

# Second year project I

# Solving Sudoku

Group 22

K. Shen

J. Willekes

# Table of Contents

- ▶ Introduction
- ▶ General implementation
  - Cell Class
  - Sudoku Class
    - Logic Rule
    - Generalization
    - Enumeration
  - Solver Class
- ▶ Elaboration of Enumeration
- ▶ Conclusion

# Introduction

8								
		3	6					
	7			9		2		
	5				7			
				4	5	7		
			1				3	
		1					6	8
		8	5				1	
	9					4		

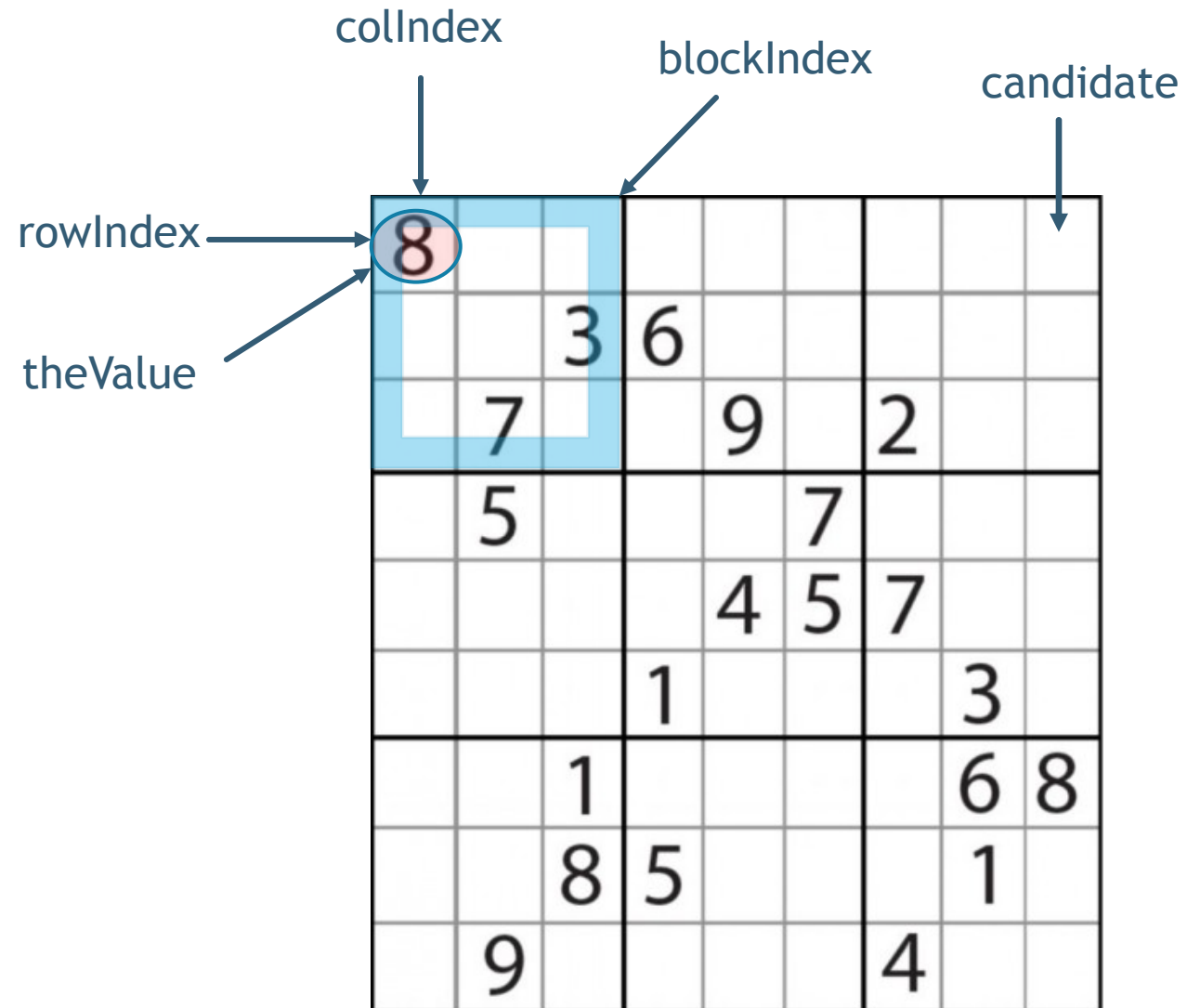
Methods:

- Logic Rule
- Generalization
- Enumeration

# General Implementation

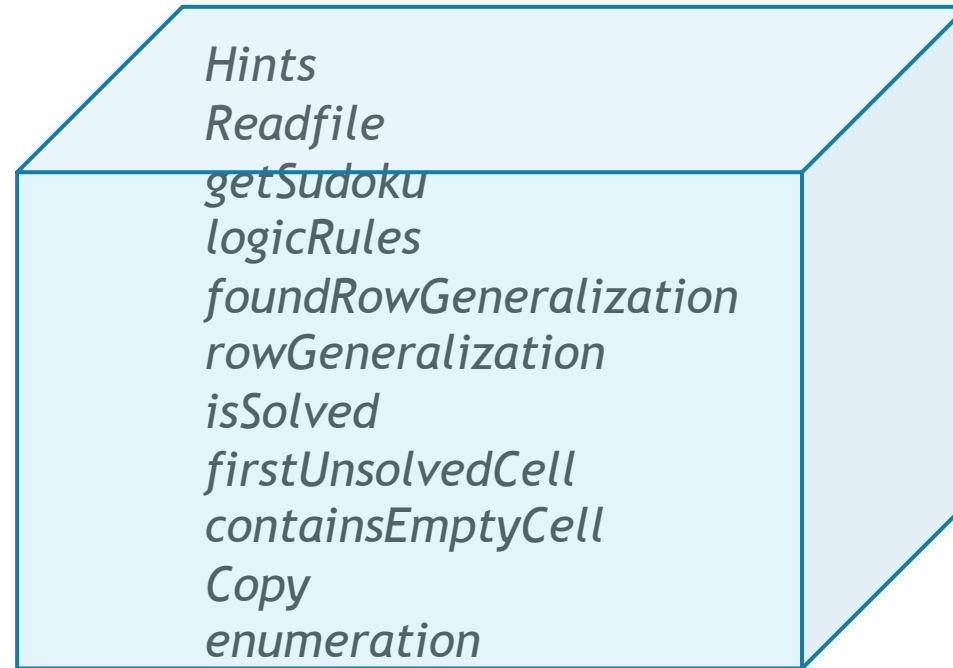
- Cell Class
- Sudoku Class
  - Logic Rule
  - Generalization
  - Enumeration
- Solver Class

