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

Introduction

As a team, our goal is to develop the online Kakuro game. Through each iteration, we have been implementing uses cases, UI components and refactoring the structure of the code. By iteration 3, our goal is to have every component of the current online game implemented in our rendering of the game.

Purpose

With this document, we will be presenting the design of the Kakuro game for the course COMP 354. In the following pages, we first explain in depth our choice of the Model View Controller (MVC) for the architectural design and implied subsystems. Next, we present the modules of the entailed detailed design of the MVC. Finally, we list dynamic scenarios which will describe the interaction between subsystems and units through execution scenarios.

Scope

In order to provide detailed design specifications of the Kakuro game and proceed efficiently with the development of the game, we present this documentation. The detailed explanations, definitions, UML diagrams and sequence diagrams provide the support needed for coherent implementation and communication between each role of the team. The documentation serves as a basis for depicting the software architecture of the game and understanding the implementation of the modules in the detailed design. The scope of this document is it to provide such support, which enables iteration 3 of our project for the Kakuro game.

2 Architectural Design

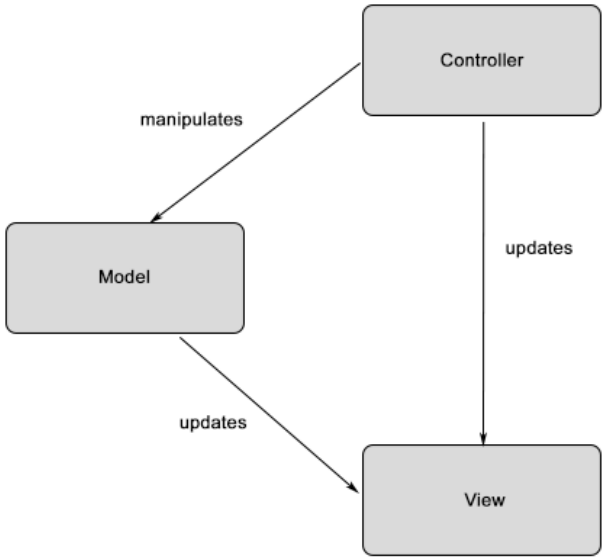

Figure 1: Architecture Diagram

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.

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) : Creates grid of cells by passing a 2D array of cells. Populates it properly and creates the GUI for it.
- getMaxNumberValid() : Returns the maximum number valid for the player to use. In this game, the maximum value is 9
- getMinNumberValid(): Returns the minimum number valid for the player to use. In this game, the minimum value is 1
- getSavedInput() : Returns the UI component that are the cells of the boardgame
- hideBoard(): Makes the current board not visible to the player
- settingTextField(JTextField) : Sets the text field with colors and borders
- showBoard() : Makes the current board visible to the player
- updateView(): Removes a panel if one is already attached to the frame and save a reference to the new panel

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(): Sets up all the buttons (pause, resume, etc) by attaching an action listener to it
- getMainPannel() : Returns the main panel of the game
- toggleMenu(): Changes the state of the game by hiding the load and choose game buttons and setting visible all the other buttons (save, pause, resume, etc.)

2.2.1.3 ChronoView

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

- getTimerLabel() : Returns the timer label associated to the chronometer
- setTimerLabel(String) : Sets the timer label given a string

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 generates the board game given the rules of the game. It contains the following methods:

- getColumns() : Returns an integer value of the number of rows in the game
- getRows() : Returns an integer value of the number of rows in the game
- initBoard(): Creates the board game with the desired number of rows and columns and leave the cells empty

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(): Returns a string of the time given the increments of seconds, minutes and hours

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(GameModel) : Given a GameModel, it initiate the game by initiating the board and it creates a chronoController and a menubarController
- loadGame(): Upload the loaded game progress and display it in the GUI if it's available
- loadInputInModel(): For every cells of every columns and rows, it loads it in the model when they are filled by the player
- loadPreconfigureGame(int)
- 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. If the solution is correct, it returns true. Else it returns false

- 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() : Returns the buttonMenuView
- getView() : Returns the main panel
- isPaused() : Returns true if the game is paused. Else, it returns false
- load() : Loads the game in the GUI
- loadPreconfigureGame(GameDifficulty g): Given a game difficulty, loads it accordingly
- pause(): Pauses the game
- resume() : Resumes the game
- save() : Saves the game in the database
- submit() : Sends the current game for validation

2.2.3.3 ChronoController

The ChronoController class handles the actions that can be perform by the player and by the system itself on the chronometer. It contains the following methods:

- 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

3 Detailed Design

The Karuro system consists of three subsystems: the View module, the Model module, and the Controller module. This MVC model has been designed and implemented since the iteration 1. 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. The database is included in the Model module. The Controller module works like a router, routing the data flows between the Model module and the View module.

