OOPS PROJECT REPORT

“SUDOKU GAME IN C++”

NAME: VAISHNAVI SHARMA

ROLL NO.: 23179

BRANCH: CSE

SEMESTER: 4

SUBMITTED TO: Prof. PANKAJ KUMARI

Introduction

Sudoku (数独) is a logic-based, combinatorial number placement puzzle game. The objective of this game is to fill a $N \times N$ grid with digits so that every row, column and inner box is filled with number from 1 to N without duplicates. The shape of inner boxes may vary, but all must have exactly N blocks. In most situations, a Sudoku game should have a unique solution. However, there can be cases that a Sudoku game has multiple solutions. Usually a game with multiple solutions is considered to be more difficult than the one that has a unique solution.

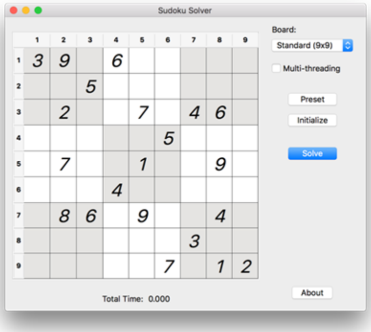
Sudoku originated in the late 16th century. La France developed the embryonic form of the modern Sudoku of 9 by 9 grid with 9 3×3 inner boxes, and later, it was believed that the modern Sudoku was mostly designed anonymously by Howard Garns in 1979. However, this game became a worldwide phenomenon not until Nikoli introduced it to Japan in 1984 and named it as 数字は独身に限る. The name was later abbreviated to 数独 (Sudoku) by Maki Kaji. Sudoku is a trademark in Japan and the puzzle is generally referred to as Number Place, or NumPla in short.

Variants

Though the 9×9 grid with 3×3 inner boxes is by far the most common Sudoku game, many other variants exist. Our program natively implemented 8 variants including the standard one. In the following variants, different inner boxes are represented by different colors. In general, 2 colors are needed to mark all inner boxes. Also the number of inner boxes is N in a $N \times N$ game for most variants, there can be exceptions.

Standard Sudoku

- Grid size: 9×9
- Number of inner boxes: 9
- Inner box shape: 3×3 Square
- Number range: 1 to 9



Board: Standard (9x9)

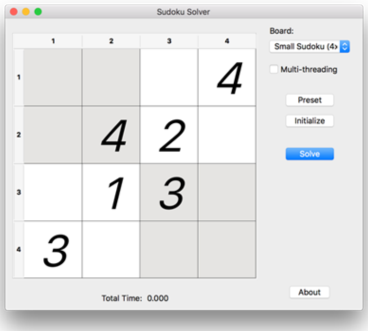
	1	2	3	4	5	6	7	8	9
1	3	9		6					
2			5						
3		2			7		4	6	
4						5			
5		7			1			9	
6				4					
7		8	6		9			4	
8							3		
9						7		1	2

Total Time: 0.000

Small Sudoku

Small Sudoku is probably the smallest and easiest Sudoku variant.

- Grid size: 4×4
- Number of inner boxes: 4
- Inner box shape: Square
- Number range: 1 to 4



Board: Small Sudoku (4>

	1	2	3	4
1				4
2		4	2	
3		1	3	
4	3			

Total Time: 0.000

Mini Sudoku

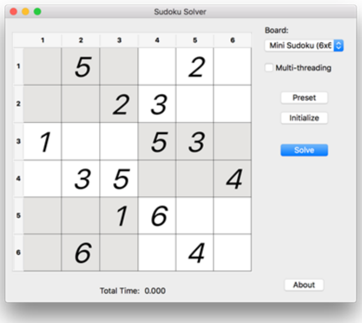
Mini Sudoku is a relatively easy variant with non- square inner boxes.

· Grid size: 6×6

· Number of inner boxes: 6

· Inner box shape: 3×2 Rectangle

· Number range: 1 to 6



	1	2	3	4	5	6
1		5			2	
2			2	3		
3	1			5	3	
4		3	5			4
5			1	6		
6		6			4	

Total Time: 0.000

Jigsaw Sudoku

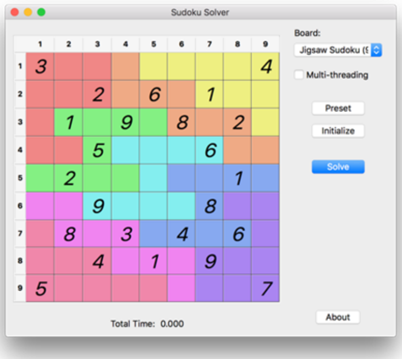
Jigsaw Sudoku (or Nonomino Sudoku) is almost the same as the standard Sudoku. The only difference to the standard Sudoku is the shape of the inner blocks. The difficulty is considered to be the same as the standard Sudoku. There are also several other variants like the Jigsaw Sudoku, where the inner boxes are not rectangles. These variants usually use multiple colors to distinguish different inner boxes.

