Sudoku - Design Manual

Tsugunobu Miyake , Doruk Ozar , Reva Sharma , Kit Jackson
Prof. Brian King
Software Engineering and Design

Introduction:

Our final project is a sudoku puzzle game. We have designed the game logic and a very remarkable user interface. We started our game logic by finding sudoku puzzles from an online website in the form of a CSV file. Then we read the CSV files and combined them with the user interface that we have developed. We have also made an algorithm where it picks a random sudoku puzzle every time we click on the play button from the database and displays it on the UI. We also have implemented an algorithm where we check the user input so that we can check if it is a valid input or not. If the input is not valid, we color the number that is inputted and also the cell red. If it is valid then we color it to blue so that the user won't confuse himself/herself with the given numbers at the beginning of the game. We have to implement an enumeration class where we have the states of the cells. It can have 6 different states: Default, Selected, Relevant, Valid Input, Invalid Input, and Conflicting. When the state of a cell is Default the user cannot change the number in that cell. If the state is Selected, that means that the cell is selected and all the row and column and the 3x3 square that the cell is in becomes a Relevant state so we color those to gray so that the user can see clearly where they are inputting.

User Stories:

We designed three user stories for this project.

Özkan Pektemek: (Complete)

- He is a high school student. He just started playing sudoku and is interested in solving sudoku puzzles, however, he doesn't know where to start. He is a beginner and does not have any experience with the game. He just knows about the rules.
- He comes from a family where no one plays sudoku but he has played other brain games such as tic tac toe and minesweeper.

- He prefers some kind of tutorials or instructions so that he can go and check them if he gets stuck on a puzzle.

Chloe Taylor: (Complete)

- She is a college student. She has a fair amount of knowledge and experience with sudoku puzzles. She comes from a family where some of the family members have played sudoku before.
- She likes playing sudoku but she hates it when she gets stuck so she prefers easier puzzles or hints with the more challenging puzzles.
- She has been playing sudoku for several years. She can complete easy puzzles but she doesn’t like to spend more time on more complex puzzles.

Marius Madsen: (Incomplete)

- He is a math professor. He prefers harder and more complicated puzzles. He likes to have a timer because he wants to challenge himself every time he solves a sudoku puzzle. He is almost an expert, and does not need any tutorials or help with the game instructions. He just wants puzzles that challenge him.
- He is very familiar with sudoku and comes from a family where almost everyone solved sudoku puzzles in their lives. He likes to solve more challenging puzzles and he is really competitive.
- He has a Ph.D. Computer Science and Mathematics from Harvard University. He likes logic games and he wants them to be accessible to everyone.

We have completed the first and the second user stories. We have implemented a game where you press the play button and it will give you a random sudoku puzzle that is not hard, it will point out your mistakes by highlighting them in red color and also counting them so that users can improve themselves. Also, we have implemented an undo, give up, and a tutorial button where you can use undo if you want to change the number that you input, or you can use the give up button to start over with a new sudoku puzzle. Users can also use the tutorial button in order to learn about the sudoku game rules and our game’s properties. We also implemented a timer where people can time themselves to get better and be more competitive. We have also implemented a counter where it counts the number of remaining cells so that people can know how many cells they have to fill in. The other last one is incomplete because, For the hard-level user, we did not have enough time to implement a level-selecting algorithm so we did not have hard or medium-level puzzles.

Relevant CRC Cards:

Class: SudokuDB

Responsibilities	Collaborators
Reads CSV file containing sudoku quizzes	
Stores sudoku quizzes	
Generates a sudoku board with a random quiz (for each time user plays a new game)	

Class: SudokuModel

Responsibilities	Collaborators
Initializes new game with a random sudoku quiz	SudokuDB
Initializes the 9x9 grid of the board with default state	SudokuGameLogic
Puts the number inputted by user on the board	SudokuCell
Deletes a number from the board	GridState
Undo a movement made on the board	SudokuStopWatch
Updates cells on the board	
Keeps track of remaining cells and mistakes made	
Stores reference to the stopwatch used to measure the elapsed time to solve sudoku	

