Project Report

Group 6, Team 3

Members: - IMT2020553 Abhinav Mahajan, IMT2020546 Aamod BK, IMT2020532 Anwit Damale, IMT2020550 Yathin Kumar, IMT2020541 Vamshidhar Reddy

Abstract: -

‘Sudoku Solver’ is an application, which takes an unsolved Sudoku as an input, solves it, and gives the user the ability to either see the answer, or try solving it using the GUI provided by the application. The GUI has several features, upon clicking on the appropriate button, the user will know what entries he has made so far are correct or wrong. He can also receive hints (a maximum of 3) upto his own convenience. The GUI can really help user’s solve Sudoku’s, as they can receive help, or know when they’re going wrong while solving Sudoku’s. This can build their abilities to solve more Sudoku’s. Many applications have been made in application repositories, like app store, play store. These applications give a GUI to solve a Sudoku which the application gives, this on the other hand will give the user to try solving any Sudoku of their preference, which they might have encountered anywhere.

The Project helped the team members learn a lot. This was the first time we wrote code for enabling a robust, fault tolerant, smooth flowing GUI. Due to a short time-span for submission, good team work, team values and cooperation was needed, and were implemented.

Introduction: -

Using drop down lists, the question Sudoku is input by the user. Set, Unblock button disable/enable respectively the input. Once the Sudoku is set, the ‘solve’ button solves the Sudoku. The Sudoku solving algorithm is infact very simple. One thing to be noted is the Sudoku’s can’t be solved only with algorithms, because they are endless. Upon going through Sudoku’s solving experts, they say ANY Sudoku can be solved under 30 minutes using few algorithms, and a few smart assumptions. Getting inspired by this, the program does exactly that. First “Option elimination” is done, if a number is filled, the row, column, box, it lies in cannot have that number again. Next 4 filling algorithms are applied. If a number (1-9) occurs only once in a row, The block in which its only occurrence is encountered, is filled, then again the elimination method is called. The same concept is applied to columns, and the 9 boxes. The 4th algorithm goes through each unfilled box and counts number of possible entries in it. If the number of entries possible is 1, the number which is possible is filled in that box. This concludes “Primitive Solving”. This takes nothing but a few milliseconds to execute, and this simplifies the question Sudoku a lot. Now assumptions need to be taken, First number of unfilled boxes in counted. Depending on the number of unfilled boxes, a corresponding number of assumptions are made. For the toughest of toughest Sudoku’s, generally 15 assumptions are needed. Multiple programs are written on the internet which go box by box permuting all cases possible (Brute forcing, which we don’t like). This may take upto 15-20 minutes for execution! This program AVOIDS this, by not making more than 15 assumptions. The program makes assumptions in places with LEAST possible number of possible entries (Thereby making the assumptions made, “smart”). By sorting appropriate data structures, it achieves this. And the moment, the assumption it makes forms an invalid Sudoku, it breaks out of the loop until a valid assumption is made. After making assumptions, multiple cases arise. One case NEEDS to be correct, because the Sudoku can have only one solution. When that case is reached, the computer simply applies the 4 algorithms used before to finish solving the Sudoku. This way you can see, the program uses good balance between algorithms, and computational power. In under \*30 seconds\*, 99.99% Sudoku’s can be solved. To be noted, Using appropriate checks, first it is checked whether Sudoku input is valid or not, and whether or not it has enough entries to solve, this way the program is free of crashes or being stuck in incredibly long loops.

From here on, GUI coding is done. The GUI frame will give the user nearly every option he wants, he will know whether his entry is correct or wrong. He can take a maximum of 3 hints too. He will be timed too, and each wrong entry made will be counted.

The user may also just see the answer, which is printed in a neat and appealing way. To be noted, the user can either only see the answer, or use the GUI, not both. The point of this was to disable him from cheating by seeing the answers.

The user can click on ‘solve’ button only ONCE. Once the program is started, and the tkinter window opens, it is meant to be used only for one Sudoku for every time it is started. This stays true with the clear, uni-functionality (Either “solve yourself” or “see answer”) of the program. It is impossible to break the code or crash it, by these controls.

System Requirements: -

Windows OS/Ubuntu and a python runtime.

Space requirement: - 4 GB RAM minimum

Flowchart: -

Initially, 5 tabs are visible. 4 are disabled and only input tab can be used.

Using drop down lists, inputs can be made. Set button freezes the options, Unblock unfreezes the options. Once the user is satisfied, Solve button is clicked. ‘Set’ is automatically called, in case the user did not himself do it. Unblock if used again, will not result in anything. The solve button can only be clicked once. An appropriate message also will be displayed if this case occurs. Solve button enables a tab. If Sudoku is valid, correct entries are made, a tab called solved is enabled. Else, unsolved is enabled. The user needs to go to the corresponding tab enabled.

In the unsolved tab, a message is printed, and the user is expected to click on exit on page 1.

