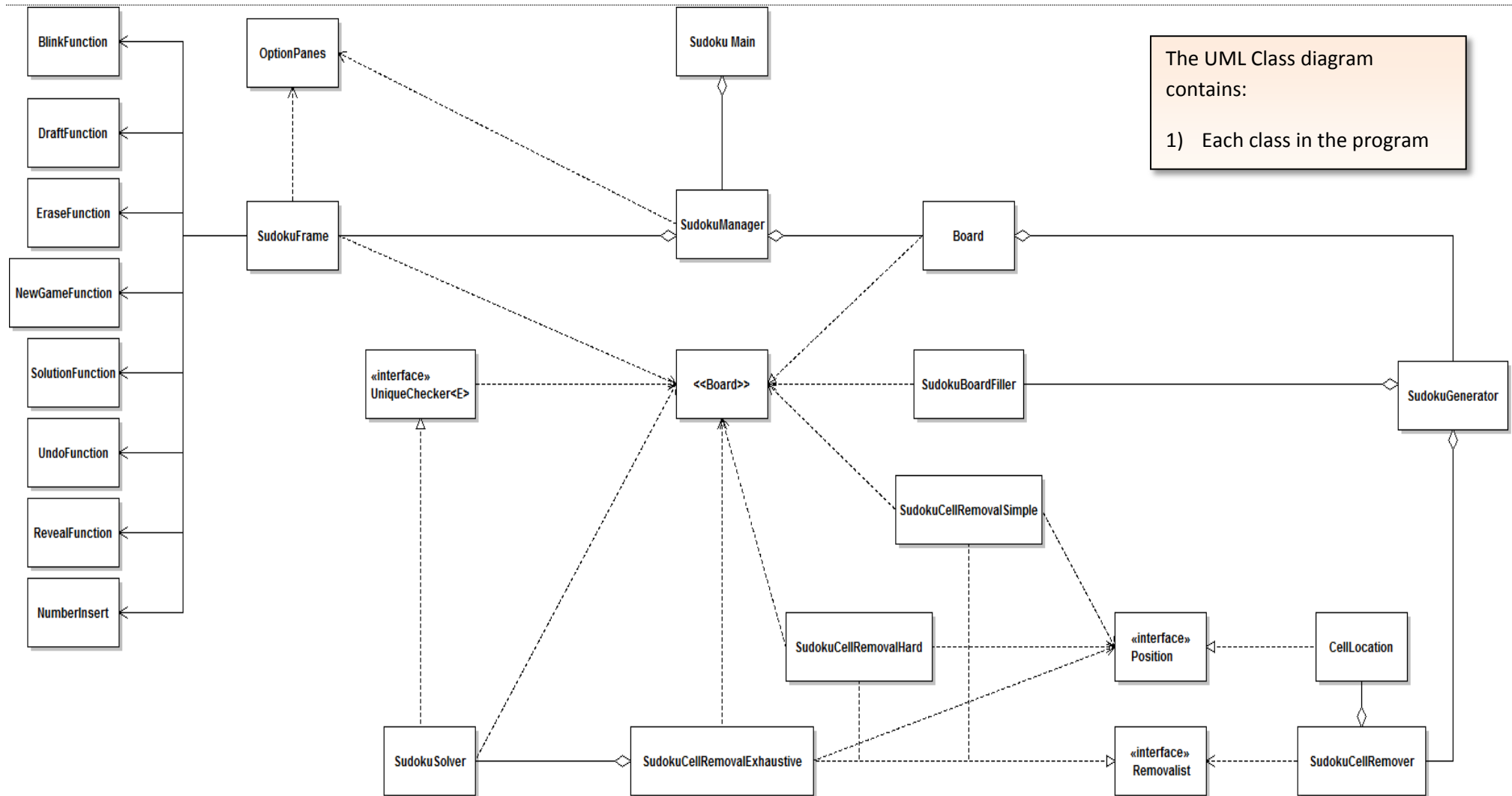## UML Class Diagram (Slim)



The UML Class diagram contains:

1) Each class in the program

# UML Class Diagram (Full)

**BlinkFunction**
c:color
b:JButton
actionPerformed(ActionEvent):void

**DraftFunction**
actionPerformed(ActionEvent):void

**EraseFunction**
actionPerformed(ActionEvent):void

**NewGameFunction**
actionPerformed(ActionEvent):void

**SolutionFunction**
actionPerformed(ActionEvent):void

**UndoFunction**
actionPerformed(ActionEvent):void

**RevealFunction**
actionPerformed(ActionEvent):void

**NumberInsert**
actionPerformed(ActionEvent):void

**OptionPanes**
start:icon
grumpy:icon
awesome:icon
puzzled:icon
geticon(String):Icon
chooseLevel:int
closedOption:int
confirmationPopUp(Icon,String,Strin):boolean
exitWinGame():void

**SudokuFrame**
buttonInputs: LinkedList<JButton>
buttonRow: LinkedList<JButton>
currentValue: String
board: Board
makeCommandButton(String,JPanel,ActionListener,String):JButton
setUpGrid():void
labelGrid():void
setUpButtonInputs():void

**Sudoku Main**
Initialises program
main

**SudokuManager**
BOARD_SIZE:int
DIFFICULTY_EASY:int
DIFFICULTY_MEDIUM:int
DIFFICULTY_HARD:int
newGame():void

**SudokuBoard**
board:ArrayList<Cell>
boardSize:int
public generate(int) : void
public removeCellValue(int, int) : void
public getCellValue(int, int) : int
public setCellValue(int, int, int) : void
public hasInput(int, int) : boolean
public isInitiallySet(int, int) : boolean
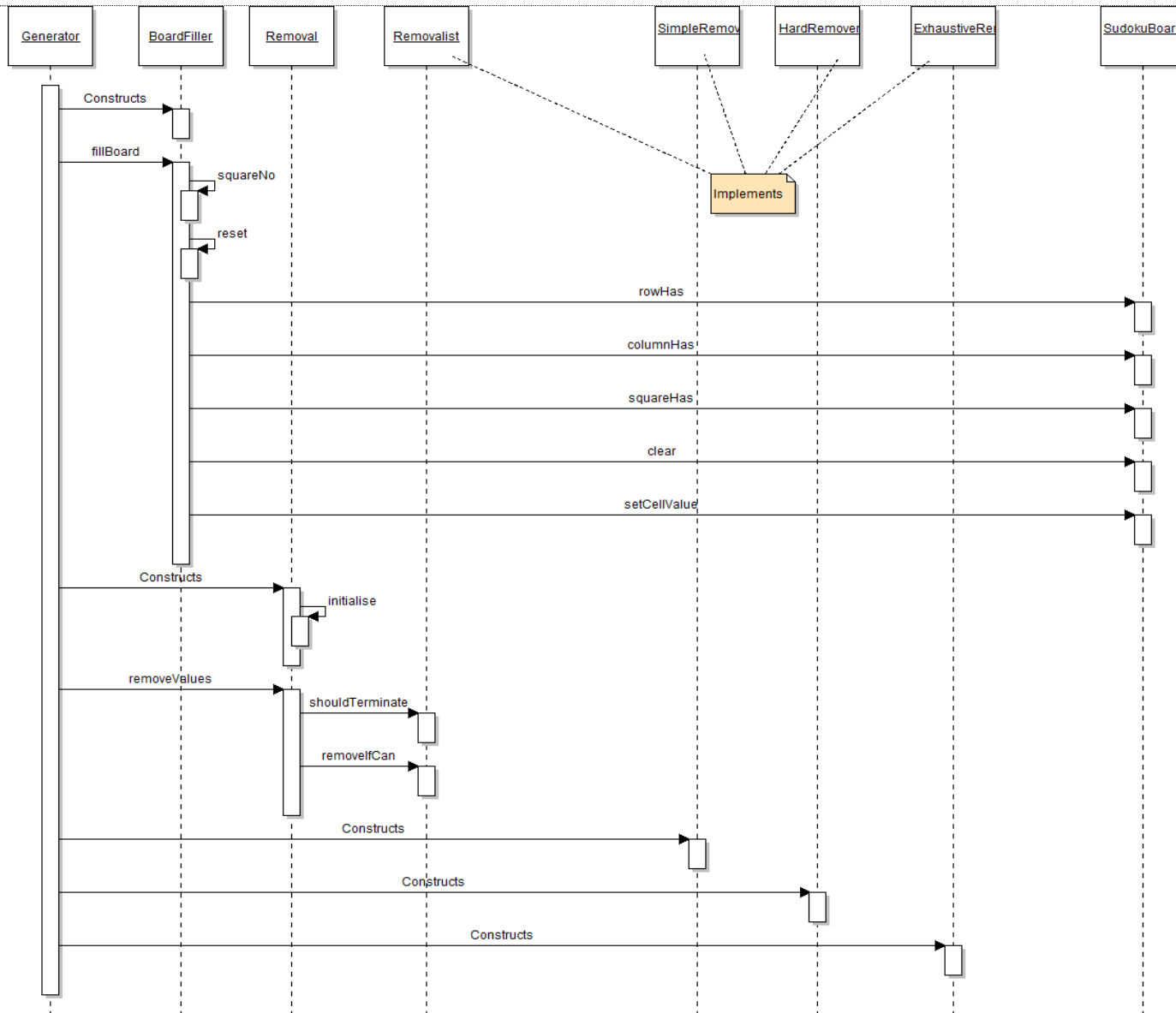public setIfInitiallySet(int, int, boolean) : void
public isEmptyCell(int, int) : boolean
public isVisibleCellDraft(int, int, int) : boolean
public setCellDraftVisibility(int, int, int, boolean) : void
public isCorrectBoard() : boolean
public isDifficultyEasy() : boolean
public isDifficultyMedium() : boolean
public isDifficultyHard() : boolean
public getBoardSize() : int
public clear() : void
public restart() : void
public rowHas(int, int) : boolean
public columnHas(int, int) : boolean
public squareHas(int, int) : boolean
public getDifficulty : int
public undoLast() : void
public isCorrectInputForCell(int, int, int) : boolean
public isFilledBoard() : boolean
private squareSectionId(int, int) : int
private addCellState(int, int) : void
private clearCell(int, int) : void
private getCell(int, int) : Cell
private getCell(double, double) : Cell

