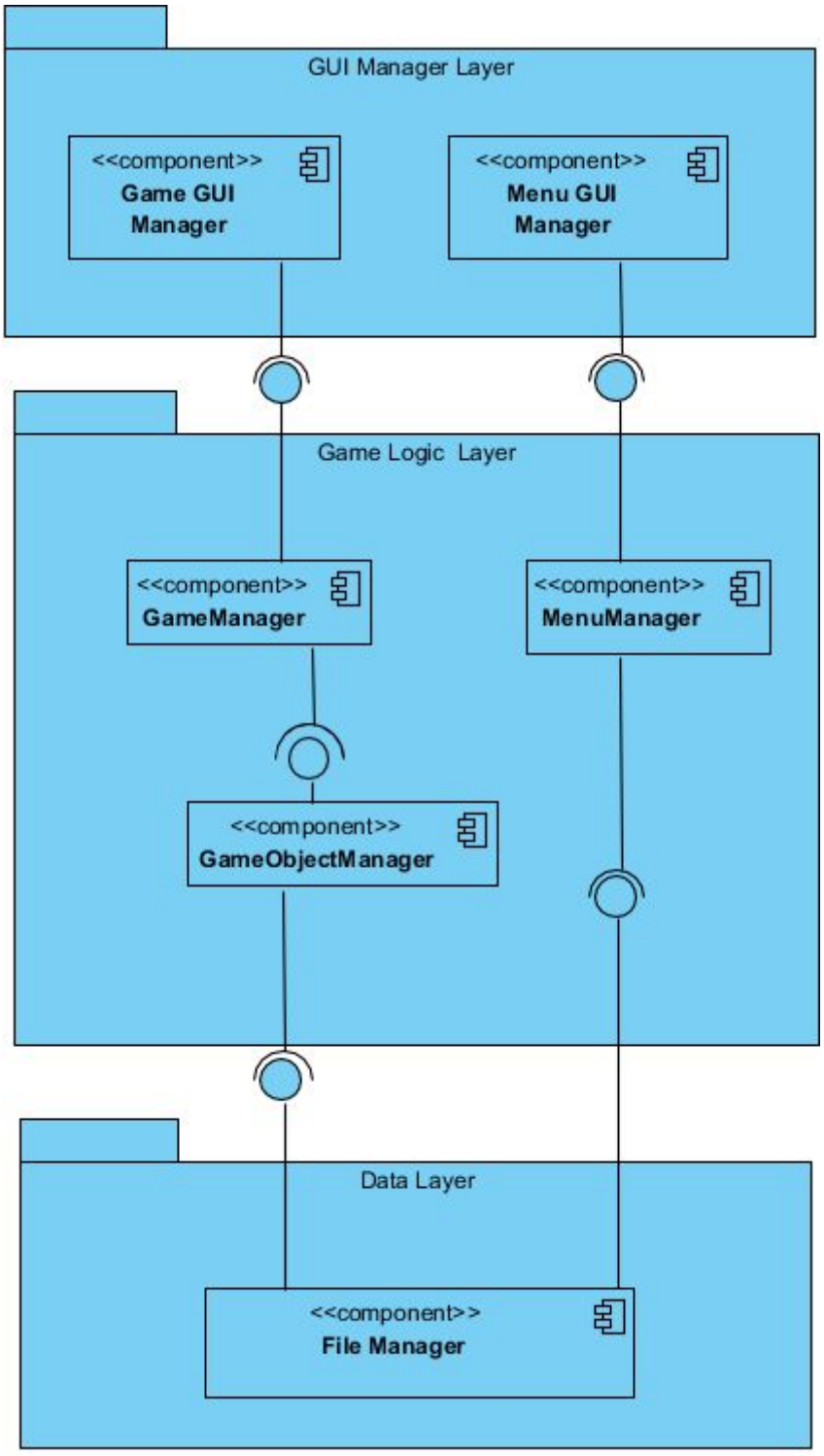


Figure 1: High level software architecture showing GUI Manager, Game Logic and Data Layers

2.2 Hardware/software mapping

Katamino v.2.0 game will be implemented in Java 8, and we will use JDK 1.8+. We will use JavaFX libraries to implement GUI. JRE 1.8 is required to run the game. By using JRM, our game will be able run in Linux, Mac and Windows environments. Additionally, a mouse and keyboard will be enough to interact with the game. GUI Manager subsystem needs a mouse to get input for Game and Menu GUI Managers. Also, a keyboard hardware is necessary for Menu GUI Manager to get input about the Player identity. Data Manager subsystem is working with operating system for file management that occur during saving and loading game. Figure 2 shows that whole system works on the same machine. Relationship between subsystems will be explained in detail on the system decomposition part.

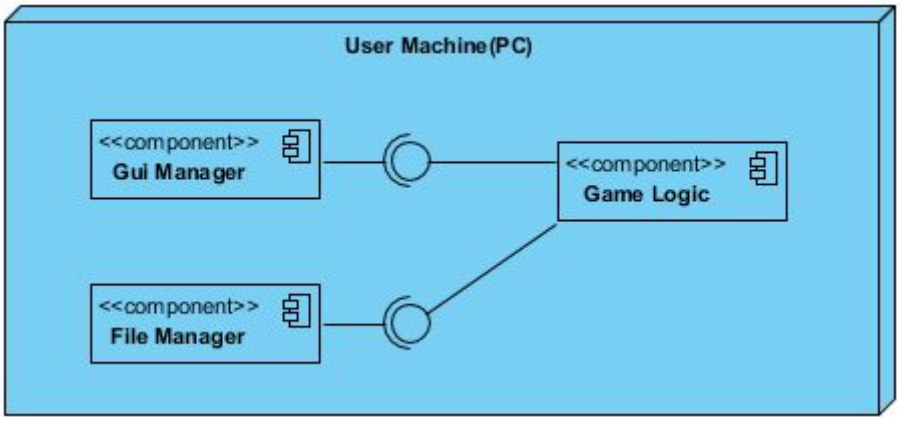


Figure 2: Deployment Diagram showing the hardware mapping of the software

2.3 Persistent data management

In Katamino v2.0, we save user data in a text file with JSON format in local storage. We use gson library of java. The save will include themes, player names and their level,highest score of players,information related to latest game of all player to continue the latest game a certain player played also file system contains all levels with their board information.
We chose to use file system for persistent data management because our data has low intensity, since we only save some board information and score information per user. And also using a file system decreased the complexity of our software.

2.4 Access control and security

Katamino v2.0 does not handle any valuable information about the user, and the multiplayer side of the game is local. Therefore, we do not prioritize security. We only store user names to serve multiple of people on one computer. Also, there is only one actor which is the player who can reach all the functionality in the game.

2.5 Boundary conditions

2.5.1 Initializing Katamino v2.0

Players will use an executable file to start Katamino v2.0. Main menu will be shown without any requirements. After selecting the single player mode, if the folder that contains player information does not exist or is corrupted, then the game creates that file, and game initialization continues, but player saves will be lost .
If any of the player save files is removed then the player won't be shown the players list. Additionally, if the selected player's save file is corrupted, then the game will show an error message, loading will fail and initialization will continue from player selection menu, also corrupted file will be removed.

Other than those cases where a single game mode selected, the system will read selected player's data, and initialize game.

When multiplayer game mode selected, system will initialize the game by reading the multiplayer board from the filesystem. If this file does not exist or is corrupted then the game will show an error message and return to main menu.
When high score displaying selected in main menu, the file contains high score data in local is removed or corrupted, loading will fail and an error message will be displayed. After displaying the error, initialization will continue from main menu also corrupted file will be removed.

Another initialization failure occurs when a current Katamino.exe is running and another Katamino.exe is attempted to be started. In this case initialization of second Katamino will not be allowed and cause an error message displaying. After message, system will shut down the second Katamino window.

2.5.2 Shutdown

Players will be able to exit from "Exit game" button on the main menu. They can also exit game by using the "X" mark on the window. In both cases the game will automatically save before exiting.

2.5.3 Error Behavior

If the game cannot load boards from board files because of a corruption, the game will show an error message, and ask the user to select another board.

Another error can occur when the player data is changed during the current player game. If the folder of the current player is deleted, the game will update player data on the disk with the data on the game.

When the system has a runtime error while player is inside a gameplay, the system will first save the current state of the game to enable loading when the player reopens the game.

3. Subsystem services

3.1 GUI Manager Layer

This layer contain two subsystems as Menu GUI Manager and Game GUI Manager.

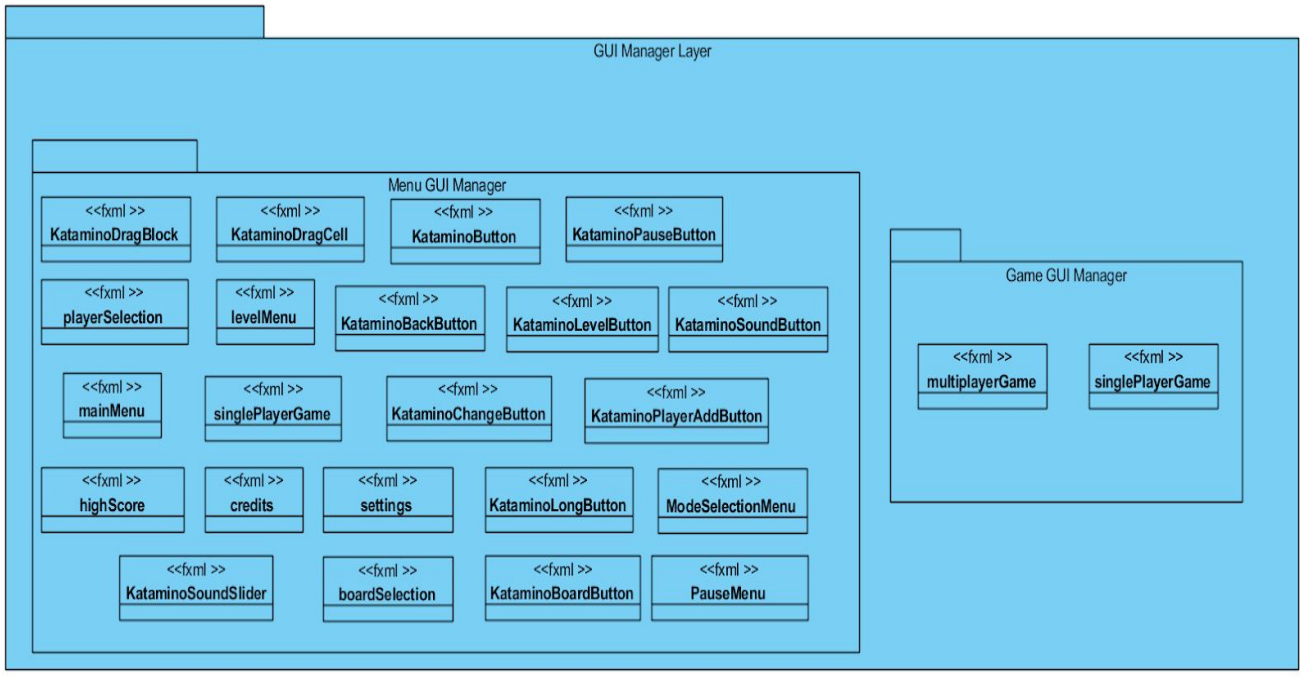


Figure 3: GUI Manager Layer with included subsystems.