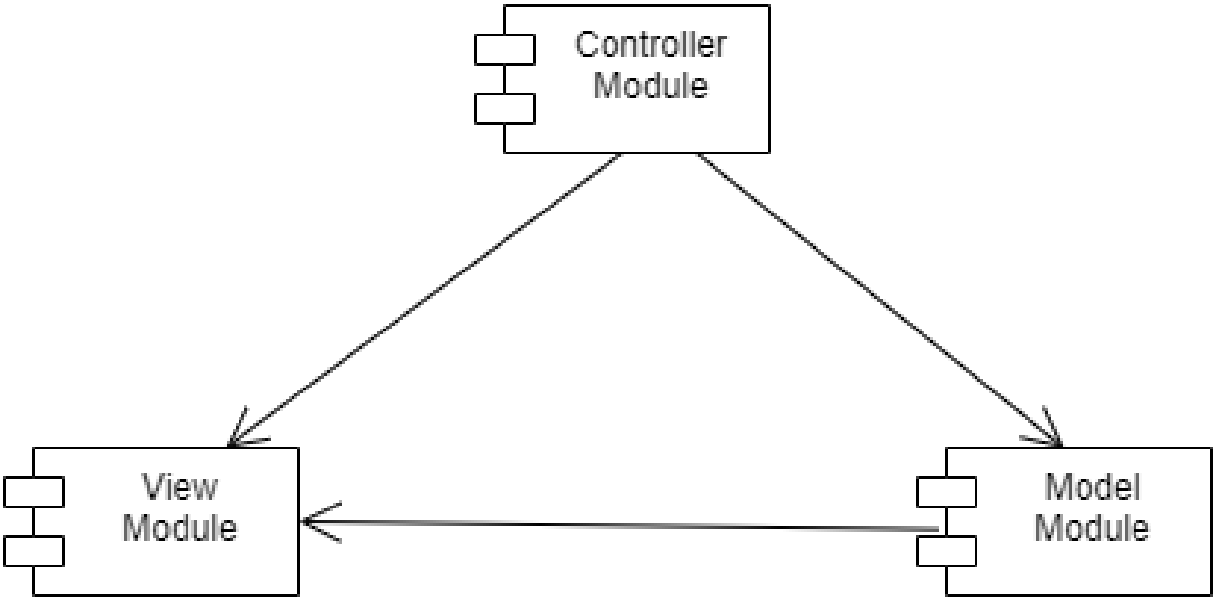

Figure 2: UML of Kakuro Subsystems

The three subsystem or modules composite into one integrated system, the Karuro system. On the other hand, the three subsystems are independent of each other and interact with each other view their respective interfaces. This design practices the principles of high cohesion and low coupling. In this section the detailed design of the three modules will be described separately. Figure.3 shows all classes in the whole Kakuro system.

