Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Querétaro

Using the nature of prolog to our advantage.

María de los Ángeles Contreras Anaya

A01700284

TC2006. Programming Languages

Benjamín Valdés Aguirre PhD

May 25, 2021.

Abstract

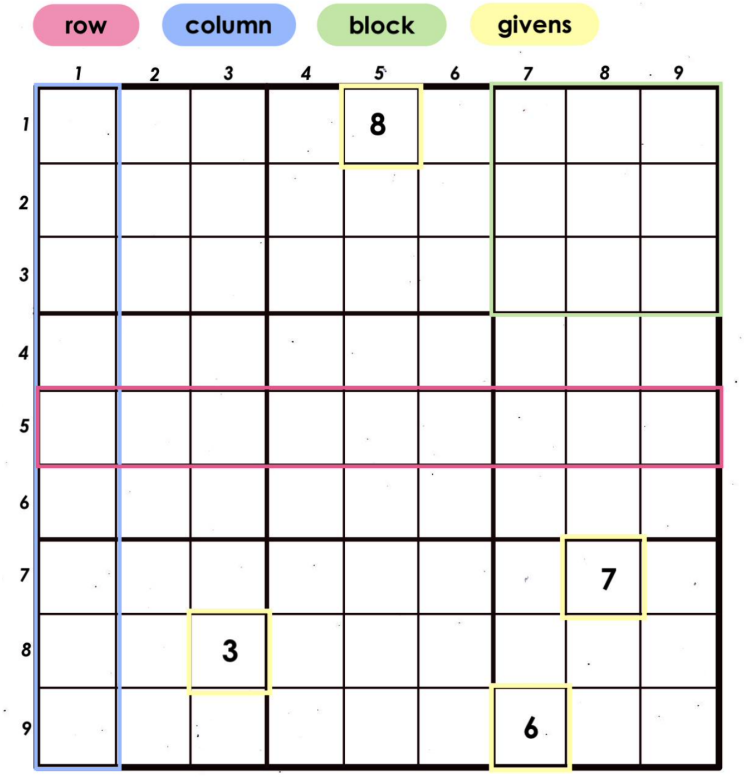
The present project makes usage of the constraint logic programming paradigm to be able to design a Sudoku solver with symbolic AI. The solution is implemented using Prolog for the logic of the program and java for the GUI, consequently, I used the JPL library to connect both of the aforementioned. The solver consists of a combination of constraint propagation and labeling (search) for the solution to be truly effective. This program was tested using easy, medium and the hardest Sudoku puzzle ever according to The Telegraph to prove its efficiency. The java GUI was created to allow users to enter their inputs(hints) and observe the solution given by the Prolog program easily, hence making the program more user-friendly.

Sudoku puzzles: Understanding

Sudoku is a puzzle game invented by Howard Garns in 1979 as "Numbers in place" but renamed after the 1984 Japanese publication by Maki Kaji as "Sudoku". The puzzle is formed by Latin squares which are defined as a grid of order n filled with n symbols so that each symbol appears only once in each row and each column, for this project we will be implementing the standard version of Sudoku which will be explained below.

This version of the Sudoku puzzle consists of a 9 by 9 Latin square grid subdivided by 3x3 blocks. Each cell inside the grid has 9 different symbols, in this case, the integers from 1 to 9, and all elements of the board should fulfill the *One Rule*:

Every row, column, and block must contain all integers.



Givens. The numbers that are provided within the puzzle while assuring the one rule is fulfilled.

A Sudoku puzzle aims to fulfill the whole grid with the help of the *givens* while assuring the one rule is satisfied. To solve a Sudoku puzzle by hand there is not a proper or official technique, each person can solve it in a completely different way but any technique used will always be limited by the human brain.

According to Miller's law, the average human brain can only hold 7+/-2 things in the immediate memory, that is why solving a Sudoku puzzle by hand will always be challenging because the human memory is not capable of holding all the possible numbers of each of the 81 cells of the grid of a Sudoku. In consequence, in order for the average human to get to a solution, he/she has to take notes of these possible answers to be able to backtrack when the one rule is broken, hence the process of getting a solution is slowed down.

Solution

I decided to implement a Sudoku solver using Prolog because the nature of the language makes it easy to solve combinatorial tasks like Sudoku puzzles, a task of this type involves finding a grouping, ordering, or assignments of a discrete, finite set of objects that satisfy all given conditions. In Prolog, these given conditions can be represented easily as facts, and they can be evaluated through the definition of a rule. Moreover, the usage of variables, unification, and recursion also helps in the implementation of the algorithm.

Furthermore, in every combinatorial task there exist "candidate solutions", this type of solution is defined as solutions that apparently fit the given conditions but miss fulfilling one or more of them. In a Sudoku puzzle, one may encounter many of these solutions because there are cases where a cell can contain one or more different numbers and in this case, the only way to make progress is to choose randomly one and keep going until the grid is filled, or the given conditions are broken, when encountering the latter cases backtracking comes into hand.

Concepts worth knowing

Prolog

Non-procedural (declarative) programming language conceived by Alain Colmerauer in 1973 and further developed by Robert Kowalski. The language makes use of the resolution theorem-proving method of automated deduction developed in 1963 by J. A. Robinson.

Prolog is popular in AI, natural language processing and is the most widely used language for the logic programming paradigm because it allows the programmer to engage in controlled deduction, meaning that the solutions obtained rely on run-time engine, hence programmers should describe the space to be explored by the engine rather than the processes to calculate results directly.

Facts

As said earlier, the conditions that a Sudoku puzzle has to comply with can be represented as facts. A fact is a predicate expression that makes a declarative statement about the domain of a problem, an example in this project is shown below:

```
length(Rows, 9).
```

This predicate declares that every row of the Sudoku puzzle must be of length 9, where *Rows,* is a variable that is matched to item 9 through unification.

Rules

Rules are predicates that use logical implication to describe the relationship among facts, for this particular program the Sudoku solver takes the form of a rule and the facts that it describes are the constraints that make a grid of numbers a Sudoku puzzle. For example, the rule of the implemented program takes this form:

```
sudokuSolver(Sudoku) :-
  length(Rows, 9),
  (...)
  maplist(labeling([ff], Rows).
```

Backtracking

The process that takes place when one or more of the given conditions are broken, backtracking is to retrace all steps until the point where all conditions were satisfied.

Declarative language

A non-procedural language is one where the programmer defines what the program should accomplish rather than the steps to accomplish that goal, thus the compiler is the one in charge of figuring out how to accomplish the goal.

Implementation

For the implementation of the Sudoku solver, I used constraint logic programming, which is a form of logic programming that includes concepts of constraint satisfaction. The algorithm combines constraint propagation and labeling in order for it to be truly effective, and its effectiveness can be proven by solving the hardest Sudoku puzzle according to The Telegraph newspaper.

Constraint propagation

A technique used over problems of a finite domain that consists of a reduction of the domain by determining elements that are inconsistent with the defined constraints until there can not be more domain reductions either because there are no more variables to be reduced or because the domain becomes empty, and a failure occurs because the problem does not have any solution.

