BUILDING AND VISUALIZING SUDOKU

A Real-Time Research Project Report

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING (AIML)

By

VALLAMALLA TRIVENI	**22VD1A6651**
CH SRIVATHSAVI	**22VD1A6645**
BANDARI THRISHA	**22VD1A6650**
M VARSHITHA	**22VD1A6654**

Under the guidance of

G. SRIDHAR

Assistant Professor

Department of Computer Science and Engineering.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AIML)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
UNIVERSITY COLLEGE OF ENGINEERING MANTHANI

Centenary Colony, Pannur (Vill), Ramagiri (Mdl), Peddapalli-505212, Telangana (India).

2023-2024

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

UNIVERSITY COLLEGE OF ENGINEERING MANTHANI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AIML)

DECLARATION BY THE CANDIDATE

We,

VALLAMALLA TRIVENI	**22VD1A6651**
CH SRIVATHSAVI	**22VD1A6645**
BANDARI THRISHA	**22VD1A6650**
MULUKA VARSHITHA	**22VD1A6654**

hereby declare that the project report entitled "**BUILDING AND VISUALIZING SUDOKU**" under the guidance of **Mr. G. Sridhar, Assistant Professor**, **Department of Computer Science and Engineering, JNTUH University College of Engineering Manthani** submitted in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering(AIML).

This is a record of bonafide work carried out by us and the results embodied in this project report have not been reproduced or copied from any source. The results embodied in this project have not been submitted to any other University or Institute for the award of any degree or diploma.

VALLAMALLA TRIVENI	**22VD1A6651**
CH SRIVATHSAVI	**22VD1A6645**
BANDARI THRISHA	**22VD1A6650**
MULUKA VARSHITHA	**22VD1A6654**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

UNIVERSITY COLLEGE OF ENGINEERING MANTHANI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AIML)

CERTIFICATE

This is to certify that the project report entitled “WEATHER FORECAST APPLICATION”being submitted by

VALLAMALLA TRIVENI	**22VD1A6651**
CH SRIVATHSAVI	**22VD1A6645**
BANDARI THRISHA	**22VD1A6650**
MULUKA VARSHITHA	**22VD1A6654**

in the partial fulfillment of the requirements for the award of the Degree of **BACHELOR OF TECHNOLOGY** in **Computer Science and Engineering(AIML)** to the **JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD UNIVERSITY COLLEGE OF ENGINEERING MANTHANI** is a record of bonfide work carried out by them during the academic year 2023-2024.

The results of investigation enclosed in this report have been verified and found satisfactory. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

SUPERVISER **HEAD OF THE DEPARTMENT**

ACKNOWLEDGMENT

We express our sincere gratitude to **Dr. Ch. Sridhar Reddy, Professor of Mechanical Engineering, Principal & Head of the department of Computer Science and Engineering(AIML), JNTUH University College of Engineering Manthani** for encouraging and giving permission to accomplish our project successfully.

We express our profound gratitude and thanks to our project guide **Mr. G. Sridhar, Assistant Professor, Department of Computer Science and Engineering, JNTUH University College of Engineering Manthani** for his constant help, personal supervision, expert guidance and consistent encouragement throughout this project which enabled us to complete our project successfully in time.

We also take this opportunity to thank other faculty members of CSE Department for their kind co-operation.

We wish to convey our thanks to one and all those who have extended their helping hands directly and indirectly in completion of our project.

VALLAMALLA TRIVENI	**22VD1A6651**
CH SRIVATHSAVI	**22VD1A6645**
BANDARI THRISHA	**22VD1A6650**
MULUKA VARSHITHA	**22VD1A6654**

ABSTRACT

This project involves the development of a Sudoku game using Python. Sudoku is a popular number puzzle game that consists of a 9x9 grid divided into nine 3x3 subgrids. The objective is to fill the grid so that each row, column, and subgrid contains all digits from 1 to 9, without repetition. The primary goal of this project is to implement a Sudoku puzzle generator that creates valid and solvable Sudoku grids. Develop a Sudoku solver that can efficiently solve any valid Sudoku puzzle using backtracking algorithm. The project demonstrates the application of algorithmic problem-solving technique in Python . It also highlights the use of Python's standard libraries to create a fully functional and enjoyable game. This project serves as an educational tool for understanding backtracking algorithms, and the logical structure of Sudoku puzzles.