· Grid size: 9×9

· Number of inner boxes: 9

· Inner box shape: Nonomino (polyomino of order 9)

· Number range: 1 to 9



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

Total Time: 0.000

Sudoku Solving Algorithm

Brute Force Search

The brute force search algorithm is probably the most fundamental method to solve any type of puzzle game. To solve a Sudoku game by brute force search, the following steps are performed:

1. Find the first block that is empty

2. List all values 1 to N for that block

3. Check all the other blocks in the **row** that contains this block, remove any used values

4. Check all the other blocks in the **column** that contains this block, remove any used values

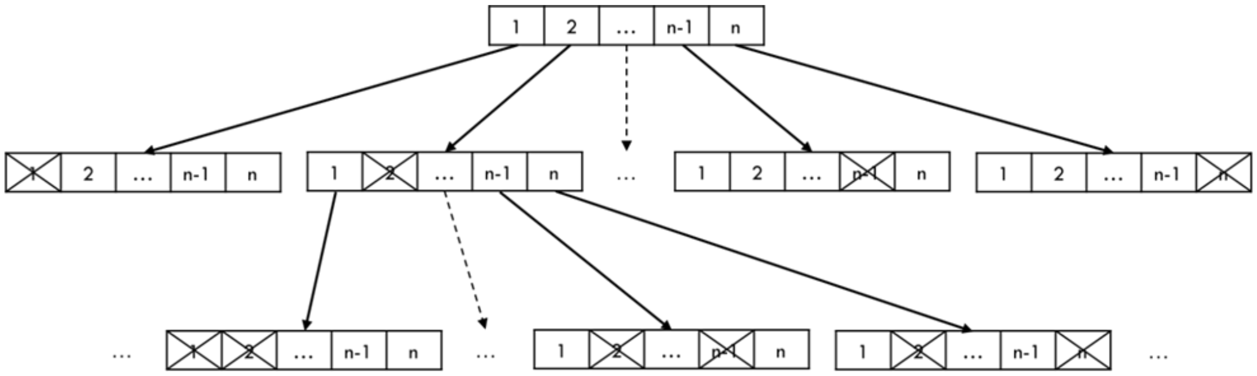
5. Check all the other blocks in the **inner boxes** that contains this block, remove any used values

6. If there is only one value remaining, assign the value to the block, then go back to step 7. If there are multiple remaining values, choose one of the values and assign to it, and keep a record of which value is chosen for the block and all the remaining values for that block, then go to step 7. If there is no value remaining, go to step 8.

7. Find the next empty block, and repeat the above steps. If no empty block can be found, then the game is solved.

8. If in any previous steps, there is one block that has been assigned a value from a list, choose the last block that satisfies the condition, assign another value from the remaining list and remove all the values assigned after the block, then repeat the above steps. If all remaining values have been tested, choose the second last block that has multiple remaining values, and test another value, until a solution is found, or no solution if all blocks having multiple remaining values are checked. Based on the explanations above, we can observe that the brute force search is to build a tree

that with maximum height of N^2, as every block needs to be checked. An abstract tree structure is shown below (each level is a block).



However, if we look the game row by row, for each row, we have N possible values for the first block, and $N - 1$ possible values for the second block, $N - 2$ possible values for the third block, and 1 value for the last block. Hence the computational complexity for filling one row is

$O(N \times (N - 1) \times \ldots \times 1) = O(N!)$. As there are N rows, the total computational complexity is

$O((N!)^2)$. In fact, this can be further reduced, but it is hard to find a short formula to represent it, so we will just keep this as the final computational complexity of brute force search.

Heuristic Search

An alternative way to solve a Sudoku game is heuristic search. In every step, we try to find a block that has the least possible values. The detailed steps are given as follows:

1. Scan all empty blocks, for every empty block, maintain a list of possible values by checking the row, column and inner boxes that contain the block.

2. Find a block that has least possible values.

3. If the block has only one possible value, assign that value to the block, and repeat from step 1. If the block has multiple possible values, assign one value from the list, and keep a record of which value has been assigned and all the other remaining values, then repeat from step 1. If the block has no possible value, go to step 4.

4. If there is one block that has multiple possible values, assign another value in the list and remove all the values after the block. If all values have been tested, check the second last block that has multiple possible values, until a solution is found, or no solution if all blocks having multiple possible values are checked.