Labeling

Used over finite domains to check satisfiability or partial satisfiability of constraints. It consists of a search over the variables of a domain with the intention of finding out an assignment that satisfies all constraints. This search is typically done with backtracking as the program encounters inconsistencies.

In order to do this type of programming, I utilized a library by Marcus Triska called CLP(FD) which includes very helpful constraints for the Sudoku solver. The constraints and predicates that I used from this library are listed below:

- ins/2 (+Vars ins +Domain): let us explicitly state domains of CLP(FD) variables.
- labeling/2 (labeling(+Options, +Vars): systematically trying out values for the finite domain variables Vars until all of them are ground

In this case, I decided to use First Fail as the strategy to specify which variables should be labeled next, this strategy labels the leftmost variable with the smallest domain next, in order to detect infeasibility early.

- all_distinct/1 (all_distinct(+Vars)): returns true if variables are pairwise distinct, this predicate makes usage of both weak and intelligent propagation.

In order to be able to connect the prolog part of the code with the java GUI I made use of an API called JPL which provides a bidirectional interface between prolog and a Java Virtual Machine. The classes and methods I used are listed below:

- Query class: A Query instance is created by an application in order to query the Prolog database (or to invoke a built-in predicate).
- Atom class: Atom is a specialized Term, representing a Prolog atom with the same name constructed with a String and used as an argument of Compound Terms.

- Term class: Term is the abstract base class for Compound, Atom, Variable, Integer, and Float, which comprise a Java-oriented concrete syntax for Prolog and corresponds to the goal of a Query.
- Variable class: This class supports Java representations of Prolog variables.
- hasSolution(): This method will attempt to call this query's goal within an available Prolog engine.
- oneSolution(): Returns the first solution, if any, as a (possibly empty) map of variable name-to-term bindings, else null.

Finally, some predicates that I used that are part of prolog language to be able to make the query to the Sudoku solver of prolog, were:

- assert(+Term): predicate to place the fact in the database
- :- dynamic puzzle/2: predicate to inform the interpreter that the definition of the predicate(s) may change during execution.

Java GUI Integration

I decided to use Java to create an interface for the user, so anyone can use the solver to find out the solution to any Sudoku puzzle. When the program is launched the application displays a menu to the user *(see illustration 1)*, when the user clicks on start, a Sudoku grid is shown, and it allows the user to enter the givens of the puzzle *(see illustration 2)*.

The user should follow the following steps when launching the Java application:

1. Press the *START* button

2. Fill in the Sudoku grid with the givens.

3. Press the *Solve* button

4.1 If the entered givens are correct, the Sudoku solver displays the solution in the same grid and a successful operation message.

4.1.1 The user may close the application or press the *Reset* button to clear the board and repeat from step 2 and forward.

4.2 If the entered givens are incorrect, a message is displayed in the same window.

4.2.1 The user may close the application or press the *Reset* button to clear the board and repeat from step 2 and forward.

Illustration 1

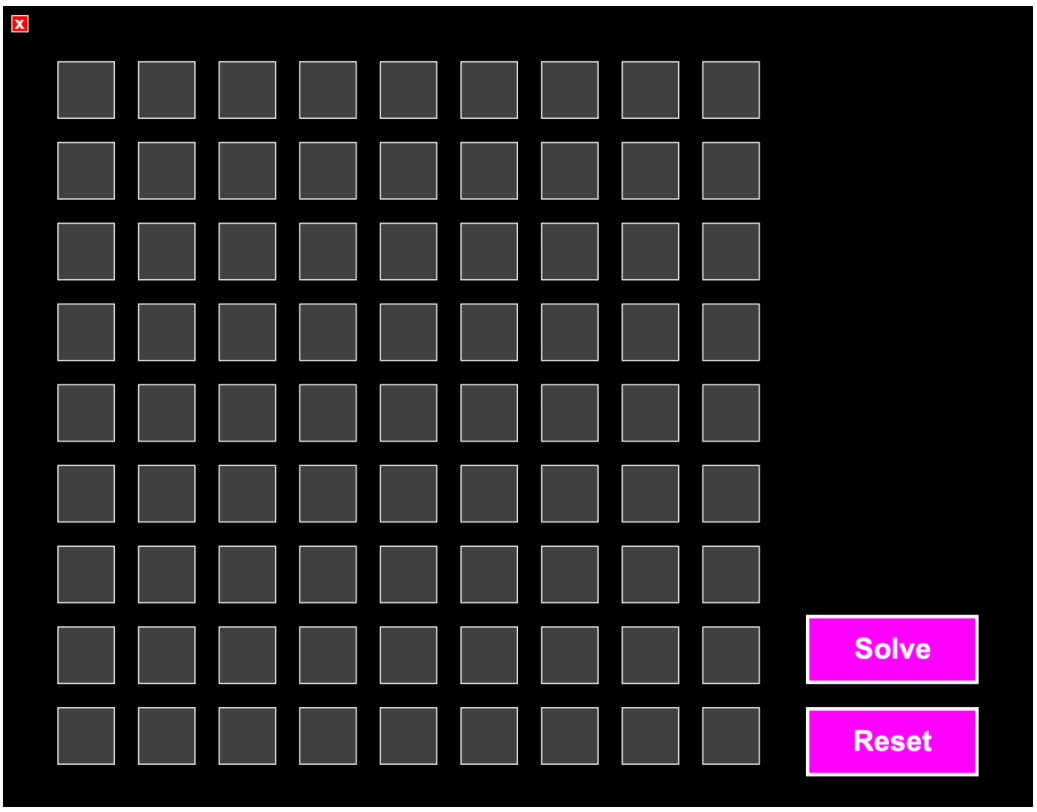


Illustration 2

Additionally, I make a brief explanation of my solver method, because I believe it is crucial to explain how I did the connection between java and prolog and how I handled the user inputs to be able to pass them to prolog and then receive them and display them to the user.

1. The user enters the givens(hints) in the GUI and I store them in a matrix of text fields **num**

```
static JTextField[][] num;
num = new JTextField[9][9];
for(int i=0;i<9;i++){
  for(int j=0;j<9;j++){
        num[i][j]=new JTextField();
        num[i][j].addKeyListener(new KeyAdapter() {
          public void keyTyped(KeyEvent e) {
            char caracter = e.getKeyChar();
            if (((caracter < '1') || (caracter > '9')) && (caracter != '\b')) {
              e.consume();
            }
          }
        });
        add(num[i][j]);
      }
}
```

2. I compile the prolog file and verify that the compilation was successful

```
Query q1 = new Query(
  "consult",
  new Term[] {new Atom("src\\sudoku.pl")});

q1.hasSolution();
```

3. I get the givens(hints) out of the matrix of text fields **num** and store them with the right format in an array of strings **arr**.

```
String[] arr = {"","","","","","","","","",""};

for(int i=0; i<9; i++){
  for(int j=0; j<9; j++){
    if(num[i][j].getText().equals("")) {
      arr[i] += "_";
    } else {
      arr[i] += num[i][j].getText();
    }
    if(j != 8) {
      arr[i] += ",";
    }
  }
}
```

