Dominic Espina jr

Parsons 343

Program 4

**LINUX COMPILATION INSTRUCTIONS**

**g++ driver.cpp**

**./a.out driver.cpp**

The program will then print the following

**Size of each generation:**

type any number **>=10**

*Example:* **Size of each generation: 100**

Then the program will then print the following

**Maximum amount of generations:**

Type any number >= 1

*Example:* **Maximum amount of generations: 1000**

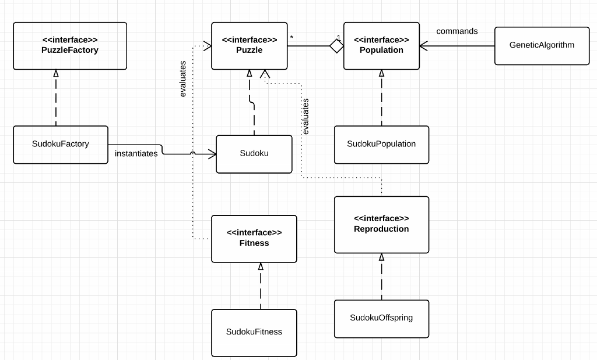
For Best results set size to **10** and generations to **500000**

Takes about 3 minutes to run but can reach fitness levels close to 0 and in some case’s are 0

**Design**

The goal of this assignment was to simulate evolution using Sudoku. The result, is supposed to get the best Sudoku possible which is based on its fitness level. The closer a Sudoku puzzle is to 0 the more “fit” the puzzle is.

I based my whole project on the following diagram given to us in the program instructions



Firstly, creating a genetic algorithm based off evolution, I used the following simple pseudocode

//create first sudoku from file

//fill population with first sudoku to (size)

//for (# of generations)

//cull x percent of the generation

//breed 10 per each one thats left

//print best sudoku puzzle

I had 90% percent of the population culled therefore when repopulating the population to near the original size, I just created 10 copies of each of the 10% of Sudoku puzzles left in the population.

My next step was creating the sudoku puzzle as the population was made of sudoku puzzles

**Designing the sudoku puzzle**

Each puzzle I decided was going to have its own 9 x 9 board of squares in the form of a 2d array

The square is a struct I created which holds a number and a bool showing if the number is fixed or not fixed

Then a sudoku puzzle will hold a fitness int which will be calculated by population

Finally, a sudoku puzzle will hold a string representing 81 numbers which represents the sudoku board

I did this in order for there to be the same way of initiating a board for all boards Sudoku(string data)

When the constructor Sudoku(string data) is called the sudoku class will call a fillboard() method which will read the string and translate it into the 9 x 9 board of squares.

**Designing the population class**

Designing the population class, it only has one variable which is a vector of Sudoku

The population class can interact with the vector of sudoku in the following ways

*Cull*

The purpose of this method is to delete x percent of the container leaving only 1 – x percent left of Sudoku puzzles within the container

Here is the following pseudocode to accomplish this task

//get number that should be left in the list (size - cull percent of (size))

//for number that should be left

//get the best in list

//add it to temp container

//remove it from old container

//delete all thats left in old container

//set old to temp

At the end of the method the size of container should be 10% of the old size. And should contain the top 10% of sudoku puzzles as I have the fixed cull amount to 90%.

Calculating the best in list is done by seeing which puzzle in the list has the best fitness

Calculating fitness is done using the fitness class which I will get into later

*newGeneration*

The purpose of this method is to be called after culling and is to repopulate the vector<sudoku> container to the normal size by creating copies of itself, 5% of these copies will be mutants in which the board within the sudoku puzzle will be randomly changed using the reproduction class which I will get into later

Here is the following pseudocode for this method

//make temp list from old container

//clear old container

//for container size

//get sudoku at i

//add 10 of these sudoku puzzles based on sudoku at i

//calculate all new puzzles finesses

At the end, I calculate their fitness because the new puzzles that have been added, have not had their fitness’s calculated. Again, this is done using the fitness class which I will get into later.