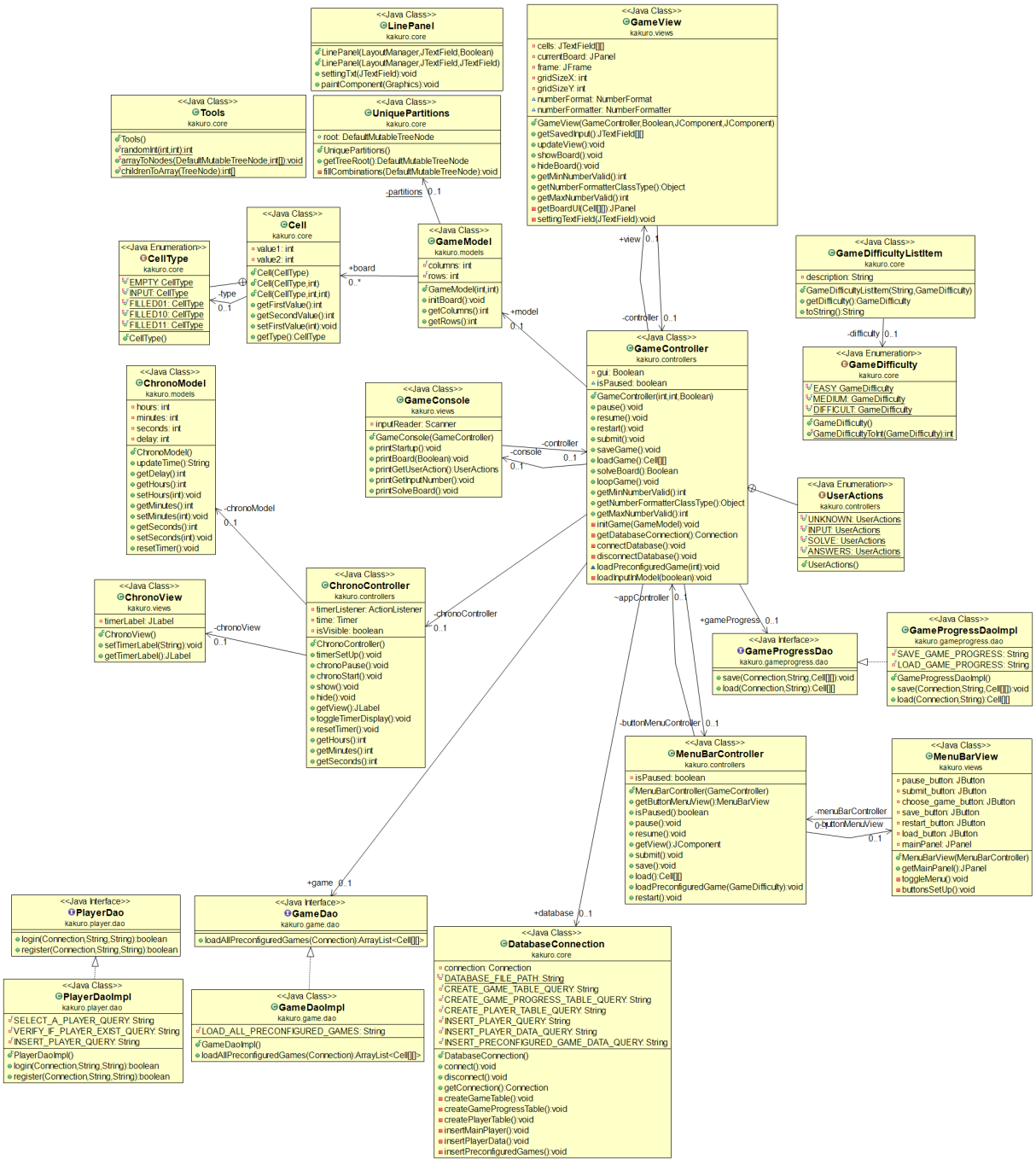

Figure 3: UML of Kakuro System Class

3.1 View Module

Detailed Design Diagram

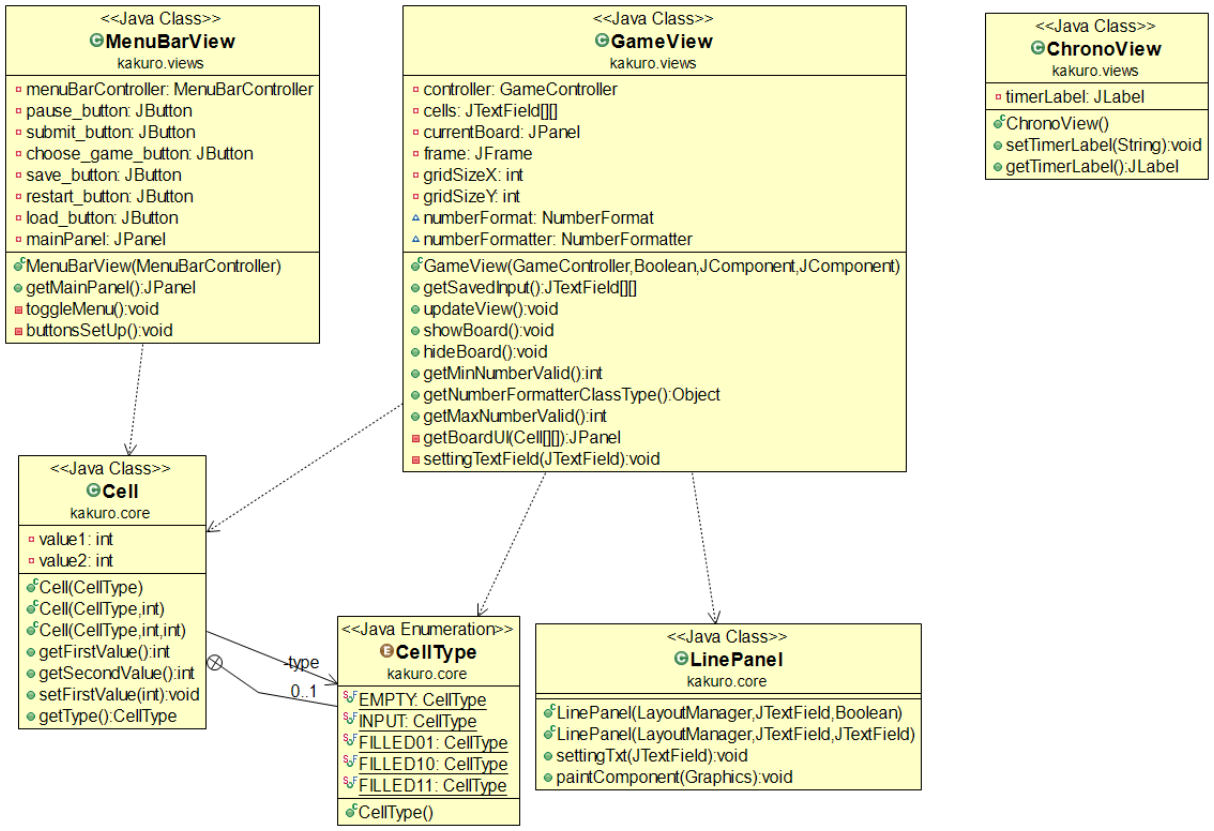

Figure 4: UML of View Module

The View module can be changed easily without impacting the functionality of the Kakuro system because it takes advantage of the MVC architecture. There are three view components at the iteration 2, they are the Game view, the Menu bar view and the Chrono view, which it a timer. The View model displays the data from the Model module, and also updates itself controlled by the Controller module. The Game View module incules the LinePane and Cell class.

Units Description

Class Name	**GameView**		
Inherits from	None		
Description	Game view class which handles the Kakuro game interface		
Attributes	Visibility	Data Type	Name
	Private	GameController	controller
	Private	JPanel	currentBoard
	Private	JFrame	frame
	Private	JTextField[ ][ ]	cells
	Private	int	gridSizeX
	Private	int	gridSizeY
	Package	NumberFormat	numberFormat
	Package	NumberFormatter	numberFormatter
Methods	Visibility	Method Name	
	Public	GameView(GameController, Boolean, JComponent, JComponent)	
	Public	getSaveInput()	
	Public	updateView()	
	Public	showBoard()	
	Public	hideBoard()	
	Public	getNumberFormatterClassType()	
	Public	getMinNumberValid()	
	Public	getMaxNumberValid()	
	Private	getBoardUI(Cell[ ][ ])	
	Private	settingTextField(JTextField)	