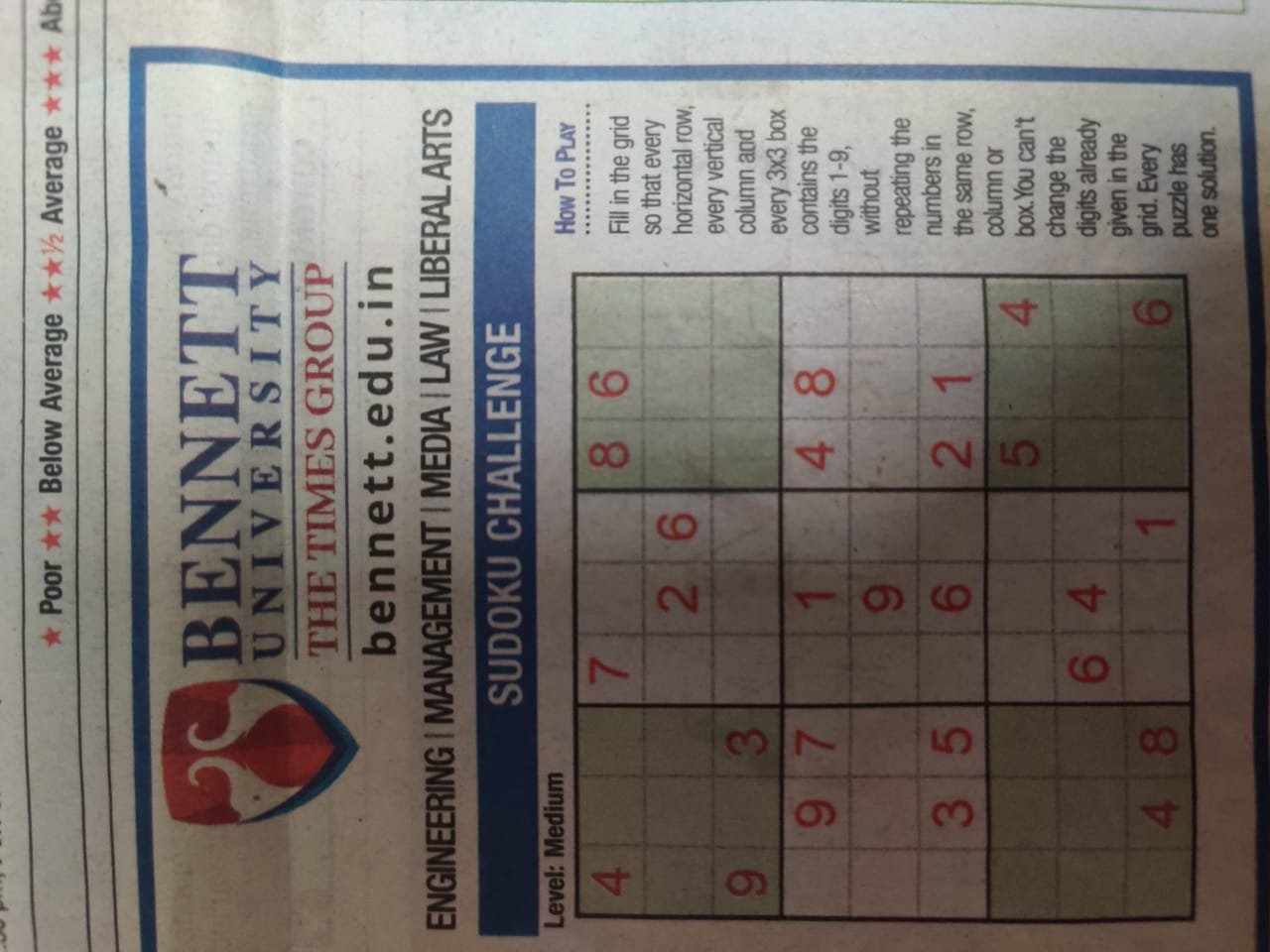
In the solved tab, buttons are given, “enable GUI”, or “see answer”. Enable GUI enables GUI tab which allows the user to solve it himself using our GUI. ‘see answer’ enables “Show Answer” tab which allows the user to see the answer of the Sudoku. During the execution, only one of the 2 tabs can be enabled.

“Show Answer” tab shows the solved Sudoku. The user can click on exit on page 1 after this.

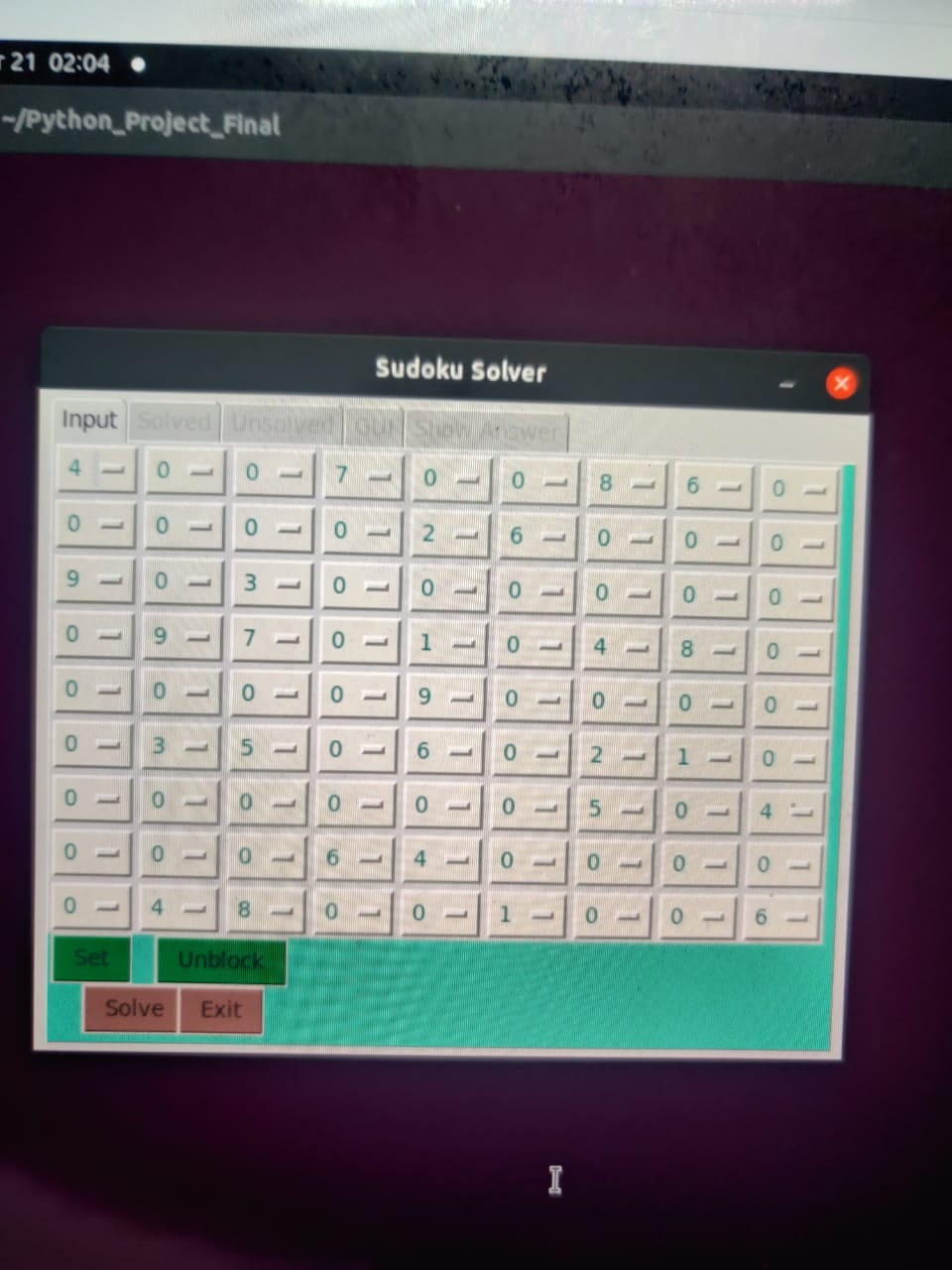
“GUI” tab has a matrix of drop down lists, and “Check Entries” button. The drop down lists have numbers from 0-10. 0 means undefined(empty)/default value. 10 means you want a hint at that position. After clicking check entries, the correct answers drop down lists will be blocked, wrong answers won’t be and an error message will come telling entry at a coordinate is wrong, change it 0. For hint (10), the answer at that coordinate will be shown via a message. Coordinates are 0 indexed (for example, first block, 1st row and 1st column is (0, 0)). Once 3 hints are exhausted, a warning message will flash anytime else another hint is used. Once Sudoku is complete and if solved, “Check Entries” button will flash a message that you’ve won, with the time taken, hints used, wrong values entered. Exit on page 1 can be clicked now.