I created a separate method for adding 10 of these sudoku puzzles based on the seed at i

Within this method I did the following

//seed

//for (10)

//if a mutant

//get mutant child of seed

//else

//get regular child who is a clone of seed

//add child to list

So there is an initial seed, using this seed I will add 10 children of the seed

This child can either be

1. A mutant child which occurs 5% of the time, this mutant child is created using SudokuReproduction class which I will explain later
2. 95% of the time a regular child is created which is a complete copy of the seed using SudokuFactory class

**SudokuFactory class**

This class is called when reproducing a perfect copy of a puzzle in the newGeneration method in population

This class only has one method which is createPuzzle which takes in a string data and returns a new puzzle based on the data

**Designing the SudokuOffspring class**

The purpose of this class was to take in a sudoku and alter about 5 of its blank spaces (0) to a random valid number

This is done using the method makeOffspring(Puzzle\* puzzle) which takes in a sudoku puzzle and returns a slightly altered sudoku puzzle

***Creating the makeOffspring method***

The main idea of this method is the following

1. Find all blank spaces within the board
2. Get list for all valid numbers that can be in this row
3. Get list for all valid numbers that can be in this column
4. Get list for all valid numbers that can be in this 3 x 3 square
5. Choose a random blank space from the step 1 blank space list
6. Create list of all common numbers in the previous 3 lists.
7. Choose random number from the common number list
8. Remove random number from row list, column list, 3x3 list
9. Set board at chosen blank space to the random number

**Step 1:** Find all blank spaces within the board

For this step I created a list called vector<int> chooseList[9];

It is size 9 because there are 9 rows and within each row there is a vector list which is because each row can have (0-9) number of blank spaces (0). The int in the vector represents the column index at which there is a 0 within the current vector row

Firstly, I initiate choose list

This is done by creating a vector<int> initiatelist which contains the number 1 to 9

Then I set each row in chooseList to initiatelist thus every vector<int> in chooselist now has the numbers 1-9

Then I iterate over the board square by square, when there is a number more than 0, I go into the row its found at in vector<int> and remove the following column index number from the vector<int> chooseList[row].

After this choose list should only contain column indexes in which there are 0

**Step 2:** Get list for all valid numbers that can be in this row

For this step I created a vector<int> rowLists[9];

This is to store the same as chooseList except each number in each vector represents the numbers that aren’t in the corresponding row

I initiate rowLists the same way as chooseList

Then within the same iteration of the board, square by square as chooseList, when the square is != 0 remove the number from the corresponding vector<int> in rowLists[row];

**Step 3:** Get list for all valid numbers that can be in this column

For this step I created a vector<int> columnLists[9];

This holds the same purpose as rowLists except contains all the valid numbers that aren’t in the corresponding column

I initiate this the same way as rowLists

Then within the same iteration of the board, square by square as chooseList, when the square is != 0 remove the number from the corresponding vector<int> in rowLists[row];

**Step 4:** Get list for all valid numbers that can be in this 3 x 3 square

For this step I created a vector<int> squareLists[9];

This holds the same purpose as rowLists and colLists except each vector<int> contains all the numbers that aren’t in the corresponding 3 x 3 square

The numbering of the squares corresponding to the number in the list goes like this

0 1 2

3 4 5

6 7 8

I initiated this list separately from rowLists and coLlists

I had to traverse the board from each 3 x 3 square 0 to 8

This went like this

for each square I remove the number from the corresponding vector<int> in the squarelist in the corresponding vector<int> 0 to 8, 3x3 square.

**Step 5:** Choose a random blank space from the step 1 blank space list

This is done by choosing a random row that’s not empty within the chooseList vector list

Within that chosen row, choose a random number from the vector column index

Your blank space is at [randrow][randcol]

**Step 6:** Create list of all common numbers in the previous 3 lists.

I created a method which returns a list of the similar numbers in the inserting two lists

This method is done by the following pseudocode

//colList

//rowList

//while neither collist or row list is empty

//if they are both the same, valid number that can be put in the spot so

//push the number to the return list

//erase both lists fronts

//else if collists number[0] is smaller than the row lists number[0]

//erase colList front

//else

//erase rowList front

The essence of this method is it compares the first values of two lists

If front is same add number to return list, remover front from both list

Else if first list front is smaller than second list front remove front from first list

Else remove front from second list

So I then I have to figure out what value can be put in this spot so

I got a list of the numbers that aren’t in this row from rowLists[randrow]

I got a list of the numbers that aren’t in this column from columnLists[randcolumn]

I got a list of the numbers that aren’t in this 3 x 3 block from squarelist[sqIndex()]

Sqindex() finds what 3x3 the square is in (0 – 8)

Then I had to merge these Lists

I merged the rowLists and ColLists into a temp List

Then I merged the squareList and tempList into a addList

This addList represents all the numbers that can be put in this board[randnum][randcol]

**Step 7:** Choose random number from the common number list

I got a rand number from this list which is the number that will be inserted

**Step 8:** Remove random number from row list, column list, 3x3 list

I got removed the number from the corresponding

rowLists[row]