Class Name	**MenuBarView**		
Inherits from	None		
Description	Menu bar view class which handles buttons in the view interface		
Attributes	Visibility	Data Type	Name
	Private	MenuBarController	menuBarController
	Private	JButton	pause_button
	Private	JButton	submit_button
	Private	JButton	choose_game_button
	Private	JButton	save_button
	Private	JButton	restart_button
	Private	JButton	load_button
	Private	JPanel	mainPanel
Methods	Visibility	Method Name	
	Public	MenuBarView(MenuBarController)	
	Public	getMainPanel()	
	Private	toggleMenu()	
	Private	buttonSetUp()	

Class Name	**ChronoView**		
Inherits from	None		
Description	Timer view class called Chrono, which acts as our view interface		
Attributes	Visibility	Data Type	Name
	Private	JLabel	timerLabel
Methods	Visibility	Method Name	
	Public	ChronoView()	
	Public	setTimerLabel(String)	
	Public	getTimerLabel()	

Class Name	**Cell**		
Inherits from	None		
Description	Cell class that is defines each cell in our board		
Attributes	Visibility	Data Type	Name
	Private	int	value1
	Private	int	value2
	Package	enum	CellType
Methods	Visibility	Method Name	
	Public	Cell(CellType)	
	Public	Cell(CellType, int)	
	Public	Cell(CellType, int, int)	
	Public	getFirstValue()	
	Public	getSecondValue()	
	Public	setFirstValue(int)	
	Public	getType()	

Class Name	**LinePanel**	
Inherits from	JPanel	
Description	Line panel class used to populate non-playable board cells: adds numbers and diagonal line. It is also extending JPanel to use the graphics with paintComponent() method overriding	
Methods	Visibility	Method Name
	Public	LinePanel(LayoutManager, JTextField, Boolean)
	Public	LinePanel(LayoutManager, JTextField, JTextField)
	Public	settingTxt(JTextField)
	Public	paintComponent(Graphics)