Enu: GridState

Responsibilities	Collaborators
Stores state of each cell after user input	

Class: SudokuGameLogic

Responsibilities	Collaborators
Works with a copy of the current sudoku board	SudokuChecker
Checks to see if cell can be edited based on the rules of the game	SudokuMovement
Undo a movement made on the board	
Deletes a number from the board	
Keeps track of number of remaining cells on the board	
Stores history of movements made on the board	

Class: SudokuChecker

Responsibilities	Collaborators
Checks for the inputted number in a given row	ErrorType
Checks for the inputted number in a given column	SudokuCell
Checks for the inputted number in a 3x3 grid	
Stores the conflicting cell if input is invalid	

Class: ErrorType

Responsibilities	Collaborators
Stores the types of error based on the row, column, and grid the number is entered in	

Class: SudokuMovement

Responsibilities	Collaborators
Stores each movement made on the board	

Class: SudokuCell

Responsibilities	Collaborators
Stores the coordinate info of a particular cell	

Class: StopWatch

Responsibilities	Collaborators

Measures the time elapsed to finish the sudoku game	

Class: AppController

Responsibilities	Collaborators
Controller used to refer to the main JavaFX application	SudokuMain
	SudokuModel

Class: SudokuGameController

Responsibilities	Collaborators
Controls the actual game screen of the app	SudokuMain

Class: MenuController

Responsibilities	Collaborators
Controls the menu screen of the app i.e. the first screen after app is launched	SudokuMain

