Requirements for the Kakuro project

Iteration 2 COMP354

Team PK-A

15 March 2020

Table 1: Team Members

Name	Role	ID Number
Tiffany Ah King	Quality Control	40082976
Isabelle Charette	Documenter	40008121
Brian Gamboc-Javiniar	Coder	40033124
Vsevolod Ivanov	Organizer	40004286
Chang Liu	Quality Control	40056360
Nolan Mckay	Coder	27873557
Nalveer Moocheet	Quality Control	40072605
Hoang Thuan Pham	Coder	40022992
Audrey-Laure St-Louis	Documenter	27558783
Jia Ming Wei	Documenter	40078192

Table of Contents

1 Introduction

Purpose

The purpose of this document is to present the design of the Kakuro game for the course COMP 354.

Scope

This document is intended to provide detailed design specifications of the Kakuro game.

2 Architectural Design

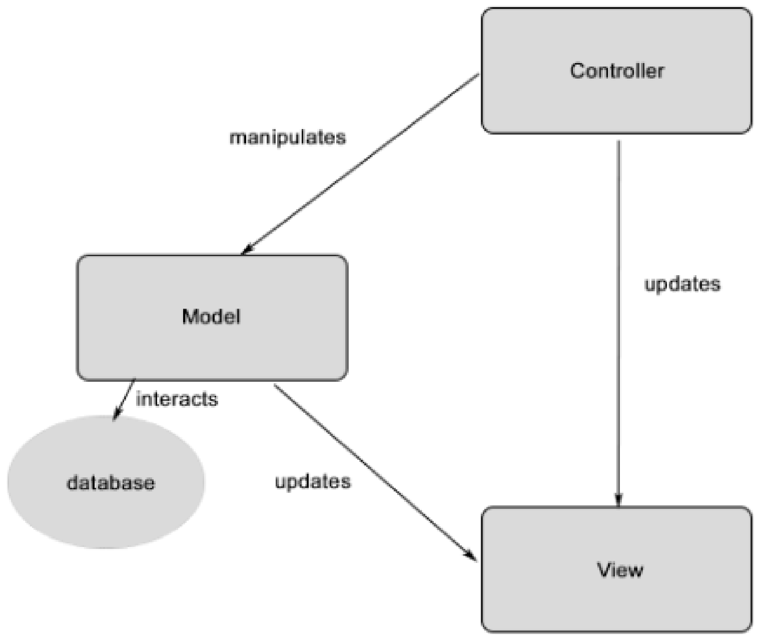

Figure 1: Architecture Diagram

2.1 Rationale

The architecture chosen for the Kakuro game is the Model View Controller model (MVC). The MVC architecture is constructed of three separate components: the model, the view and the controller.

The model is the central component of the game. It stores the data and it changes depending on the state of the game. In Kakuro, the model stores the information of the cells from the rows and the columns displayed by the graphical user interface. The view is the graphical user interface (GUI) of the game. It displays the grid, the buttons, and the text elements but it also displays the date from the model. It allows the player to enter numbers to play the game. Whenever the model changes, the GUI reacts to these changes by updating itself. For example, if the player presses the button restart, the model will be updated by clearing its input cells and therefore, the GUI will react by showing empty input cells to the user.

The controller manages the interactions with the user and decides which functions should be called given an action. The controller will use the model's data, he will take action on those depending on the user's action and he will send it to the view for it to show it in the GUI.

2.2 Subsystem Interface Specifications

2.2.1 View Interface

2.2.1.1 GameView

The GameView class is the interface used for the View. With its constructor, it initializes the GUI of our Kakuro game. The following methods are available for this interface:

- getBoardUI (Cell)
- getMaxNumberValid()
- getMinNumberValid()
- getNumberFormatterClassType()
- getSavedInput()
- hideBoard()
- settingTextField(JTextField)
- showBoard()
- updateView()

2.2.1.2 MenuBarView

The MenuBarView class is used for the different buttons displayed on the GUI of the game. The MenuBarView class contains the following methods:

- buttonsSetUp()
- getMainPannel()
- toggleMenu()

2.2.1.3 ChronoView

The ChronoView class handles the chronometer of the game. It contains the following methods:

- getTimerLabel()

- setTimerLabel()

2.2.2 Model Interface

The interface between the controller and model is called whenever the controller receives input from the player. The kakuro.models package contains all the classes from the model interface. The following methods are part of the interface:

2.2.2.1 GameModel

The GameModel class is the interface used for the Model of our system. It interacts with the database and it handles all the functions that are implementing the rules of the game. It contains the following methods:

- getColumns(int, int)
- getRows()
- initBoard()

2.2.2.2 ChronoModel

The ChronoModel class is the model of the chronometer of our Kakuro game. It contains the following methods:

- getDelay() : Returns the delay used for the chronometer
- getHours() : Returns the hours value of the chronometer
- getMinutes(): Returns the minutes value of the chronometer
- getSeconds() Returns the seconds value of the chronometer
- resetTimer() : Brings the chronometer to the value zero for its seconds, minutes and hours
- setHours() : Sets the hours of the chronometer
- setMinutes() : Sets the minutes of the chronometer
- setSeconds() : Sets the seconds of the chronometer
- updateTime()

2.2.2.3 PlayerModel

The PlayerModel class handles the player's information. It contains the following methods:

- getPlayerPassword() : Returns the password of the player
- getPlayerUsername() : Returns the username of the player
- setPlayerPassowrd() : Sets the password of the player
- setPlayerUsername(): Sets the username of the player

2.2.3 Controller Interface

The controller interface is a package (kakuro.controllers) composed of the following three classes:

2.2.3.1 GameController
The controller accepts input from the player and performs simple validations on it. The class contains the following methods :

- connectDatabase() : Connects the game to the database
- disconnectDatabase() : Disconnect the database from the game
- getDatabaseConnection() : Returns the connections established from the database.
- getMaxNumberValid() : Returns the maximum number aloud for the player to use during the game
- getMinNumberValid() : Returns the minimum value aloud for the player to use during the game
- getNumberFormatterClassType()
- initGame() :
- loadGame()
- loadInputInModel
- loadPreconfigureGame(int)
- loopGame()
- pause() : Pause the game which stops the chronometer and blocks the player from entering data in the game
- restart() : Clears the board and restart the chronometer
- resume() : Starts the chronometer and allows the player to continue playing by entering value in the board game
- saveGame() : Save the state of the game in the database
- solveBoard() : Solves the board by calculating the sums of the rows and the columns and checks if it brings to a correct solution.
- submit() : Lets the user get feedback from the system to know if he got the right solution or not

2.2.3.2 MenuBarController
The MenuBarController Class accepts input from the player through the menu bar and performs actions depending on the button that is being pressed. The class contains the

following methods:

- getButtonMenuView()
- getView()
- isPaused()
- load()
- loadPreconfigureGame(GameDifficulty g)
- pause()
- resume()
- save()
- submit()

2.2.3.3 ChronoController

- chronoPause() : Pause the chronometer
- chronoStart() : Starts the chronometer
- getHours() : Returns the hours of the chronometer
- getMinutes() : Returns the minutes of the chronometer
- getSeconds() : Returns the seconds of the chronometer
- getView(): Returns the label of the chronometer.
- hide() : Hides the chronometer on the GUI of the game.
- resetTimer() : Resets the chronometer to bring it to zero (hours, minutes and seconds) show() : Display the chronometer in the GUI of the game
- timerSetUp() : Set up the chronometer by attaching an action listener to it.
- toggleTimerDisplay() : Sets the chronometer visible or hides it dependanding on the state he's in

2.2.4 Other

To support our MVC architecture, we created the following classes.

Core Package

2.2.4.1 Cell

- getFirstValue()
- getSecondValue()
- getType()
- setFirstValue(int)

2.2.4.2 DatabaseConnection

- connect()
- createGameProgressTable()
- createGameTable()
- createPlayerTable()
- disconnect()
- getConnection()
- insertMainPlayer()
- insertPlayerData()
- insertPreconfiguredGames()

2.2.4.3 GameDifficulty

- gameDifficultyToInt(GameDifficulty) : Transform the level of difficulty chosen to a integer value and returns it

2.2.4.4 GameDifficultyListItem

- getDifficulty(): Returns the level of difficulty chosen by the player toString(): Returns a description of the difficulty chosen

2.2.4.5 LinePanel

- paintComponent(Graphics): Draws the diagonal line in the black cells
- settingTxt(JTextField): Sets the background and the foreground color of the game

2.2.4.6 Tools

- arrayToNodes(DefaultMutableTreeNode)
- childrenToArray(TreeNode)
- randomInt() : Generates a random integer value

2.2.4.7 UniquePartitions

- fillCombinations(DefaultMutableTreeNode)
- getTreeRoot()

GameProgresse DAO package

2.2.4.8 GameProgressDao

- load(Connection, String)
- save(Connection, String, Cell)

Game DAO package

2.2.4.8 GameDao

- loadAllPreconfiguredGames(Connection)

Player DAO package

2.2.4.9 PlayerDAO

- Login (Connection, String, String)
- Register(Connection, String, String)

3 Detailed Design

The Karuro system consists of three subsystems: Game-Puzzle, Registration, and Ranking subsystems. The Game-Puzzle subsystem is implemented in the iteration 1 and iteration 2. During the iteration 1, this subsystem is implemented using the UI and the console. During the iteration 2, a SQLite server is integrated in the libraries so that the input data and solution data are possible to be stored in the database. Therefore, having a database server is essential to implement the Registration subsystem and Ranking subsystem in the iteration 3.