3.2 Model Module

Detailed Design Diagram

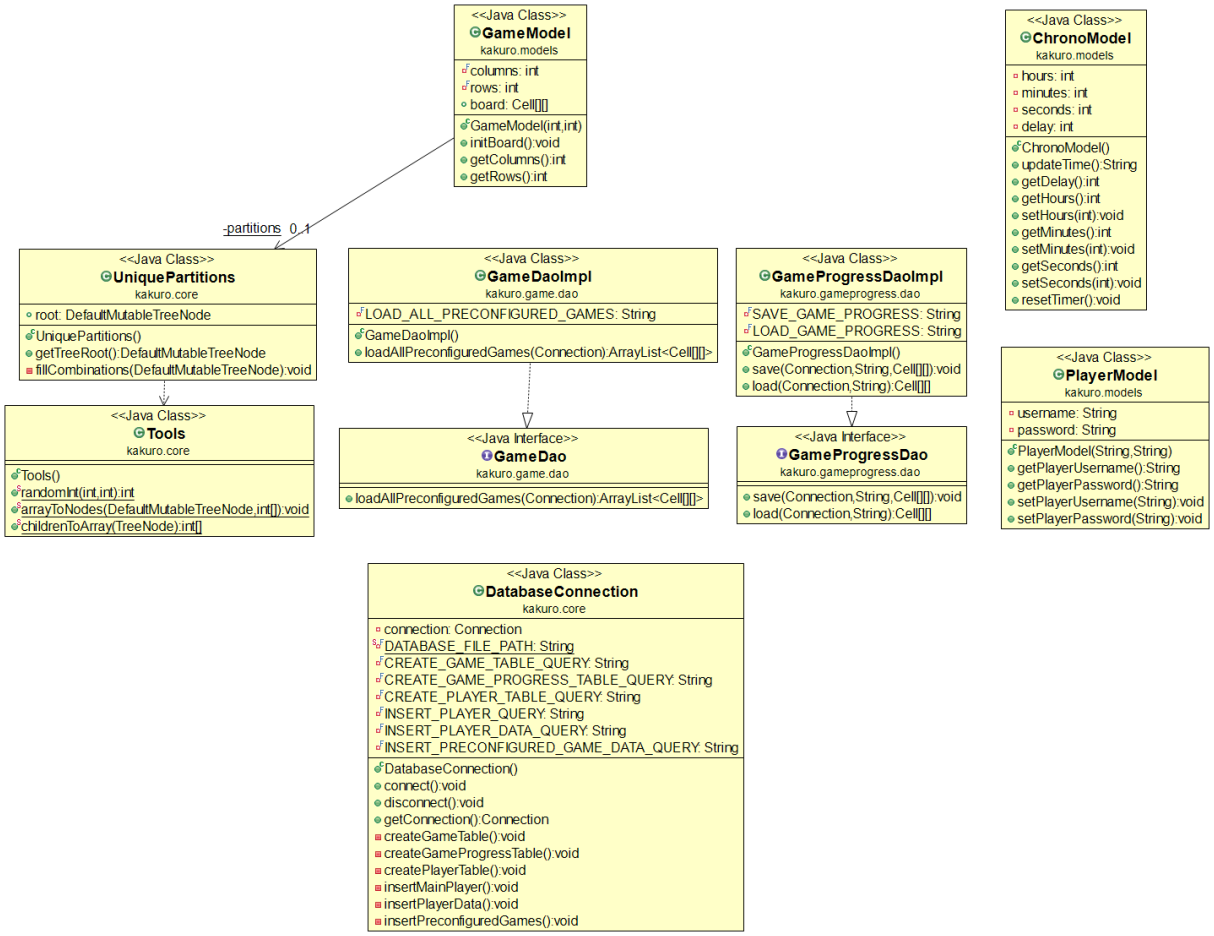

Figure 5: UML of Model Module

The core part of the Kakuro system design is the Model module. This module is used to represent the data of the game, such as the game state, actions, timer, player data etc. Database Connection sits in this module. This module consists of three classes, GameModel, ChronoModel, and PlayerModel. Initial game board and get column and rows data are the key part of the game system. The time information are related to the game score which will be implemented in the iteration 3. The player model handles the username and password information will be used to login and register the game system, which will be implemented in iteration 3. In the iteration 2, the Game model performs the main functionality. The Model module provides the data requests from the View module, and receives and stores the data from the Controller module.The Game Model includes the UniquePartitions, Tools, DatabaseConnection, GameDaoIml and GameProgressDaoImpl class.

Units Description

Class Name	**GameModel**		
Inherits from	None		
Description	Game model class, which acts as our representation data for the object		
Attributes	Visibility	Data Type	Name
	Private	int	columns
	Private	int	rows
	Private	Cell[ ][ ]	board
	Private	UniquePartitions	partitions
Methods	Visibility	Method Name	
	Public	GameModel(int, int)	
	Public	initBoard()	
	Public	getColumns()	
	Public	getRows()	

Class Name	**PlayerModel**		
Inherits from	None		
Description	Player model class, which acts as our representation data for the object		
Attributes	Visibility	Data Type	Name
	Private	String	username
	Private	String	password
Methods	Visibility	Method Name	
	Public	PlayerModel(String, String)	
	Public	getPlayerUsername()	
	Public	getPlayerPassword()	
	Public	setPlayerUsername(String)	
	Public	setPlayerPassword(String)	

Class Name	**ChronoModel**		
Inherits from	None		
Description	Timer model acts as our representation data for the object		
Attributes	Visibility	Data Type	Name
	Private	int	hours
	Private	int	minutes
	Private	int	seconds
	Private	int	delay
Methods	Visibility	Method Name	
	Public	ChronoModel()	
	Public	updateTime()	
	Public	getDelay()	
	Public	getHours()	
	Public	setHour(int)	
	Public	getMinutes()	
	Public	setMinutes(int)	
	Public	getSecounds()	
	Public	setSeconds(int)	
	Public	resetTimer()	

Class Name	**UniquePartitions**		
Inherits from	None		
Description	Lists all possible answers in a Tree ADT		
Attributes	Visibility	Data Type	Name
	Private	DefaultMutableTreeNode	root
Methods	Visibility	Method Name	
	Public	UniquePartitions()	
	Public	getTreeRoot()	
	Public	fillCombinations(DefaultMutableTreeNode)	

Class Name	**Tools**	
Inherits from	None	
Description	Tools for general utilities	
Methods	Visibility	Method Name
	Public	Tools()
	Public	randomInt(int, int)
	Public	arrayToNodes(DefaultMutableTreeNode, int[ ])
	Public	childrenToArray(TreeNode)

Class Name	**GameDaoImpl**		
Implements from	GameDao		
Description	Game implementation class that implements the abstract class for operations in the database		
Attributes	Visibility	Data Type	Name
	Private	String	LOAD_ALL_PRECONFIGURED_GAMES
Methods	Visibility	Method Name	
	Public	GameDaoImpl()	
	Public	loadAllPreconfiguredGames(Connection)	

Class Name	**GameProgressDaoImpl**		
Implements from	GameProgressDao		
Description	Game Progress implementation class that implements the abstract class for operations in the database		
Attributes	Visibility	Data Type	Name
	Private	String	SAVE_GAME_PROGRESS
	Private	String	LOAD_GAME_PROGRESS
Methods	Visibility	Method Name	
	Public	GameProgressDaoImpl()	
	Public	save(Connection, String, Cell[ ][ ])	
	Public	load(Connection, String)	

Class Name	**DatabaseConnection**		
Inherits from	None		
Description	Database connection class to create the connection to our database, which creates the database, tables and inserts preloaded data if they do not exist		
Attributes	Visibility	Data Type	Name
	Private	Connection	connection
	Private	String	DATABASE_FILE_PATH
	Private	String	CREATE_GAME_PROGRESS_TABLE_QUERY
	Private	String	CREATE_PLAYER_TABLE_QUERY
	Private	String	INSERT_PLAYER_QUERY
	Private	String	INSERT_PLAYER_DATA_QUERY
	Private	String	INSERT_PRECONFIGURED_GAME_DATA_QUERY
Methods	Visibility	Method Name	
	Public	DatabaseConnection()	
	Public	connect()	
	Public	disconnect()	
	Public	getConnection()	
	Private	createGameTable()	
	Private	createGameProgressTable()	
	Private	createPlayerTable()	
	Private	insertMainPlayer()	
	Private	insertPlayerData()	
	Private	insertPreconfiuredGames()	