Keywords: Puzzle Generator, Solver, Backtracking algorithm

TABLE OF CONTENTS

S.NO	TITLE	PAGE NO.
i	**TITLE PAGE**	**i**
ii	**DECLARATION**	**ii**
iii	**CERTIFICATE**	**iii**
iv	**ACKNOWLEDGEMENT**	**iv**
v	**ABSTRACT**	**v**
vi	**TABLE OF CONTENTS**	**vi,vii**
vii	**LIST OF FIGURES**	**viii**
1.	**INTRODUCTION**	**1**
	1.1 Sudoku	2
	1.2 Problem Statement	2
	1.3 Scope	2
	1.4 Objective	3
	1.5 Motive	3
2.	**LITERATURE SURVEY**	**4**
	2.1 what is python?	5
3.	**SYSTEM ANALYSIS**	**6**
	3.1 Existing system	7
	3.1.1 Limitations of the existing system	7-8
	3.2 Proposed System	8

	3.3 Feasibility Study	9
4.	**SYSTEM REQUIREMENT SPECIFICATION**	**10**
	4.1 Non-Functional Requirements	11-12
	4.2 Software Requirements	12
	4.3 Hardware Requirements	12
5.	**SYSTEM DESIGN**	**13**
	5.1 Low-Level Design	14
	5.2 Introduction to UML	14
	5.2.2 Activity Diagram	15
6.	**IMPLEMENTATION**	**16-18**
	6.1 Explanation of the algorithm	18-45
	6.2 Converting main.py into .exe file	46
7.	**TESTING**	**48**
	7.1 Test case	48
8.	**RESULTS**	**49**
	8.1 Home page	50
	8.1.1 If we loose game	50
	8.1.2 If we win game	51
10.	**CONCLUSION**	**53**
11.	**BIBLIOGRAPHY**	**55**

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
5.2.2.1	ACTIVITY Diagram	15
6.1.1	Installing Pygame	17
6.1.2	Files used	18
6.2.1	Installing py installer	46
6.2.2	Converting .py into .exe	46
8.1.1	Home Page	50
8.1.2	Loose game	50
8.1.3	Win game	51

INTRODUCTION

1. INTRODUCTION

Building and Visualizing Sudoku : A Python Perspective", explores the intersection of Python and Sudoku game. Focused on Pygame libraries and Convolutional Neural Networks (CNNs) , creating the game interface and visualization and recognizing the digits in the sudoku board.

1.1 SUDOKU

Sudoku Game had been created with the help of a module named "Pygame". Basically, in this, a random grid will be produced containing digits from 1 to 9 . To play this game: Each row, column, and square needs to be filled out with numbers 1-9, without repeating any numbers within the row, column, or square. Sudoku is quite a logical game. This game has different modes depending upon the size of the game. The standard size is 9x9. Moreover, there are 25 different types of Sudoku available having different rules according to them. The primary rule is that in each square of the nine boxes, there must be numbers between 1 to 9 and no number can repeat itself. Moreover, we should also consider that each line of 9 boxes horizontally and vertically should also contain 1 to 9 numbers, without any number repeating itself.

1.2 PROBLEM STATEMENT

Sudoku is a widely enjoyed logic-based number puzzle game. Our job is to place a number into every empty box so that each row across, each column down, and each small 9-box square within the large square (there are 9 of these) will contain each number from 1 through 9. Remember that no number may appear more than once in any row across, and column down, or within any small 9 box square. The numbers will be filled with help of pattern matching.

1.3 SCOPE

The purpose of the proposed project is to improve thinking skills. The game keeps track of all the records you set during the game and you can choose between easy and difficulty. You can create your own Sudoku and always go back to the previous steps to find a solution. It's a very difficult task to do manually. This is because it requires a lot of memory, and mathematical calculations. The game "Sudoku" improves the thinking and eyesight of the brain.

1.4 OBJECTIVE

The objective of Sudoku is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 subgrids contain all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which typically has a unique solution.Sudoku requires the player to use logical reasoning and deduction to figure out which number can go in each blank.