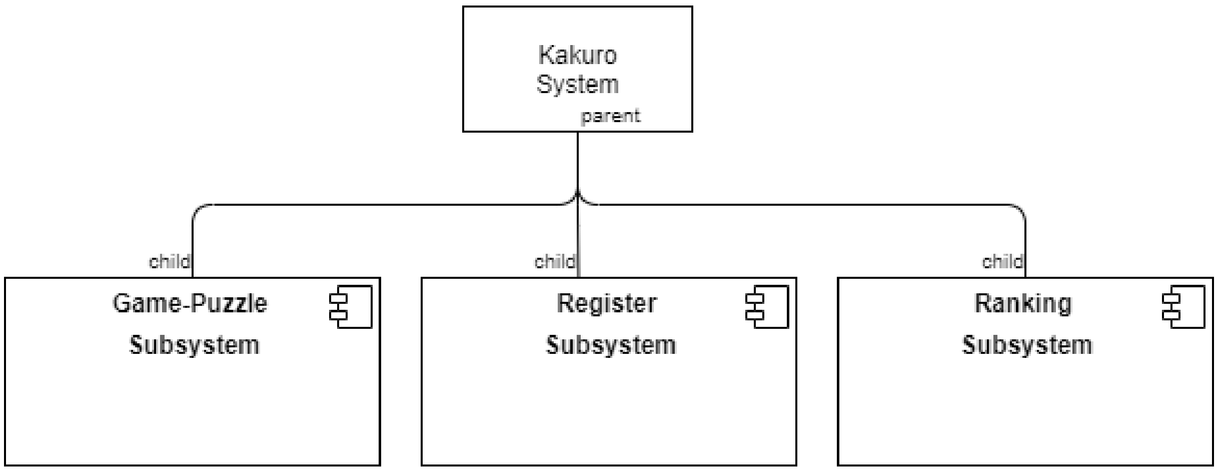

Figure 2: UML of Kakuro Subsystems

The three subsystem are derived from the whole system, the Karuro system. On the other hand, the three subsystems are independent of each other. This design practice the principles of high cohesion and low coupling. The three subsystems are also three components of this software systems. The three subsystems present three different parts of view, apply different models, and use different parts of controller. The Ranking subsystem intersects with the Registration subsystem in the Player class, database connection class, and the MainFrame class, and both of them have dependency with the Game-Puzzle subsystem because the game scores come from the game puzzle.

3.1 Game-Puzzle Subsystem

Detailed Design Diagram

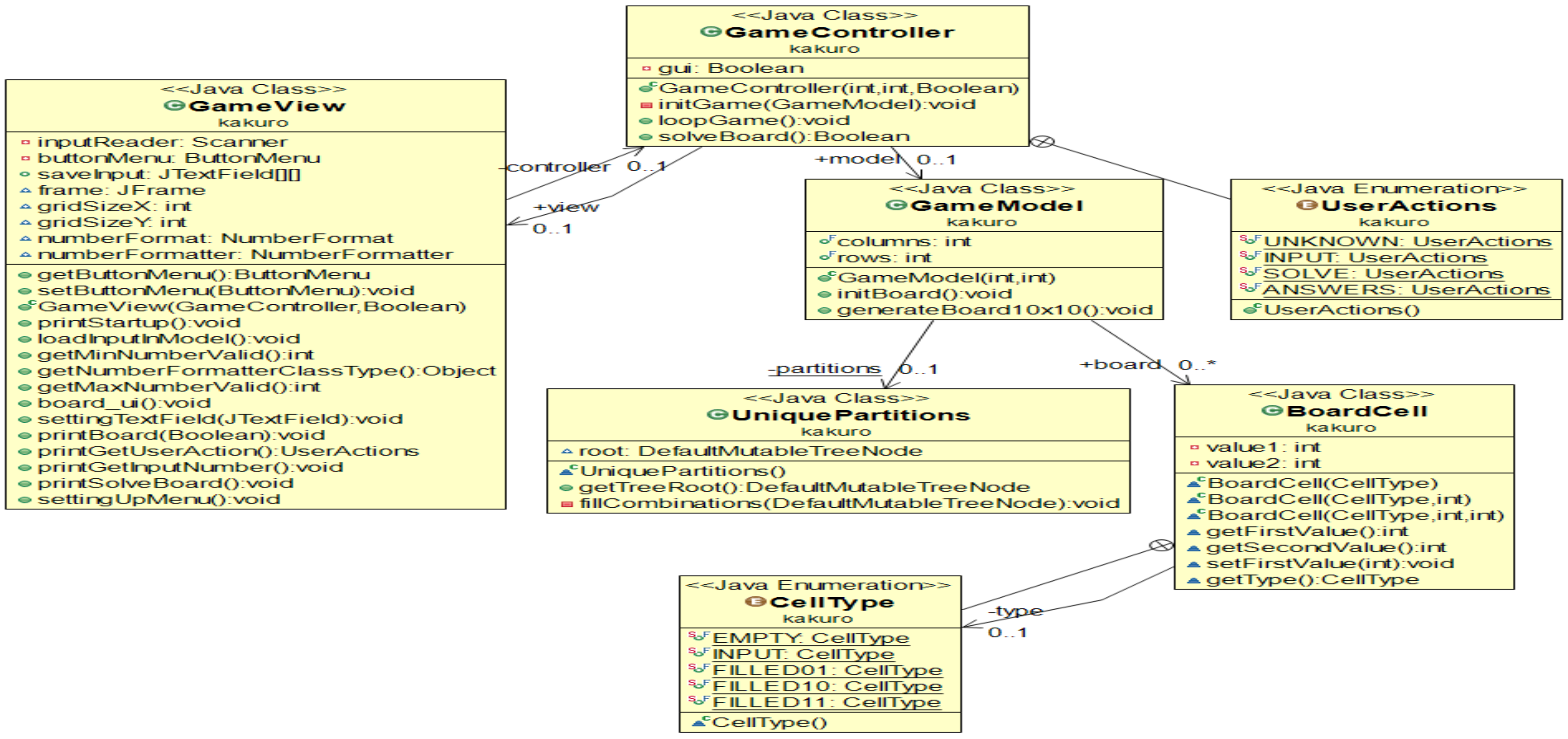

Figure 3: UML of Geme-Puzzle Subsystems

The Game-Puzzle subsystem design follows the MVC model. The GameView is the user interface of this system, it includes line_panel, BoarCell, CellType and ButtonMenu classes. The Chrono class provides the timer function that is not used in this subsystem and will be used in the Ranking subsystem. The model part includes GameModel, UniquePartitions, Tools, and database part which includes GameDaoImpl and GameProgressDaoImpl. The Controller part is the GameController class.