When a hint is introduced it should get the integer and when the text field is empty it should store an underscore. In between every element of each row, there should be a comma, except for the last element of the respective row.

```
Array of strings that holds the entered givens
1,_,_,_,_,_,_,_,_
_,_,_,_,_,_,_,_,_
_,_,_,_,_,_,_,_,_
_,_,_,_,_,_,_,_,_
_,_,_,_,_,_,_,_,_
_,_,_,_,_,_,_,_,_
_,_,_,_,_,_,_,_,_
_,_,_,_,_,_,_,_,_
_,_,_,_,_,_,_,_,_
```

4. Then I store each string of the array inside a 2D array that is used in the prolog query.

```
String rows[][] = {{arr[0]}, {arr[1]}, {arr[2]}, {arr[3]}, {arr[4]}, {arr[5]},
{arr[6]}, {arr[7]}, {arr[8]}};
```

```
2D array that is used in prolog query
[[1,_,_,_,_,_,_,_,_], [_,_,_,_,_,_,_,_,_], [_,_,_,_,_,_,_,_,_], [_,_,_,_,_,_,_,_,_], [_,_,_,_,_
```

5. I build and place the dynamic fact inside the prolog file and verify if the operation was successful.

```
String puzzle = "puzzle(" + puzzleNum + ", " + Arrays.deepToString(rows) + ")";
Query q2 = new Query("assert(" + puzzle + ")");
q2.hasSolution();
```

6. I make the query to prolog's Sudoku solver and store the result as a String **result**

```
Query  q3 = new Query("puzzle(" + puzzleNum + ",Rows), sudoku(Rows).");
result = q3.oneSolution().get("Rows").toString();
```

```
The result returned by prolog as a String
[[1, 2, 3, 4, 5, 6, 7, 8, 9], [4, 5, 6, 7, 8, 9, 1, 2, 3], [7, 8, 9, 1, 2, 3, 4, 5, 6], [2, 3, 1, 6, 7,
```

7. I read the string returned by the prolog query character by character, and then I store them in a matrix of integers **sol**

```
sol = new int[9][9];

for(int i=0; i<result.length(); i++) {
  if(aux == 9) {
    aux = 0;
    j++;
  }
  if(j == 9) {
    j = 0;
  }
  a = Character.getNumericValue(result.charAt(i));

  if(a >= 1 && a <= 9) {
    sol[j][aux] = a;
    aux++;
  }
}
```

```
Get characters from the solution(String) and parse them to integer
 1 2 3 4 5 6 7 8 9
 4 5 6 7 8 9 1 2 3
 7 8 9 1 2 3 4 5 6
 2 3 1 6 7 4 8 9 5
 8 7 5 9 1 2 3 6 4
 6 9 4 5 3 8 2 1 7
 3 1 7 2 6 5 9 4 8
 5 4 2 8 9 7 6 3 1
 9 6 8 3 4 1 5 7 2
```

8. Finally, I get the integers of the matrix **sol** parse them as a String, and then place them in the matrix of text fields **num** in order to display the solution to the user.

```
for(int i=0; i < sol.length; i++){
  for(int k=0; k < sol[i].length; k++){
    num[i][k].setText(String.valueOf(sol[i][k]));
  }
}
```

Tests

Furthermore, I have placed some examples of tests that I did in order to prove the effectiveness of my Sudoku solver.

Test 1. Easy puzzle from Sudoku.com

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

Exhibit A. Using only labeling

```
time((puzzle(1,Sudoku), search(Sudoku), maplist(portray_clause, Sudoku))).
543,801,085 inferences, 80.603 CPU in 80.603 seconds (100% CPU, 6746689 Lips)
** Execution aborted **
```

Exhibit B. Using only constraint propagation

```
time((puzzle(1,Sudoku), constraintP(Sudoku), maplist(portray_clause, Sudoku))).
[2, 5, 4, 6, 7, 3, 8, 9, 1].
[1, 3, 9, 5, 4, 8, 2, 6, 7].
[8, 6, 7, 1, 9, 2, 3, 4, 5].
[6, 4, 5, 9, 3, 7, 1, 2, 8].
[3, 8, 1, 2, 6, 5, 9, 7, 4].
[7, 9, 2, 8, 1, 4, 5, 3, 6].
[4, 7, 8, 3, 5, 9, 6, 1, 2].
[9, 2, 6, 7, 8, 1, 4, 5, 3].
[5, 1, 3, 4, 2, 6, 7, 8, 9].
256,294 inferences, 0.057 CPU in 0.057 seconds (100% CPU, 4506645 Lips)
Sudoku = [[2, 5, 4, 6, 7, 3, 8, 9, 1], [1, 3, 9, 5, 4, 8, 2, 6, 7], [8, 6, 7, 1, 9, 2, 3, 4, 5], [6, 4, 5, 9, 3, 7, 1, 2, 8], [3, 8, 1, 2, 6, 5, 9, 7, 4], [7, 9, 2, 8, 1, 4, 5, 3, 6], [4, 7, 8, 3, 5, 9, 6, 1, 2], [9, 2, 6, 7, 8, 1, 4, 5, 3], [5, 1, 3, 4, 2, 6, 7, 8, 9]]
```

Exhibit C. Using constraint propagation and labeling (the combined solution)

```
time((puzzle(1,Sudoku), sudokuSolver(Sudoku), maplist(portray_clause, Sudoku))).
[2, 5, 4, 6, 7, 3, 8, 9, 1].
[1, 3, 9, 5, 4, 8, 2, 6, 7].
[8, 6, 7, 1, 9, 2, 3, 4, 5].
[6, 4, 5, 9, 3, 7, 1, 2, 8].
[3, 8, 1, 2, 6, 5, 9, 7, 4].
[7, 9, 2, 8, 1, 4, 5, 3, 6].
[4, 7, 8, 3, 5, 9, 6, 1, 2].
[9, 2, 6, 7, 8, 1, 4, 5, 3].
[5, 1, 3, 4, 2, 6, 7, 8, 9].
257,096 inferences, 0.057 CPU in 0.057 seconds (100% CPU, 4505967 Lips)
Sudoku = [[2, 5, 4, 6, 7, 3, 8, 9, 1], [1, 3, 9, 5, 4, 8, 2, 6, 7], [8, 6, 7, 1, 9, 2, 3, 4, 5], [6, 4, 5, 9, 3, 7, 1, 2, 8], [3, 8, 1, 2, 6, 5, 9, 7, 4], [7, 9, 2, 8, 1, 4, 5, 3, 6], [4, 7, 8, 3, 5, 9, 6, 1, 2], [9, 2, 6, 7, 8, 1, 4, 5, 3], [5, 1, 3, 4, 2, 6, 7, 8, 9]]
```

With an easy ranked Sudoku puzzle both the constraint propagation method Exhibit B. and the full method Exhibit C. took the same time (0.057 seconds) to fully complete the puzzle given but the pure labeling method one was not even able to partially solve it. That is why the usage of pure labeling was out of the question for this particular task.