1.5 MOTIVE

The primary motives of creating and engaging with a Sudoku game encompass a variety of educational, cognitive, and recreational aspects. The motives behind Sudoku game are multifaceted, blending cognitive development, educational benefits, and recreational enjoyment. By combining logical problem-solving with an engaging user experience, Sudoku serves as both a stimulating mental exercise and a valuable educational resource.

Sudoku can be a calming and soothing activity, allowing players to focus on the puzzle and temporarily escape from the stresses of daily life.Sudoku has been used as a therapeutic tool for individuals with anxiety, depression, and ADHD, helping to reduce symptoms and improve mood. Sudoku presents a personal challenge, allowing players to push themselves, set goals, and track their progress. Sudoku is often referred to as a "brain game" because it helps build cognitive reserve, reducing the risk of age-related cognitive decline and dementia.

LITERATURE SURVEY

2.LITERATURE SURVEY

The study of implementing and solving Sudoku puzzles using Python involves exploring various aspects of computer science, algorithms, and software development. Below is a comprehensive literature study on the topic, focusing on key areas such as algorithm design, data structures, and graphical user interface development. Sudoku is a rich field of study, bridging recreational mathematics and serious academic research. Its simple rules yet complex solution space make it an enduring and widely studied puzzle.

The literature study of Sudoku by Python covers a broad spectrum of topics, from algorithm design and data structures and practical implementations. The resources highlighted provide a solid foundation for understanding the theoretical and practical aspects of developing a Sudoku solver and interactive application in Python. By using these materials, developers and researchers can gain insights into efficient problem-solving techniques and best practices for creating engaging and educational software.

2.1 What is Python?

Python is a high-level, interpreted programming language known for its readability, simplicity, and versatility. Created by Guido van Rossum and first released in 1991, Python emphasizes code readability and allows programmers to express concepts in fewer lines of code compared to languages such as C++ or Java.

Python is a versatile and powerful programming language that is easy to learn and use. Its extensive libraries, active community, and support for multiple programming paradigms make it suitable for a wide range of applications, from web development and data science to automation and artificial intelligence. Whether you are a beginner or an experienced programmer, Python offers tools and frameworks to help you achieve your programming goals efficiently.

Python has a rich ecosystem of libraries for game development, and one of the most popular among them is Pygame. Pygame is a set of Python modules designed for writing video games. It provides functionalities for creating games and multimedia applications. Pygame is a great starting point for Python game development due to its simplicity and extensive functionality

SYSTEM ANALYSIS

3.SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

For developing a Sudoku game, you can consider using various existing systems and platforms depending on your programming skills, preferences, and target platform (like web, mobile, desktop).

Printed Puzzle Books: Sudoku puzzles are widely available in printed puzzle books and magazines.

Mobile Apps: There are numerous Sudoku apps available on both iOS and Android platforms.

Web-based Platforms: Many websites offer Sudoku puzzles that can be played directly in a web browser.

Online Communities: There are online communities dedicated to Sudoku where players can discuss strategies, share puzzles they've created, and participate in tournaments or challenges.

3.1.1 LIMITATIONS OF EXISTING SYSTEM

- ➢ **Static Content:** The puzzles are fixed and cannot be updated or changed. Once you solve all the puzzles, the book is finished.
- ➢ **No Interactivity:** Printed puzzles lack interactive features such as hints, instant feedback, or the ability to check answers.
- ➢ **Portability and Durability:** While books are portable, they can be bulky to carry around. They can also wear out, tear, or get damaged over time.
- ➢ **Cost:** Buying multiple printed puzzle books can be more expensive than subscribing to a digital service with a vast collection of puzzles.
- ➢ **Battery Life:** Mobile apps need to be optimized to minimize battery consumption, which can limit their functionality and continuous usage.
- ➢ **Hardware Constraints:** Mobile devices have limited processing power, memory, and storage compared to computers, which can affect app performance and capabilities.
- ➢ **Update Cycles:** Users may not always update to the latest version of an app, leading to fragmentation and inconsistencies in the user experience.