Units Description

Class Name	**GameController**			
Inherits from	None			
Description	The controller of subsystem			
Attributes	Visibility	Data Type	Name	Description
	Public	enum	UserActions	User actions
	Public	DatabaseConnection	database	A database connection
	Private	Boolean	gui	GUI or console
Methods	Visibility	Method Name	Description	
	Public	GameController(int, int, Boolean)	Constructor	
	Public	loopGame()	A loop that keep game running	
	Public	solveBoard()	To check if the answer is correct	
	Private	initGame(GameModel)	To initiate a new game	
	Public	loadInputInModel(boolean)	Loads input to model	
	Public	loadPreconfiguredGame(int)	Loads a configured game	
	Public	connectDatabase()	Connects to database	
	Public	disconnectDatabase()	Disconnects to database	

Class Name	**GameModel**			
Inherits from	None			
Description	The view of subsystem			
Attributes	Visibility	Data Type	Name	Description
	Public	int	columns	The columns of the board
	Public	int	rows	The rows of the board
Methods	Visibility	Method Name	Description	
	Public	GameModel(int, int)	Constructor	
	Public	initBoard()	To initiate a new board	
	Public	generateBoard10x10()	To generate a 10x10 board	

Class Name	**GameView**			
Inherits from	None			
Description	The view of the subsystem			
Attributes	Visibility	Data Type	Name	Description
	Private	Scanner	inputReader	A input reader
	Private	ButtonMenu	buttonMenu	A ButtonMenu object
	Private	JFrame	frame	A JFrame object
	Public	JTextField[][]	saveInput	The inputs array
	Private	int	gridSizeX	X value of grid size
	Private	int	gridSizeY	Y value of grid size
	Private	NumberFormat	numberFormat	A number format instance
	Private	NumberFormatter	numberFormatter	A NumberFormatter instance
Methods	Visibility	Method Name		Description
	Public	GameView(GameController, Boolean)		Constructor
	Public	getButtonMenu()		Return a ButtonMenu object
	Public	setButtonMenu()		Set a value
	Public	printStartup()		Displays instructions in console
	Public	loadInputInModel()		To load input model
	Public	getMinNumberValid()		Return a minimum valid integer
	Public	getMaxNumberValid()		Return a maximum valid integer
	Public	getNumberFormatterClassType()		Return a class type
	Public	board_ui()		To create an user interface
	Public	settingTextField(JTextField)		To set the text fields of board
	Public	settingUpMenu()		To set up the button menu
	Public	printBoard(Boolean)		Displays input in console
	Public	printGetUserAction()		Reads user actions from console
	Public	printGetInputNumber()		Displays and validates inputs
	Public	printSolveBoard()		Displays the solution correctness

Class Name	**line_panel**		
Inherits from	JPanel		
Description	A Panel for the game board		
Methods	Visibility	Method Name	Description
	Public	line_panel(LayoutManager, JTextField, Boolean)	Constructor
	Public	line_panel(LayoutManager, JTextField, JTextField)	Constructor
	Public	settingTxt(JTextField)	Sets text fields
	Public	paintComponent(Graphics)	Paints components

Class Name	**UniquePartitions**			
Inherits from	None			
Description	Lists all possible answers in a Tree ADT			
Attributes	Visibility	Data Type	Name	Description
	Private	DefaultMutableTreeNode	root	A root node object
Methods	Visibility	Method Name	Description	
	Public	UniquePartitions()	Constructor	
	Public	getTreeRoot()	Returns a root node object	
	Public	fillCombinations(DefaultMutableTreeNode)	Fills cells with possible number combinations to solve the puzzle	

Class Name	**Tools**		
Inherits from	None		
Description	Tools for general utilities		
Methods	Visibility	Method Name	Description
	Public	Tools()	Constructor
	Public	randomInt(int, int)	Returns a random int
	Public	arrayToNodes(DefaultMutableTreeNode, int[])	Converts array to nodes
	Public	childrenToArray(TreeNode)	Converts nodes to array

Class Name	**ButtonMenu**			
Inherits from	None			
Description	A Menu of Buttons			
Attributes	Visibility	Data Type	Name	Description
	Package	JButton	pause_button	a pause button
	Package	JButton	play_button	a play button
	Package	JButton	submit_button	a submit button
	Package	JButton	newGame_button	new game button
	Package	JButton	choose_game_button	choose game botton
	Package	JButton	save_button	a save button
	Package	JButton	restart_button	a restart button
	Package	JButton	load_button	a load button
	Package	JPanel	mainPanel	a main panel
Methods	Visibility	Method Name	Description	
	Public	ButtonMenu(JFrame, int, int, GameController)	Constructor	
	Public	toggleMenu()	Toggles visibilities	
	Public	buttonsSetUp()	Adds Action Listeners to buttons	

Class Name	**BoardCell**			
Inherits from	None			
Description	A cell of game board			
Attributes	Visibility	Data Type	Name	Description
	Private	int	value1	A value of cell
	Private	int	value2	A value of cell
	Package	enum	CellType	Five cell types in game board
Methods	Visibility	Method Name	Description	
	Public	BoardCell(CellType)	Constructor	
	Public	BoardCell(CellType, int)	Constructor	
	Public	BoardCell(CellType, int, int)	Constructor	
	Public	getFirstValue()	Returns value1	
	Public	getSecondValue()	Returns value2	
	Public	setFirstValue(int)	Sets value1	
	Public	getType()	Retutns a cell type	

Class Name	**GameDaoImpl**			
Implements from	GameDao			
Description	Data Access Objects (DAO) of games			
Attributes	Visibility	Data Type	Name	Description
	Private	String	LOAD_ALL_PRECONFIGURED_GAMES	A query statement
Methods	Visibility	Method Name		Description
	Public	GameDaoImpl()		Constructor
	Public	loadAllPreconfiguredGames(Connection)		Loads game data