3.1.1 Game GUI Manager

GUI of the game handled by fxml files and internal libraries. Fxml files are connected to controller classes to bind user input to game logic layer.

3.1.2 Menu GUI Manager

This Layer consist of Menus, settings, credits, scoreboard and special Katamino components. Menus have direct interaction with users thought fxml files which bind the input to game logic layer, specifically to Menu Manager Subsystem, for Controller and Model classes. Components in these layer are the basic fxml files that manage the directions for multiple menus and game objects that has a special view and used in menus.

3.2 Game Logic Layer

Game Logic Layer has 3 subsystems Game Manager,Game Object Manager and Menu Manager.

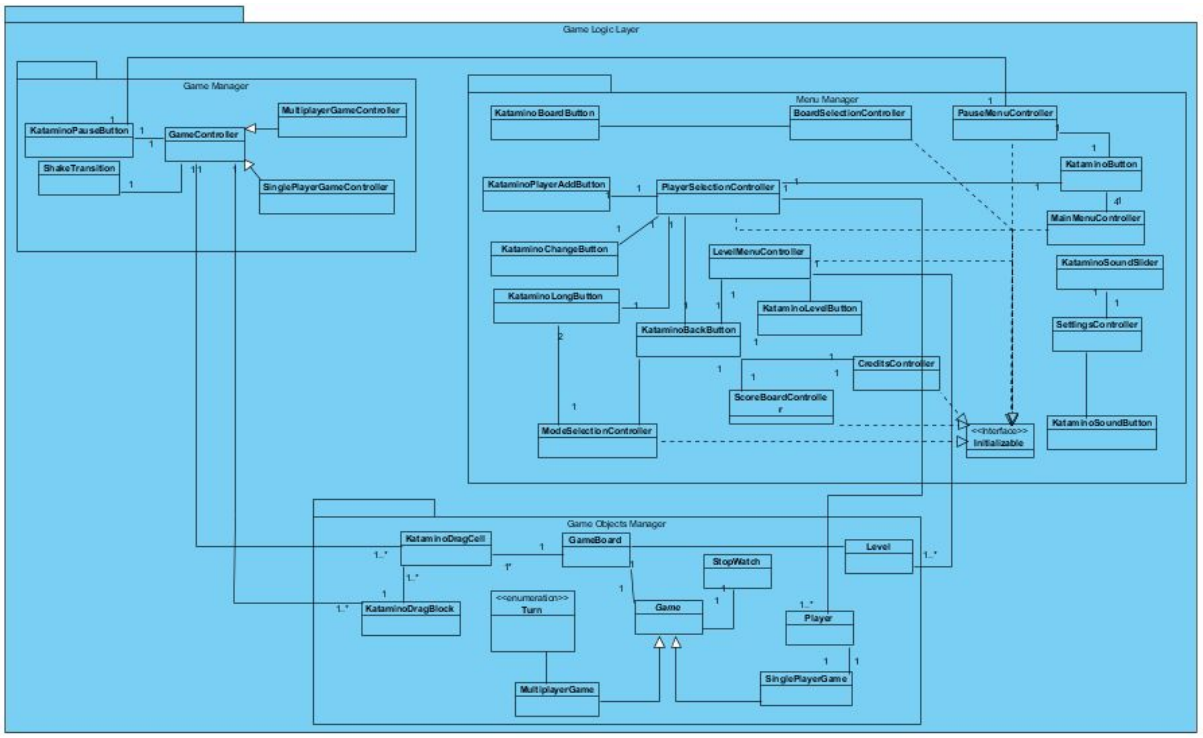


Figure 4: Game Logic Layer Overall Class Diagram

3.2.1 Game Manager

Game Manager is responsible subsystem for controllers of the real game, Katamino v2.0. It uses the models in Game Object Manager subsystem.

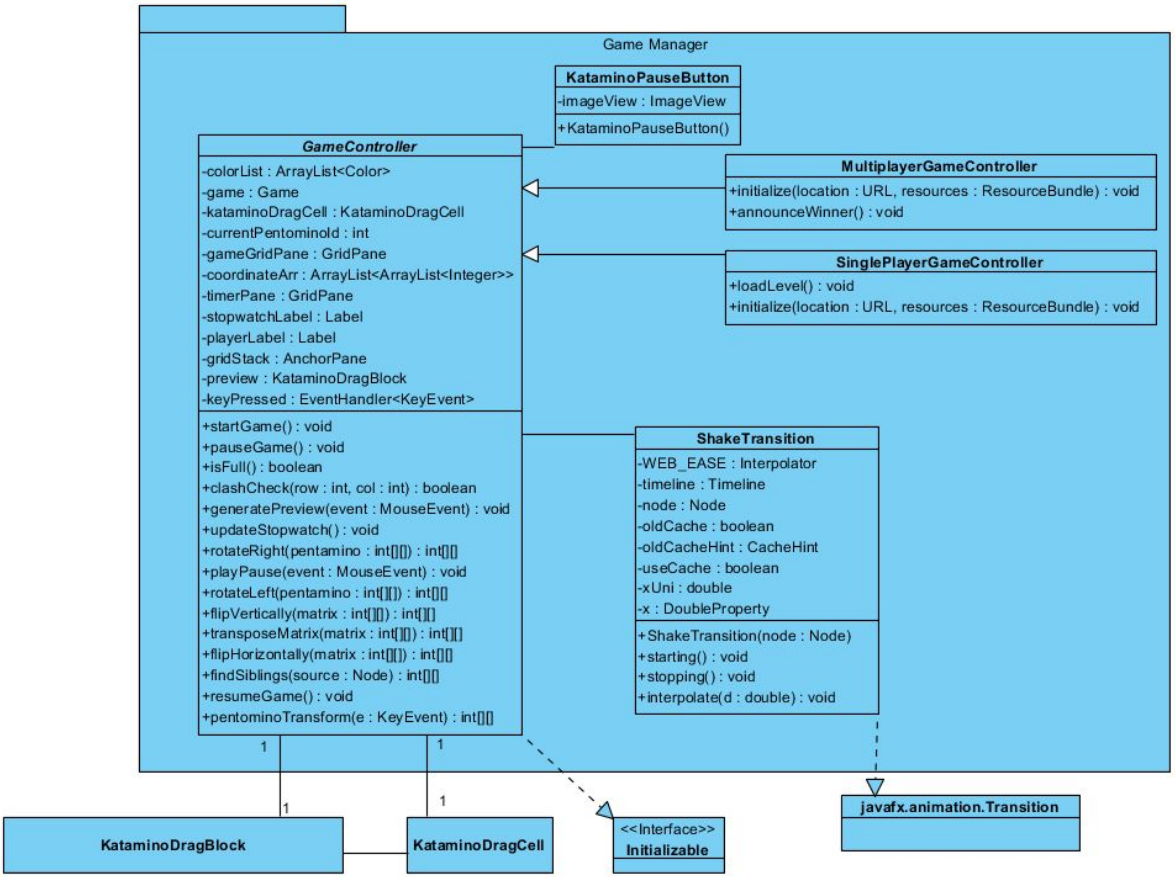
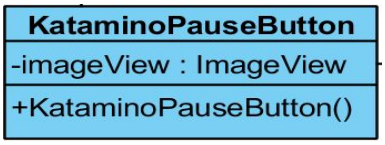


Figure 5: Game Manager subsystem

KataminoPauseButton



Attributes:

private ImageView imageView : This is the image of the KataminoPauseButton.

Constructor:

public KataminoBackButton(): This constructor connects fxml and controller.

GameController

Attributes:

private ArrayList<Color> colorList: This attribute holds twelve distinct colors that represents the twelve different pentomino tiles.

private Game game: This attribute holds the game logic information that is used throughout the game and is an instance of the Game model class. It is responsible for the data such as the game board, player, score etc. and transfer its to the controller for the interface visualization.

private KataminoDragCell kataminoDragCell: This attribute is used during the game logic operations such as dragging and clash checking. It represents the source node of the pentomino tile that has been operated.

private int currentPentominoId: This attribute holds the pentomino id of the source node of the pentomino tile that has been operated which represents the shape of the pentomino tile and is used for identification purposes during game operations.

private ArrayList<ArrayList<Integer>> coordinateArr: This attribute is the transformed representation the pentomino tile inside the game grid into as a 2D matrix during the game operations and is used for the calculating the sibling tiles of the pentomino.

private GridPane timerPane: This attribute represents the grid pane in the fxml document used as an UI element.
private GridPane gameGridPane: This attribute represents the game grid which constructed by individual drag cells which the gameBoard in the fxml document used as an UI element.
private Label stopwatchLabel: This UI element represents the stopwatch in the game UI and shows the elapsed time in seconds to the player.
private Label playerLabel: This UI element represents the name of the current player that is playing the game.
private AnchorPane gridStack: This attribute represents the Anchor Pane in the fxml document used as an UI element to combine different UI layers during drag operations.
private KataminoDragBlock preview: This attribute displays the dragged pentomino block as a preview on top of the game board as a preview.
private EventHandler<KeyEvent> keyPressed: This attribute represents the keyboard key press event used during the rotation and flip of the pentomino tiles.