**SudokuBoardCell**
private boardSize : int
private valueFinal : int
private valueInput : int
private isInitiallyVisible : boolean
private isCurrentlyVisible : boolean
private tempNumbers : boolean[]
public getCorrectValue() : int
public getInputValue() : int
public cloneBack(Cell) : void
public cloneCell() : Cell
public setCorrectValue(int) : void
public setInputValue(int) : void
public removeCorrectValue() : void
public isCurrentlyVisible() : boolean
public isInitiallyVisible() : boolean
public isSetDraft(int) : boolean
public setDraft(int, boolean) : void
public clear() : void
public restart() : void
public isEmptyCorrect() : boolean
public isEmptyInput() : boolean
public isCorrect() : boolean
public getBoardSize() : int

**«interface» Cell**
public getCorrectValue() : int
public getInputValue() : int
public cloneBack (Cell) : void
public cloneCell() : Cell
public setCorrectValue(int) : void
public setInputValue(int) : void
public removeCorrectValue() : void
public isCurrentlyVisible() : boolean
public isInitiallyVisible() : boolean
public isSetDraft(int) : boolean
public setDraft(int, boolean) : void
public clear() : void
public restart() : void
public isEmptyCorrect() : boolean
public isEmptyInput() : boolean
public isCorrect() : boolean
public getBoardSize() : int

**<<Interface>> Board**
public generate(int) : void
public removeCellValue(int, int) : void
public getCellValue(int, int) : int
public setCellValue(int, int, int) : void
public hasInput(int, int) : boolean
public isInitiallySet(int, int) : boolean
public setIfInitiallySet(int, int, boolean) : void
public isEmptyCell(int, int) : boolean
public isVisibleCellDraft(int, int, int) : boolean
public setCellDraftVisibility(int, int, int, boolean) : void
public isCorrectBoard() : boolean
public isDifficultyEasy() : boolean
public isDifficultyMedium() : boolean
public isDifficultyHard() : boolean
public getBoardSize() : int
public clear() : void
public restart() : void
public rowHas(int, int) : boolean
public columnHas(int, int) : boolean
public squareHas(int, int) : boolean
public getDifficulty : int
public undoLast() : void
public isCorrectInputForCell(int, int, int) : boolean
public isFilledBoard() : boolean