Class Name	**GameProgressDaoImpl**			
Implements from	GameProgressDao			
Description	DAOs of game progress			
Attributes	Visibility	Data Type	Name	Description
	Private	String	SAVE_GAME_PROGRESS	A query statement
	Private	String	LOAD_GAME_PROGRESS	A query statement
Methods	Visibility	Method Name		Description
	Public	GameProgressDaoImpl()		Constructor
	Public	save(Connection, String, BoardCell[][])		Sets game progress data
	Public	load(Connection, String)		load game progress data

3.2 Registration Subsystem

Detailed Design Diagram

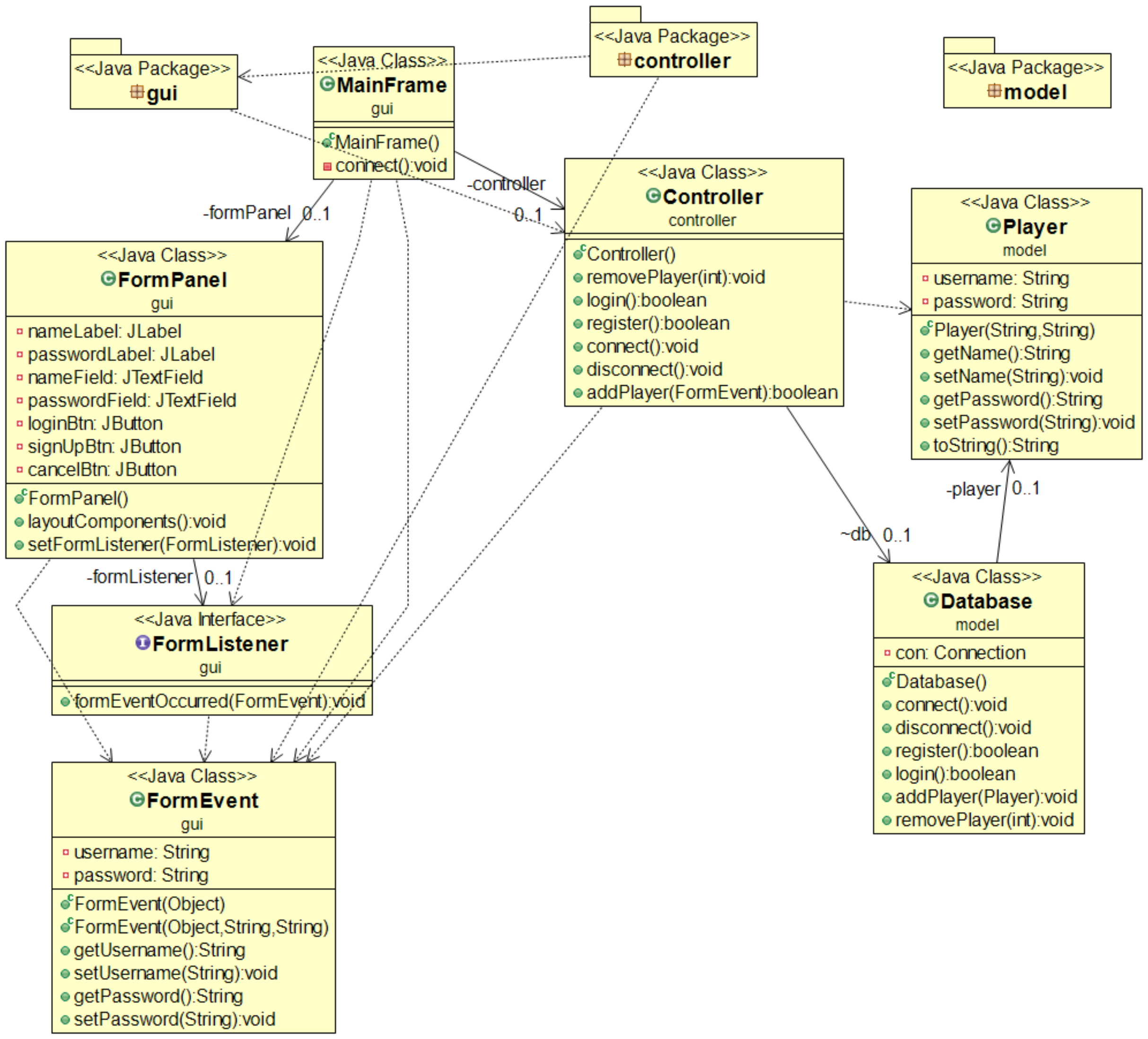

Figure 4: UML of Registration Subsystems

Registration subsystem will be implemented in the iteration 2. This design follows MVC model. The model consists of the Player class and the Database class. The Database class offers the connections to the SQLite database. The gui package is the view of the MVC model, and it is the user interface for the player to login and register. The Controller class controls the data flow between model and view.

Units Description

Class Name	**Player**			
Inherits from	None			
Description	Players can login and register to this game system			
Attributes	Visibility	Data Type	Name	Description
	Private	String	username	Player’s username
	Private	String	password	Player’s password
Methods	Visibility	Method Name	Description	
	Public	Player(String, String)	Constructor	
	Public	getName()	Returns username	
	Public	setName(String)	Sets username	
	Public	getPassword()	Returns password	
	Public	setPassword(String)	Sets password	
	Public	toString()	Returns a string	

Class Name	**Database**			
Inherits from	None			
Description	A database connection			
Attributes	Visibility	Data Type	Name	Description
	Private	Connection	con	A connection to database
Methods	Visibility	Method Name	Description	
	Public	Database()	Constructor	
	Public	connect()	Connects to database	
	Public	disconnect()	Disconnects to database	
	Public	register()	Registers to game	
	Public	login()	Login to game	
	Public	addPlayer(Player)	Inserts a player into database	
	Public	removePlayer	Deletes a player from database	