- **Screen Size:** The smaller screen size of mobile devices can restrict the user interface design and limit the amount of information that can be displayed at one time.
- **Internet Dependency:** They require a reliable internet connection. Without it, users can't access the platform.
- **Performance Issues:** Web-based applications can be slower and less responsive compared to native applications, especially with complex tasks or large data sets.
- **Security Risks:** Web platforms are more vulnerable to cyberattacks, including hacking, phishing, and data breaches.
- **Scalability:** Handling a large number of simultaneous users can be more challenging on web platforms without proper infrastructure.
- **Information Overload:** The vast amount of information available can be overwhelming and difficult to manage or verify.
- **Privacy Concerns:** There is a risk of personal information being exposed or misused.
- **Lack of Personal Interaction:** Online communication lacks non-verbal cues like body language and tone, which can lead to misunderstandings.
- **Addiction and Time Management:** Spending excessive time online can lead to addiction and poor time management.

3.2 PROPOSED SYSTEM

Visualization and algorithmic libraries in Python with Pygame are essential tools for developing interactive applications, data analysis, and solving complex problems. Python is known for which simplicity and readability, making it easy to implement the logics and rules of sudoku. Pygame provides a straightforward way to handle 2D graphics, which is sufficient for rendering the sudoku grid and basic user interfaces. Python and Pygame applications can run on multiple platforms like Windows, macOS, Linux which ensured broad accesibilty for users. Python has a large and active community, with numerous resources, libraries and framework available.

3.3 FEASIBILITY STUDY

The project aims to develop an Sudoku game using deep Python libraries and frameworks. Developing a Sudoku solver and player application is feasible from technical, operational, economic, and scheduling perspectives. Python's ease of use and powerful libraries, combined with Pygame's capabilities for interactive graphics.

Five key considerations involved in the feasibility analysis are

i. **Economic Feasibility :** The primary expenses involve the time investment of developers. Using Python and Pygame, both free and open-source, minimizes software costs. These are minimal, especially if the application is distributed as a desktop or mobile app. Hosting costs are also low if using free or low-cost platforms

ii. **Technical Feasibility :** The combination of Python's ease of use and Pygame's capabilities for interactive graphics ensures an efficient development process and robust performance.Use of popular IDEs such as PyCharm or VSCode for efficient development.

iii. **Social Feasibility :** Sudoku's social feasibility lies in its universal appeal, cognitive benefits, potential for social interaction, stress-relieving qualities, accessibility, and educational value. These factors contribute to its widespread popularity and integration into various aspects of social and educational environments

iv. **Operational Feasibility :** The required personnel with expertise in data collection, preprocessing, and model maintenance are available within the project team or can be easily recruited. The implementation of the brain tumor classification system using the selected algorithms is practical and can be integrated.

v. **Legal and Ethical Feasibility :**Sudoku puzzles themselves are not protected by copyright, as they are considered logical or mathematical concepts. However, specific implementations, software code, and unique puzzle designs may be subject to copyright, trademark, or patent laws. Developers must ensure that they do not infringe on existing intellectual property when creating their own versions of Sudoku games.

SYSTEM REQUIREMENTS SPECIFICATION

4.SYSTEM REQUIREMENTS SPECIFICATION

Software Requirement Specification Software Requirements Specification plays an important role in creating quality software solutions. Specifications are a representation process. Requirements are represented in a manner that ultimately leads to successful software implementation. Requirements may be specified in a variety of ways. Representation format and content should be nested Diagrams and other notational forms should be restricted in number and consistent in use. Representations should be revisable.

4.1 NON-FUNCTIONAL REQUIREMENTS

✧ **Usability:**

Usability is the ease of use and learning ability of a human-made object. The object of use can be a software application, website, book, tool, machine, process, or anything a human interacts with. A usability study may be conducted as a primary job function by a usability analyst or as a secondary job function by designers, technical writers, marketing personnel, and others.

✧ **Reliability:**

The probability that a part, equipment, or system will satisfactorily perform its intended function under given circumstances, such as environmental conditions, limitations as to operating time, and frequency and thoroughness of maintenance for a specified period.

✧ **Performance:**

Accomplishment of a given task measured against present standards of accuracy, completeness, cost, and speed.

✧ **Supportability:**

The design characteristics of a stand or support system meet the operational requirements of an organization.

✧ **Implementation:**

Implementation is the realization of an application, or execution of a plan, idea, model, design, specification, standard, algorithm, or policy.

✧ **Interface:**

An interface refers to a point of interaction between components and is applicable at the level of both hardware and software. This allows a component whether a piece of hardware such as a graphics card or a piece of software such as an internet browser to function independently while using interfaces to communicate with other components via an input/output system and an associated protocol.