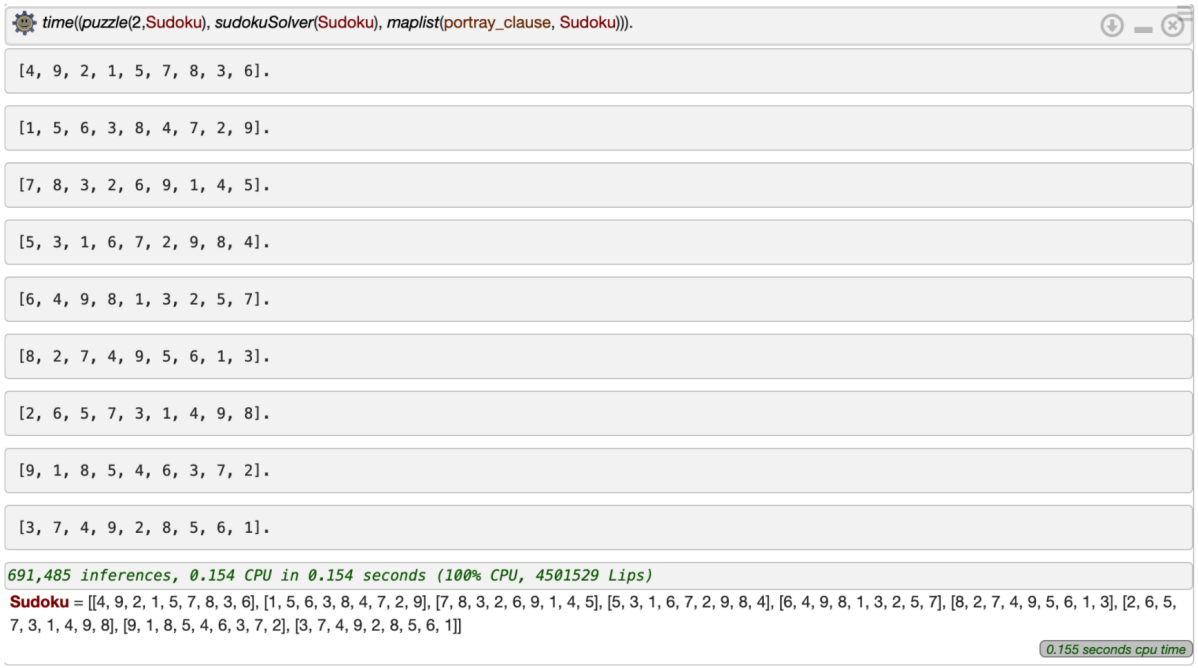
Test 2. Medium difficulty puzzle from Sudoku.com

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

Exhibit A. Using only constraint propagation

```
time((puzzle(2,Sudoku), constraintP(Sudoku), maplist(portray_clause, Sudoku))).
[4, 9, _, 1, _, 7, 8, 3, _].
[_, _, _, 3, _, _, 7, _, _].
[7, _, 3, 2, _, _, 1, _, _].
[5, 3, 1, 6, 7, 2, 9, 8, 4].
[_, 4, _, 8, _, _, 2, _, _].
[8, 2, 7, _, _, _, 6, 1, 3].
[_, 6, _, _, _, _, _, _, 8].
[_, _, _, _, _, _, _, _, _].
[3, 7, 4, 9, 2, 8, 5, 6, 1].
556,471 inferences, 0.124 CPU in 0.124 seconds (100% CPU, 4479554 Lips)
Sudoku = [[4, 9, _9886, 1, _9898, 7, 8, 3, _9922], [_9934, _9940, _9946, 3, _9958, _9964, 7, _9976, _9982], [7, _10000, 3, 2, _10018, _10024, 1, _10036, _10042],
[5, 3, 1, 6, 7, 2, 9, 8, 4], [_10114, 4, _10126, 8, _10138, _10144, 2, _10156, _10162], [8, 2, 7, _10192, _10198, _10204, 6, 1, 3], [_10234, 6, _10246, _10252, _10258,
_10264, _10270, _10276, 8], [_10294, _10300, _10306, _10312, _10318, _10324, _10330, _10336, _10342], [3, 7, 4, 9, 2, 8, 5, 6, 1]],
_9886 in 2\/5..6,
```

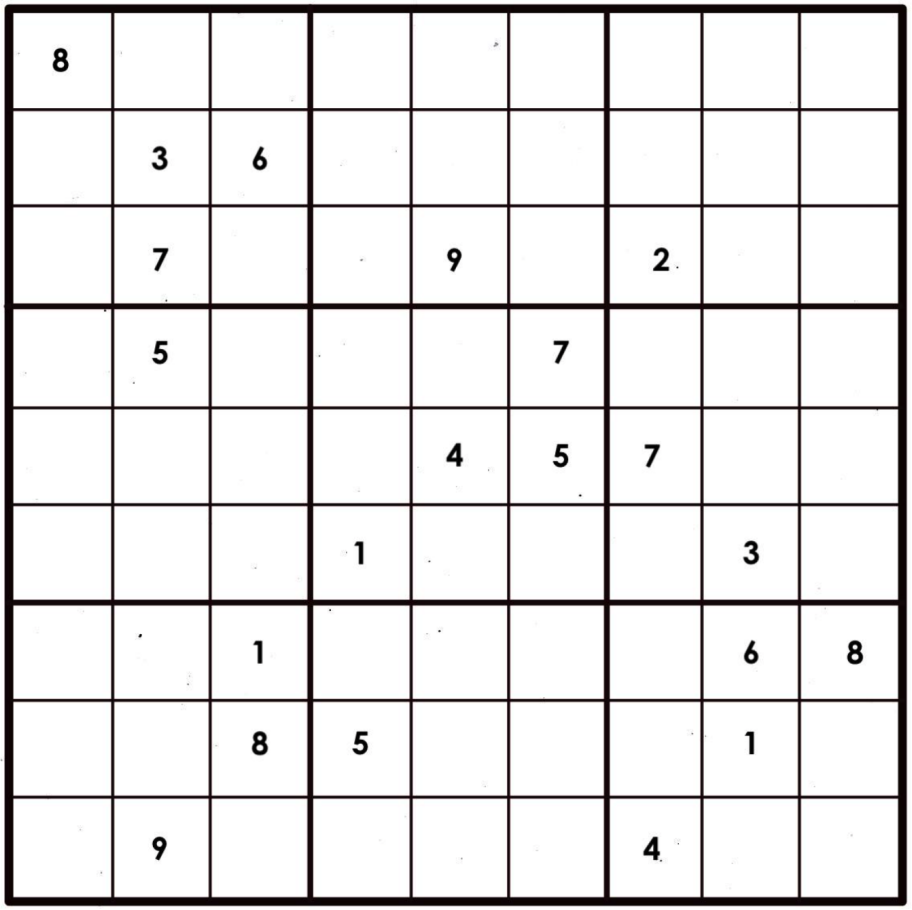
Exhibit B. Using both labeling and constraint propagation (the combined solution)



When working with medium difficulty Sudoku puzzles the constraint propagation solution fell short because even tho it took quite less time to solve the puzzle, the solution was only partial. Hence, my decision to try the combination of both constraint propagation and labeling, which in this case performed very well.