**«interface» UniqueChecker<E>**
public isUniqueSolution(E) : boolean

**SudokuBoardFiller**
public fillBoard() : void
private reset(int, int) : int
private squareNo(int, int) : int

**SudokuGenerator**
public generateBoard(Board) : void

**SudokuSolver**
private static final POSSIBLE : int
private static final NOT_POSSIBLE : int
private static final EMPTY : int
private board : int[][]
private temp : int[][][]
public isUniqueSolution(Board)
public solved() : boolean
public solve() : boolean
private solveSimpleSteps() : boolean
private solveMediumSteps() : boolean
private pointingPairsAndTriplesAlgorithm() : boolean
private getSection(int, int) : int
private getSectionFirstCol(int) : int
private getSectionFirstRow(int) : int
private getSectionRowPossibleVals(int, int) : LinkedList<Integer>
private getSectionColPossibleVals(int, int) : LinkedList<Integer>
private getColPossibleVals(int) : LinkedList<Integer>
private getRowPossibleVals(int) : LinkedList<Integer>
private getSectionPossibleVals(int, int) : LinkedList<Integer>
private getTempFinalVal(int, int) : int
private removeTempVal(int, int, int) : boolean
private removeDuplicates(LinkedList<Integer>) : void
private numberOccurances(int, LinkedList<Integer>) : int
private setTempFinalVal(int, int, int) : void
private debug(Object) : void
private possibleVals(int, int) : LinkedList<Integer>
private valid() : boolean
private clearTemp(int, int) : void
private updateCellsFilled() : int
private updateTempVals() : void
private updateBoardVals() : boolean
private inRow(int, int) : int
private inColumn(int, int) : int
private inSection(int, int, int) : int
private canBe(int, int, int) : boolean

**SudokuCellRemoverSimple**
public removeIfCan(Position) : boolean
public shouldTerminate() : boolean
private squareNo(int, int) : int

**SudokuCellRemoverHard**
public removeIfCan(Position) : boolean
public shouldTerminate() : boolean
private findAdj(int) : int[]
private adjFull(int[], int[], int) : boolean

**«interface» Position**
getRow() : int
getCol() : int

**CellLocation**
getRow() : int
getCol() : int

**SudokuCellRemoverExhaustive**
public removeIfCan(Position) : boolean
public shouldTerminate() : boolean

**«interface» Removalist**
public removeIfCan(Position) : boolean
public shouldTerminate() : boolean

**SudokuCellRemover**
public removeValues(Removalist) : void
private initialise() : void

The UML Class diagram contains:

1) Each class in the program
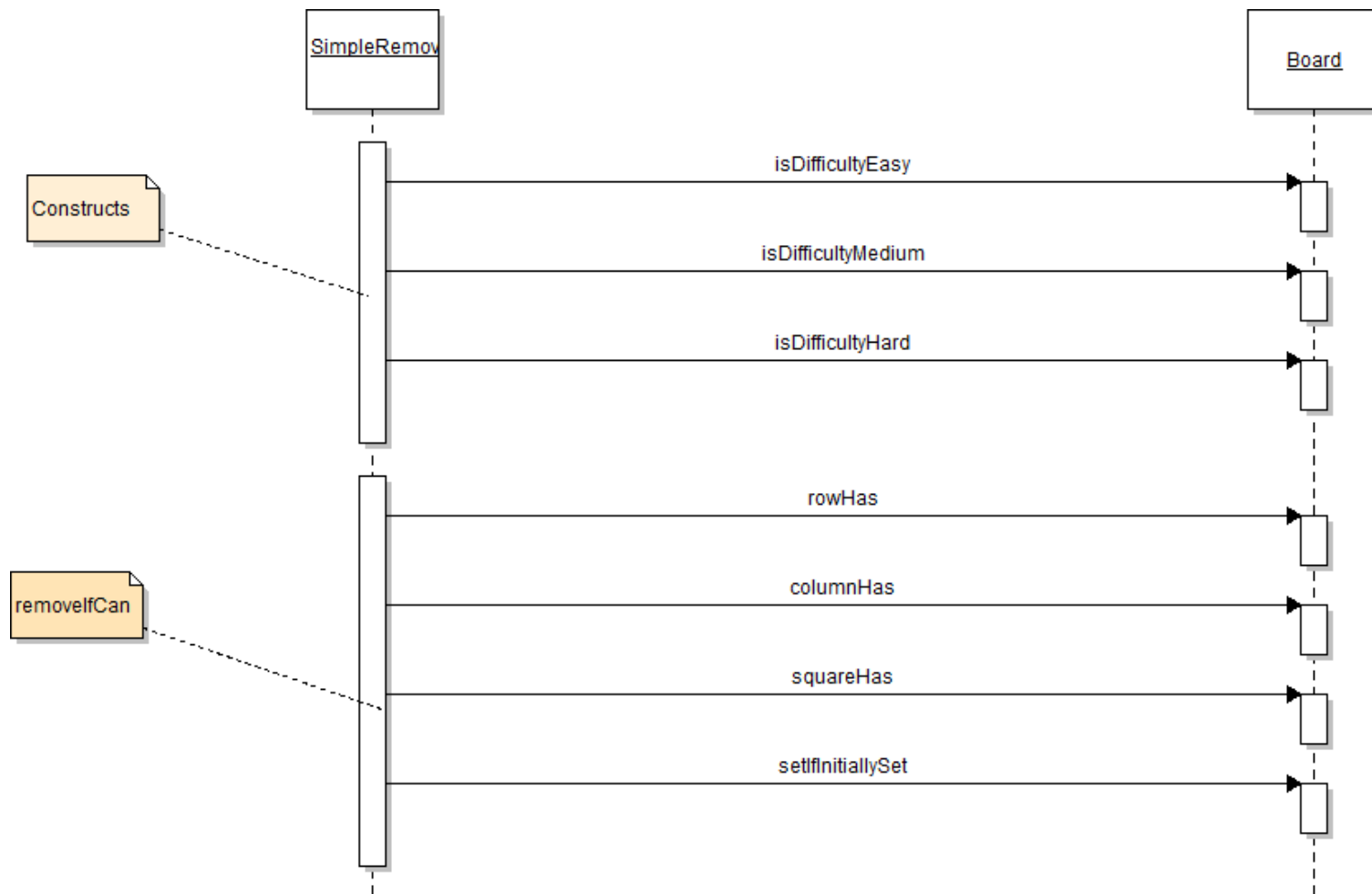2) Each class' methods / fields
3) Each class' multiplicites