✧ **Legal:**

It is established by or founded upon law or official or accepted rules of or relating to jurisprudence; "legal loophole". Having legal efficacy or force", "a sound title to the property" Relating to or characteristic of the profession of law, "the legal profession". Allowed by official rules; "a legal pass receiver".

4.2 HARDWARE REQUIREMENTS

Processor: 11th Generation Intel®Core™i7.

Random Access Memory (RAM): 4GB or more.

Solid State Drive (SSD): 512GB

4.3 SOFTWARE REQUIREMENTS

Operating System: Windows 11.

Language: Python 3.11.14.

Web Browser: Latest version of Chrome.

Python libraries and frameworks-Pygame

Open CV

Convolutional Neutral Network(CNN)

PyInstaller

SYSTEM DESIGN

5.SYSTEM DESIGN

Systems Design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It involves translating user requirements into a detailed blueprint that guides the implementation phase. The goal is to create a well-organized and efficient structure that meets the intended purpose while considering factors like scalability, maintainability, and performance.

Mastering Systems Design is crucial for anyone looking to build robust and scalable systems. Our comprehensive Systems Design course provides you with the knowledge and skills to excel in this area. Through practical examples and expert insights, you'll learn how to effectively translate user requirements into detailed designs that can be successfully implemented.

In any development process, be it Software or any other tech, the most important stage is Design. Without the designing phase, you cannot jump to the implementation or the testing part. The same is the case with the System as well.

Systems Design not only is a vital step in the development of the system but also provides the backbone to handle exceptional scenarios because it represents the business logic of software.

5.2 LOW-LEVEL DESIGN

5.2.1 INTRODUCTION TO UML

Unified Modelling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying. visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. UML is a very important part of developing object- oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

Unified Modelling Language is a general-purpose modeling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. We use UML. diagrams to portray the behavior and structure of a system. UML. helps software engineers, businessmen, and system architects with modeling, design, and analysis. The project includes the following UML. diagrams:

- Activity Diagram

5.2.2 ACTIVITY DIAGRAM

An activity diagram shown in Fig-5.2.2.1 is defined as a UML diagram that focuses on the execution and flow of the behavior of a system instead of implementation. It is also called an object-oriented flowchart. Activity diagrams consist of activities that are made up of actions that apply to behavioral modeling technology.