Demonstration: -

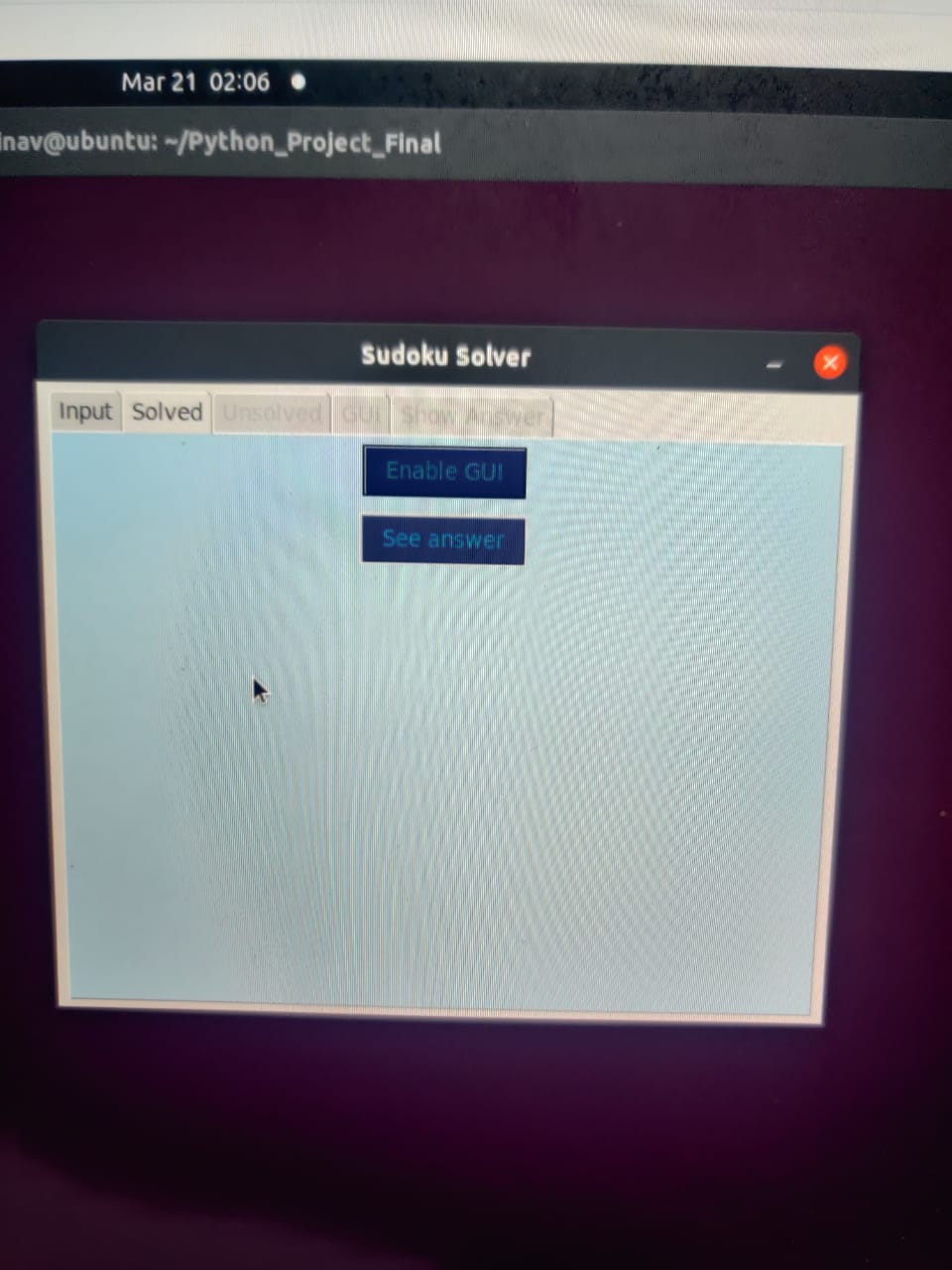
This is the sample Sudoku used: -



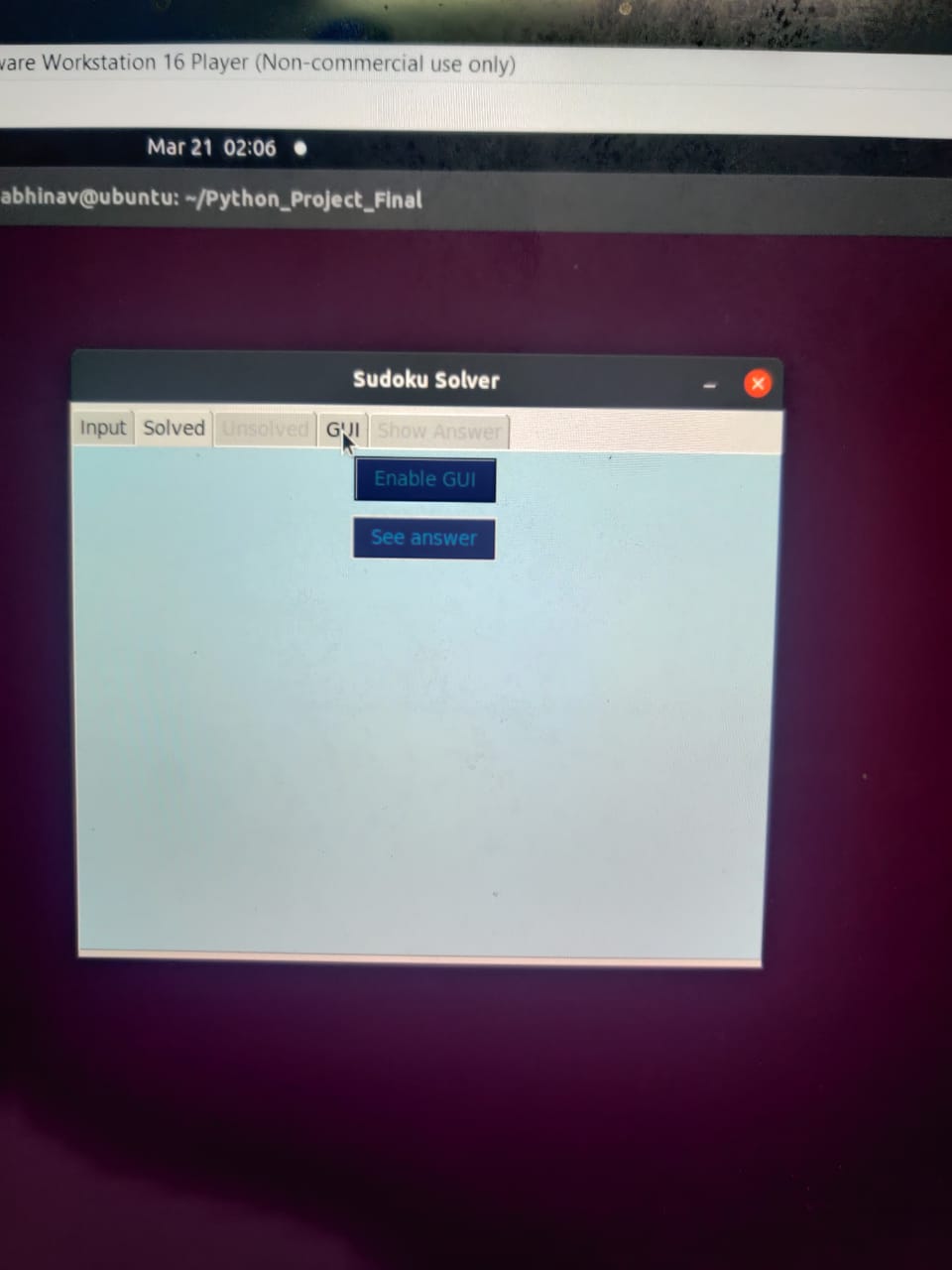
On running the program: -