Methods:
public void startGame(): This function starts the game by starting the stopwatch after the level is loaded to the game board.
public void pauseGame(): This function pauses the stopwatch and the game. While paused the player is unable to play the game.
public void resumeGame(): This function is used for resuming the game from where it was and also resumes the stopwatch.
public boolean isFull(): This function checks all the cells inside the gameboard that are identified by the “onBoard” property and returns true if all the tiles are placed in the gameboard with no empty cells else it returns false.
public boolean clashCheck(int row, int column): This function checks if there are any clashes when dropping a pentomino tile to a new place on the grid. It checks the weather the new position of the tile is occupied or not and if it is occupied it returns true and does not place the tile and also gives a visual warning to the player.
public void generatePreview(MouseEvent event): This function generates a transparent preview of the selected pentomino tile during the game operations and gives the player a visual tool to position the tile on the game board. Figure 6 is an example of both moving and placed pentomino depending of the perspective.

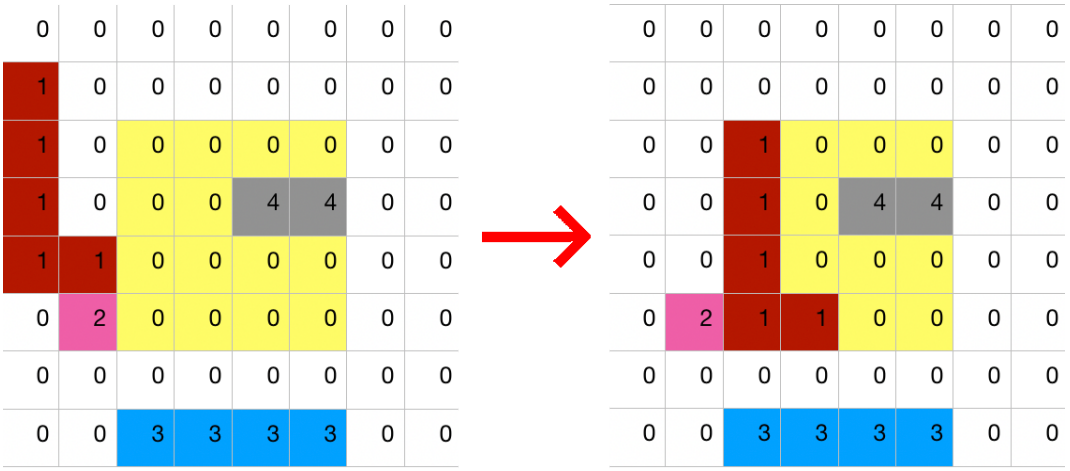


Figure 6: Moving and Placing a Pentomino

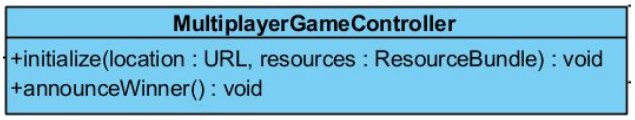
public void updateStopwatch(): This function updates the stopwatch label in the UI by calling the getElapsedTime function from the game model during the gameplay.
public int[][] rotateRight(int[][] pentomino): This function rotates the selected pentomino tile during the drag to 90° right and visually represents the rotated tile as a preview.
public int[][] rotateLeft(int[][] pentomino): This function rotates the selected pentomino tile during the drag to 90° left and visually represents the rotated tile as a preview.
public int[][] flipVertically(int[][] matrix): This function flips the selected pentomino tile during the drag vertically and visually represents the flipped tile as a preview.
public int[][] flipHorizontally(int[][] matrix): This function flips the selected pentomino tile during the drag horizontally and visually represents the flipped tile as a preview.
public int[][] transposeMatrix(int[][] matrix): This function is used for the game logic operations during the rotation and flip of the pentomino tile that returns the transpose of the inputted 2D matrix which represents the pentomino tile.
public int[][] pentominoTransform(KeyEvent e): This function is responsible for the flip and rotate operations in the gameplay and calls these functions according to the pressed key.
public int[][] findSiblings(Node source): This function is used for the game logic operations such generating the preview. It returns the sibling tiles of the same pentomino node as a 2D integer array.
public void playPause(MouseEvent event): This function is responsible for the play and pause operations in the gameplay and pauses or resumes the game.

SinglePlayerGameController

Methods:
public void loadLevel(): This function loads the selected level of the player using the FileManager and Level classes which returns a 2D array of the game board from the JSON file and translates it to the game UI by shaping the game board accordingly.

public void initialize(location : URL, resources : ResourceBundle): This functions is called when the singlePlayerGame fxml is loaded. It initializes the game by setting the level, player and prepares the game board for the gameplay.

MultiplayerGameController

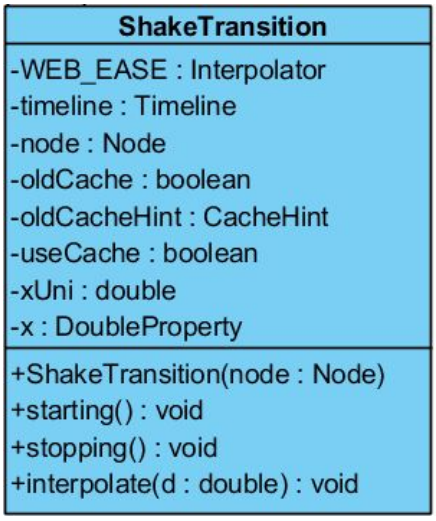


Methods:

public void announceWinner(): This function announces the winner of the multioplayer game when one of the components is out of moves.

public void initialize(location : URL, resources : ResourceBundle): This functions is called when the multiPlayerGame fxml is loaded. It initializes the game by setting the level, players and prepares the game board for the gameplay.

ShakeTransition



Attributes:

private Interpolator WEB_EASE: This attribute represents a spline curve for the ease out animation.

private Timeline timeline: This attribute holds the timeline with predefined keyframes for the shake animation.

private Node node: This attribute represents the node that the animation will be applied to.

private boolean oldCache: This attribute states whether there is a cache for the animation or not.

private CacheHint oldCacheHint: This attribute holds a cache hint used during the cache mechanism.

private boolean useCache: This attribute states whether the stored cache should be used or not.

private double xUni: This property holds the initial x position of the node before the animation begins.

private DoubleProperty x: This property is used during the keyframe setup of the animation.

Consturctor:

public ShakeTransition(Node node): The constructor of the ShakeTransition class takes a node as input and initializes the node attribute of the class.

Methods:

public void starting(): This function is called when the animation is starting.
public void stopping(): This function is called when the animation is stopping.
public void interpolate(double d): This function controls the animation and takes a double value as input and interpolates the animation accordingly.

3.2.1 Menu Manager

This subsystem contain Menu Controllers and classes used in these controllers which are used for making the game more visual than a normal javafx game. Those are specialized javafx buttons generally.

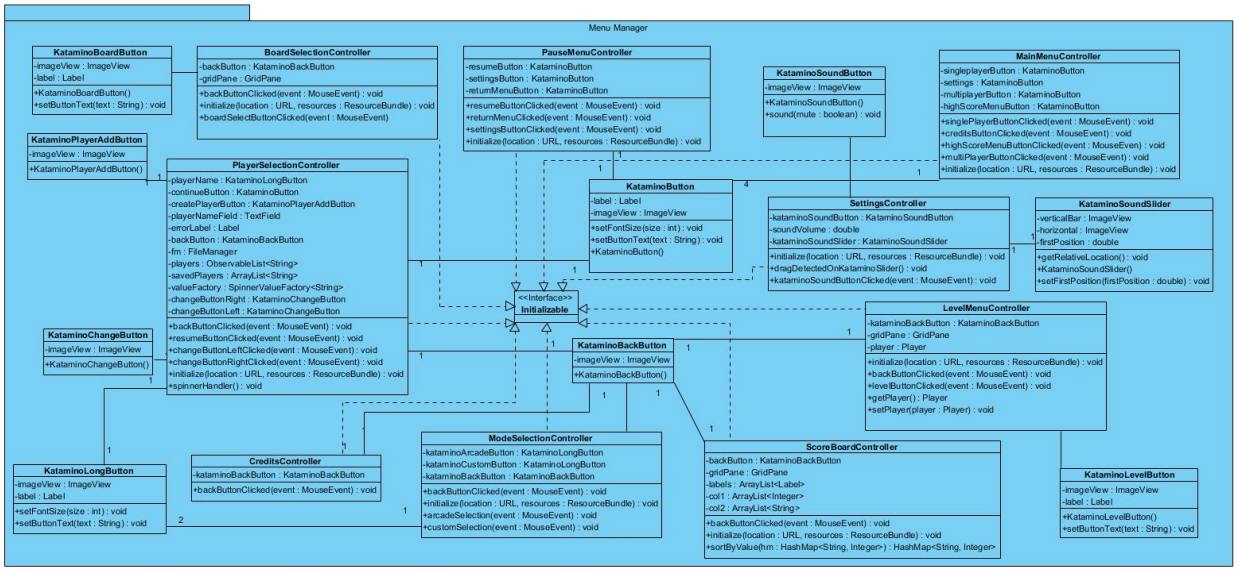
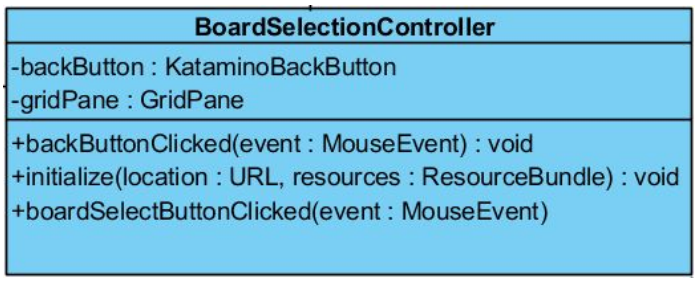


Figure 7: Menu Manager subsystem

BoardSelectionController

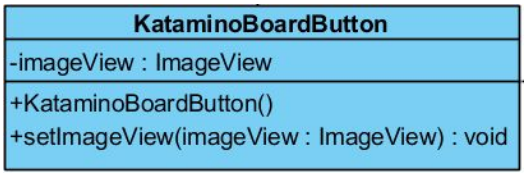


Attributes:

private KataminoBackButton backButton : This is the back button that has gui features.
private GridPane gridPane: This is the main container of the BoardSelectionController.

Methods:
public void backButtonClicked(MouseEvent event): This method is invoked by board selection controller fxml with MouseEvent when the user clicks the backButton. It changes the scene to the main menu.
public void boardButtonClicked(MouseEvent event) : This function is invoked by board selection fxml with MouseEvent when the user clicks the boardButton. It changes the scene to the multiplayer game.
public void initialize(URL location, ResourceBundle resources): This method is called when the board selection fxml is loaded. Moreover, all boardButtons' images are setted.

KataminoBoardButton



Attributes:
private ImageView imageView : This is the image of the KataminoBoardButton .