ColumnList[col]

squareList[sqIndex()]

**Step 9:** Set board at chosen blank space to the random number

Just set board at [randrow][randcol] to the random number

**Overall**

The pseudocode looks like this

//create new clone of input

//create initiate list

//create array of vector of rows and colums [9]

//create array for sqaure in sudoku [9]

//initiate them with the numbers 1 to 9

//adjust them to represent output

//MAIN ALGORITHM

//for (5)

//get a random row where its not full

//within that row get a random empty spot

//if the row isn't full yet

//get list of possible numbers that could be in this spot [r][c]

//remove the number form there respective lists

//update output board

//update output data

**Designing the fitness class**

The main purpose of this class is to find the fitness of a sudoku puzzle

This is done by its method howfit(Puzzle p)

Here is the pseudocode for this method I will explain what each aspect mean

//for (r = 0 r <= 6 r = r + 3)

//for (c =0 to 2)

//total = get conflict rows

//total = get conflict columns

//total = get conflicts boxes

//return total

I decided to iterate the top for loop with the numbers 0, 3, 6 instead of the traditional 0,1,2 for the conflicts boxes method as there are only 3 rows the top left corner of a 3 x 3 box can be in which is 0, 3 and 6.

The bottom is 0 to 2 (0,1,2) to iterate 3 times

These two loops Iterates through all numbers 0-8 by doing r + c

These numbers are sent to tell which row and which column to find conflicts in

To find what box it is in send c, r

Here is how I get the corresponding conflict aspect.

**Get conflicts per Row**

I created a bool list of the size 10 with all spots of false

I then iterate through the corresponding row, when I see the number in the bool list. When I iterate through the row and I see a number if the boolList[number] is true add to total else set boolList[number] to true

//for (int i = 0 to <= 8 i++)

//get num row[r][i]

//if list[num]

//total++

//else

//list[num] = true

//return total

**Get conflicts per column**

Same as the conflicts per row but this time check the corresponding column for conflicts

//bool list used check

//for (int i = 0 to <= 8 i++)

//get num row[i][c]

//if list[num]

//total++

//else

//list[num] = true

//return total

**Get conflicts per 3x3 square**

Same method as before but I need to traverse the list in a 3 x 3 fashion instead of a row and column fashion

So passed in is the r and c

R can be the numbers 0,3,6

C can be the numbers 0,1,2

//bool list used check

//for (r to r +2)

//for (c \* 3 to c \* 3 + 2

//get num at board [r][j]

//if list[num]

//total++

//else

//list[num] = true

//return total

These for loops traverse the box r to r + 2. In addition traverse c \* 3 to ( c \* 3 ) + 2