Class Name	**Controller**		
Inherits from	None		
Description	A controller of the Registration subsystem		
Methods	Visibility	Method Name	Description
	Public	Controller()	Constructor
	Public	connect()	Connects to database
	Public	disconnect()	Disconnects to database
	Public	register()	Registers to game
	Public	login()	Login to game
	Public	addPlayer(FormEvent)	Inserts a player into database

Class Name	**MainFrame**		
Inherits from	JFrame		
Description	An user interface of the Registration subsystem		
Methods	Visibility	Method Name	Description
	Public	MainFrame()	Constructor
	Public	connect()	Connect to the Controller

Class Name	**FormPanel**			
Inherits from	JPanel			
Description	The form panel for players to login and register			
Attributes	Visibility	Data Type	Name	Description
	Private	JLabel	nameLabel	A label
	Private	JLabel	passwordLabel	A label
	Private	JTextField	nameField	A text field
	Private	JTextField	passwordField	A text field
	Private	JButton	loginBtn	A button
	Private	JButton	signUpBtn	A button
	Private	JButton	cancelBtn	A button
Methods	Visibility	Method Name	Description	
	Public	FormPanel()	Constructor	
	Public	layoutComponents()	Sets the layout components parameters	
	Public	setFormListener(FormListener)	Sets an event listener	

Class Name	**FormEvent**			
Inherits from	FormListener			
Description	The form events			
Attributes	Visibility	Data Type	Name	Description
	Private	String	username	A username
	Private	String	password	A password
Methods	Visibility	Method Name	Description	

	Public
	Public
		FormEvent(Object)
		FormEvent(Object, String, String)
			Constructor
			Constructor	

	Public	getUsername()	Returns a username	
	Public	setUsername()	Sets a username	
	Public	getPassword()	Returns a password	
	Public	setPassword()	Sets a password	

3.3 Ranking Subsystem

Detailed Design Diagram

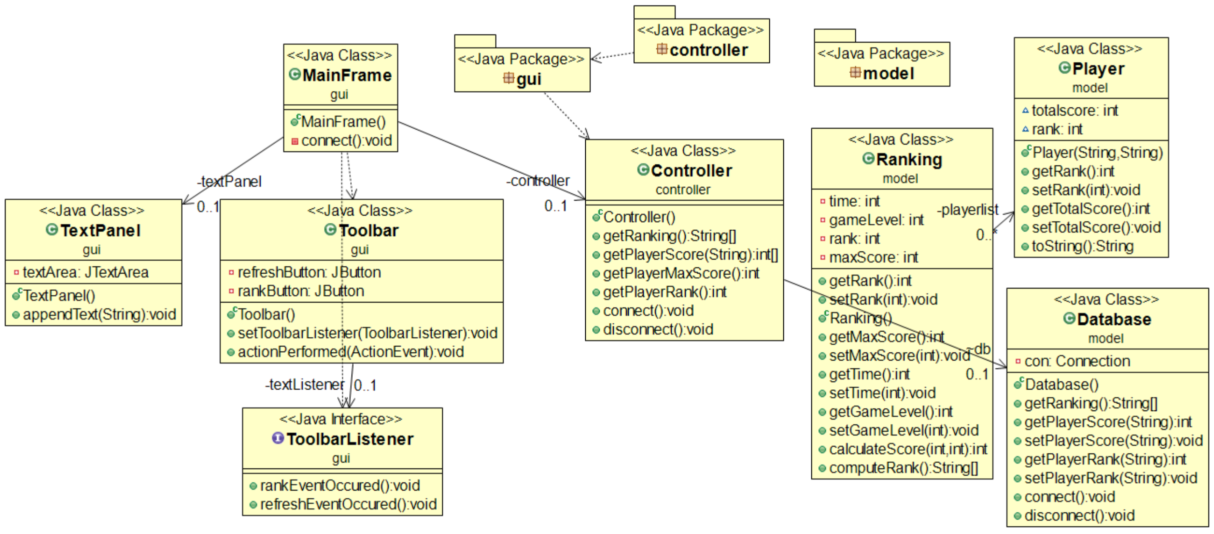

Figure 5: UML of Ranking Subsystems

The Ranking subsystem design follows the MVC model. The view part consists of the MainFrame, TextPanel, Toolbar and ToolbarListener. The control part is the Controller class and it controls the data flow between the database connection to the MainFrame. The model part consists of the Ranking, Player and Database class. The Ranking class provides the computing ranking functionality. The ranking rules include the time used, the game difficulty level and the correctness of solutions.

Units Description

Class Name	**Ranking**			
Inherits from	None			
Description	To compute the ranking of the players			
Attributes	Visibility	Data Type	Name	Description
	Private	int	time	The time used in a game
	Private	int	gameLevel	The game difficulty level
	Private	int	rank	A player's rank position
	Private	int	maxScore	The maximum score in the system
Methods	Visibility	Method Name	Description	
	Public	Ranking()	Constructor	
	Public	getRank()	Returns a rank	
	Public	setRank()	Sets a rank	
	Public	getMaxScore()	Returns a highest score	
	Public	setMaxScore(int)	Sets a value	
	Public	getTime()	Returns a time	
	Public	setTime()	Sets a value	
	Public	getGameLevel()	Returns a game difficulty level	
	Public	setGameLevel(int)	Sets a value	
	Public	calculateScore()	Returns a total score	
	Public	computeRank()	Returns a rank according to the rules	