# Cell Class



## Sudoku class

- Logic rule
- Generalization
- Enumeration



## Solver class

- the usage of the three methods
- Print out the results

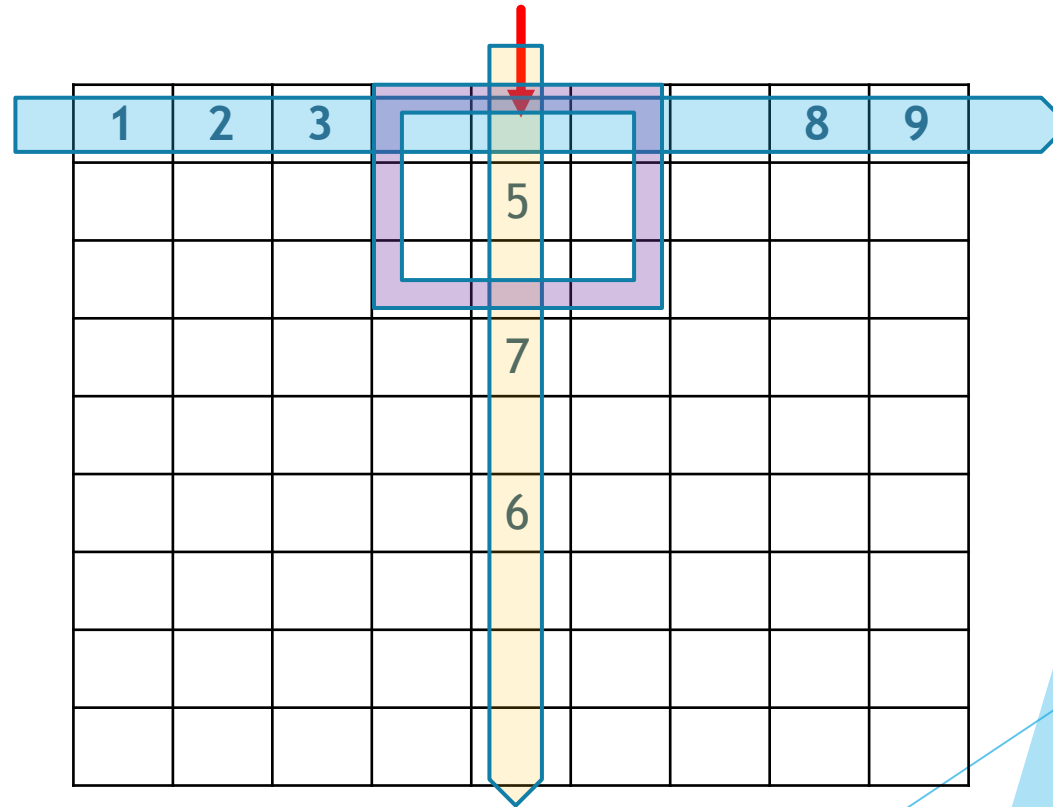
# Sudoku class

- Logic rule
- Generalization
- Enumeration

# Logic rule

- Check by row, column and block
- Eliminate duplicated candidates  
Comparing with the hints
- Find a naked single

For Cell[1][5]:  
Candidate: 1, 2, 3, 4,  
5, 6, 7, 8, 9





# Logic rule

Take row for instance

If

```
dequeued.colIndex() !== j  
!dequeued.isEmpty()  
!mySudoku[dequeued.rowIndex()][j].isEmpty()
```

```
mySudoku[dequeued.rowIndex()][j].remove(  
  dequeued.theValue());
```

If

```
mySudoku[dequeued.rowIndex()][j].theValue() == 0  
mySudoku[dequeued.rowIndex()][j].isSolved()){
```

```
mySudoku[dequeued.  
rowIndex()][j].setValue( mySudoku[dequeued.rowIndex()][j  
].getCandidates().get(0));
```

hint

Remove value of the duplicated  
candidates

Unsolved cell  
now only one candidate left

Assign value to the unsolved cell

Enqueue hints

# Generalization

- Check by row, column and block
- Find hidden single

[illegible]

# Generalization

For  
If

```
hints.isEmpty()  
int i=0; i<9; i++  
int k=1; k<10;k++  
int count = 0;  
int j =0; j<9; j++
```



If

```
mySudoku[i][j].getValue() ==0  
mySudoku[i][j].contains(k)
```



If

Count ++



Count ==1  
true

Take row for instance

Fix a row  
Go through columns



Unsolved cell  
Contains same candidates



Number of times that a  
value k occurs



Find a hidden single  
Enqueue as a new hint

# Enumeration

- Branching method
- make a guess for an unsolved cell
- And then apply logic rule and generalization again

Cell[1][5]:4,6

Cell[1][4]: 4,6

Cell[1][6]:4,6

1	2	3					8	9
				5				
				7				
			7					
			8					
					5			
			5					

# Enumeration

Check if it is solved



Look for first  
unsolved cell



Make a copy  
Assign values



Make a guess for the  
first unsolved cell

```
int i=0; i<9; i++  
int j =0; j<9; j++  
this.finalSudoku[i][j].getValue() !=0  
solvedCells++
```



```
outerLoop:  
int i=0; i<9; i++  
int j=0; j<9; j++  
finalSudoku[i][j].getValue()  
==0
```



```
copy = finalSudoku[i][j];  
break outerLoop;}
```

# Enumeration

Use copy to branch out

No solved

Logic rule  
Generalization

If

Sudoku.containsEmptyCell(copyOf  
Sudoku.mySudoku)

continue

If

foundColGeneralization(hints,  
copyOfSudoku.mySudoku)

foundRowGeneralization(hints,  
copyOfSudoku.mySudoku)

foundBoxGeneralization(hints,  
copyOfSudoku.mySudoku)

int i=0; i<numCandidates; i++  
Make a copy for each candidate

!hints.isEmpty()  
&& !copyOfSudoku.isSolved()

If (...)  
true continue;  
false

# Conclusion

- Logic rule is a basic method
- Generalization method helps to find more hidden singles
- Enumeration method makes a guess for the remaining unsolved cell, branches out and applies logic rule and generalization again
- Improvement!