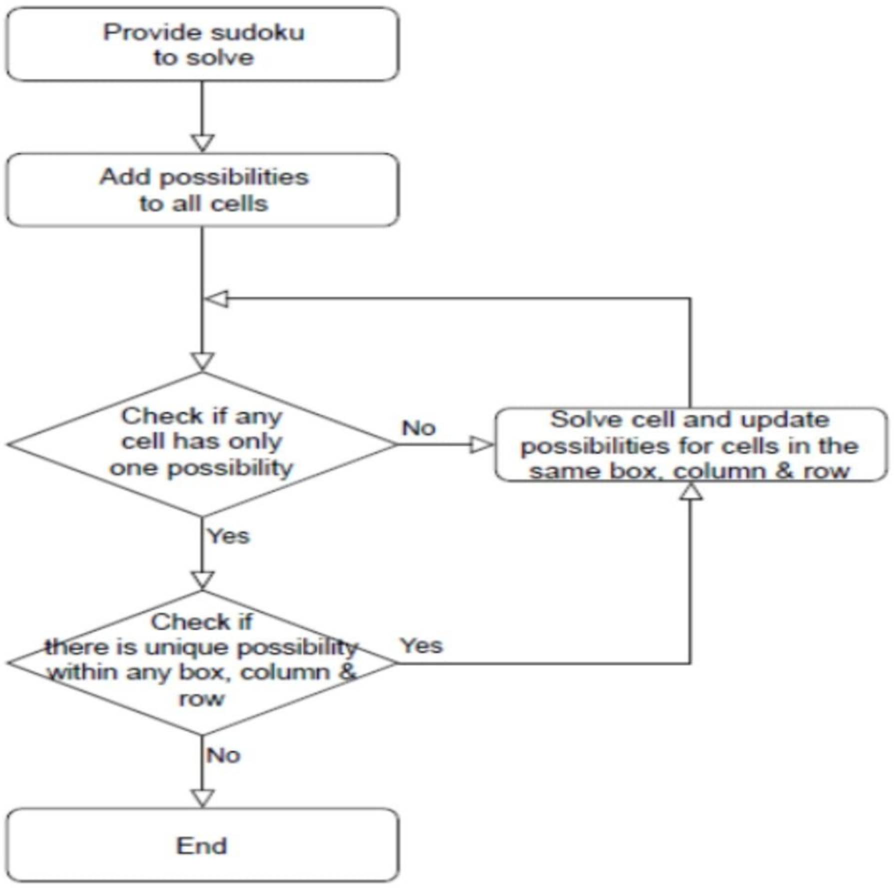

Provide sudoku to solve
Add possibilities to all cells
Check if any cell has only one possibility
No
Solve cell and update possibilities for cells in the same box, column & row
Yes
Check if there is unique possibility within any box, column & row
Yes
No
End

Fig-5.2.2.1:Activity Diagram

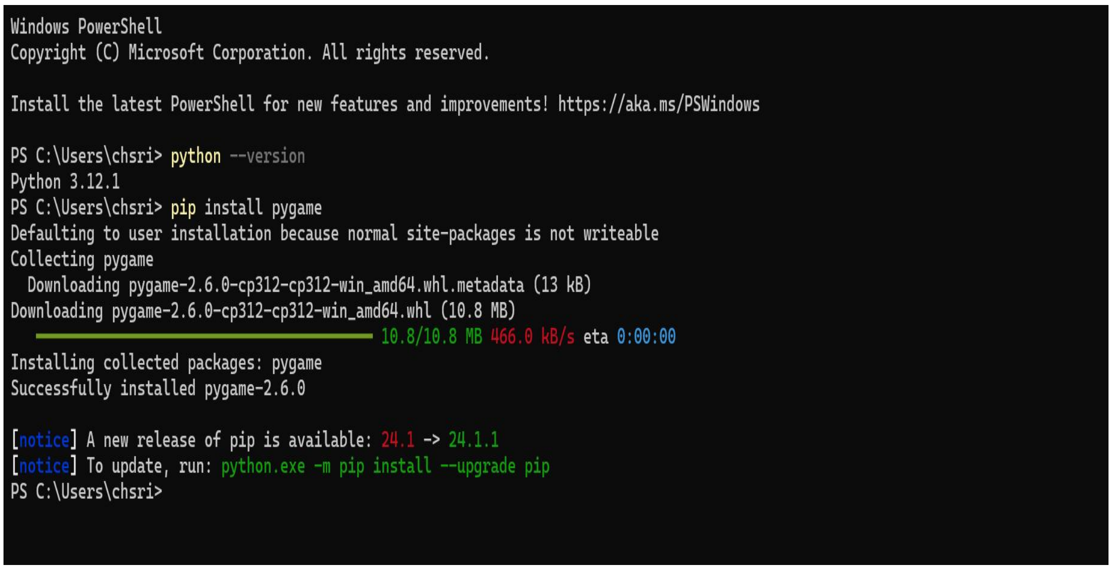
IMPLEMENTATION

6.IMPLEMENTION

Implementing a Sudoku game involves several components, including board representation, input validation, user interface, and a solving algorithm. Below is an overview of the theory behind each component and how they come together to form the game,

Step 1:Installation and Setup

Let's start by making sure Pygame is installed on the computer; head to the terminal and install pygame module using pip.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\chsri> python --version
Python 3.12.1
PS C:\Users\chsri> pip install pygame
Defaulting to user installation because normal site-packages is not writeable
Collecting pygame
  Downloading pygame-2.6.0-cp312-cp312-win_amd64.whl.metadata (13 kB)
Downloading pygame-2.6.0-cp312-cp312-win_amd64.whl (10.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.8/10.8 MB 466.0 kB/s eta 0:00:00
Installing collected packages: pygame
Successfully installed pygame-2.6.0

