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

2.1 Architectural Diagram

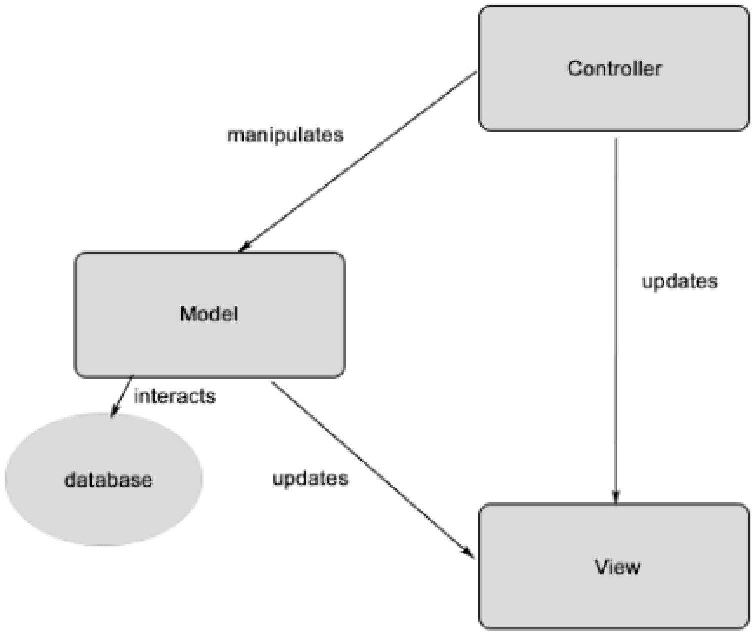

Figure 1: Architecture Diagram

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

Specification of the software interfaces between the subsystems, i.e. specific messages (or function calls) that are exchanged by the subsystems. These are also often called "Module Interface Specifications". Description of the parameters to be passed into these function calls in order to have a service fulfilled, including valid and invalid ranges of values. Each subsystem interface must be presented in a separate subsection.

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.

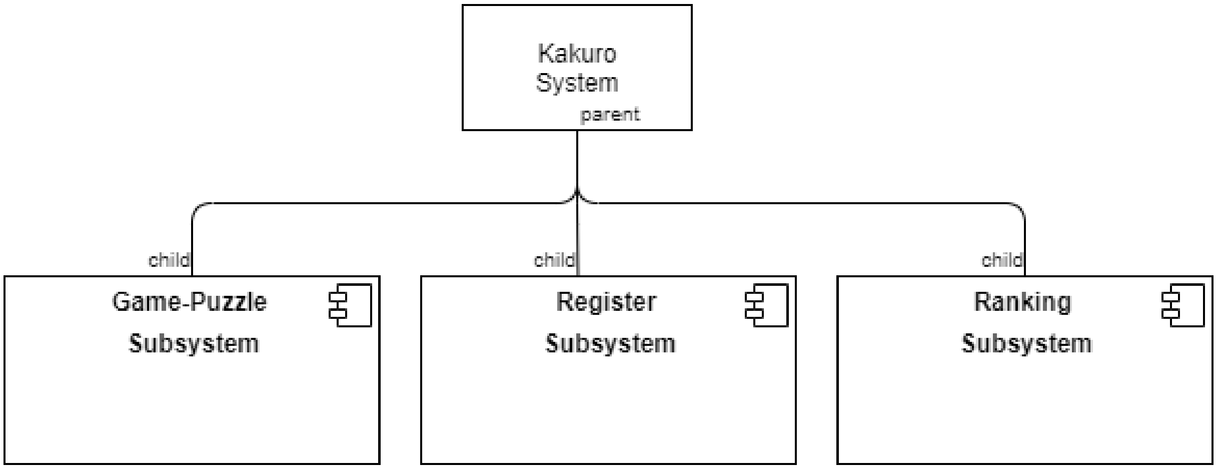

Figure 2: UML of Kakuro Subsystems

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 views, apply different models, and use different controller. The Ranking subsystem intersects with the Registration subsystem in the username-score part, and both of them have dependency with the Game-Puzzle subsystem in the aspect of game score coming from the game result and the specific user of that game.