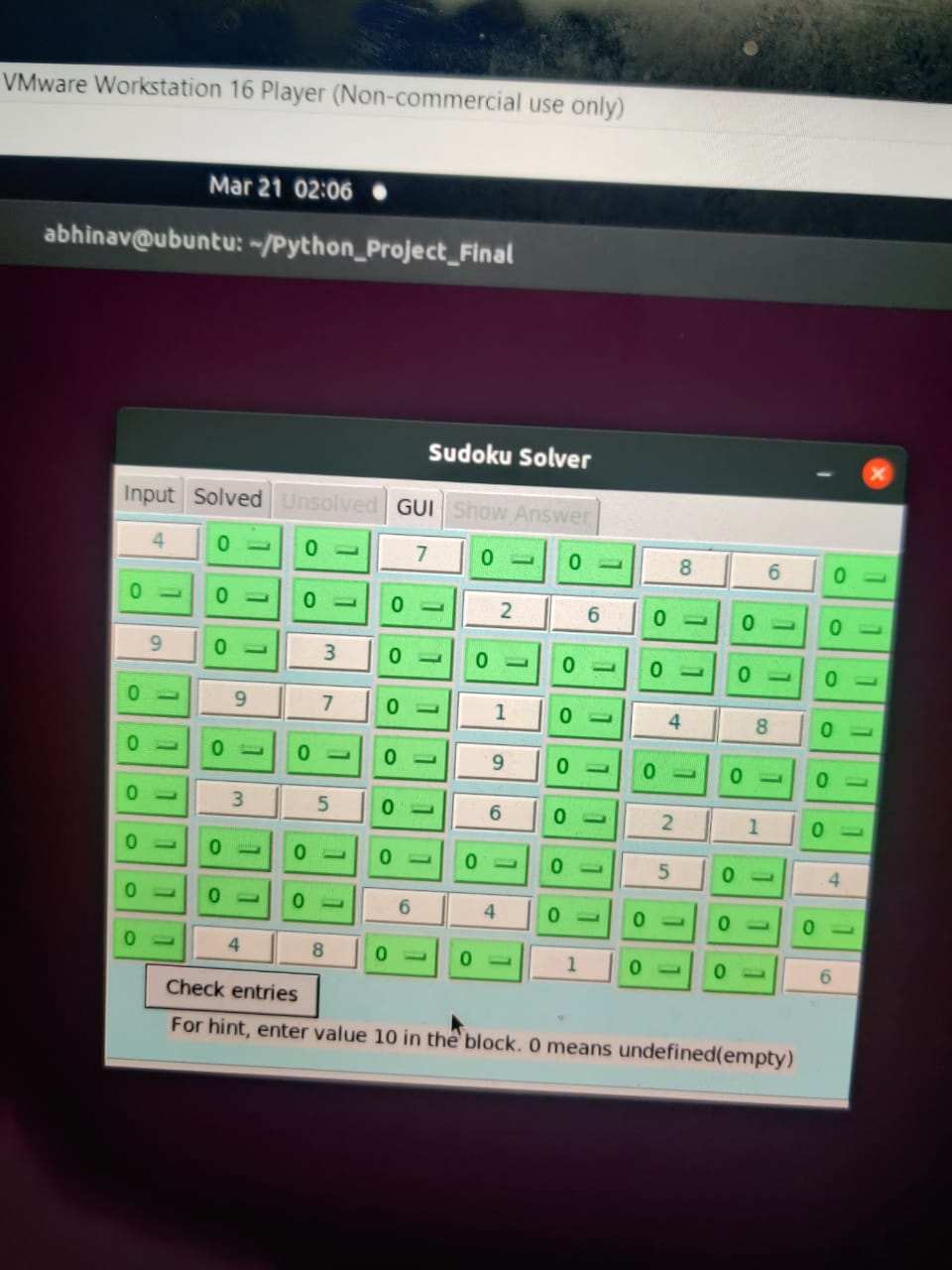
(This is how the input will be put, Set is clicked, then Solve, Since Sudoku is solved, Solved tab opens)