[notice] A new release of pip is available: 24.1 -> 24.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\chsri>
```

Step 2 : Board Representation

The Sudoku board is a 9x9 grid, divided into nine 3x3 sub-grids. Each cell in the grid can contain a number from 1 to 9 or be empty (typically represented by 0).

Step 3 :Input Validation

For a Sudoku game, it’s crucial to validate the player's input to ensure that the numbers are placed according to the rules of Sudoku.

Rules:

Each number from 1 to 9 must appear exactly once in each row.

Each number from 1 to 9 must appear exactly once in each column.

Each number from 1 to 9 must appear exactly once in each 3x3 sub-grid.

Step 4 : User Interface

The user interface allows players to interact with the game. This can be text-based or graphical. In a simple implementation, a text-based interface is often used where players input their moves via the console. It displays the board, Accepting and validating user input and providing feedback and game status updates.

Step 5 : Solving Algorithm

A Sudoku solver is useful for generating puzzles and checking the solution. The most common algorithm used is backtracking. Find an empty cell. Try placing numbers 1 to 9 in the empty cell. Check if placing a number violates Sudoku rules. If placing a number is valid, recursively attempt to solve the rest of the board. If no number can be placed without violation, backtrack and try the next number. Continue until the board is completely filled or determined unsolvable.

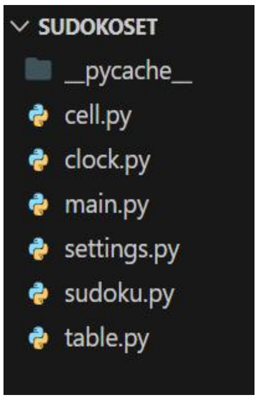
Step 6 : Game Logic

Initializing the board with a valid puzzle. Allowing the player to make moves. Checking for game completion. Optionally providing hints or auto-solving the puzzle.

6.1 EXPLANATION OF THE ALGORITHM

Let's start by making sure Pygame is installed on the computer; head to the terminal and install pygame module using pip.

After that, create a directory for the game and create the following .py file inside it; settings.py, main.py, sudoku.py, cell.py, table.py, and clock.py.



Let's define the game variables and useful external functions in settings.py:

```
# setting.py

from itertools import islice



WIDTH, HEIGHT = 450, 450

N_CELLS = 9

CELL_SIZE = (WIDTH // N_CELLS, HEIGHT // N_CELLS)



# Convert 1D list to 2D list

def convert_list(lst, var_lst):

    it = iter(lst)

    return [list(islice(it, i)) for i in var_lst]
```

Next, let's create the main class of our game. This class will be responsible for calling the game and running the game loop:

```
# main.py

import pygame, sys

from settings import WIDTH, HEIGHT, CELL_SIZE
from table import Table


pygame.init()


screen = pygame.display.set_mode((WIDTH, HEIGHT + (CELL_SIZE[1] * 3)))

pygame.display.set_caption("Sudoku")


pygame.font.init()


class Main:

    def __init__(self, screen):

        self.screen = screen

        self.FPS = pygame.time.Clock()

        self.lives_font = pygame.font.SysFont("monospace", CELL_SIZE[0] // 2)

        self.message_font = pygame.font.SysFont('Bauhaus 93', (CELL_SIZE[0]))

        self.color = pygame.Color("darkgreen")


    def main(self):

        table = Table(self.screen)

        while True:
            self.screen.fill("gray")

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    pygame.quit()

                    sys.exit()

                if event.type == pygame.MOUSEBUTTONDOWN:

                    if not table.game_over:

                        table.handle_mouse_click(event.pos)

            # lower screen display

            if not table.game_over:

                my_lives = self.lives_font.render(f"Lives Left: {table.lives}", True,
pygame.Color("black"))

                self.screen.blit(my_lives, ((WIDTH // table.SRN) - (CELL_SIZE[0] //
2), HEIGHT + (CELL_SIZE[1] * 2.2)))

            else:

                if table.lives <= 0:

                    message = self.message_font.render("GAME OVER!!", True,
pygame.Color("red"))

                    self.screen.blit(message, (CELL_SIZE[0] + (CELL_SIZE[0] // 2),
HEIGHT + (CELL_SIZE[1] * 2)))

                elif table.lives > 0:

                    message = self.message_font.render("You Made It!!!", True,
self.color)

                    self.screen.blit(message, (CELL_SIZE[0] , HEIGHT +
(CELL_SIZE[1] * 2)))

            table.update()

            pygame.display.flip()

            self.FPS.tick(30)



if __name__ == "__main__":