Constructor:
public KataminoBoardButton(): This constructor connects fxml and controller.

Methods:
public void setImageView(ImageView imageView): This sets the image of KataminoBoardButton.

KataminoPlayerAddButton

Attributes:
private ImageView imageView : This is the image of the KataminoPlayerAddButton.

Constructor:
public KataminoPlayerAddButton(): This constructor connects fxml and controller.

PlayerSelectionController

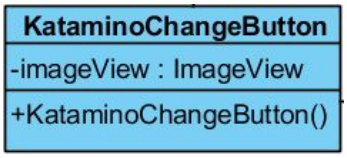
Attributes:

private KataminoLongButton playerName: This button shows the name of the player.
private KataminoChangeButton changeButtonLeft : This is the left change button for players.
private KataminoChangeButton changeButtonRight : This is the right change button for players.
private TextField playerNameField : This is the text field that user enters player name.
private KataminoButton continueButton : This is the continue button with gui elements.
private KataminoPlayerAddButton createPlayerButton : This is the player add button.
private Label errorLabel : This label keep error message when player name is not valid(empty or clashing names). Shows “Invalid Player Name!”.
private KataminoBackButton backButton : This is the back button that has gui features.
private FileManager fm : Filemanager fetches the players from the players.json that has whole players in it.
private ObservableList<String> players : This contains player names that are taken from savedPlayers in the form of observable list.
private ArrayList<String> savedPlayers : This contains player names which fetched from file manager.
private SpinnerValueFactory<String> valueFactory : This is the spinner factory for spinner (KataminoLongButton) takes players.

Methods:

public void backButtonClicked(MouseEvent event) : This function is invoked by player selection fxml with MouseEvent when the user clicks the backButton. It changes the scene to the main menu.
public void changeButtonLeftClicked(MouseEvent event) : This method changes the text of the label to previous player.
public void changeButtonRightClicked(MouseEvent event) : This method changes the text of the label to next player.
public void spinnerHandler(): This handler converts (make them act) KataminoLongButton and KataminoChangeButtons to spinner.
public void createButtonClicked(MouseEvent event): When this method is invoked, it saves new player to the player.json using filemanager.
public void continueButtonClicked(MouseEvent event): This function is invoked by player selection fxml with MouseEvent when the user clicks the backButton. It changes the scene to the single player game.
public void initialize(URL location, ResourceBundle resources): This method is called when the player selection fxml is loaded. It initializes attributes of the class.

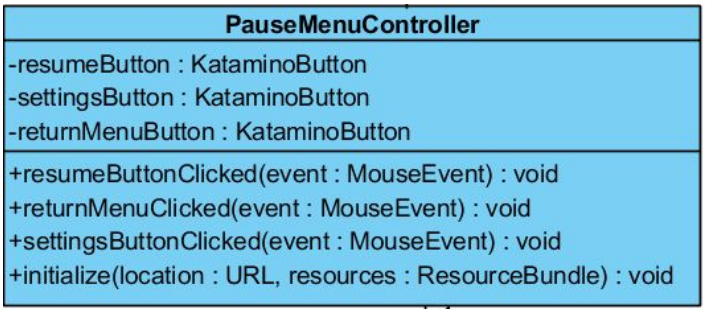
KataminoChangeButton



Attributes:
private ImageView imageView : This is the image of the KataminoChangeButton.

Constructor:
public KataminoChangeButton() : This constructor connects fxml and controller.

PauseMenuController



Attributes:
private KataminoButton resumeButton: This is the resume button that has GUI features.
private KataminoButton settingsButton: This is the settings button that has GUI features.
private KataminoButton returnMenuButton: This is the return menu button that has GUI features.

Methods:

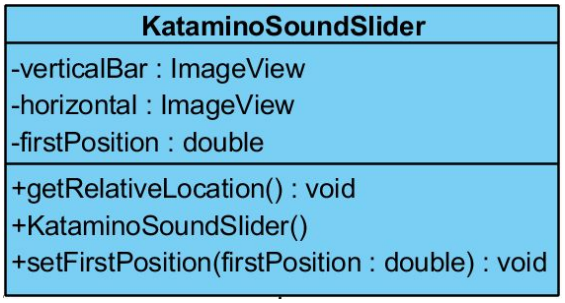
public void resumeButtonClicked(MouseEvent event) : This function is invoked by pause menu fxml with MouseEvent when the user clicks the resumeButton. It changes the scene to the current game screen.

public void settingsButtonClicked(MouseEvent event) : This function is invoked by pause menu .fxml with MouseEvent when the user clicks the settingsButton. It changes the popup pane to the settings.

public void returnMenuClicked(MouseEvent event) : This function is invoked by pause menu fxml with MouseEvent when the user clicks the returnMenuButton. It changes the scene to the main menu.

public void initialize(URL location, ResourceBundle resources) : This functions is called when the pause fxml is loaded. It sets the labels of the KataminoButtons.

KataminoSoundSlider



Attributes:

private ImageView horizontal: This is slider button which is dragged over vertical bar.

private ImageView verticalBar: This is the vertical bar that

private double firstPosition: This is the current position of the slider.

Constructor:

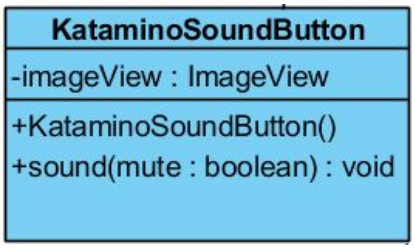
public KataminoSoundSlider(): This constructor connects fxml and controller. This constructor also sets drag events to bar.

Methods:

public double getRelativeLocation(): This function returns relative location of the slider button out of 100.

public void setFirstPosition(double firstPosition): This method sets the current position of the slider button.

KataminoSoundButton



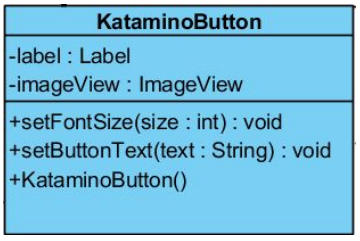
Attributes:

private ImageView imageView : This is the image of the KataminoSoundButton.

Constructor:
public KataminoSoundButton(): This constructor connects fxml and controller.

Methods:
public void sound(boolean mute) : This method changes the imageView in terms of mute value if the mute is true, imageView will be set to mute icon. Otherwise, imageview will be set to unmute icon.

KataminoButton

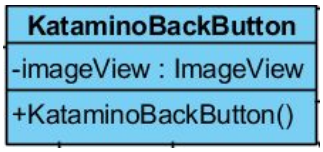


Attributes:
private ImageView imageView : This is the image of the KataminoButton.
private Label label : This is the label of the KataminoButton.

Constructor:
public KataminoButton(): This constructor connects fxml and controller.

Methods:
public void setFontSize(int size): This changes the current font size of the KataminoButton.
public void setButtonText(int size): This method sets the label text of the KataminoButton.

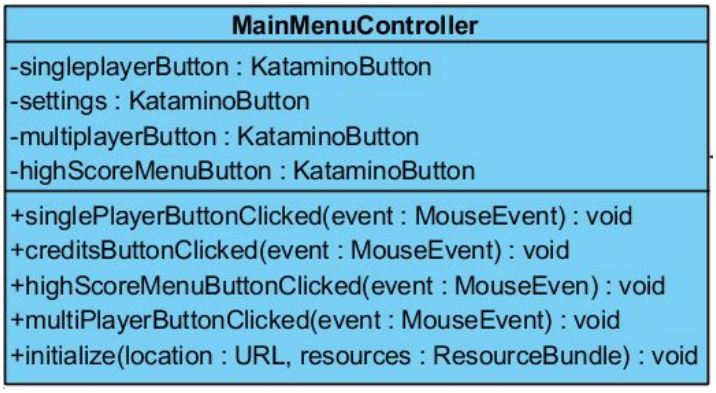
KataminoBackButton



Attributes:
private ImageView imageView : This is the image of the KataminoBackButton.

Constructor:
public KataminoBackButton(): This constructor connects fxml and controller.

MainMenuController



Attributes:

private KataminoButton singleplayerButton : This is the singleplayer button that has gui features.

private KataminoButton settings : This is the settings button that has gui features.

private KataminoButton multiplayerButton : This is the multiplayer button that has gui features.

private KataminoButton highScoreMenuButton : This is the highScoreMenu button that has gui features.

Methods:

public void singlePlayerButtonClicked(MouseEvent event): This function is invoked by main menu fxml with MouseEvent when the user clicks the singlePlayerButton. It changes the scene to the mode selection menu.

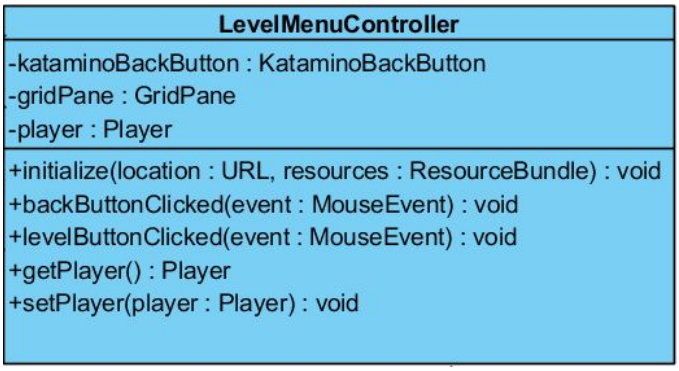
public void creditsButtonClicked(MouseEventevent): This function is invoked by main menu fxml with MouseEvent when the user clicks the creditsButton. It changes the scene to the credits.

public void highScoreMenuButtonClicked(MouseEvent event): This function is invoked by main menu fxml with MouseEvent when the user clicks the highScoreMenuButton. It changes the scene to the highscore.

public void multiPlayerButtonClicked(MouseEvent event): This function is invoked by main menu fxml with MouseEvent when the user clicks the multiPlayerButton. It changes the scene to the board selection menu.

public void initialize(URL location, ResourceBundle resources) : This functions is called when the main menu fxml is loaded. It sets the labels of the KataminoButtons.

LevelMenuController



Attributes:

private KataminoBackButton kataminoBackButton : This is the back button that has gui features.

private GridPane gridPane : This is the container of KataminoLevelButtons.

private Player player : This is the current player of the game.

Methods:

public void backButtonClicked(MouseEvent event) : This function is invoked by level menu fxml with MouseEvent when the user clicks the backButton. It changes the scene to the mode selection menu.

