Day 1, Thursday 1/3/19

I created the Block class that controls the methods of a single cell. It is a 3X3 array list of Strings and each item represents a number in the cell. I created a contains method which returns true if the Block already has the number and false otherwise. Then I created the add and remove method which will allow the user to edit if their insertion follows the rules of the game as given by the contains method. I then created a toString() which formats the array into a block of a Sudoku puzzle. I did a soft test on the Block class and everything seems to be working. I will probably run a more extensive test on Saturday.

Day 2, Friday 1/4/19

I created the Grid class which is a collection of 9 blocks and represents the entire Sudoku puzzle. It is a 3X3 array list of Blocks and has the same methods as the Blocks class. The main difference between the Blocks and Grid Class is that the contains method for the Grid class checks if the number being placed is already in the row or column the user is trying to place it in. Add and Remove methods call on their parent methods with the condition that the insertion returns false for contains. Testing revealed that my contains method is flawed and will need to be worked on. On Sunday, I will begin the Sudoku class which will serve as the interface for the user.

Day 3, Saturday 1/5/19

I have been running tests on Grid and Block class and the contains method for Grid class is the only issue. It ignores the Sudoku rules and allows the player to insert a number into the same column or row as a pre existing number.

Day 4, Sunday 1/6/19

I have fixed the contains method for Grid class. All that’s left for this project is the interface portion. I have been reading through Mr.K’s terminal demo and have been using some of the code he has- I will make sure to cite him in the READ.ME and in the code. The Sudoku puzzle can now constantly update and you can add and remove numbers in but the coordinates of the terminal don’t correspond with the indices of the Grid class so I’m going to have to figure out how to fix that. I’m considering changing the toString() of the Grid and Block class so that it fits more smoothly in the coordinate system of the terminal but only if I have to. I have a convert method in the Sudoku class and it will read in the file submitted, and will use that information to create a Sudoku puzzle. The link for where I’m getting the Sudoku puzzles will be in the READ.ME.

Day 5, Monday 1/7/19

I have fixed the coordinate problem from day 4 by creating an array for x and y coordinates. Then by using the indexOf method, I could figure out which index of the arraylist you’re on and can therefore convert the coordinate system of the terminal to my puzzle. I think I have met all of my goals for the minimum viable product and I will probably now work on creating a good menu, a more user friendly interface, and picking Sudoku puzzles for the user to complete. Once I’m satisfied with that which will probably be Wednesday, I’m going to move on in creating the generator method.

Day 6, Tuesday 1/8/19

The menu is up and you can select your difficulties easy, medium, hard by pressing E, M and H respectively. Then it will randomly select a file from the difficulty chosen and add the elements to the interface puzzle. I chose the Sudoku text files and added them to github. Tomorrow I will work more on designs and begin the generator method.

Day 7, Wednesday 1/9/19

I fixed a few bugs on the ReadPuzzle method and tested it so it should be working now. The problem was that I had to convert the coordinates received by the Scanner into the coordinates used by the add method. There was a slight glitch in the menu method and I found the source of the problem.

Day 8 Thursday 1/10/19

I couldn’t work on the project because I needed to study for the CS test but I did run through what I wanted to do for the demo and how I will explain the code.

Day 9 Friday 1/11/19

I think I’m going to hold off on working on the generator until after the demo. It’s better if I work on making the designs of the interface better than rushing to add a new feature to the project. I wrote down some things I should say on an index card.

Day 10 Saturday 1/12/19

I’m building a comprehensive test for my project to see if it is ready for the demo. I merged all of the branches and updated the READ.me so that it gives instructions on how to download the project.

Day 11 Sunday 1/13/19

I typed up what I’m going to say for the demo and how I’ll explain the code. I’m going to hold off on adding more commits until after Monday since I might make it not work. On Monday, I’m going to work more on the generator and make it more efficient. This, I think, will be the final step in completing the project.

Day 12 Monday 1/14/19

I added the puzzle method to generator.java. This will produce a puzzle by removing numbers and then filling them in 50 times and seeing if each attempt gives the same grid. Basically it checks for uniqueness in a puzzle and removes random elements. The only major run time complexity is the hidden complexity of the generator function as the puzzle method calls that multiple times. So optimization of generator.java is completely reliant on the generator method.

Day 13 Tuesday 1/15/19

The reason why the generator isn't efficient is because the grid contains method doesn't work properly. If I can fix that, the efficiency will improve greatly. Also, I optimized it a lot by not having it recursively call upon the generator method until it gets a correct sudoku puzzle. That took up a lot of unnecessary memory and always caused a stack overflow error. Also, I found the minimum number of numbers needed to have a determined puzzle, 77 so once my generator reaches a grid of size 77 it will know to stop which will save it from a lot of not needed insertions. The number of failed insertions also increases exponentially as the grid gets filled so taking away 4 moves needed to fill the grid will make the generator run very fast.

Day 14 Wednesday 1/16/19

I worked on the contains method and I’m drawing out what exactly is happening in the method. I think it should work but I can’t figure out where I went wrong in making the code. By holding the first and third index constant, we can iterate using a double for loop through a row of the grid. It sounds right in my head but I will need to examine it further to figure out the main problem.

Day 15 Thursday 1/17/19

The contains method works in nearly all cases on the Sudoku interface. I checked Sudoku.java and it should work so I’m assuming that it’s a glitch or lag in the terminal that’s causing them because contains method passes all of the tests in GridTest.java. As for the generator method, it’s taking a very long time to finish and I’m considering starting it from scratch.

Day 16 Friday 1/18/19

I started from scratch and created a new generator that is much more efficient than my old one. First you create a random permutation of the sequence [1,9]. Then, you go through a single block of a grid and for each square, you try to add elements from your random sequence until the add method returns true. When this happens, you remove that element from the sequence, which isn't necessary but efficient. This is much more efficient than pure random bashing. An upper limit for this generator is 729 iterations as you can't get stuck and reach a contradiction since all sudoku puzzles have at least 1 solution and we assume that each square will take the most number of tries, 9, giving 9\*81 = 729. If the size of the sudoku puzzle was generalized to be an n by n puzzle with [1,n] in each row, column, or box, we would have n^3 iterations at an O(n^5) algorithm (since generator has add method in each iteration). This method also can easily fit into the puzzle method.

Day 17 Saturday 1/19/19

Two commits ago, I might have jumped the gun in saying that my generator has O(n^5). I incorrectly assumed that as long as you follow the rules of the sudoku puzzle, you will be able to fill it without any revisions. I then tested this out for myself and realized that it was very easy to create a puzzle that leads to an unsolvable contradiction. So, I edited my code such that if none of the numbers 1-9 work on a specific square, you set the block back to its original input and create a new arrangement of the sequence to put in. Then, if that doesn't work 5 times, start the grid over from the beginning. If it still is polynomial time, it would have an astronomically high coefficient but since I now have it start over from the beginning, it's sort of acts like a recursive algorithm as it starts from where the function first began giving it a very fast growing run time complexity. Thankfully, the generator isn't meant to be used in general cases so we don't really need to worry too much about efficiency.

Day 18 Sunday 1/20/19

I wasn't making good enough progress with the generator so I put back the readPuzzle method. The final product for this project will probably be an interface but I'm also thinking of using Java to posts the time taken to complete the puzzle on Twitter. That way people can see how they fared against others and it can be a fun component of the project. I already downloaded the twitter libraries that Sudoku.java will use and it seems like an extra 2 lines of code so it won’t be too hard.