3.1 Game-Puzzle Subsystem

Detailed Design Diagram

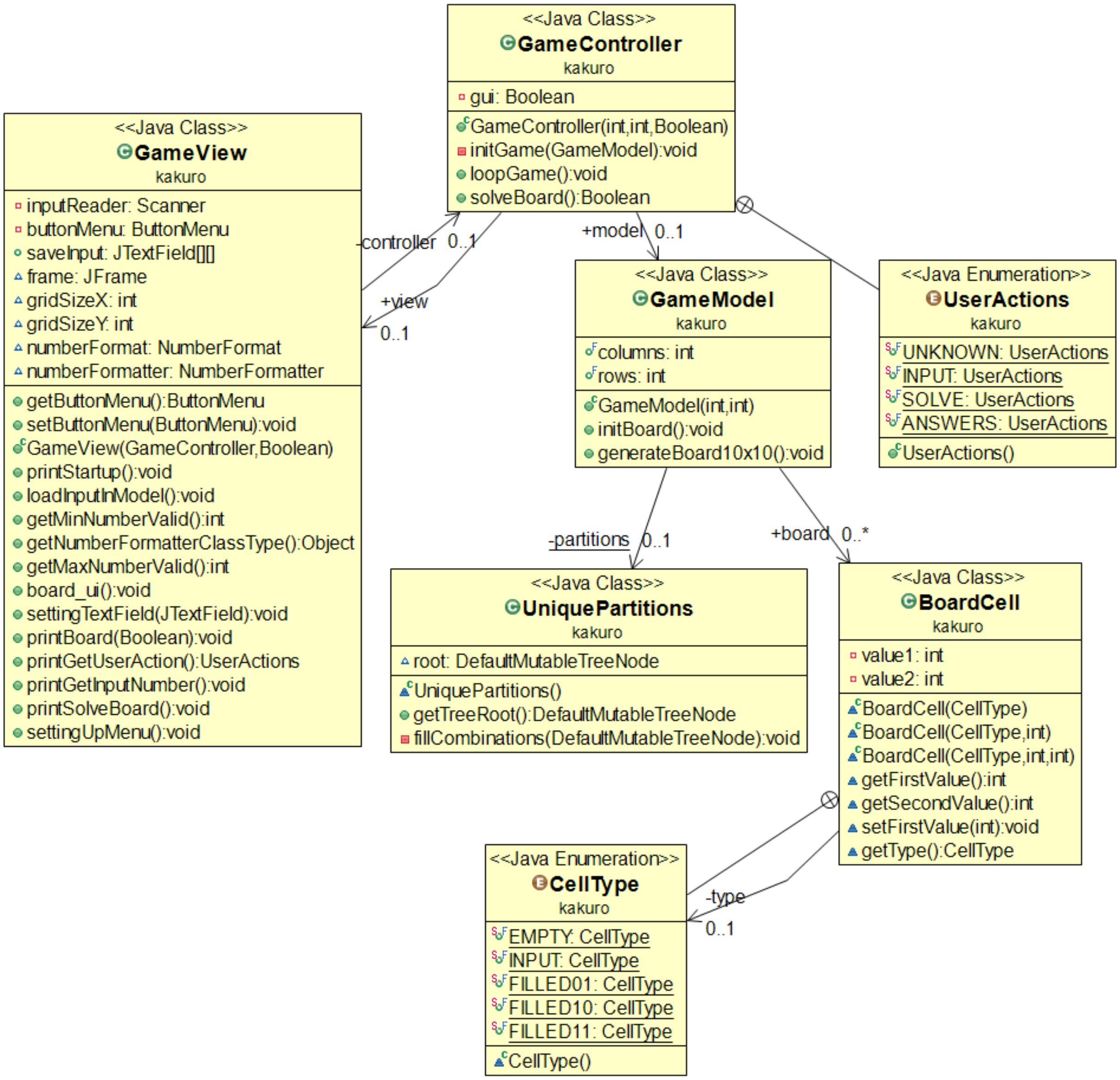

Figure 3: UML of Geme-Puzzle Subsystems

Units Description

List each class in this subsystem and write a short description of its purpose, as well as notes or reminders useful for the programmers who will implement them. List all attributes and functions of the class.

Class Name	**GameController**			
Inherits from	None			
Description	The controller of subsystem			
Attributes	Visibility	Data Type	Name	Description
	Public	enum	UserActions	User actions
	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	

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	boardUi()		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	**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	**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	

3.2 Registration Subsystem

Detailed Design Diagram

UML class diagram depicting the internal structure of the subsystem, accompanied by a paragraph of text describing the rationale of this design.