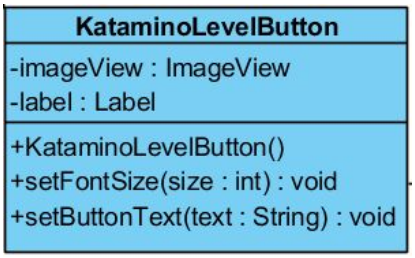
public void initialize(URL location, ResourceBundle resources) : It creates KataminoLevelButtons and set their names.

public void levelButtonClicked(MouseEvent event): This function is invoked by main level menu fxml with MouseEvent when the user clicks a levelButton. It fetches the corresponding level information from FileManager and changes the scene to the singleplayer game.

public Player getPlayer() : This is the basic function for getting current player.

public void setPlayer(Player player): This is the basic function for setting current player.

KataminoLevelButton



Attributes:

private ImageView imageView : This is the image of the KataminoLevelButton.

private Label label : This is the label of the KataminoLevelButton.

Constructor:

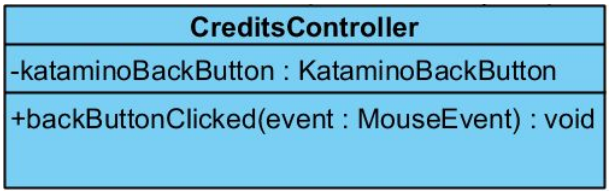
public KataminoLevelButton(): This constructor connects fxml and controller.

Methods:

public void setFontSize(int size) : This changes the current font size of the KataminoLevelButton.

public void setButtonText(String text) : This method sets the label text of the KataminoLevelButton.

CreditsController



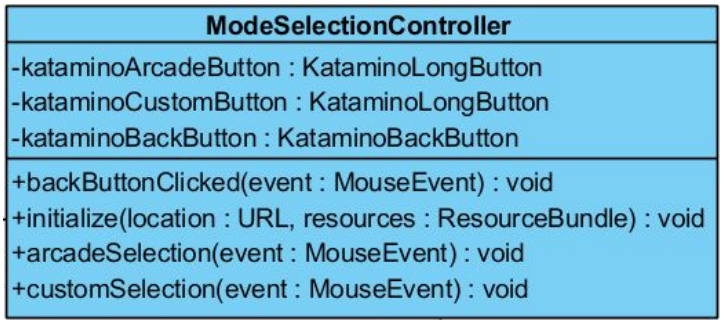
Attributes:

private KataminoBackButton kataminoBackButton : This is the back button that has gui features.

Methods:

public void backButtonClicked(MouseEvent event) : This function is invoked by level menu fxml with MouseEvent when the user clicks the backButton. It changes the scene to the main menu.

ModeSelectionController



Attributes:

private KataminoLongButton kataminoArcadeButton : This is the arcade game button that has gui features.

private KataminoLongButton kataminoCustomButton : This is the custom game button that has gui features.

private KataminoBackButton kataminoBackButton : This is the back button that has gui features.

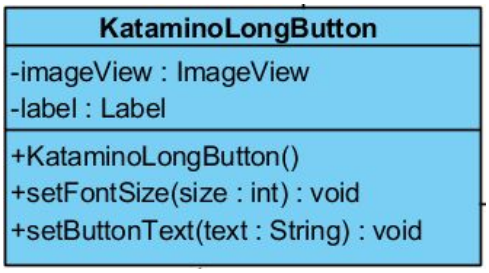
Methods:

public void backButtonClicked(MouseEvent event) : This function is invoked by mode selection menu fxml with MouseEvent when the user clicks the backButton. It changes the scene to the main menu.

public void initialize(URLlocation ,ResourceBundle resources) : This functions is called when the main menu fxml is loaded. It sets the labels of the KataminoLongButtons.

public void arcadeSelection(MouseEvent event) : It passes game mode as arcade and loads player selection fxml.
public void customSelection(MouseEvent event) : It passes game mode as custom and loads player selection fxml.

KataminoLongButton

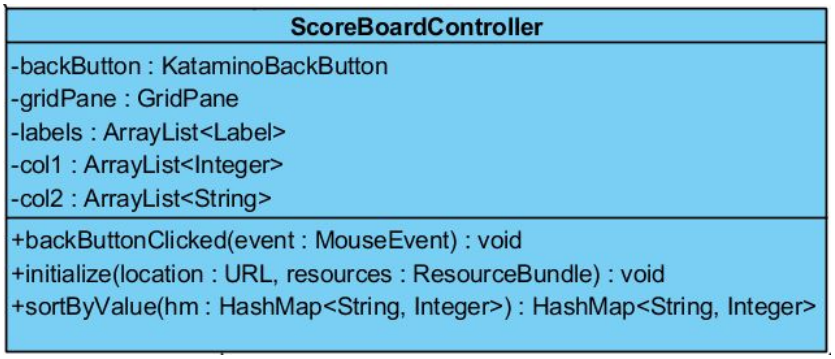


Attributes:
private ImageView imageView : This is the image of the KataminoLongButton.
private Label label : This is the label of the KataminoLongButton.

Constructor:
public KataminoLongButton(): This constructor connects fxml and controller.

Methods:
public void setFontSize(int size) : This changes the current font size of the KataminoLongButton.
public void setButtonText(String text) : This method sets the label name of the KataminoLongButton.

ScoreBoardController

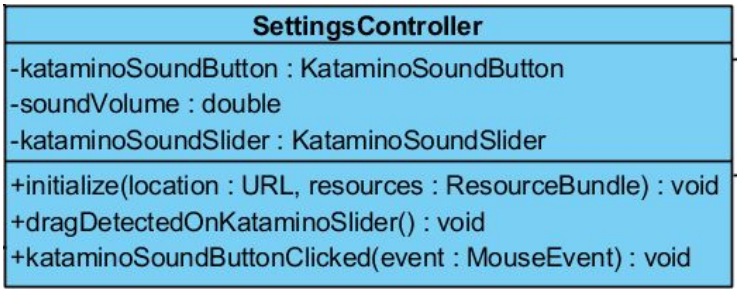


Attributes:
private KataminoBackButton backButton : This is the back button that has gui features.
private GridPane gridPane : This main container for score labels.
private ArrayList<Label> labels : This is the labels that takes player names and their order representation.
private ArrayList<Integer> col1 : This array contains player names.
private ArrayList<String> col2 : This array contains numeration.

Methods:

public void backButtonClicked(MouseEvent event) : This function is invoked by mode selection scoreboard fxml with MouseEvent when the user clicks the backButton. It changes the scene to the main menu.
public void initialize(URLlocation, ResourceBundle resources) : This functions is called when the highscore fxml is loaded. It sets labels in terms of col1 and col2.
public HashMap<String, Integer> sortByValue(HashMap<String, Integer> hm) : This method takes highscores as hashmap and maps them to col1 and col2.

SettingsController



Attributes:
private KataminoSoundSlider kataminoSoundSlider : This is the katamino sound slider that has gui features.
private KataminoSoundButton kataminoSoundButton: This is the katamino sound button that has gui features.
private double soundVolume : This attribute keeps current volume of the sound.

Methods:
public void dragDetectedOnKataminoSoundSlider() : This function increases and decreases volume in terms of relative position function of the katamino sound slider.
public void kataminoSoundButtonClicked(MouseEvent event) : This function triggers the method ,which is sound, of kataminoSoundButton. It will mute or unmute the sound with respect to current state of it.
public void initialize(URL location, ResourceBundle resources) : This functions is called when the settings fxml is loaded. It sets the current position of the slider button according to current sound level.

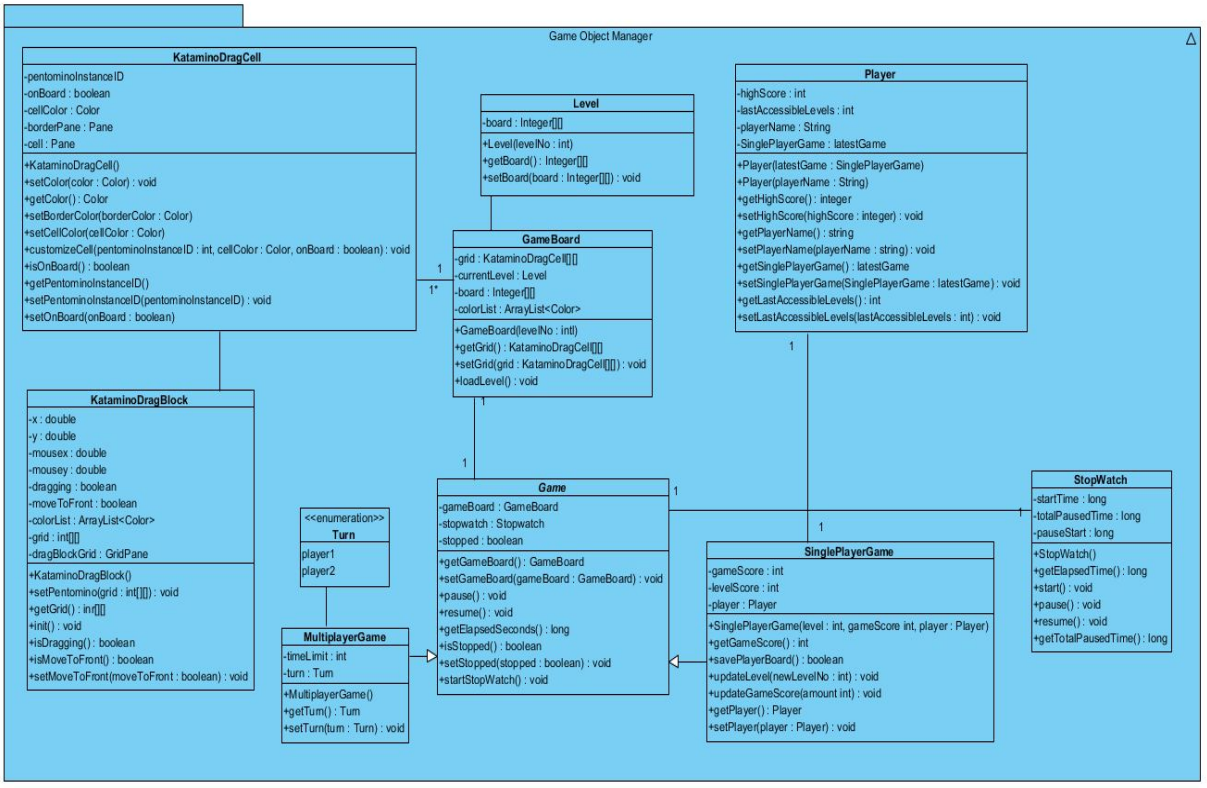


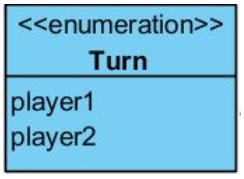
Figure 8: Game Object Manager subsystem

3.2.2 Game Objects Manager

There are 8 game objects in Katamino: Katamino Drag Cell, Katamino Drag Block, abstract Game, Single Player Game, Multiplayer Game, Game Board, Level and Player and 1 enumeration; Turn.