This traverses the corresponding box

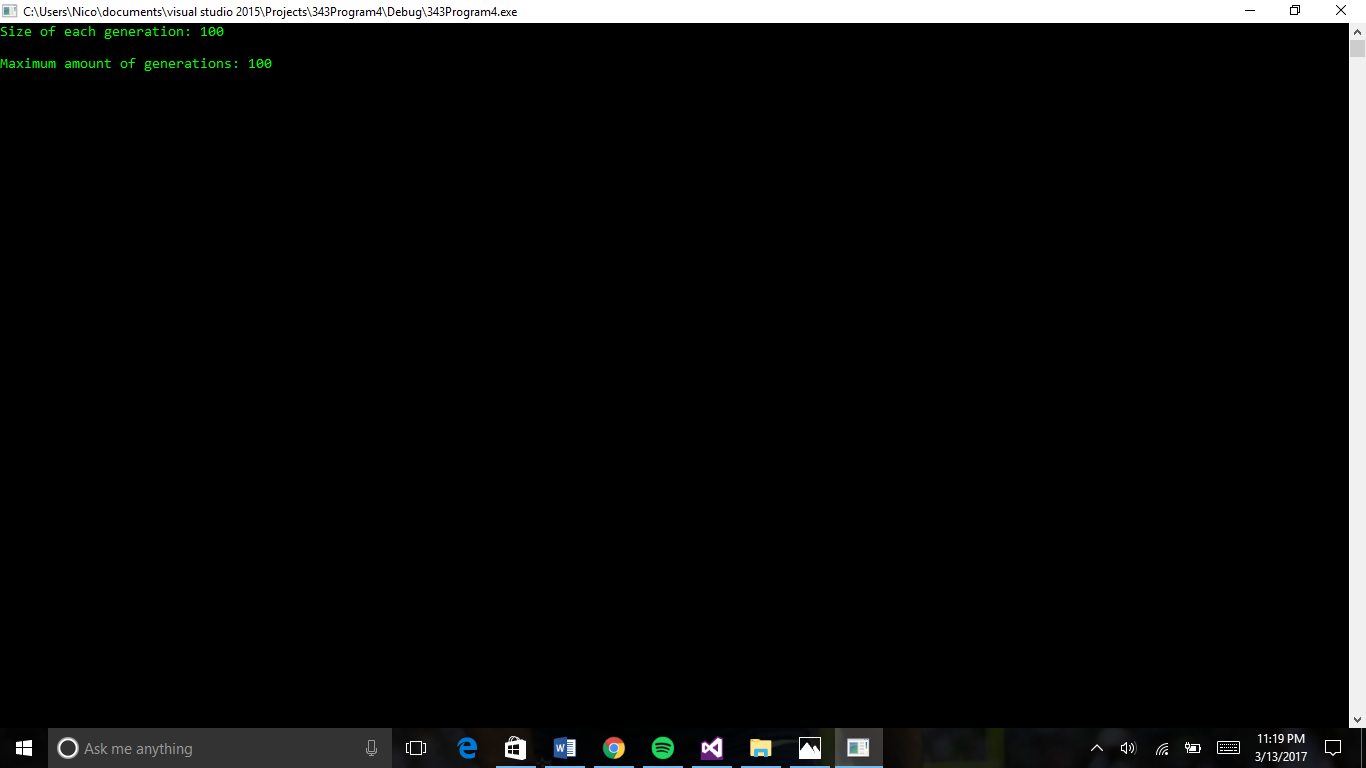
**Testing**

There were many tests and glitches especially regarding calculating the fitness and creating an offspring

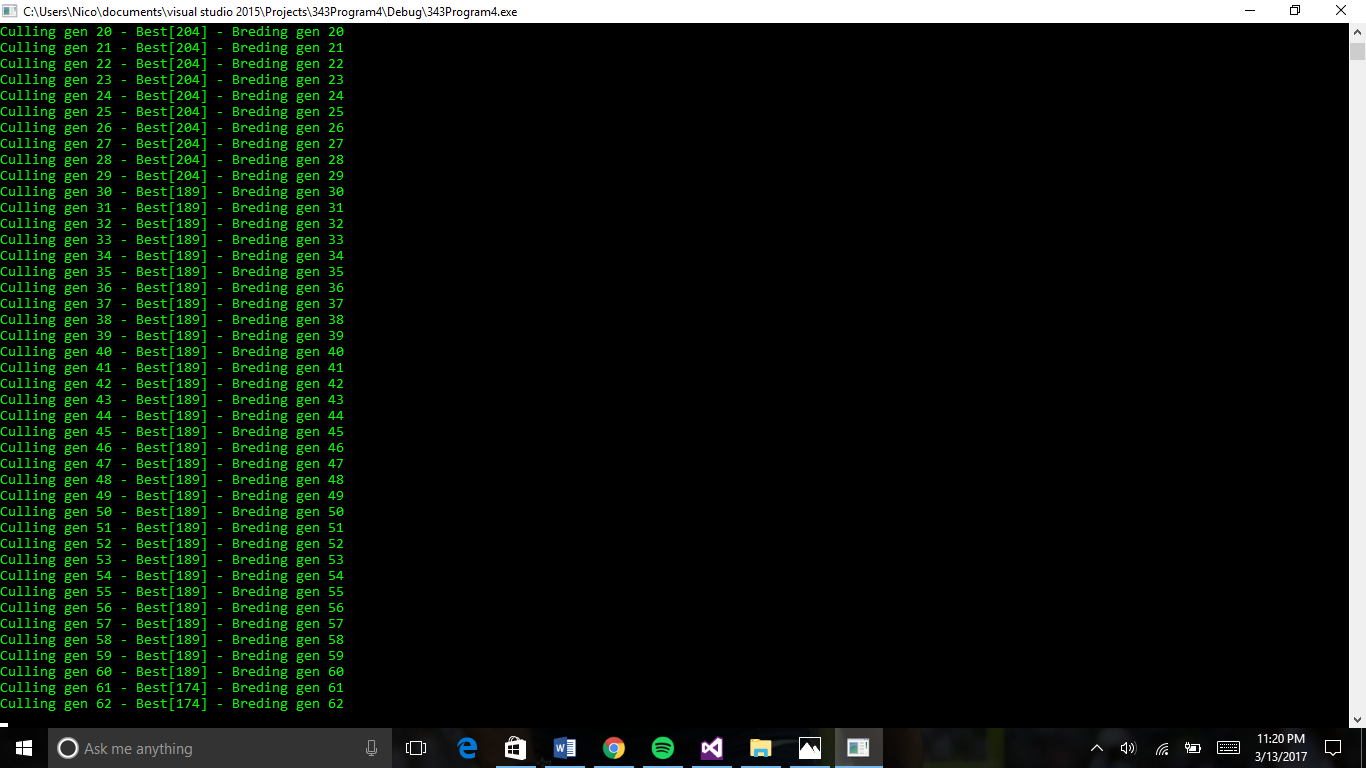
In the fitness class there were out of index errors, this was solved by checking my boundaries