It is easy to see the steps of heuristic search share some commons with brute force search. The key difference is that heuristic search tries to minimize the cost of testing possible values by introducing more tests on all empty blocks. The introduced cost may be ignored because it is marginal. Therefore, if every empty block has exactly 0 or 1 possible value, it is the best scenario with computational complexity $O(N^2)$ as every block needs to be checked once only. But the worst scenario where all blocks are empty will have computational complexity the same as brute force search, which is $O((N!)^2)$.

Filling Empty Grid

There is a special algorithm that can fill an empty board quickly with computational complexity $O(N^2)$, if the game satisfy the following conditions:

· The grid must be completely empty

· The width and height of the grid must be $N = n^2$

· There must be exactly N inner boxes

· Every inner box must be a $n \times n$ square

· No overlapping blocks

```
final int n = 3;
final int[][] field = new int[N][ N]; for (int i =
0; i < N; i++)

  for (int j = 0; j < N; j++)

    field[i][j] = (i* n + i/ n + j) % N + 1;
```

Solving Standard Sudoku

As the algorithm to solve all supported Sudoku variants is the same, we mainly test the time, CPU and memory usage for 200 preset values for standard Sudoku. Among the 200 presets, 100 are easy level and 100 are hard level. The presets are crawled from the Internet, but the difficulty levels are not quite uniform. We found that there could be some relatively hard presets in the easy level set, and there are also some extremely hard presets in the hard level set.

Since easy levels generally have a smaller c value, the time needed for solving an easy level standard Sudoku is far more less than solving a hard level. The CPU and memory usage are not very precise, but the CPU usage for easy levels is usually below 8%, and above 13% for hard levels. Also the maximum memory usage we recorded for easy levels is less than 100 megabytes, but for hard levels, the memory usage can easily go beyond 1.5 gigabytes, sometimes even 10 gigabytes.

Optimization

Exponential growth is a natural of Sudoku, which means there is no way that can really reduce the complexity to solve a badly designed game. However, there are still 2 methods that may be used.

Concurrently Processing

If a game has multiple solutions, usually these valid solutions and any other invalid solutions are independent of each other. Hence concurrently processing or multi-threading can be applied to speed up the searching process. Also possible values of any empty block can be checked concurrently. For example, the row, column, and all inner boxes that contain the block can be checked concurrently through multiple threads (1 thread for checking the row, 1 thread for checking the column, 1 or more threads for checking inner boxes).

There are also several disadvantages of using concurrently processing, specifically multi-threading (assuming we have enough cores for multi-threading). For a small grid size, the overhead of creating, executing, terminating threads may take longer than a single thread. Also each thread has its own memory space, the overall memory usage will be much higher than a single thread application. This is especially serious for this memory-consuming problem. Another problem could be when to create threads. Obviously it is impossible to create a thread for every iteration, if there is only one possible value for a block, multi-threading is not needed. Multi-threading is only needed when a block has multiple possible values. But such blocks may be found at any time in the process.

Stochastic Value Selection

Instead of testing possible values in order, a value can be randomly chosen. Theoretically this is faster and resource efficient. But still, it can also reach the worst scenario, which is the same as testing possible values in order. Also the implementation of this method will be more complicated.

Features

1. Include 8 Sudoku variants to play
2. Heuristic search method to solve the game
3. Real-time validation is supported if the game is played by human
4. The time to solve the game automatically is provided
5. Easy to load presets
6. Easy to create and edit preset, and save it to file for reuse
7. Flexible to add more Sudoku variants

Conclusion

In this project, we implemented a playable application to solve a variety of Sudoku games by using heuristic search and multi-threading. The heuristic search is more efficient than brute force search in most cases (though the worst can be the same). Our application is compatible with most Sudoku variants and is flexible to add more variants. A solution is guaranteed (or no solution will be given), though sometimes it is restricted by the hardware power. The application does not only support solving a Sudoku game, users can also play the game by themselves.

We have studied several facts that may affect the heuristic search, and proposed a stochastic heuristic search model as a future work.

References

[1] https://en.wikipedia.org/wiki/Sudoku

[2] https://en.wikipedia.org/wiki/Brute-force_search

[3] https://en.wikipedia.org/wiki/Heuristic_(computer_science)

[4] https://en.wikipedia.org/wiki/Sudoku_solving_algorithms

[5] http://doc.qt.io/qt-5/qcolor.html - predefined-colors.

[6] http://www.lamsade.dauphine.fr/~cazenave/papers/sudoku.pdf