3.3 Controller Module

Detailed Design Diagram

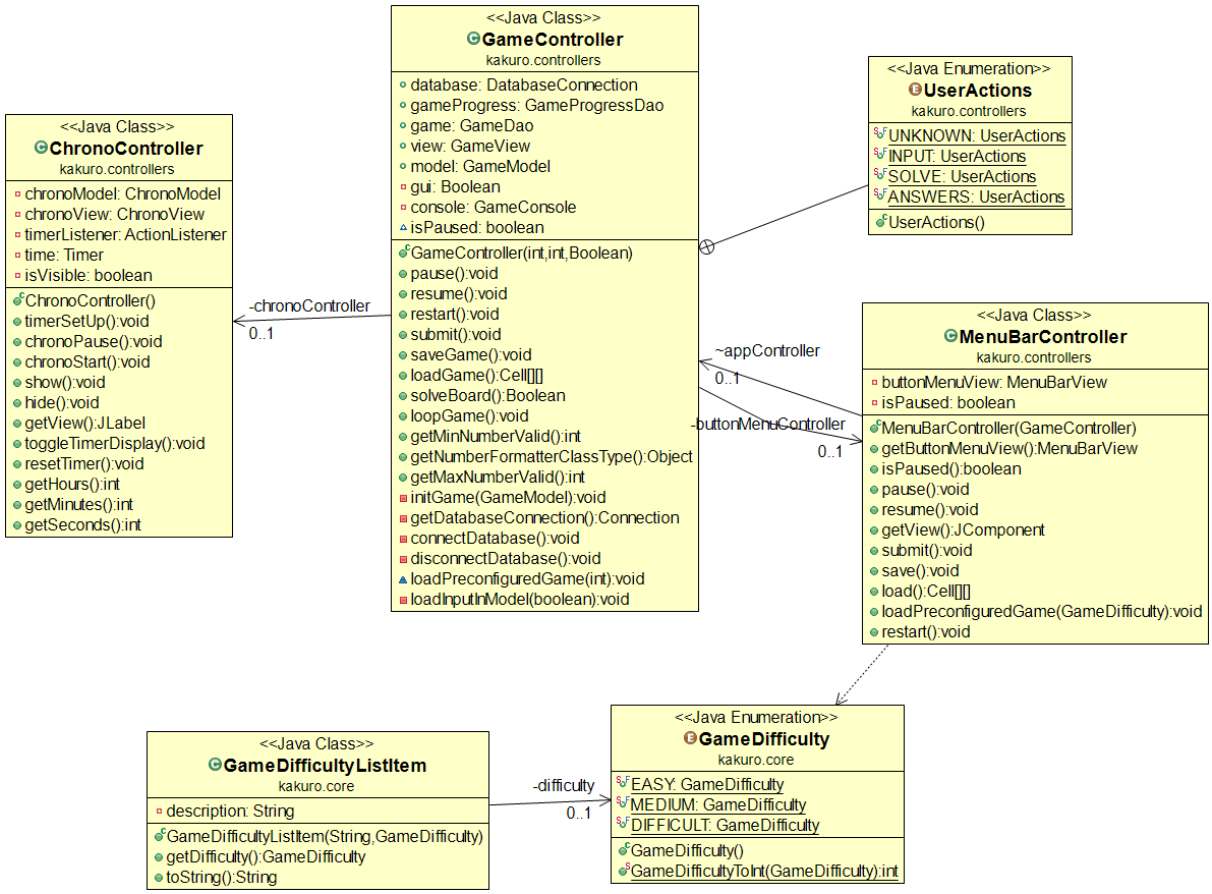

Figure 6: UML of Controller Module

The Controller module orchestrates the Model module and View module of the Karuro system. The Game Controller accepts input from the player and the button click events, and it also validates the input checking and notifies the user. The Controller module handles the data flows between the Model module and the View module. It handles connection and disconnection to the database. It also save game, load game, pause game, resume game, and restart game through the Menu bar controller. The Chrono controller is a timer controller, and it handles all time related functionality. The Game Controller module includes the MenuBarController, ChronoController and GameDifficultyListItem class.

Units Description

Class Name	**GameController**		
Inherits from	None		
Description	Basically orchestrates the model and view of the game		
Attributes	Visibility	Data Type	Name
	Public	DatabaseConnection	database
	Public	GameProgressDao	gameProgress
	Public	GameDao	game
	Public	GameView	view
	Public	GameModel	model
	Public	enum	UserActions
	Private	Boolean	gui
	Private	GameConsole	console
	Package	boolean	isPaused
Methods	Visibility	Method Name	
	Public	GameController(int, int, Boolean)	
	Public	UserActions()	
	Public	pause()	
	Public	resume()	
	Public	restart()	
	Public	submit()	
	Public	saveGame()	
	Public	loadGame()	
	Public	solveBoard()	
	Public	loopGame()	
	Public	getMinNumberValid()	
	Public	getNumberFormatterClassType()	
	Public	getMaxNumberValid()	
	Public	initGame(GameModel)	
	Private	getDatabaseConnection()	
	Private	connectDatabase()	
	Private	disconnectDatabase()	
	Private	loadInputInModel(boolean)	
	Package	loadPreconfiguredGame(int)	

Class Name	**ChronoController**		
Inherits from	None		
Description	Timer controller acts as an orchestrator between the model and the view		
Attributes	Visibility	Data Type	Name
	Private	ChronoModel	chronoModel
	Private	ChronoView	chronoView
	Private	ActionListener	timerListener
	Private	Timer	time
	Private	boolean	isPaused
Methods	Visibility	Method Name	
	Public	ChronoController()	
	Public	timerSetUp()	
	Public	chronoPause()	
	Public	chronoStart()	
	Public	show()	
	Public	hide()	
	Public	getView()	
	Public	toggleTimerDisplay()	
	Public	resetTimer()	
	Public	getHours()	
	Public	getMinutes()	
	Public	getSeconds()	