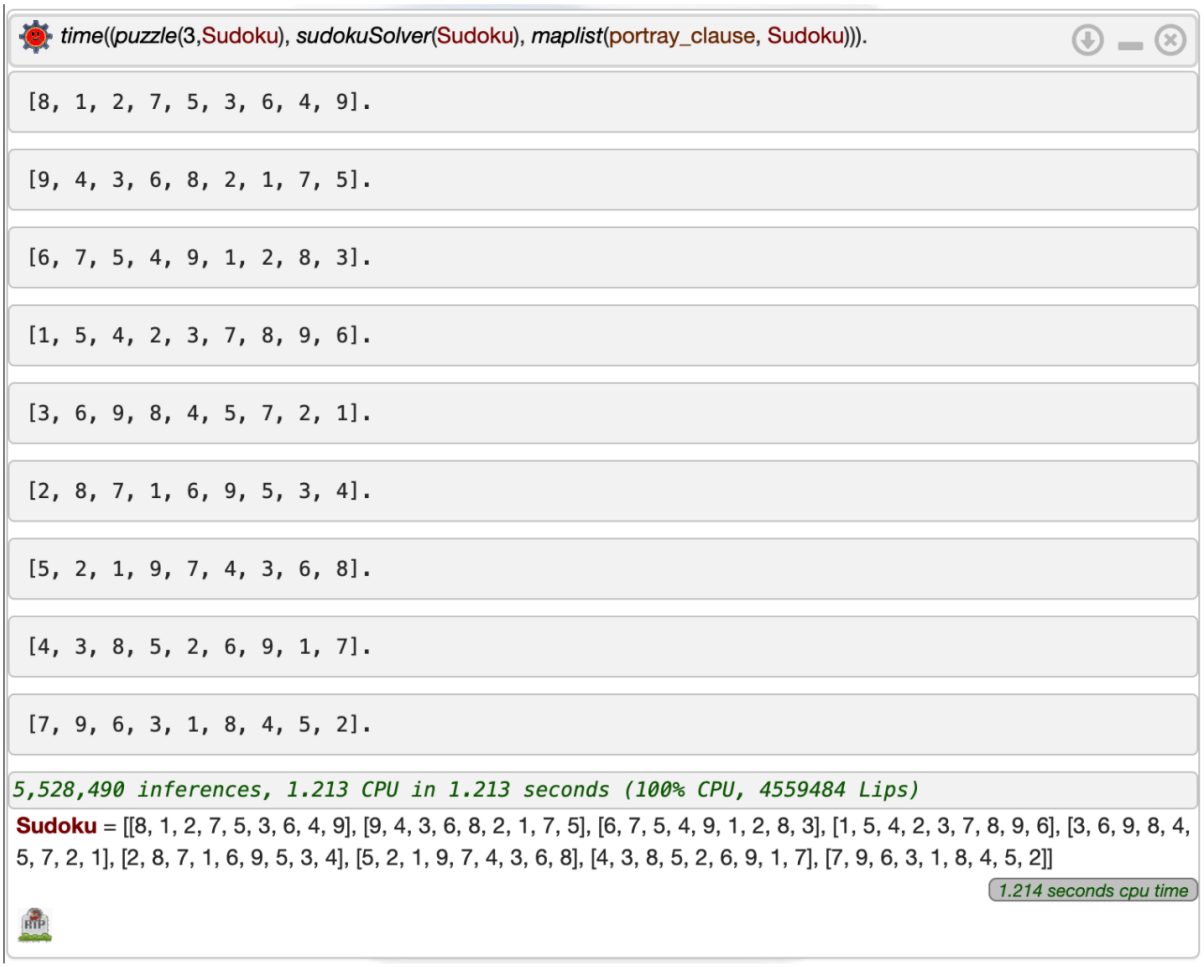
The combined solution spent just 0.154 seconds in solving the complete Sudoku puzzle.

Test 3. The hardest puzzle ever according to The Telegraph



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

Exhibit A. Using both labeling and constraint propagation (the combined solution)



As we can observe in Exhibit C. the hardest puzzle ever can be solved with the solution that combines labeling and constraint propagation easily, and it can be done in 1.213 seconds which is an incredibly short period of time in comparison to the time that it would take the average person to solve it.

Conclusions

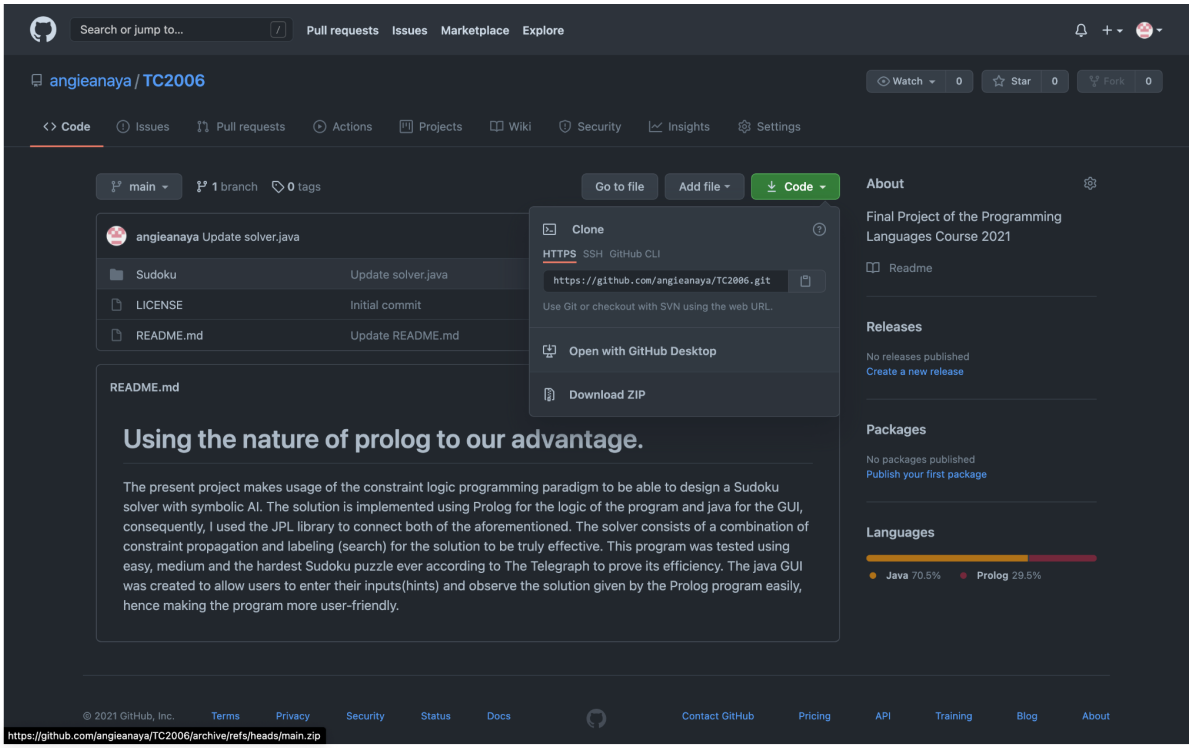
In sum, CLP is a combination between two declarative paradigms: constraint solving and logic programming. When using this paradigm, basic components of a problem are stated as constraints and the problem as a whole is represented by putting these aforementioned constraints together by means of rules.