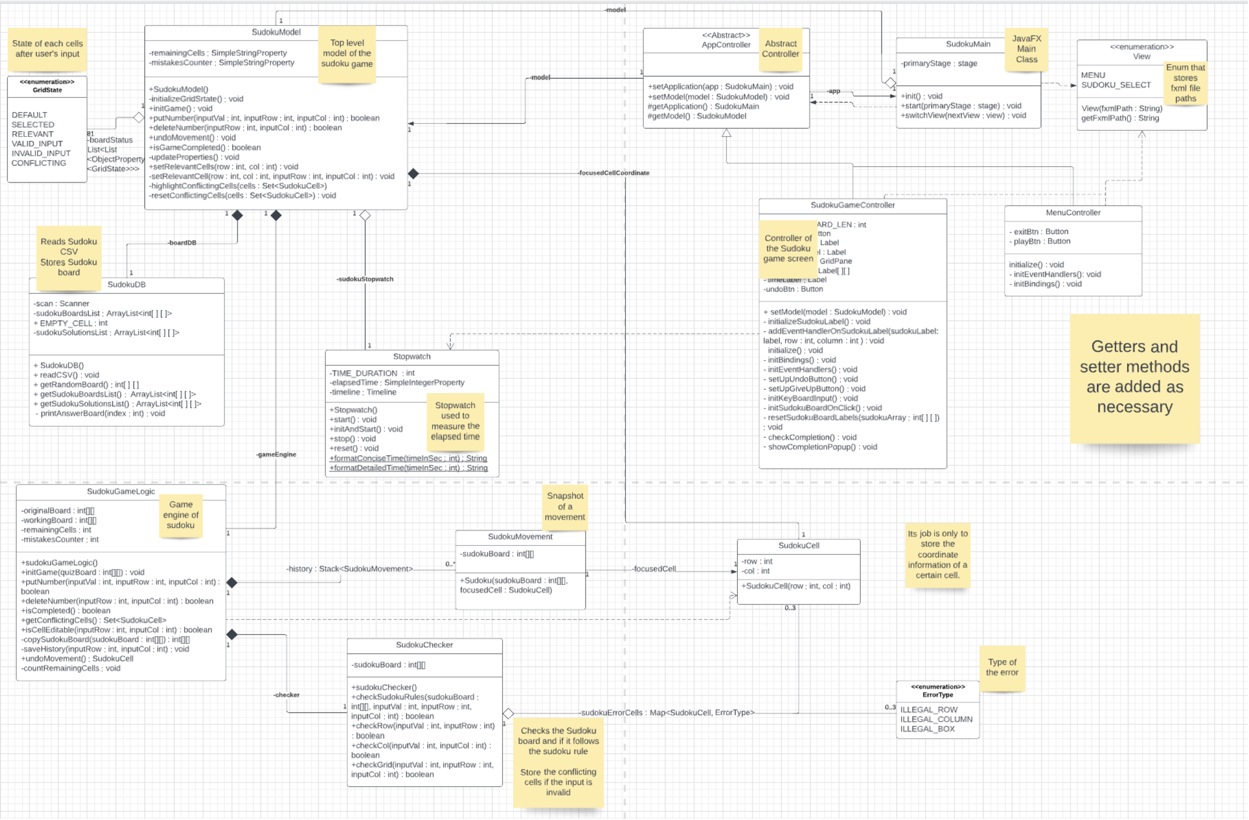
Class: SudokuMain

Responsibilities	Collaborators
Runs the actual app i.e. the main JavaFX application	SudokuModel

Enum: View

Responsibilities	Collaborators
Stores the FXML file paths for each view	

UML Diagram:


State of each cells after user's input
<<enumeration>>
GridState
DEFAULT
SELECTED
RELEVANT
VALID_INPUT
INVALID_INPUT
CONFLICTING
81
-boardStatus
List<List<ObjectProperty<GridState>>>
SudokuModel
Top level model of the sudoku game
-remainingCells : SimpleStringProperty
-mistakesCounter : SimpleStringProperty
+SudokuModel()
-initializeGridSrtate() : void
+initGame() : void
+putNumber(inputVal : int, inputRow : int, inputCol : int) : boolean
+deleteNumber(inputRow : int, inputCol : int) : boolean
+undoMovement() : void
+isGameCompleted() : boolean
-updateProperties() : void
+setRelevantCells(row : int, col : int) : void
-setRelevantCell(row : int, col : int, inputRow : int, inputCol : int) : void
-highlightConflictingCells(cells : Set<SudokuCell>)
-resetConflictingCells(cells : Set<SudokuCell>) : void
-model
-focusedCellCoordinate
-boardDB
-sudokuStopwatch
-gameEngine
Reads Sudoku CSV
Stores Sudoku board
SudokuDB
-scan : Scanner
-sudokuBoardsList : ArrayList<int[ ] [ ]>
+ EMPTY_CELL : int
-sudokuSolutionsList : ArrayList<int[ ] [ ]>
+ SudokuDB()
+ readCSV() : void
+ getRandomBoard() : int[ ] [ ]
+ getSudokuBoardsList() : ArrayList<int[ ] [ ]>
+ getSudokuSolutionsList() : ArrayList<int[ ] [ ]>
- printAnswerBoard(index : int) : void
Stopwatch
-TIME_DURATION : int
-elapsedTime : SimpleIntegerProperty
-timeline : Timeline
+Stopwatch()
+start() : void
+initAndStart() : void
+stop() : void
+reset() : void
+formatConciseTime(timeInSec : int) : String
+formatDetailedTime(timeInSec : int) : String
Stopwatch used to measure the elapsed time
<<Abstract>>
AppController
Abstract Controller
+setApplication(app : SudokuMain) : void
+setModel(model : SudokuModel) : void
#getApplication() : SudokuMain
#getModel() : SudokuModel
-app
SudokuMain
JavaFX Main Class
-primaryStage : stage
+init() : void
+start(primaryStage : stage) : void
+switchView(nextView : view) : void
<<enumeration>>
View
Enum that stores fxml file paths
MENU
SUDOKU_SELECT
View(fxmlPath : String)
getFxmlPath() : String
SudokuGameController