Class Name	**MenuBarController**		
Inherits from	None		
Description	Handles the buttons in our game application		
Attributes	Visibility	Data Type	Name
	Private	MenuBarView	buttonMenuView
	Private	boolean	isPaused
Methods	Visibility	Method Name	
	Public	MenuBarController(GameController)	
	Public	getButtonMenuView()	
	Public	isPaused()	
	Public	pause()	
	Public	resume()	
	Public	getView()	
	Public	submit()	
	Public	save()	
	Public	load()	
	Public	loadPreConfiguredGame(GameDifficulty)	
	Public	restart()	

Class Name	**GameDifficulty**	
Inherits from	None	
Description	An enumerator for different levels of game difficulties	
Methods	Visibility	Method Name
	Public	GameDifficulty()
	Public	GameDifficultyToInt(GameDifficulty)

Class Name	**GameDifficultyListItem**		
Inherits from	None		
Description	The game difficulty found in the dropdown menu when pre-loading a game		
Attributes	Visibility	Data Type	Name
	Private	String	description
	Private	GameDifficulty	difficulty
Methods	Visibility	Method Name	
	Public	GameDifficultyListItem(String, GameDifficulty)	
	Public	getDifficulty()	
	Public	toString()	

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.

4.1 Initialize Game (UI only)

In iteration 2, we do not yet have a "start game" button implemented in the UI of the game. Therefore, a use case for the current iteration is clicking of the run main button of Eclipse, which initializes the game. We also have a version of the game that can be played on the console, but for this sequence diagram, only the methods implicated in the UI of the game are involved.

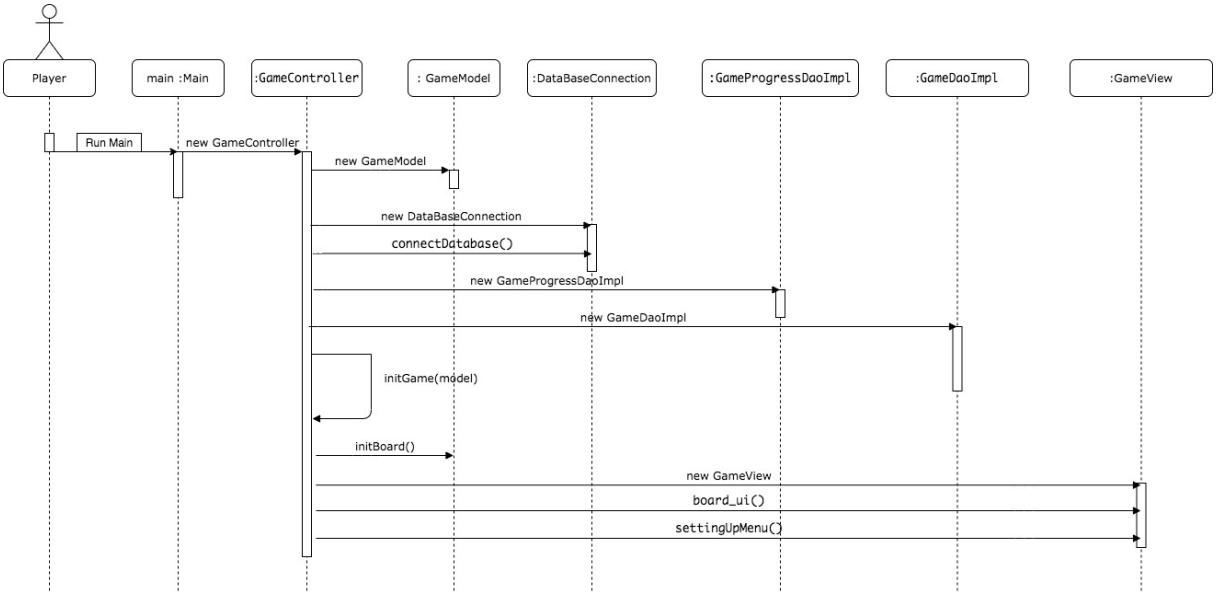

Figure 7: Sequence diagram to initialize a game

4.2 Save Game

A crucial use case for iteration 2 that has been implemented is to save the game. Once the player has initialized the game by clicking run main, they can next choose a game level and input numbers from 1 to 9 in the board cells. If they haven't finished the game and wish to continue at a later date, this is the use case that represents this eventuality of user interaction.