- Katamino Drag Cell is the main component of the game. Whole game board consists of Katamino Drag Cells as grid base. So that pentominoes and board will be combination of Katamino Drag Cells.
- Katamino Drag Block is the pentominoes which are group of 5 Katamino Drag Cells and their usage is limited with the game controllers. They are not part of Game Boards.
- Player is the model for each user. Whenever user asks for a player, this leads to creating of Player instance.
- Game is the abstract basic game that is used for common functionalities of Single Player Game and Multiplayer Game which has a Game Board.
- GameBoard has a group of KataminoCells and uses the information of Level object which is responsible to access file manager for loading initial game board information.
- Stopwatch is time measurement object of a Game.

Turn

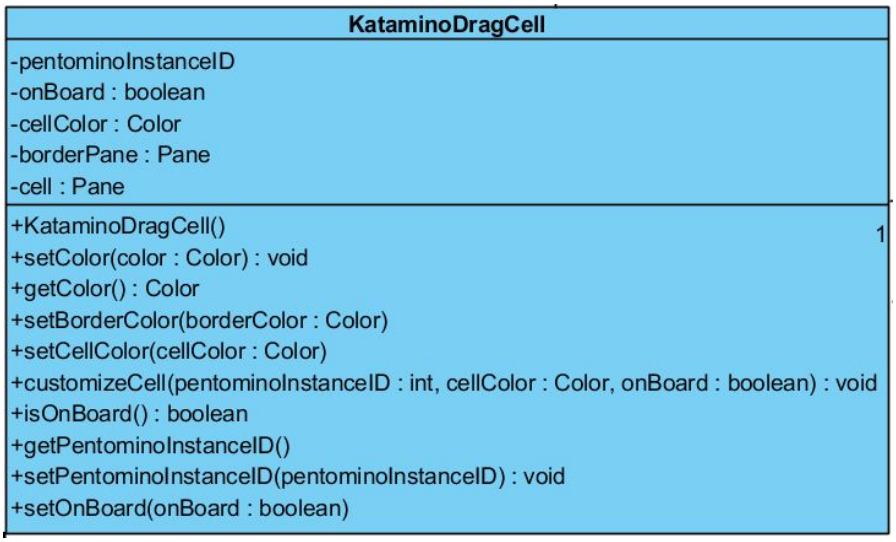


Enums:

player1: This enum stands for turn of first player.

player2: This enum stands for turn of seconds player.

KataminoDragCell



Attributes:

private int pentominoInstanceId: This attribute represents the id of the pentomino tile and is used for the identification of the individual cell.

private boolean onBoard: This attribute represents whether the current cell is on the game board or is in the game grid. If the cell is on the game board it returns true.

private Color cellColor: This attribute controls the coloring of the cell that is displayed in the game.

private Pane borderPane: This attribute represents the Border Pane in the fxml document used as an UI element.

private Pane cell: This attribute represents the Pane in the fxml document used as an UI element.

Constructor:

public KataminoDragCell(): This constructor loads the kataminoDragCell.fxml file and binds it with its controller during initialization.

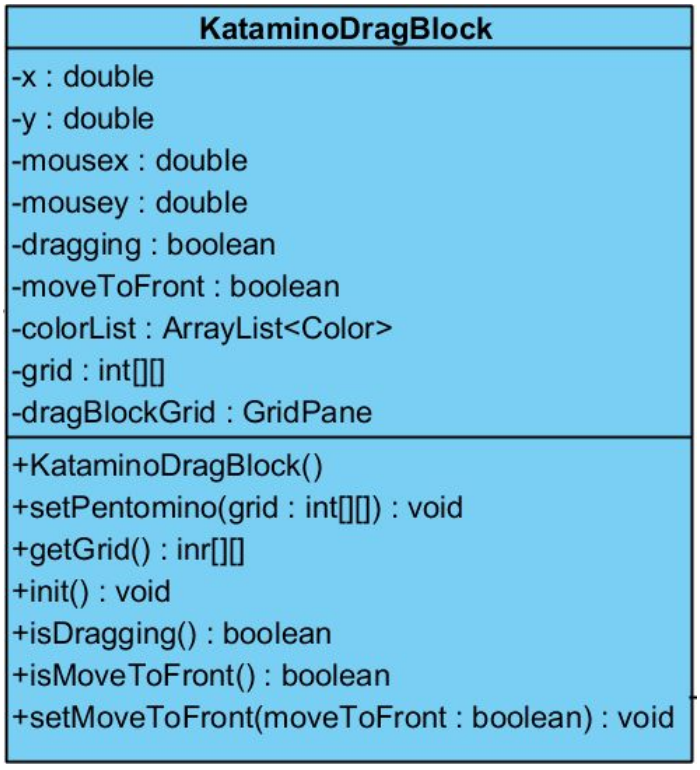
Methods:

public void setColor(Color color): This function sets the cell color attribute of the class.

public void getColor(): This function returns the cell color attribute of the class.

public void setBorderColor(Color borderColor): This function sets the border color of the cell to the inputted color.
public void setCellColor(Color cellColor): This function sets the cell color to the inputted color.
public boolean isOnBoard(): This function returns the onBoard attribute of the class.
public void setOnBoard(boolean onBoard): This function sets the onBoard attribute to the inputted value.
public void setPentominoInstanceId(int pentominoInstanceID): This function sets the pentominoInstanceId attribute of the class to the inputted value.
public int getPentominoInstanceId(): This function returns the pentominoInstanceId attribute of the class.
public void customizeCell(pentominoInstanceID : int, cellColor : Color, onBoard : boolean): This function customizes the cell by setting its pentominoInstanceId, color, and onBoard properties to the inputted values accordingly.

KataminoDragBlock



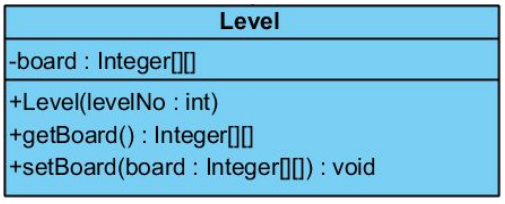
Attributes:
private double x: This attribute represents the x position of the node in the view.
private double y: This attribute represents the y position of the node in the view.
private double mouseX: This attribute represents the x position of the mouse in the view.
private double mouseY: This attribute represents the y position of the mouse in the view.
private boolean dragging: This attribute represents whether the block is dragging or not.
private boolean moveToFront: This attribute represents whether the block show be moved to front in the view or not.
private ArrayList<Color> colorList: This attribute holds twelve distinct colors that represents the twelve different pentomino tiles.

private int[][] grid: This attribute represents the game grid of the game translated as a 2D integer array.
private GridPane dragBlockGrid: This attribute represents the Grid Pane in the fxml document used as an UI element.

Constructor:
public KataminoDragBlock(): This constructor loads the kataminoDragBlock.fxml file and binds it with its controller during initialization.

Methods:
public void setPentomino(int[][] grid): This function is responsible for constructing the selected pentomino inputted as a 2D integer matrix and displays the block as a preview.
public int[][] getGrid(): This function returns the grid attribute of the class.
public void init(): This function sets the dragging operations of the node as it binds dragging functions to the mouse events and sets the position values of the block while dragging.
public boolean isDragging(): This function returns the dragging attribute of the class.
public boolean isMoveToFront(): This function returns the moveToFront attribute of the class.
public void setMoveToFront(boolean moveToFront): This function sets the moveToFront attribute of the class according to the inputted value.

Level

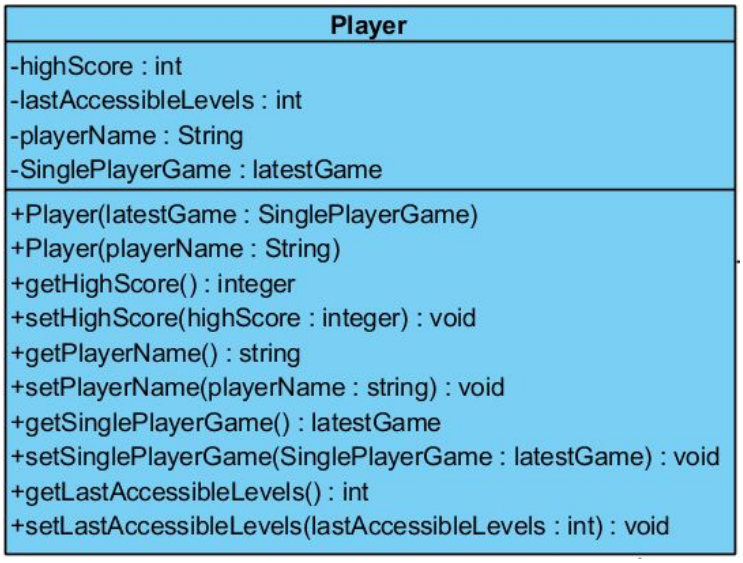


Attributes:
private Integer[][] board: This attribute keeps the game board for the specified level indicating the cell mapping of the tiles (pentominoes).

Constructors:
public Level(int levelNo): This is the constructor that takes integer mapping of the playground such as Figure 9.

Methods:
public Integer[][] getBoard(): This function getter of the board attribute returns the cell mapping of the game board as an 2D integer array.
public void setBoard(Integer[][] board): This setter function changes the board property to the given cell mapping inputted as 2D integer array.

Player



Attributes:

private int highScore: This attribute keeps the high score of the player in integer type.
private int lastAccessibleLevel: This attribute stores the last level accessible by the player, and they are stored as an int.
private String playerName: This attribute represents the name of the player.
private SinglePlayerGame latestGame: This attribute represents an instance of the game model.