Controller of the Sudoku game screen
ARD_LEN : int
tton
Label
l : Label
GridPane
Label[ ][ ]
- timeLabel : Label
-undoBtn : Button
+ setModel(model : SudokuModel) : void
- initializeSudokuLabel() : void
- addEventHandlerOnSudokuLabel(sudokuLabel: label, row : int, column : int) : void
initialize() : void
- initBindings() : void
- initEventHandlers() : void
- setUpUndoButton() : void
- setUpGiveUpButton() : void
- initKeyBoardInput() : void
- initSudokuBoardOnClick() : void
- resetSudokuBoardLabels(sudokuArray : int[ ] [ ]) : void
- checkCompletion() : void
- showCompletionPopup() : void
MenuController
- exitBtn : Button
- playBtn : Button
initialize() : void
- initEventHandlers(): void
- initBindings() : void
Getters and setter methods are added as necessary
SudokuGameLogic
Game engine of sudoku
-originalBoard : int[][]
-workingBoard : int[][]
-remainingCells : int
-mistakesCounter : int
+sudokuGameLogic()
+initGame(quizBoard : int[][]) : void
+putNumber(inputVal : int, inputRow : int, inputCol : int) : boolean
+deleteNumber(inputRow : int, inputCol : int) : boolean
+isCompleted() : boolean
+getConflictingCells() : Set<SudokuCell>
+isCellEditable(inputRow : int, inputCol : int) : boolean
-copySudokuBoard(sudokuBoard : int[][]) : int[][]
-saveHistory(inputRow : int, inputCol : int) : void
+undoMovement() : SudokuCell
-countRemainingCells : void
-history : Stack<SudokuMovement>
0..*
Snapshot of a movement
SudokuMovement
-sudokuBoard : int[][]
+Sudoku(sudokuBoard : int[][], focusedCell : SudokuCell)
-focusedCell
SudokuCell
-row : int
-col : int
+SudokuCell(row : int, col : int)
0..3
Its job is only to store the coordinate information of a certain cell.
-checker
SudokuChecker
-sudokuBoard : int[][]
+sudokuChecker()
+checkSudokuRules(sudokuBoard : int[][], inputVal : int, inputRow : int, inputCol : int) : boolean
+checkRow(inputVal : int, inputRow : int) : boolean
+checkCol(inputVal : int, inputCol : int) : boolean
+checkGrid(inputVal : int, inputRow : int, inputCol : int) : boolean
Checks the Sudoku board and if it follows the sudoku rule
Store the conflicting cells if the input is invalid
-sudokuErrorCells : Map<SudokuCell, ErrorType>
0..3
<<enumeration>>
ErrorType
Type of the error
ILLEGAL_ROW
ILLEGAL_COLUMN
ILLEGAL_BOX