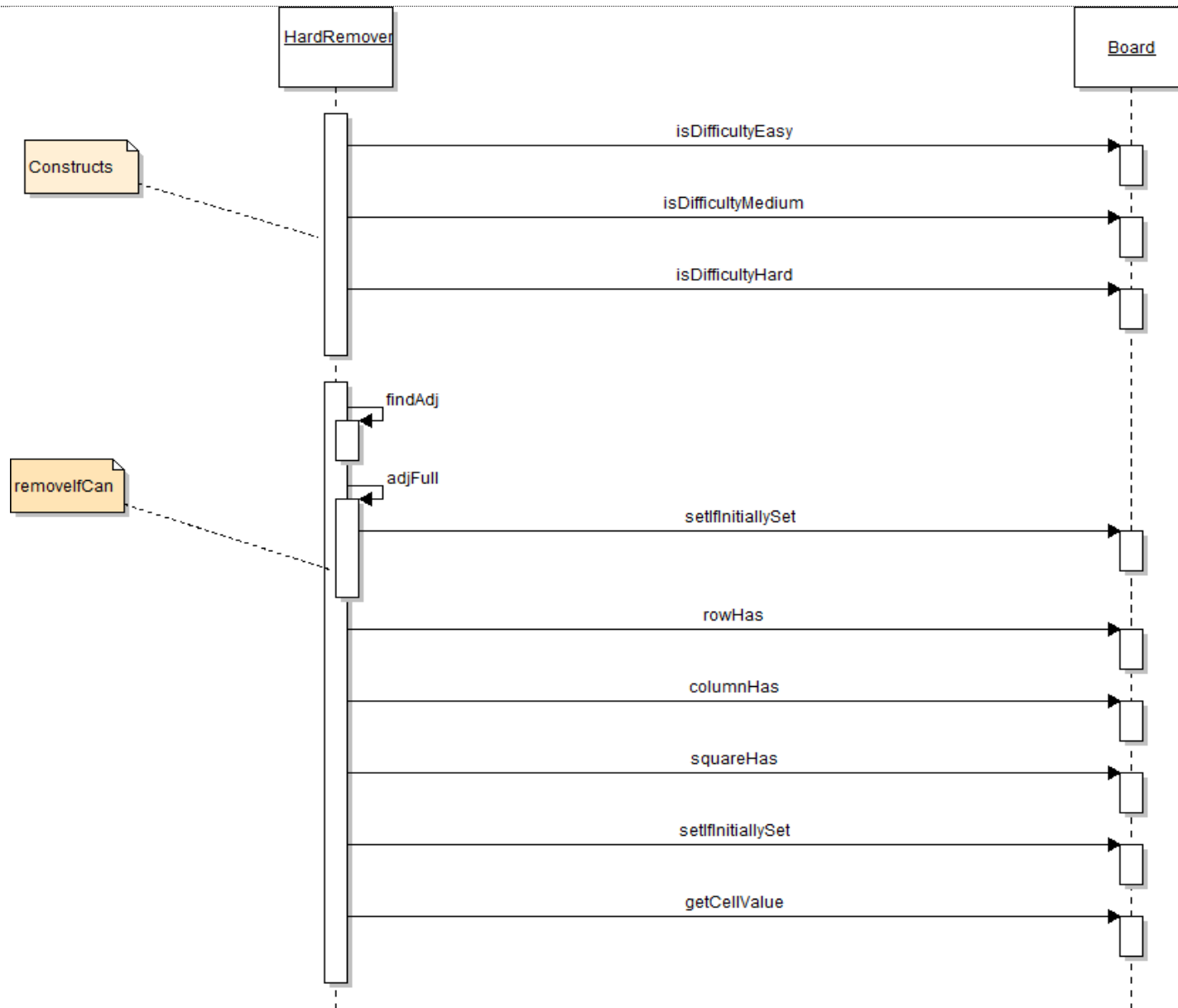## UML Sequence Diagram (Project Overview)
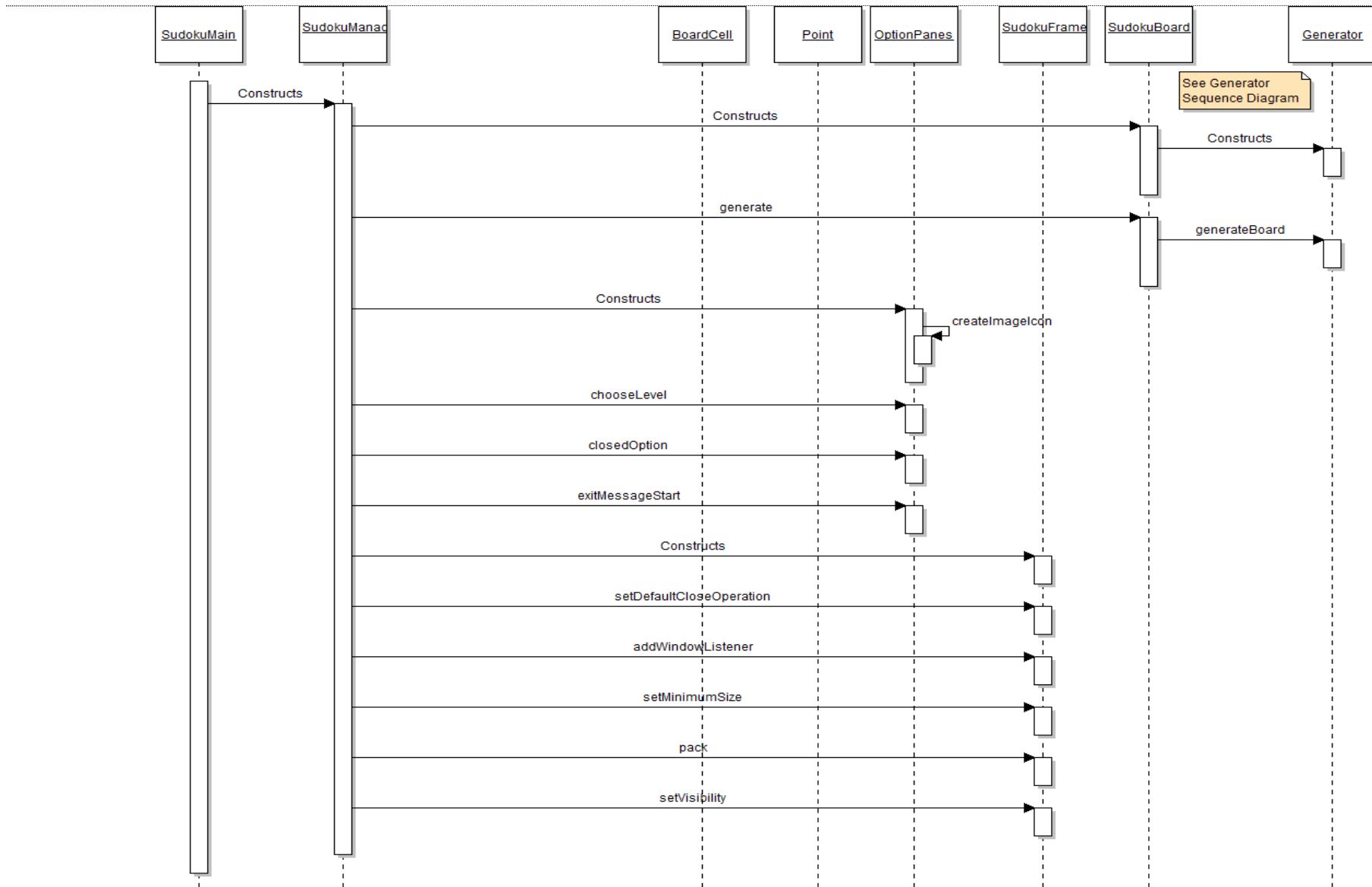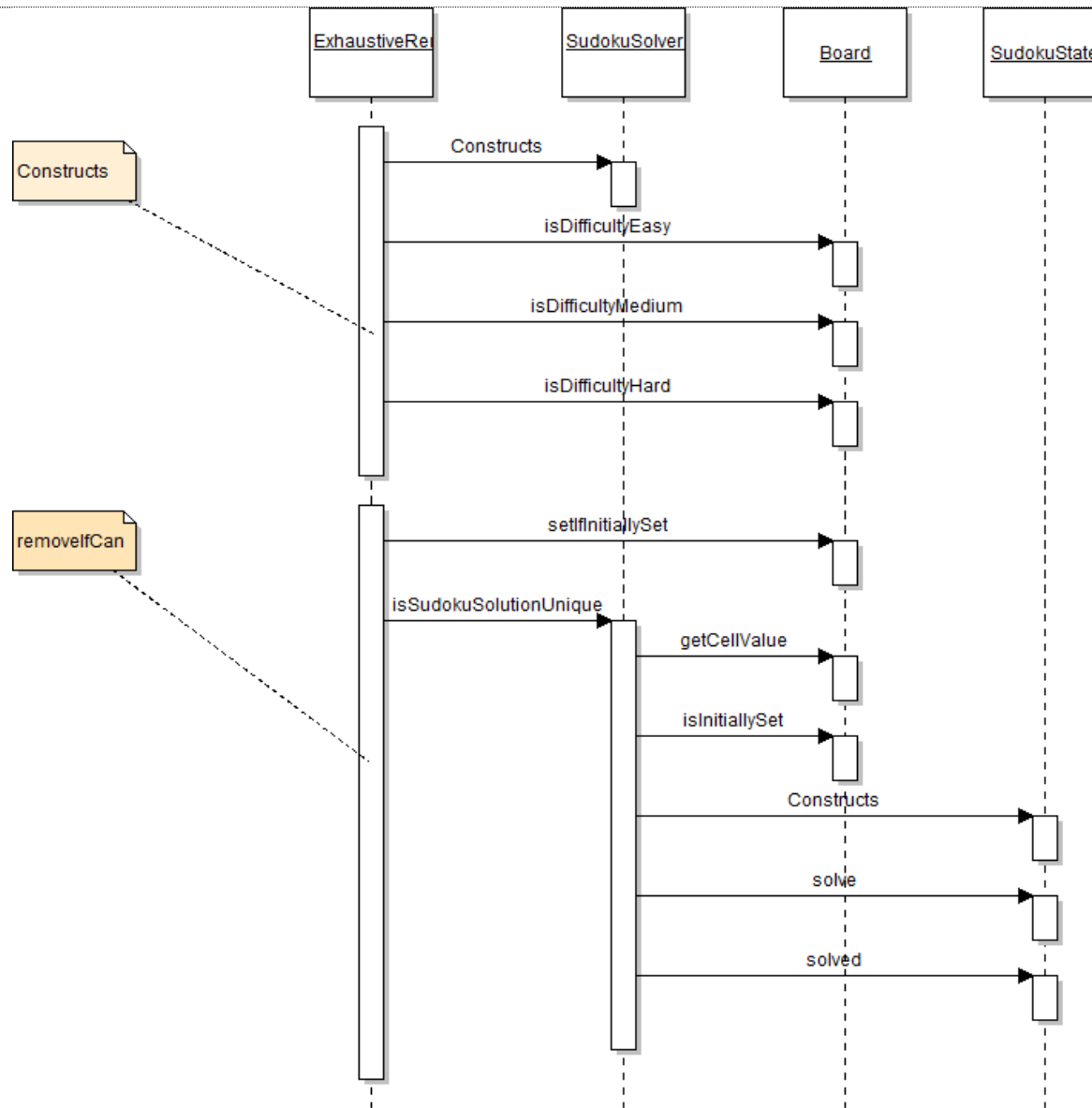