Constructors:

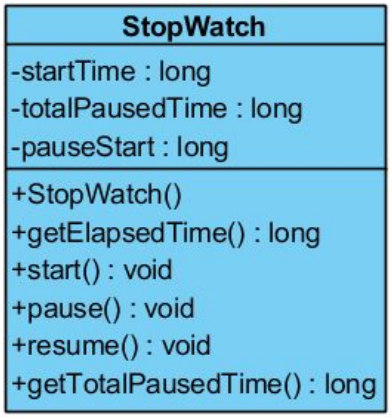
public Player(String playerName) : This is the default constructor of the Player class that creates new player
public Player(SinglePlayerGame latestGame) : This is the default constructor of the Player class that sets lastestGame and the already saved player.

Methods:

public int getHighScore: This function returns the high score of the given player stored in the highScore attribute as an integer.
public void setHighScore(int highScore): This function updates the player's high score as they progress through the game by completing the levels.The updated high score will also be reflected to the ScoreBoard of the players and rearrange the board accordingly.
public String getPlayerName(): This getter function returns the name of the player.
public void setPlayerName(String playerName): This setter function sets the player's name attribute to the given string.
public void setSinglePlayerGame(SinglePlayerGame latestGame): This setter function updates the latest game attribute of the class with the given value.
public SinglePlayerGame getSinglePlayerGame(): This getter function returns the latest game model of the player.
public int getLastAccessibleLevel(): This function returns last accessible level so that the player will be able to play previous and the current last level.

public void setLastAccessibleLevel(int lastAccessibleLevel): This function enables to set the last accessible level attribute of the class by setting it to the last level accessible to the player in the form of an int.

StopWatch

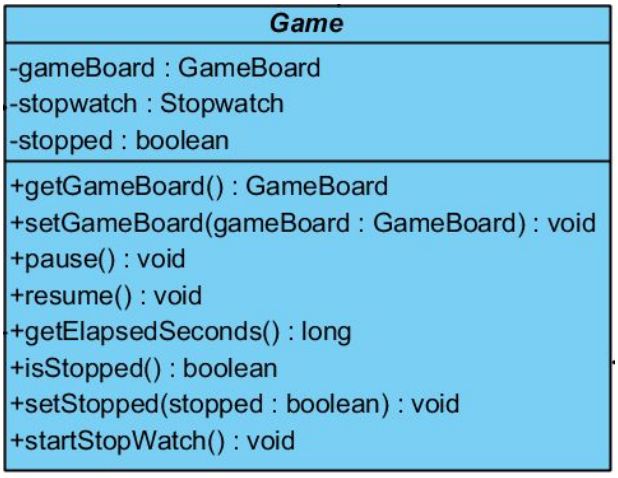


Attributes:
private long startTime : This is the property that shows the initialization time of the object.
private long totalPausedTime : This property shows the time between current time and pauseStart.
private long pauseStart : This is the the time that player stops the game.

Constructors:
public StopWatch(): This constructor initializes the attributes of the class to 0.

Methods:
public long getElapsedTime() : This method returns elapsed time from pause to current time.
public void start() : This method sets the current time to startTime.
public void pause() : This method sets the current time to pauseStart.
public void resume() : This sets the totalPaused time in terms of current time and pauseStart.
public long getTotalPausedTime() : This function returns the total paused time. In other word this is classical getter method.

Game



Attributes:

private GameBoard gameBoard : This attribute is an instance of the GameBoard class that represents the game board in which the pentomino tiles are placed.

private Stopwatch stopwatch : This attribute is an instance of the Stopwatch class and is responsible for keeping the elapsed time and it's calculations.

private boolean stopped : This attribute states whether the game is paused by the player or exits the application.

Methods:

public GameBoard getGameBoard(): This function returns the gameBoard attribute of the class.

public void setGameBoard(GameBoard gameBoard): This function sets the gameBoard attribute to the inputted value.

public void pause(): This function calls the pause function of the stopwatch instance when the game is paused.

public void resume(): This function calls the resume function of the stopwatch instance when the game is resumed.

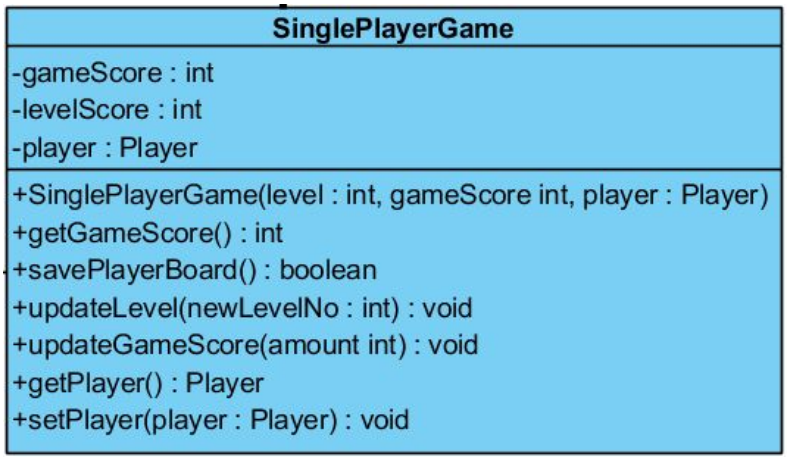
public boolean isStopped(): This function returns the stopped attribute of the class.

public void setStopped(boolean stopped): This function sets the stopped attribute of the class to the inputted value.

public long getElapsedSeconds(): This function returns the elapsed seconds from the stopwatch in long type.

public void startStopwatch(): This function starts the counting of the stopwatch when the game begins.

SinglePlayerGame



Attributes:

private int gameScore: This attribute represents the overall score of the player across the application.

private int levelScore: This attribute represents the score that the player will be gained from completing the current level.

private Player player: This attribute represents the current player instance playing the game.

Constructor:

public SinglePlayerGame(int level, int gameScore, Player player): The constructor of the SinglePlayerGame class takes level, gameScore and player information as parameters and initializes the attributes of the class accordingly.

Methods:

public int getGameScore(): This function returns the gameScore attribute of the class.

public boolean savePlayerBoard(): This function saves the current game board with all the position of the pentomino tiles in case of application termination or the player quits the game so that they could continue from where they were left.

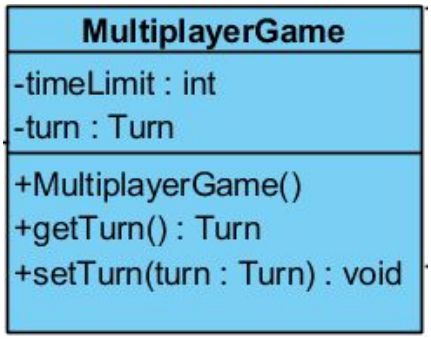
public void updateLevel(int newLevelNo): This function changes the level of the game by taking the newLevelNo as input and updates the game board accordingly to the new level.

public void updateGameScore(int amount): This function updates the gameScore attribute of the class to the inputted amount.

public Player getPlayer(): This function returns the player attribute of the class.

public void setPlayer(Player player): This function sets the player attribute of the class to the inputted value.

MultiplayerGame

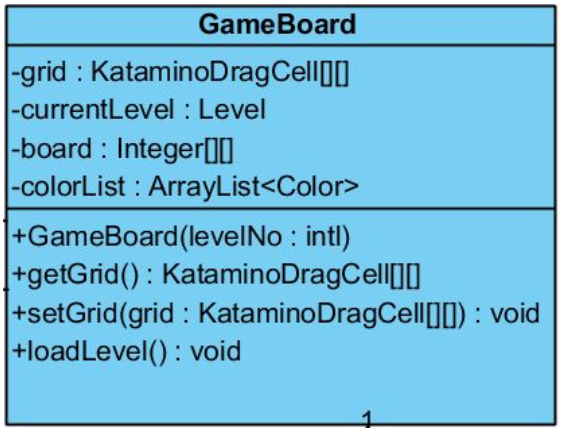


Attributes:
private Turn turn: This attribute takes Turn enum and shows whose turn it is.
private int timeLimit: This attribute keeps maximum time limit for each player's turn.

Constructor:
public MultiplayerGame(): The default constructor of the MultiplayerGame class.

Methods:
public Turn getTurn(): This function returns the turn attribute of the class.
public void setTurn(Turn turn): This function sets the turn attribute of the class to the inputted value used during changing turns between the rounds.

GameBoard



Attributes:
private KataminoDragCell[][] grid: This attribute keeps all grids on the whole view as KataminoDragCell.
private Level currentLevel : This is the current level that is loaded on the gameboard.
private ArrayList<Color> colorList : This is the list that consist of possible colors of the pentominoes.
Integer[][] board : This attribute keeps all grids on the whole view as integer mapping.

Constructor:
public GameBoard(int levelId): This is the constructor that takes levelId of the playground and initialize the level object such as Figure 9.

0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	4	4	0	0
1	1	0	0	0	0	0	0
0	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	3	3	3	3	0	0

Figure 9

Methods:

public KataminoDragCell [][] getGrid(): This is the standard getter used by GameController. By this method GameControler can access the Grid and determine the positioning of the items as a whole.

public void loadLevel() : Loads color and border of the grid. Thus grid ready to be deployed.

public void setGrid(KataminoDragCell[][] grid): This is the standard setter that used by GameController.

3.3 Data Layer

The Data layer consists of the FileManager class and it is responsible for the handling of data manipulations and basic CRUD functionalities. In this subsystem the File Manager holds ScoreBoard, Player and Level information and is responsible for loading game map files as levels, updating the score board and recording the data changes to the system.