Class Name	**Player**			
Inherits from	None			
Description	Adds rank and totalscore attributes			
Attributes	Visibility	Data Type	Name	Description
	Private	int	totalscore	The total score of all games for one player
	Private	int	rank	The rank of a player in the system
Methods	Visibility	Method Name	Description	
	Public	getRank()	Returns the rank	
	Public	setRank()	Sets a value	
	Public	getTotalScore()	Returns a total score	
	Public	setTotalScore()	Sets a value	

Class Name	**Database**			
Inherits from	None			
Description	Adds query statements for ranking			
Attributes	Visibility	Data Type	Name	Description
	Private	Connection	con	A connection to database
Methods	Visibility	Method Name	Description	
	Public	Database()	Constructor	
	Public	connect()	Connects to database	
	Public	disconnect()	Disconnects to database	
	Public	getRanking()	Returns a ranking list	
	Public	getPlayerScore(String)	Returns a score	
	Public	setPlayerScore(String)	Sets a score in database	
	Public	getPlayerRank(String)	Returns a rank from database	
	Public	setPlayerRank(String)	Sets a player's rank	

Class Name	**Controller**		
Inherits from	None		
Description	Adds methods for ranking		
Methods	Visibility	Method Name	Description
	Public	Controller()	Constructor
	Public	connect()	Connects to database
	Public	disconnect()	Disconnects to database
	Public	getRanking()	Returns a ranking list
	Public	getPlayerScore(String)	Returns a player's total score
	Public	getPlayerMaxScore()	Returns a highest score in the system
	Public	getPlayerRank(String)	Return a player's rank

Class Name	**MainFrame**		
Inherits from	JFrame		
Description	An user interface of the Registration subsystem		
Methods	Visibility	Method Name	Description
	Public	MainFrame()	Constructor
	Public	connect()	Connect to the Controller

Class Name	**TextPanel**			
Inherits from	JPanel			
Description	Displays a ranking list in the text area			
Attributes	Visibility	Data Type	Name	Description
	Private	JTextArea	textArea	A text area
Methods	Visibility	Method Name	Description	
	Public	TextPanel()	Constructor	
	Public	appendText(String)	Appends text	

Class Name	**Toolbar**			
Inherits from	JPanel			
Description	A toolbar with buttons			
Attributes	Visibility	Data Type	Name	Description
	Private	JButton	refreshButton	A refresh button
	Private	JButton	rankButton	A rank button
Methods	Visibility	Method Name	Description	
	Public	Toolbar()	Constructor	
	Public	setToolbarListener(ToolbarListener)	Sets event listener	
	Public	actionPerformed(ActionEvent)	Performs actions	

Interface Name	**ToolbarListener**		
Inherits from	None		
Description			
Methods	Visibility	Method Name	Description
	Public	rankEventOccured()	Listens the rank button
	Public	refreshEventOccured()	Listens the refresh button

4 Dynamic Design Scenarios

The following are the descriptions of the execution scenarios of the game initialization, the process of saving a game and the process of loading a game. These systems are involved in the subsystem of the puzzle mechanics.