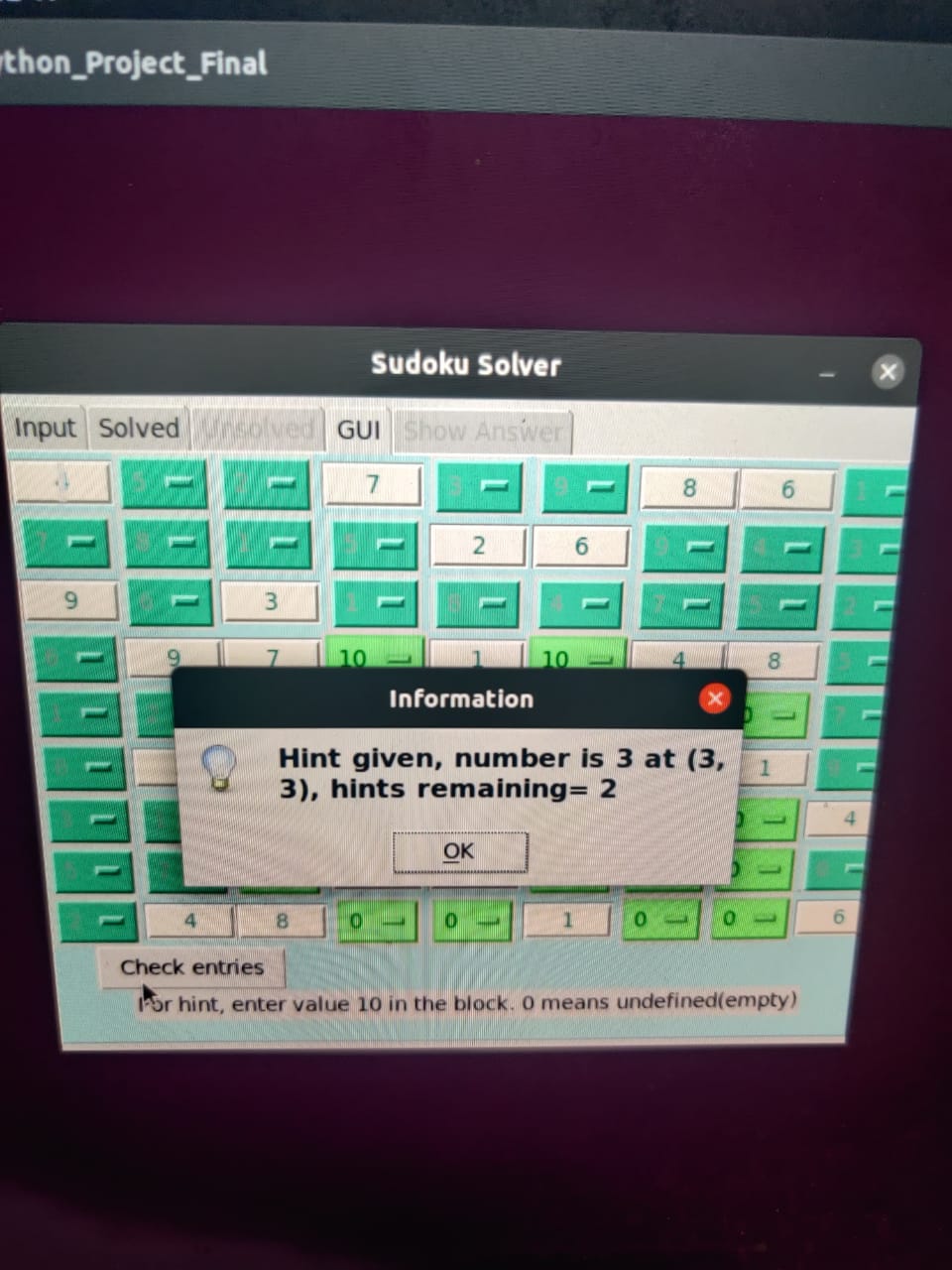
(Now enable GUI is clicked)