The usage of constraint logic programming (CLP) was crucial in the development of this project because it extends the notion of a logical variable by allowing variables to have a domain rather than a specific value, in this case, the usage of this aspect allows us to define the domain of symbols that the Sudoku grid accepts. Moreover, the constraint part of this paradigm gives the programmer the power to specify the desired rules that the value of a constrained variable must abide by, in this case, the variable Sudoku is the solver in itself because it only exists if it satisfies all stated constraints.

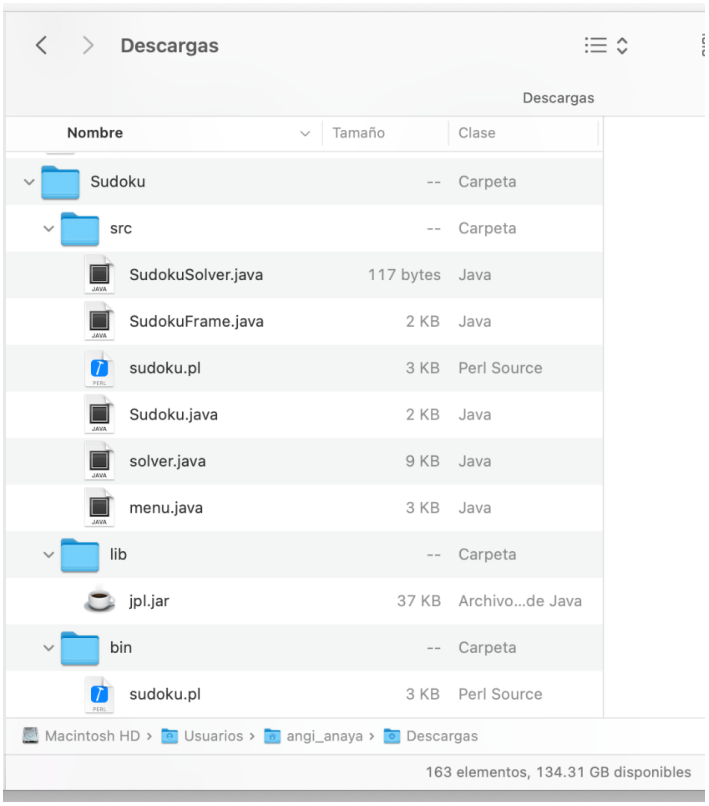
Using constraint-satisfaction methods can successfully navigate enormous search spaces which makes the calculus of the solution of a Sudoku puzzle incredibly fast and effective but the combination with the built-in backtracking characteristic of Prolog facilitates the customization of search procedures and therefore optimizes the solver algorithm.

Project Setup

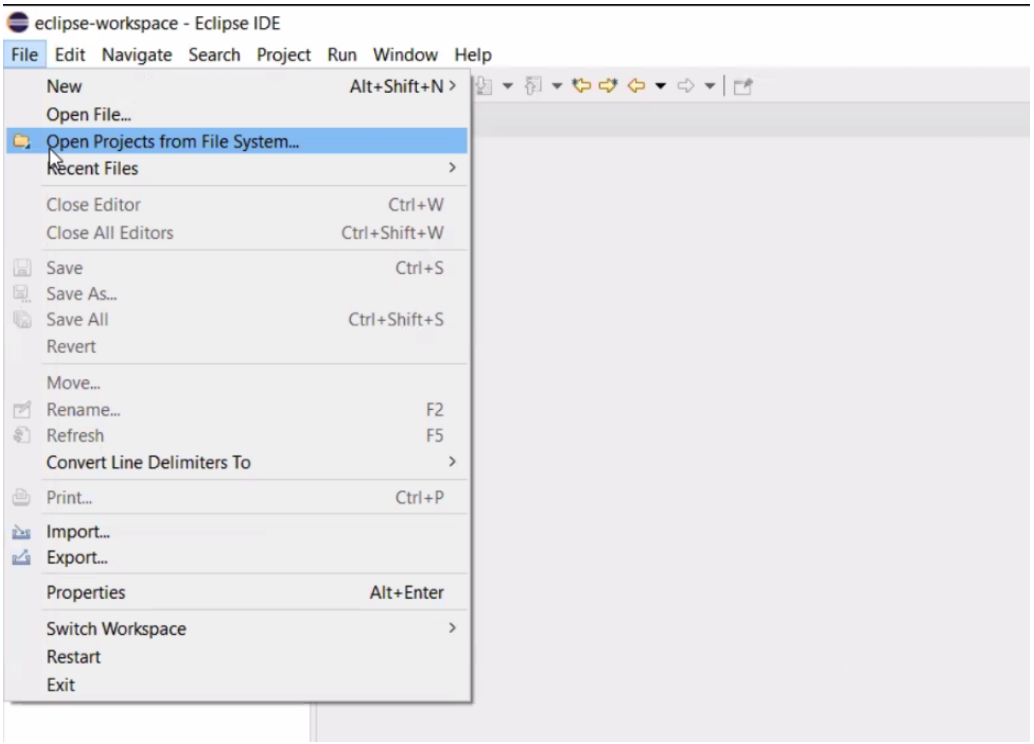
1. Enter the GitHub repository https://github.com/angieanaya/TC2006 and download the project as zip.



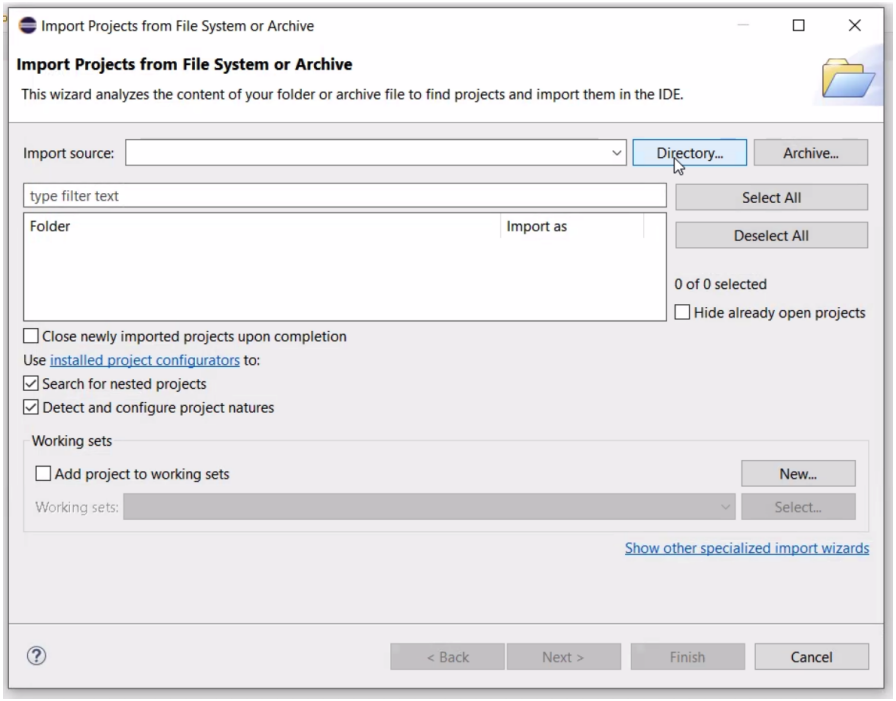
2. Once the project is downloaded, it must be decompressed, the file hierarchy will look as follows:



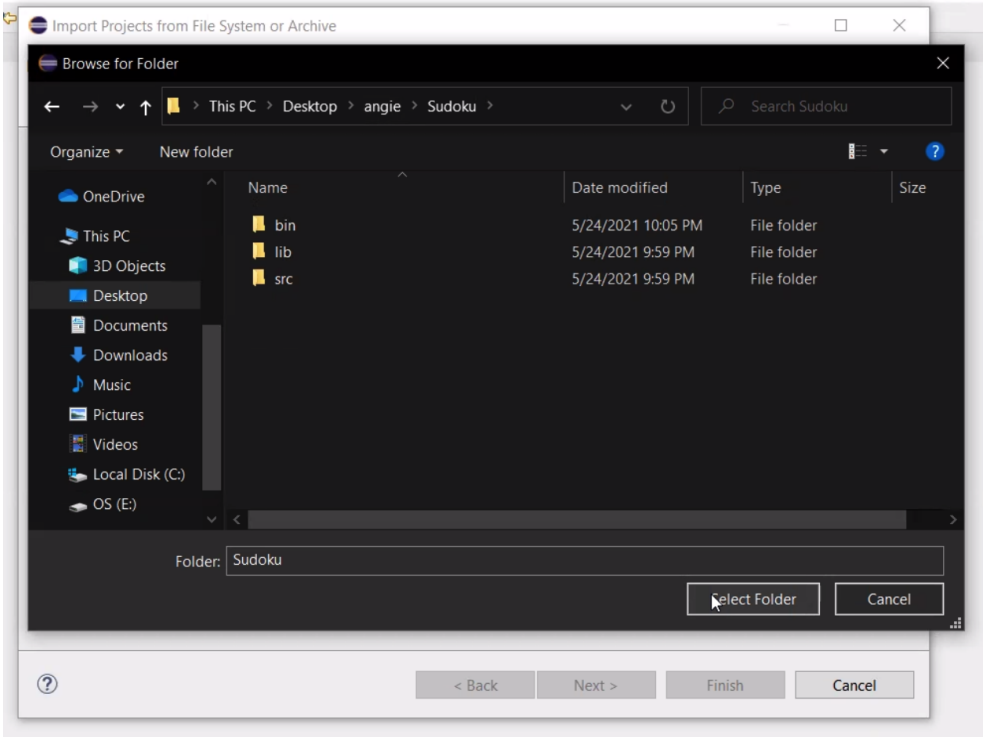
3. Open Eclipse IDE and click *File* >> *Open projects from file system*



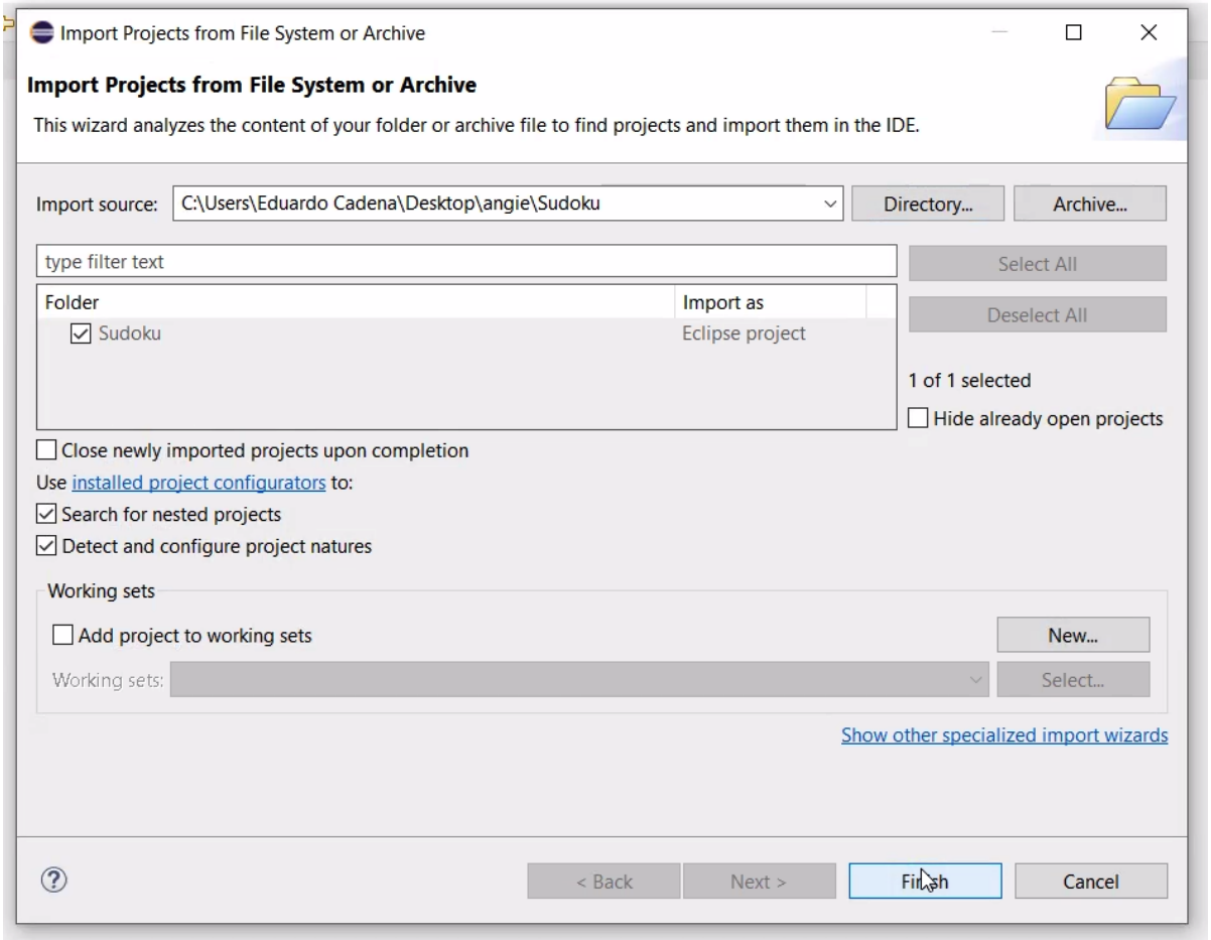
4. Click on *Directory* to open the local file system.