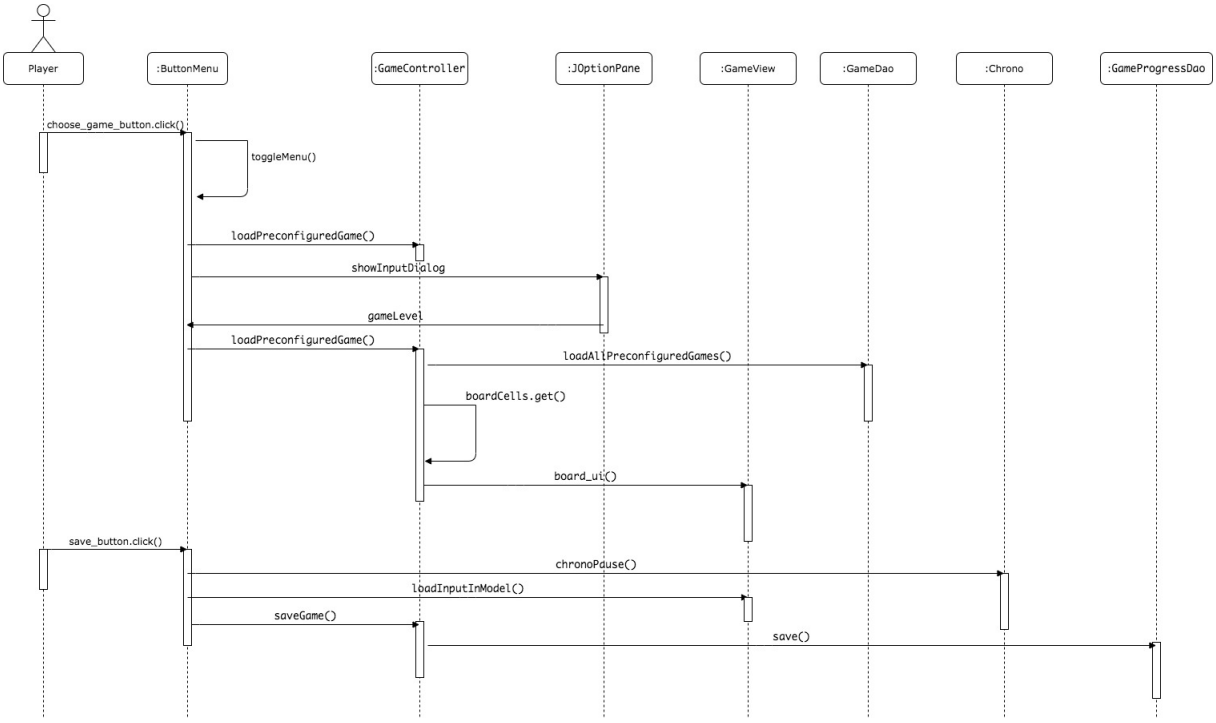

Figure 8: Sequence diagram to save a game

4.3 Load Game

A use case that goes hand in hand with saving the game is loading the game. Once a player has saved an in progess game, they can come back to it by loading the game they were previously playing. This is done by clicking the UI button "Load Game". This action populates the board of the game with saved inputs and the saved time value of the timer.

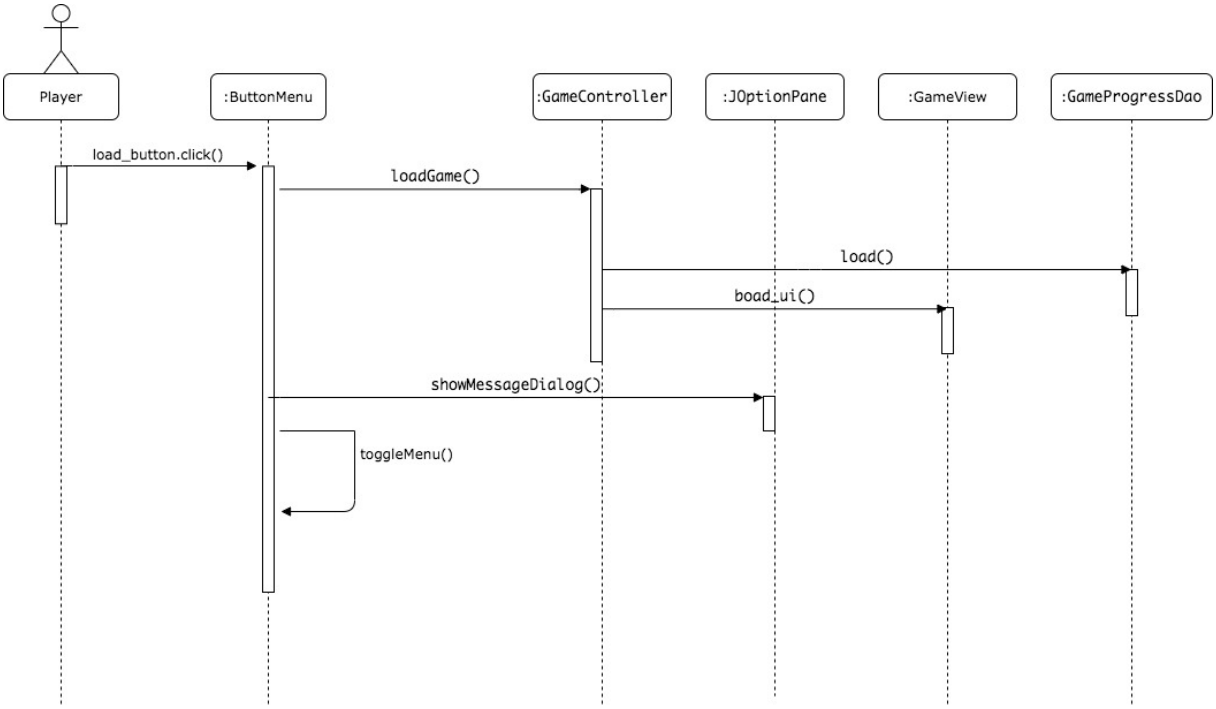

Figure 9: Sequence diagram to load a game