Units Description

List each class in this subsystem and write a short description of its purpose, as well as notes or reminders useful for the programmers who will implement them. List all attributes and functions of the class.

3.3 Ranking Subsystem

Detailed Design Diagram

UML class diagram depicting the internal structure of the subsystem, accompanied by a paragraph of text describing the rationale of this design.

Units Description

List each class in this subsystem and write a short description of its purpose, as well as notes or reminders useful for the programmers who will implement them. List all attributes and functions of the class.

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)

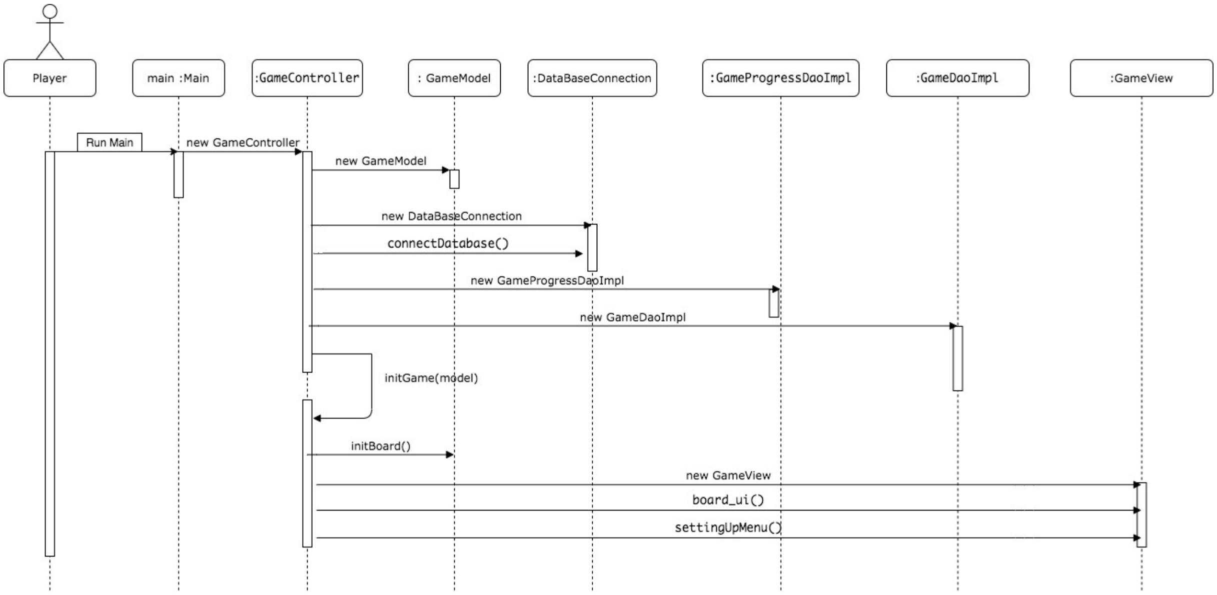

Figure 4: Sequence diagram to initialize a game

4.2 Save Game

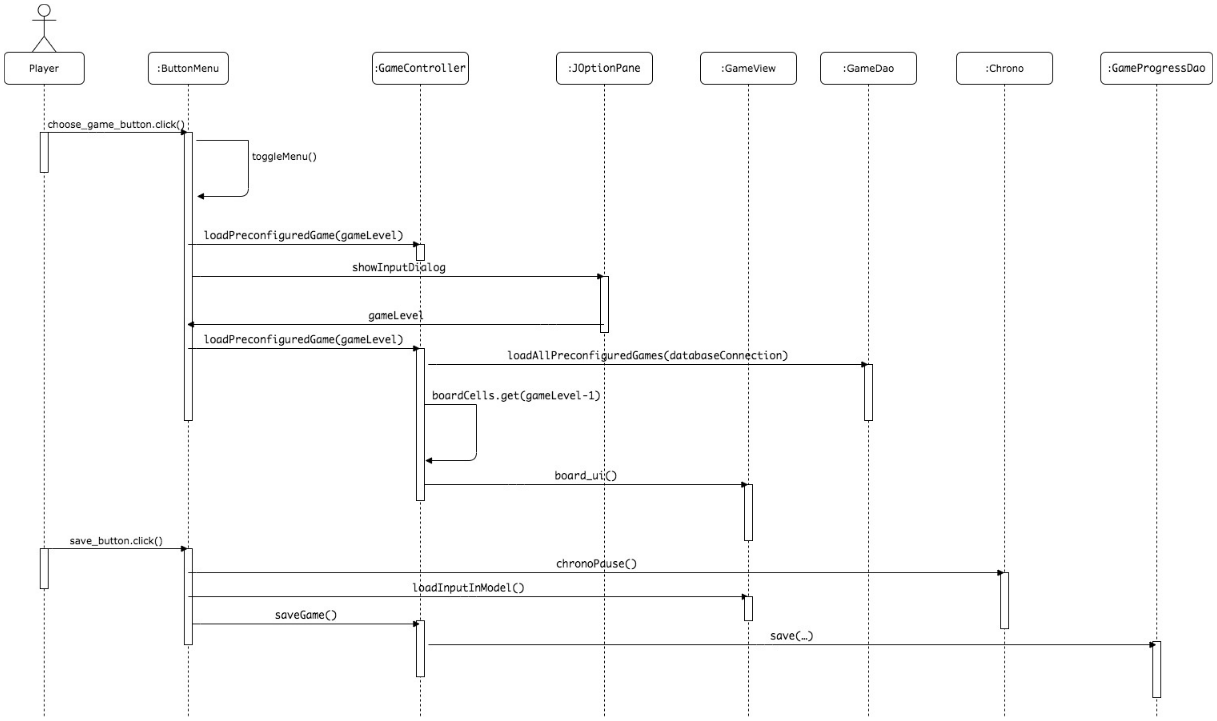

Figure 5: Sequence diagram to save a game

4.3 Load Game

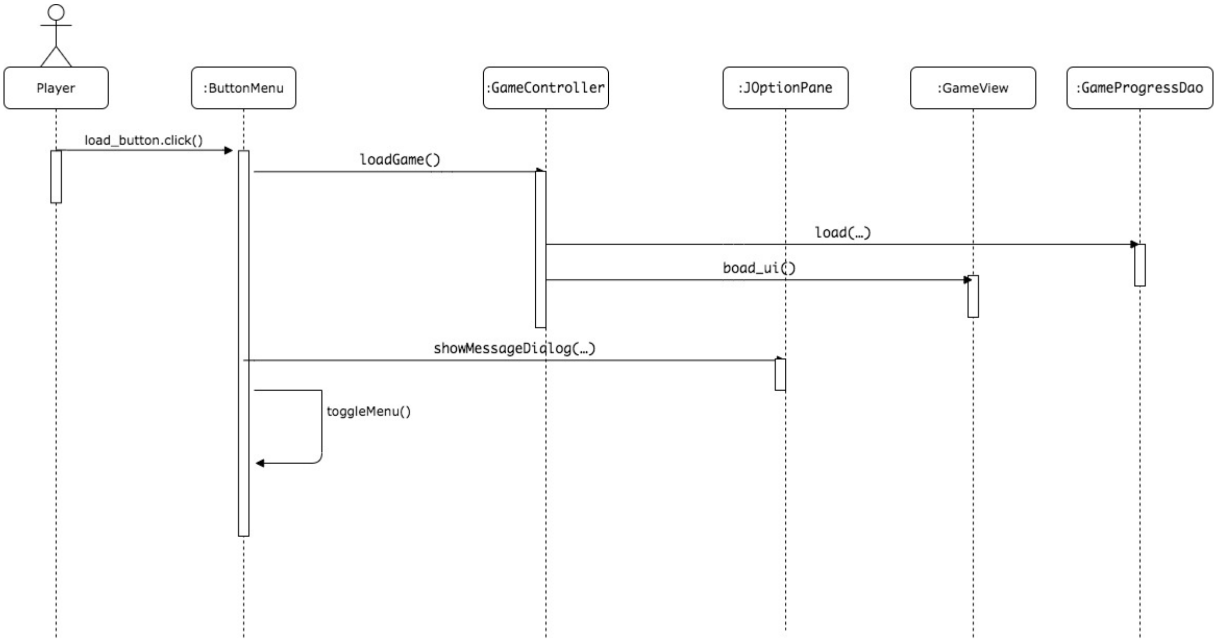

Figure 6: Sequence diagram to load a game