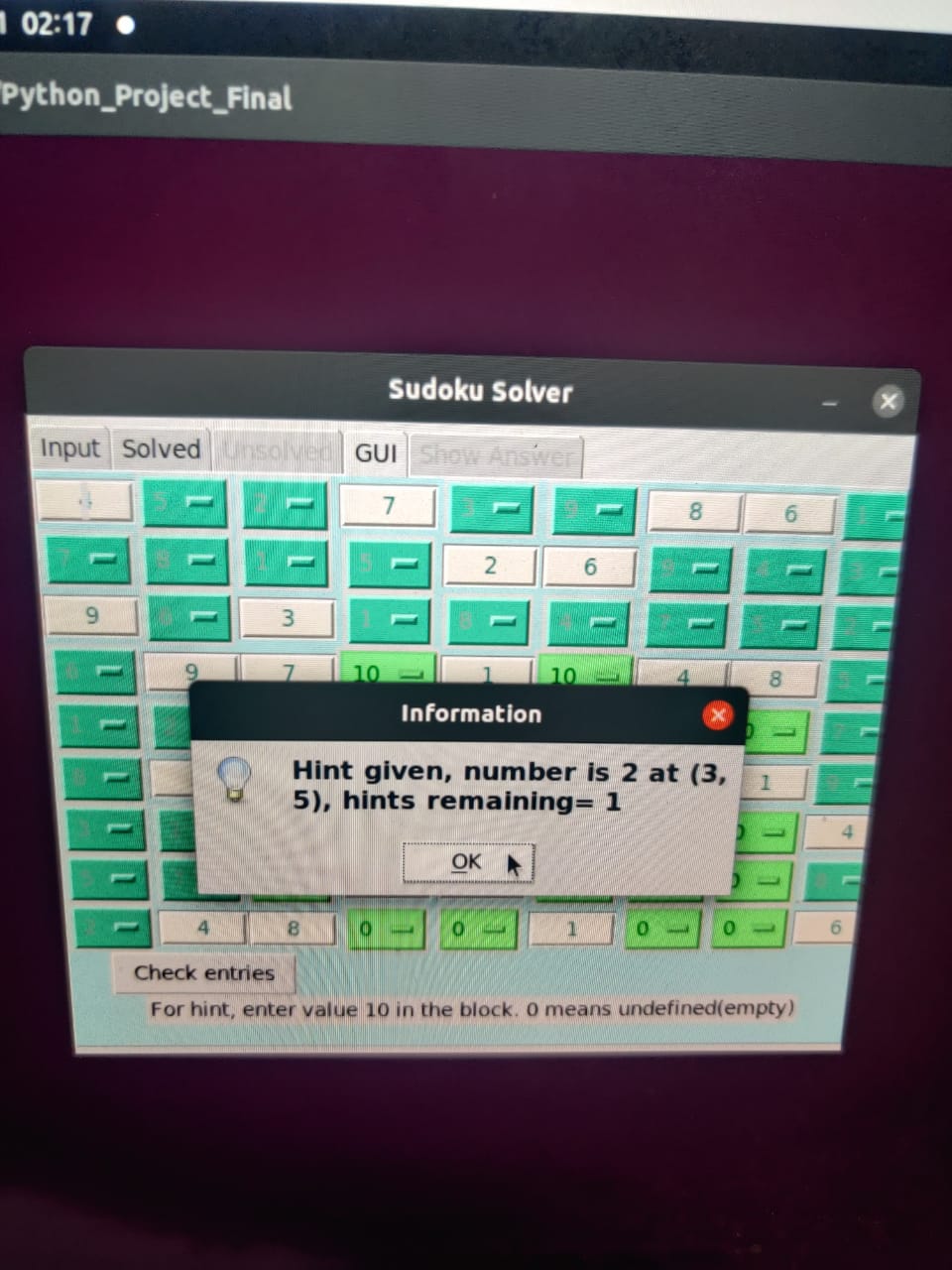
(Now GUI tab is opened, clicked on)



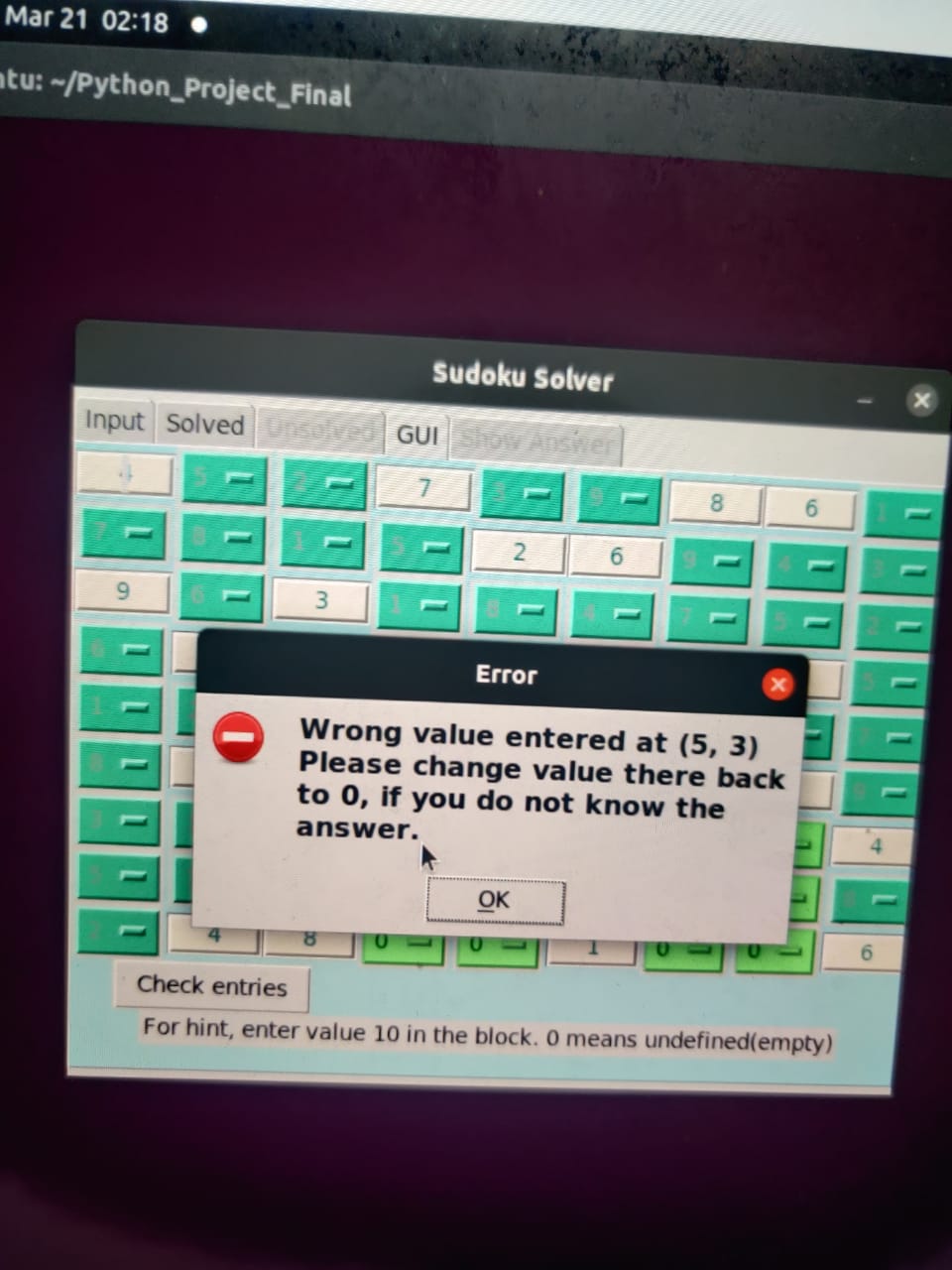
(This is how the GUI tab looks)