Model:

The top-level model of the game is SudokuModel. It consists of SudokuGameLogic, which is a game engine of Sudoku. This class launches the SudokuGameLogic game engine with a random board generated by SudokuDB class. This model updates the status of the game, such as the remaining-cells label and mistakes-made label, as well as the background colors of Sudoku cells based on the user's click and the user's keyboard input.

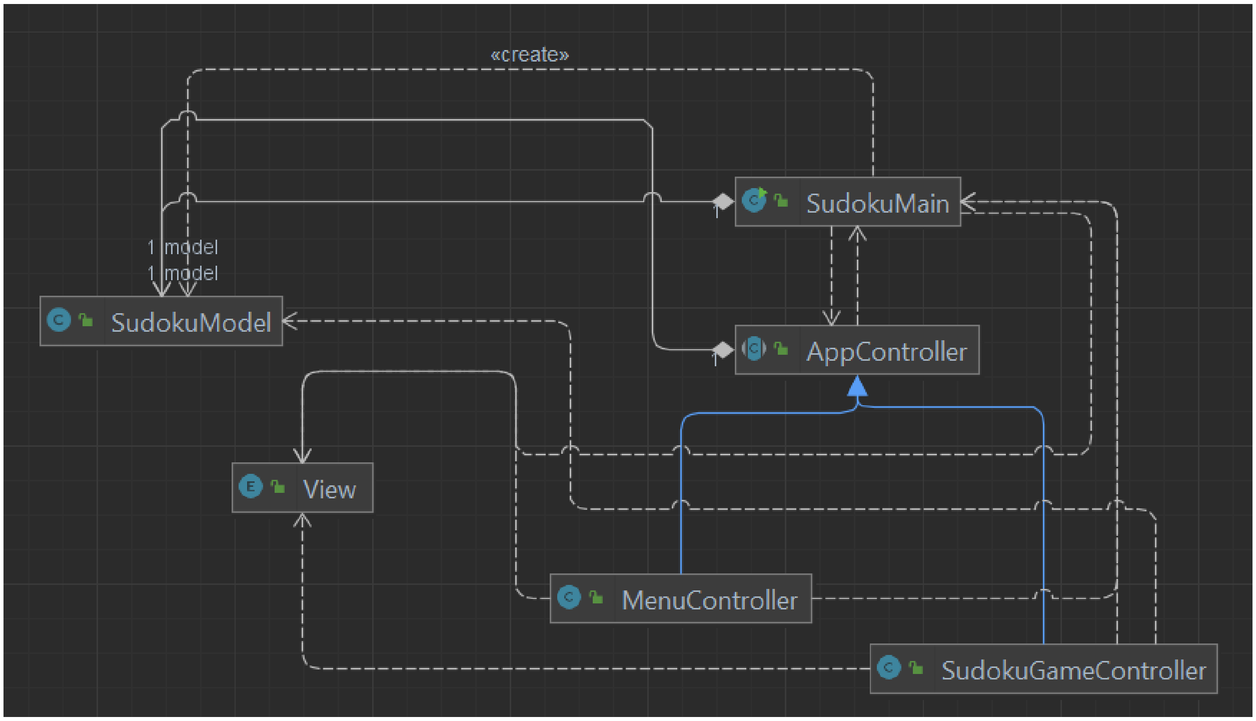
SudokuGameLogic handles user input of putting numbers, removing numbers, and undoing the previous movement. It composites SudokuChecker class that checks if the current board with the user input is valid or not. If not SudokuChecker class stores the cell that is causing the conflict. SudokuGameLogic also stores multiple SudokuMovement objects, which are snapshots of movements. These objects were used to undo a movement.

View/Controller:

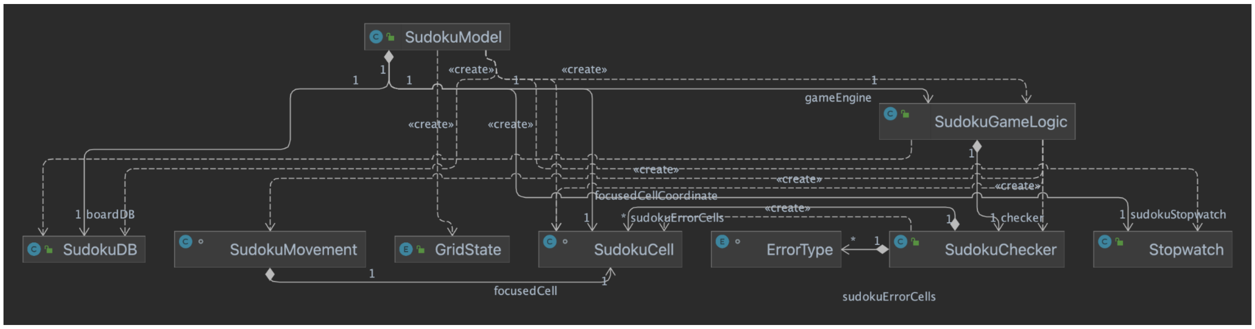
The AppController class is the abstract class of the controller, where every controller class inherits this class. It has attributes to refer to the model and the JavaFX Main class. This allows the controller class to refer to the main application and switch Scenes when some event triggers. Finally, the View of this application was made in FXML files, and an enum, View, was created to store the name of each View and the file paths of FXML files.

The concise UML diagram of the model generated by IntelliJ is shown below for your reference.

Top-Level Classes:



Model Classes:


SudokuModel
1
1
1
«create»
«create»
1
1
gameEngine
SudokuGameLogic
«create»
«create»
1
«create»
«create»
«create»
focusedCellCoordinate
«create»
1 boardDB
1
* sudokuErrorCells
1
1 checker
1 sudokuStopwatch
SudokuDB
SudokuMovement
GridState
SudokuCell
ErrorType
*
1
SudokuChecker
Stopwatch
1
focusedCell
1
sudokuErrorCells