## UML Sequence Diagram (Generator)
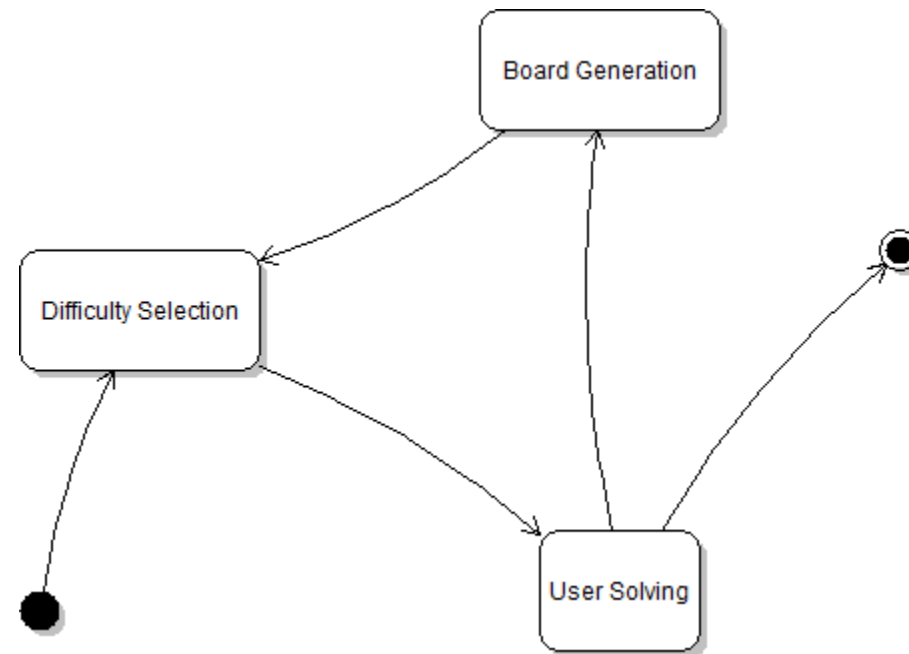
## UML Sequence Diagram (Removal - Simple)

## UML Sequence Diagram (Removal - Hard)

## UML Sequence Diagram (Removal - Exhaustive)

## UML State Diagram

Board Generation

Difficulty Selection

User Solving

The state diagram shows each general
state the program can be in. It also shows
how you can move between states.