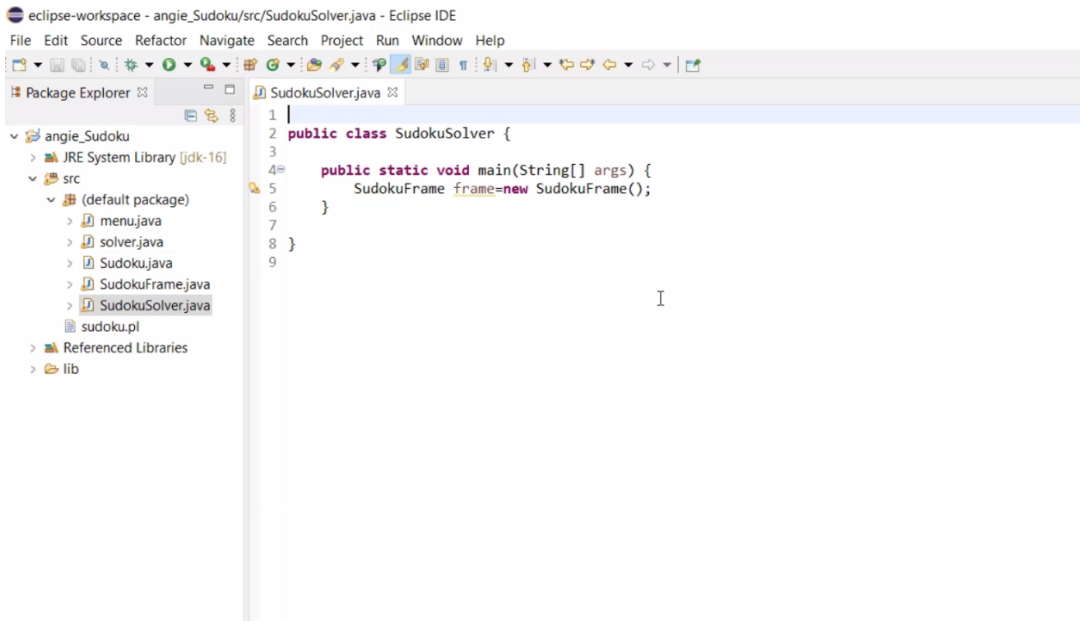
5. Select the *Sudoku* folder and click on *Select Folder*



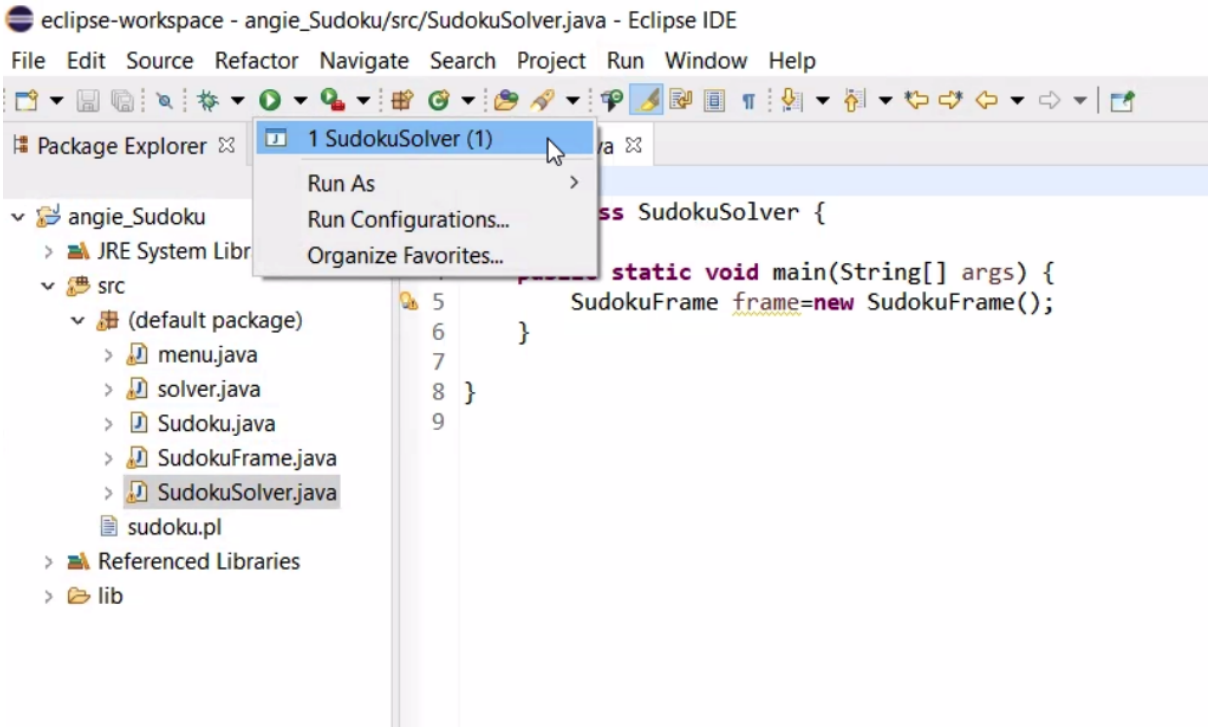
6. The window should look as shown below, then click on *Finish*



7. Open the SudokuSolver.java file from the hierarchy of files in Eclipse IDE, the file is inside the *Sudoku >> src* folder.



8. Click on *Run*



References

Backtracking in Prolog - javatpoint. (2011). Www.javatpoint.com. https://www.javatpoint.com/backtracking-in-prolog

Bessiere, C. (2006). *Constraint Propagation*. https://www.ics.uci.edu/~dechter/courses/ics-275a/spring-2014/readings/constraint-propagation-bessiere.pdf

By Telegraph Sport. (2012, June 28). *World's hardest sudoku: can you crack it?* The Telegraph. https://www.telegraph.co.uk/news/science/science-news/9359579/Worlds-hardest-sudoku-can-you-crack-it.html

CLP(FD): Constraint Logic Programming over Finite Domains. (2012). Swi-Prolog.org. https://www.swi-prolog.org/pldoc/doc/_SWI_/library/clp/clpfd.pl

Dincbas, M., Simonis, H., & Van Hentenryck, P. (1990). Solving large combinatorial problems in logic programming. *The Journal of Logic Programming*, *8*(1-2), 75–93. https://doi.org/10.1016/0743-1066(90)90052-7

JPL - Introduction. (2021). Jpl7.org. https://jpl7.org/

Play Free Sudoku online - solve daily web sudoku puzzles. (2018). Sudoku.com. https://sudoku.com/

PROLOG Facts, Rules and Queries. (2021). Trincoll.edu. http://www.cs.trincoll.edu/~ram/cpsc352/notes/prolog/factsrules.html#:~:text=A%20fact%20is%20a%20predicate,must%20end%20with%20a%20period.

The Math Behind Sudoku: Introduction to Sudoku. (2021). Cornell.edu.

http://pi.math.cornell.edu/~mec/Summer2009/Mahmood/Intro.html

PROLOG | computer language | Britannica. (2021). In *Encyclopædia Britannica*.

https://www.britannica.com/technology/PROLOG