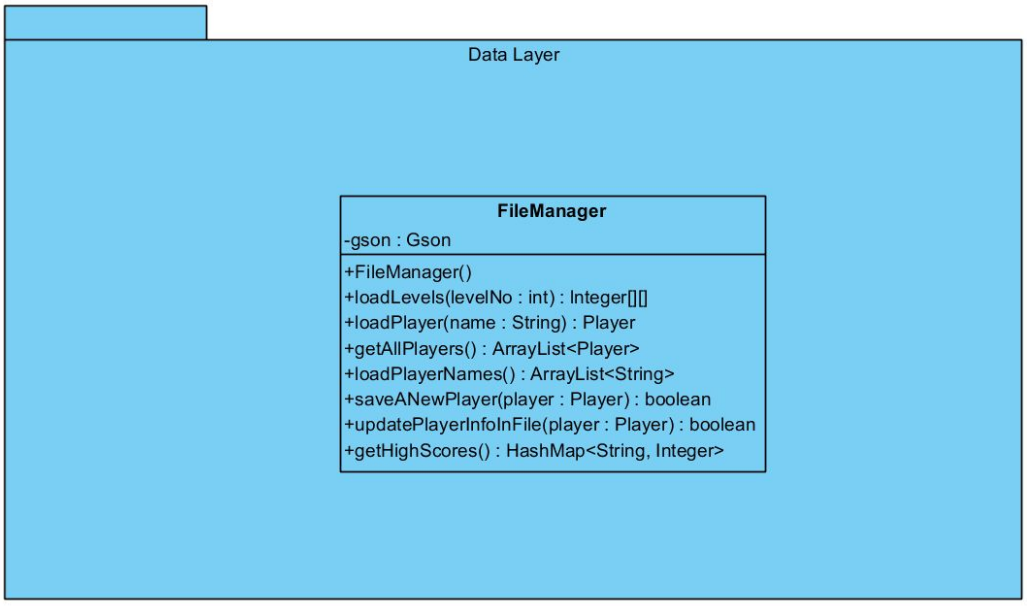
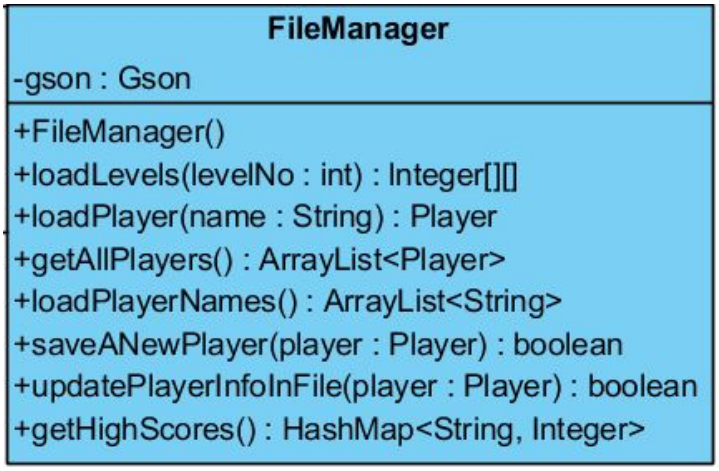


Figure 10: Data Layer subsystem

FileManager



Attributes:

private Gson gson : This is the gson object that interact with jsons.

Constructor:

public FileManager() : This is the constructor that initializes Gson instance.

Methods:

public Player loadPlayer(String name) : This is the method that takes the player name and get the information of the player from players.json, initializes and returns the player.

public ArrayList<Player> getAllPlayers() : This method fetches all players from players.json and returns them as an arraylist of players.

public ArrayList<String> loadPlayerNames() : This method fetches all player names from players.json and returns them as an arraylist of player names.

public boolean saveANewPlayer(Player player) : This function saves player object as json to the player.json.

public boolean updatePlayerInfoInFile(Player player) : This function updates the properties of the currently saved player.
public HashMap<String, Integer> getHighScores() : This method fetches highscores of the players from the players.json and return as a hashmap for efficiency.
public Integer [][] loadLevels(int levelNo) : This function loads the game levels from the txt file according to level id and returns the grid.

4. Low-level design

4.1 Object design trade-offs

Inheritance vs Composition

In the design of Katamino v2.0, we value composition over inheritance to increase flexibility of design. Additionally, with the use of composition we plan to increase the maintainability of our code.

Loose coupling vs tight coupling

In designing Katamino v2.0, we prefer loose coupling over tight coupling to decrease the dependencies between classes and ease the testing of our code. Moreover, with the use of loose coupling we plan to increase maintainability, maintainability and reusability of our code.

Centralized vs. Decentralized

In katamino v2.0's design, we mostly used centralized design. Most of the logic in the game are handled by a controller class.We chose Centralized design to increase the maintainability and maintainability of our control classes.

4.2 Final object design

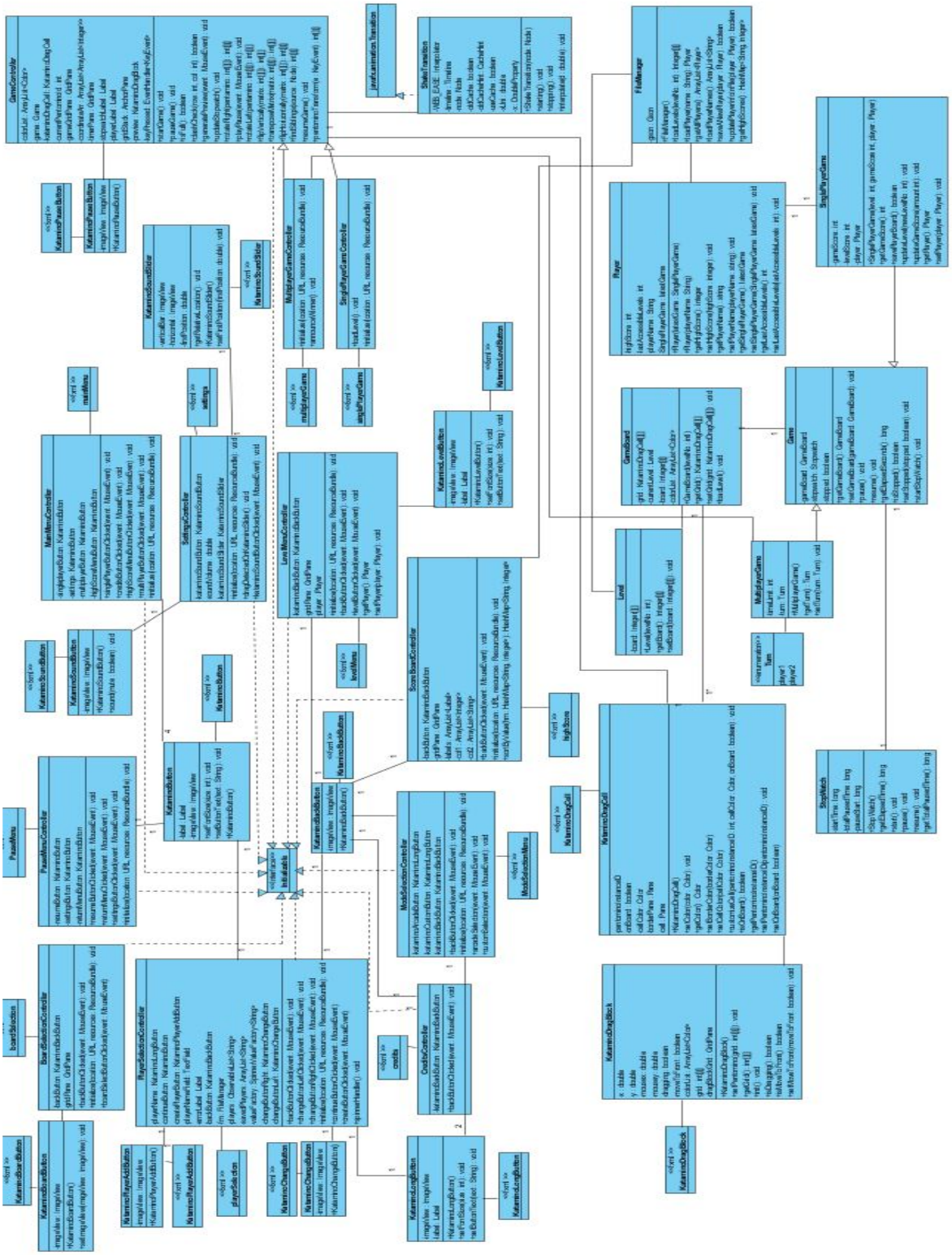


Figure 11: Final Class Diagram with all classes

4.3 Packages

Java.util

This package will be used for collections for data management on memory. Such as the current gameboard's shape and pentominoes locations on the board. From this package, we will also use IO sub-packages for loading from the and saving data to computer, such as player data, high score data and custom board data.

Javafx.scene

This package will be used for support to scene graph. From this package, we will also use sub-packages such as input, layout and image. We will use input subpackage to use handlers for mouse and keyboard events, layout to fit GUI objects, image to support images in GUI.

Javafx.event

We will use this package to handle JavaFX events, such as mouse clicks and key presses.

Javafx.application

This package will be used for the JavaFX application life cycle.

Javafx.animation

This package contains classes that are used to animate the nodes. Animation is the base class of all these classes. By using this library, transitions such as the shake transition and game board change transitions were built.

Java.com.google.gson

This library used for json operations. By using this library, game converts objects and single attributes of the objects to the json format and it converts json format to java objects and object attributes.

4.4 Class Interfaces

ActionListener

This interface is responsible for catching any action with the corresponding action events. This interface will be working with Menu class. This interface will be used to enable buttons in the menus.

MouseListener

This interface will be used to catch mouse click actions. MouseListener will be implemented by inputManager class to use the input in game. These inputs will be used to calculate movements and new positions of pentominoes.

KeyListener

KeyListener interface is responsible for catching keyboard events. KeyListener will be implemented by input manager. These inputs will be used to calculate the flips and rotations of pentominoes.

Initializable

Initializable interface is responsible for initializing an fxml controller after its root element is completely processed.It is used to interact and manipulate elements in the fxml file.

5. Improvement Summary

In the second iteration of the design report as any software could change we have to make many changes during our implementation which the improvements as listed below. Firstly, because of the additions we did in the analysis report, our objects design has changed. Also, because of other changes we did during design like adding some extra classes also affected our object design.

- We have added new packages which is used for implementations.
- Our representation of subsystem decomposition is totally changed. To show the relationship between subsystems we have used lollipop component diagrams.
- In subsystem services we have explained the subsystems with more detailed and we gave brief explanations about their components.
- We have changed the design of models to have more MVC design, and used them to save the latest game.
- We have made our implementation more reusable. Currently, we are using specification inheritance in contrast with the implementation inheritance which we had first used for Single Player Game and Multiplayer Game in the first iteration. By changing it, we aimed to have a more abstract Game and now, it specializes to Single Player Game and Multiplayer Game.
- We added new GUI elements, controllers and models that we had not foresee to have in the first iteration. These classes emerge as the implementation is progressing.
- We changed our internal class design for nearly all classes (at least one of the property is changed) because various new classes are added and the implementation has modified for some methods. For instance, the algorithms for flip and rotation functions separated from the chunk functions for the sake of the divide-and-conquer methodology for the teamwork and change.

6. Glossary & References

[1] "Overview (Java Platform SE 8)", *Docs.oracle.com*, 2018. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/overview-summary.html. [Accessed: 08- Nov- 2018].

[2] "Overview (JavaFX 8)", *Docs.oracle.com*, 2018. [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/overview-summary.html. [Accessed: 08- Nov- 2018].