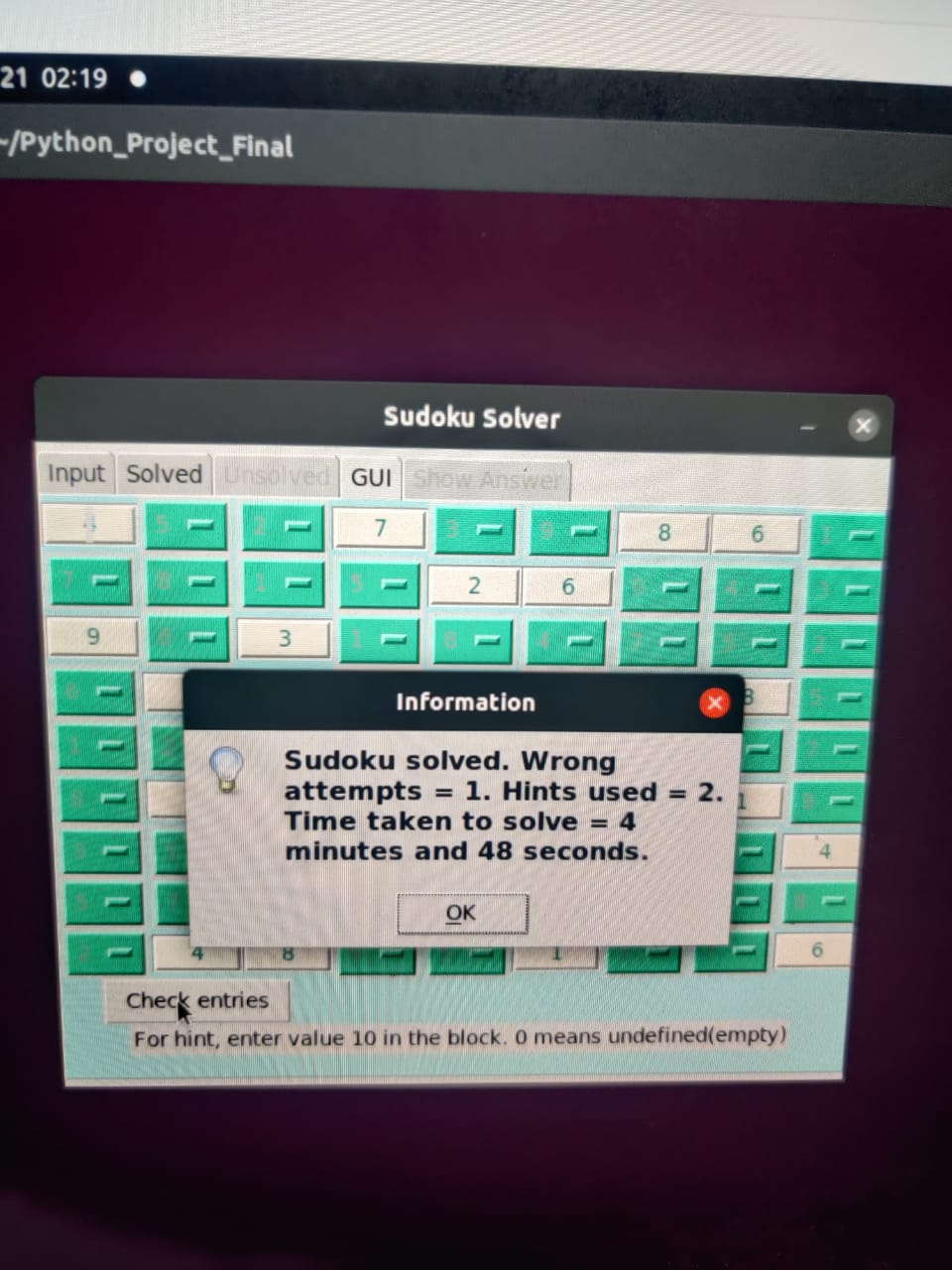
(Check entries button blocked the correct answers in dark green, which now can’t be changed. Also, this is how the hints are provided)