Creating an offspring, code was often functional but there were often repeats in rows and columns which I had to fix. Most of this was fixed by isolating the problem, then changing variable indexes found at the problem. Many of the removingNumberFromColumn or row had the wrong row or column number.

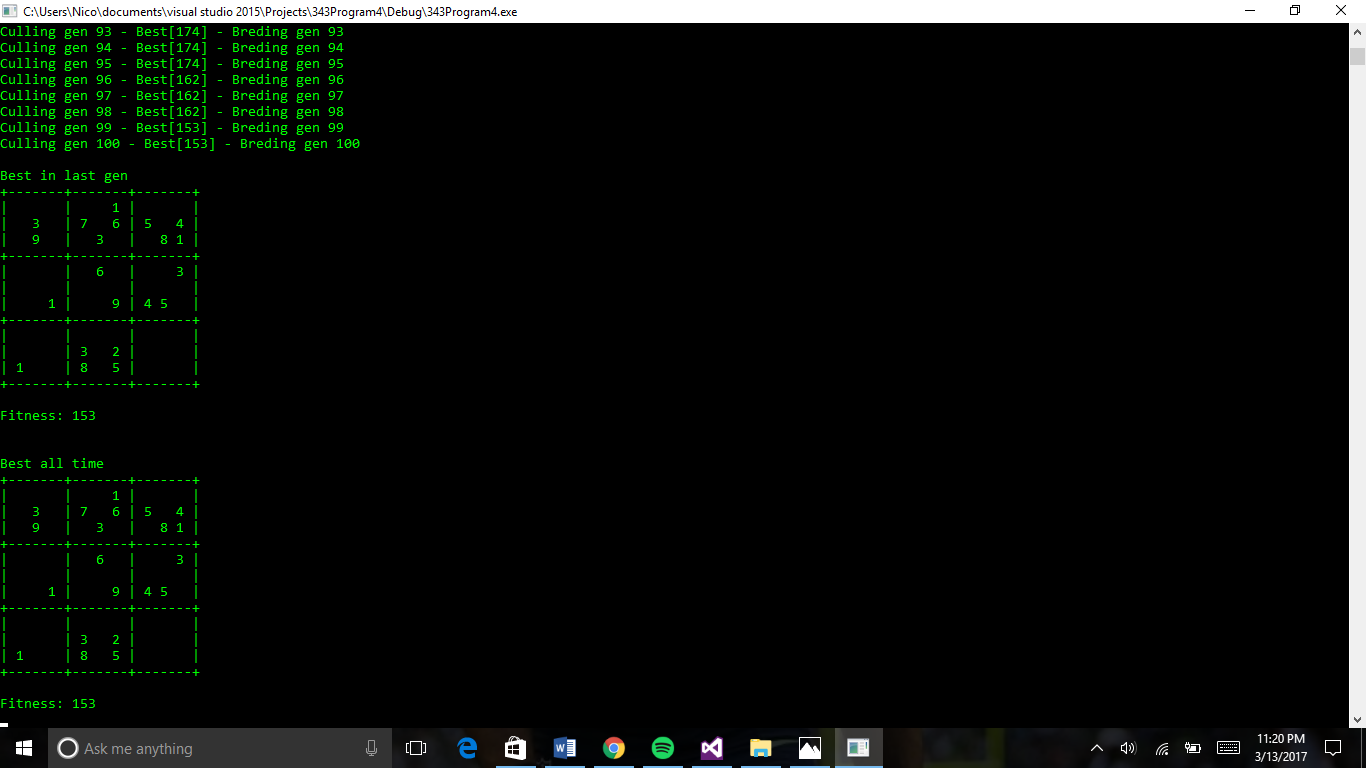
**Display**



Insert the data



Middle of making generations

Displays best in last generation + fitness

Displays best of all time + fitness