FIX initialize game **add: play GAME * **Do introduction

4.1 Initialize Game (UI only)

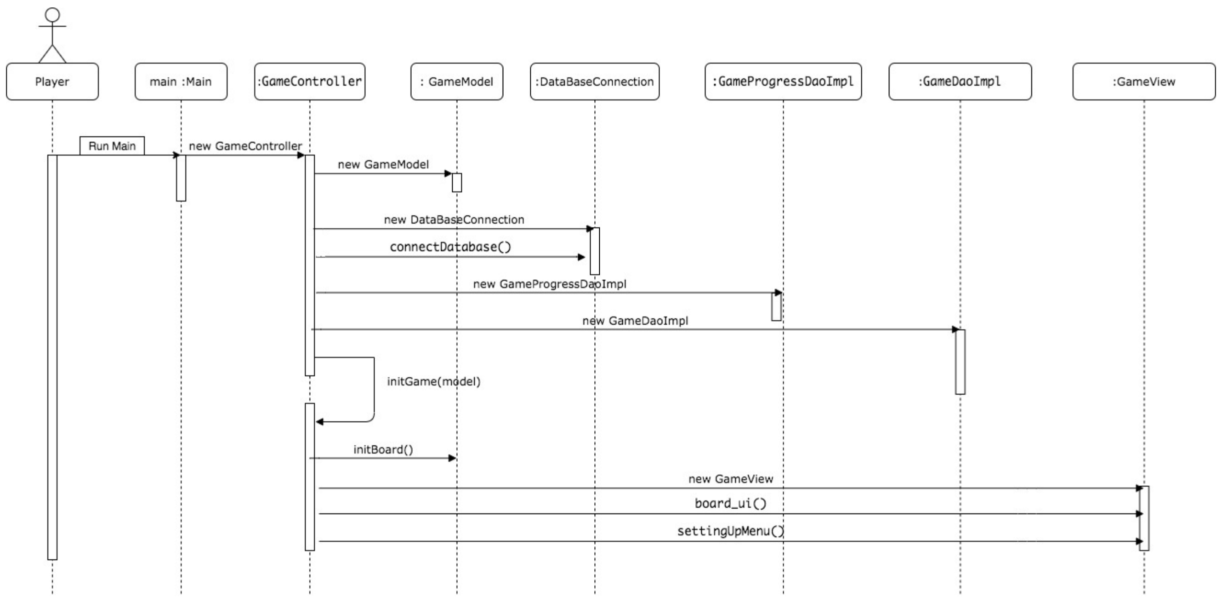

Figure 6: Sequence diagram to initialize a game

4.2 Save Game

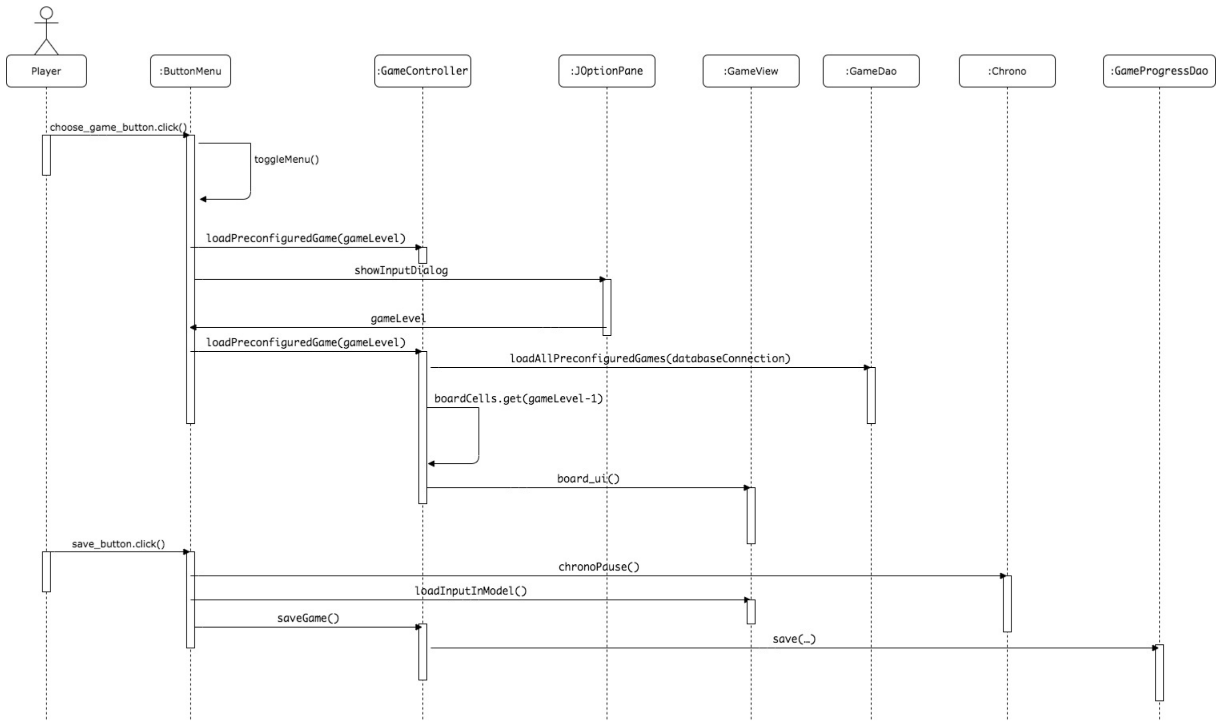

Figure 7: Sequence diagram to save a game

4.3 Load Game

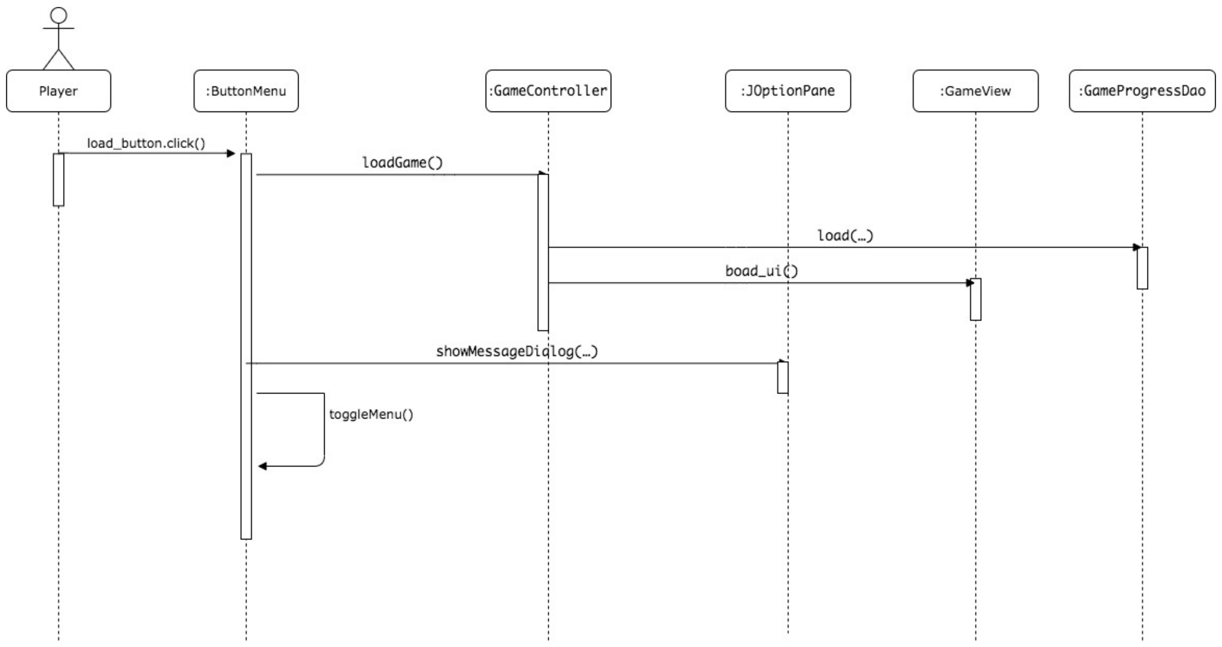

Figure 8: Sequence diagram to load a game