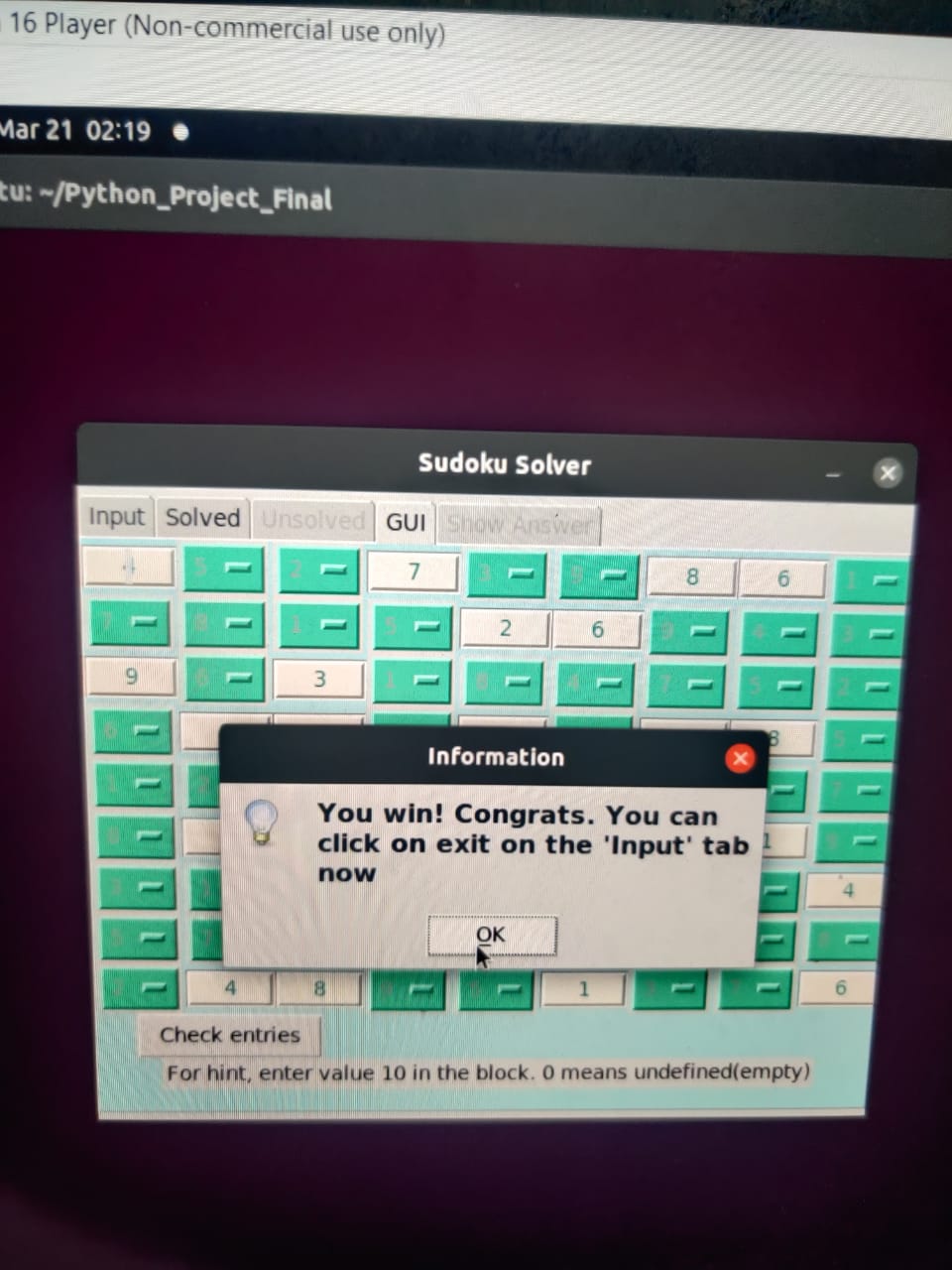


(Also, an example of multi entry check)



(Example of when a wrong value is entered)





(When the Sudoku is solved!!!!)

Please note - On the pictures, the contrast between the value inside the dark green box and the box itself SEEMS low, this is due to quality of pictures taken on phone, not indicative of the actual visibility on screen, where the contrast is much higher and distinguishable.

Future aspect: -

This program can definitely be implemented on a high scale application. To do it, further features which can be added is a web API, which can fetch Sudoku’s of different Sudoku’s online, and provide them to the user.

The input can be done via a scanner, and the user can send an image of the unsolved Sudoku, making input much faster and advanced.

GUI can have additional features, such as an option to find available numbers in a square, and have animation features too, so that it looks smooth flowing.

References: -

Not many, most were original ideas. The algorithm for solving was purely original.

GUI was inspired by the team’s thinking and the smoother flowing dropdown/comboboxes which tkinter provides.

For learning how to use tkinter libraries, Codemy’s youtube channel had AMPLE resources needed to master it (over 177 videos on tkinter exclusively). Taught us how to structure a big application.

Link: - <https://www.youtube.com/watch?v=yQSEXcf6s2I&list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV>

Small deviations from Project Plan: -

Module 1 was abolished. The GUI was maintained, but the concept of having a huge repository was abolished, only because it took away the essence of the program, having a repository of over 70 pre-solved Sudoku’s would unnecessarily make the program heavy in memory. This program, in this stage requires barely any memory. The problem was not in lack of time, or hardness of execution, but only because it took away the unique sophisticated simplicity of the project, which now behaves uni-functionally.

One GUI aspect was compromised on, because its implementation was impossible. Drop down boxes was used in our GUI frame, which only takes input from the user. In the project plan we wrote we can give temporary values, a maximum of 6 for the user’s convenience, but it wasn’t possible to implement that in a drop down box. It would require a structural change in the whole code.

Other than these 2 small deviations, we stuck to the plan, and added more features along the way to create a complete, round project.

Work Distributions:-

1 member, Abhinav Mahajan, working on the algorithm and concurrently,

4 members, Aamod BK, Yathin Kumar, Anwit Damale, Vamshidhar Reddy working on GUI aspect of the program.

90% of the work done by group calls and screen sharing, really reduced debugging time. (Lesser individual sequential work was done, and more simultaneous group work was done and this is why a Github repository was not made) One copy of code was made and circulated after the calls, and later the programs (Solving the Sudoku plus implementing the GUI) was integrated into one, with ease.

Thank you,

Respectful Sir, Professor Sujit Kumar

Dear TA, Rahul Murali Shankar