    play = Main(screen)

    play.main()
```

From the name itself, the Main class will be the main class of our game. It takes an argument of screen which will serve as the game window, for animating the game.

The main() function will run and update our game. It will initialize the Table first (serves as our puzzle table). To keep the game running without intentionally exiting, we put
a while loop inside it. Inside our loop, we place another loop (the for loop), which will catch all the events going on inside our game window, events such as key click, mouse movement, mouse button click or when the player hits the exit button.

The main() is also responsible for displaying players' "lives left" and game-over messages, whether the player wins or loses. To update the game, we call the table.update() to update the changes in our game table. Then, to render the changes, the pygame.display.flip() does the job done. The self.FPS.tick(30) controls the framerate update speed.

Generating Sudoku Puzzle

The Sudoku() class will be responsible for generating a random Sudoku puzzle for us. In sudoku.py, create a class and name it Sudoku. Let's begin by importing the necessary modules: random, math, and copy:

```
# sudoku.py

import random

import math

import copy



class Sudoku:

    def __init__(self, N, E):

        self.N = N

        self.E = E

        # compute square root of N

        self.SRN = int(math.sqrt(N))

        self.table = [[0 for x in range(N)] for y in range(N)]

        self.answerable_table = None

        self._generate_table()



    def _generate_table(self):
```

```
    # fill the subgroups diagonally table/matrices

    self.fill_diagonal()

    # fill remaining empty subgroups

    self.fill_remaining(0, self.SRN)

    # Remove random Key digits to make game

    self.remove_digits()
```

The class has an initializer (__init__()) method that takes two parameters N and E, representing the size of the Sudoku grid and the number of cells to be removed to create a puzzle. The class attributes include N (grid size), E (number of cells to remove), SRN (square root of N), table (Sudoku grid), and answerable_table (a copy of the grid with some cells removed). The _generate_table() method is immediately called upon object creation to set up the Sudoku puzzle.

Primary number filling:

```
def fill_diagonal(self):

    for x in range(0, self.N, self.SRN):

        self.fill_cell(x, x)



def not_in_subgroup(self, rowstart, colstart, num):

    for x in range(self.SRN):

        for y in range(self.SRN):

            if self.table[rowstart + x][colstart + y] == num:
```

```
            return False

    return True



def fill_cell(self, row, col):

    num = 0

    for x in range(self.SRN):

        for y in range(self.SRN):

            while True:

                num = self.random_generator(self.N)

                if self.not_in_subgroup(row, col, num):

                    break

            self.table[row + x][col + y] = num
```

The fill_diagonal() method fills subgroups diagonally by calling the fill_cell() method for each subgroup. The fill_cell() method generates and places a unique number in each subgroup cell.

```
def random_generator(self, num):

    return math.floor(random.random() * num + 1)



def safe_position(self, row, col, num):

    return (self.not_in_row(row, num) and self.not_in_col(col, num) and
self.not_in_subgroup(row - row % self.SRN, col - col % self.SRN, num))



  def not_in_row(self, row, num):

    for col in range(self.N):

      if self.table[row][col] == num:

        return False

    return True



  def not_in_col(self, col, num):

    for row in range(self.N):

      if self.table[row][col] == num:

        return False

    return True



  def fill_remaining(self, row, col):

    # check if we have reached the end of the matrix

    if row == self.N - 1 and col == self.N:

      return True

    # move to the next row if we have reached the end of the current row

    if col == self.N:
            row += 1

            col = 0

        # skip cells that are already filled

        if self.table[row][col] != 0:

            return self.fill_remaining(row, col + 1)

        # try filling the current cell with a valid value

        for num in range(1, self.N + 1):

            if self.safe_position(row, col, num):

                self.table[row][col] = num

                if self.fill_remaining(row, col + 1):

                    return True

                self.table[row][col] = 0

        # no valid value was found, so backtrack

        return False
```

Several helper methods (random_generator(), safe_position(), not_in_row(), not_in_col(), and not_in_subgroup()) are defined. These methods assist in generating random numbers, checking if a position is safe to place a number, and ensuring that a number is not already present in a row, column, or subgroup.

```
    def remove_digits(self):

        count = self.E
        # replicates the table so we can have a filled and pre-filled copy

        self.answerable_table = copy.deepcopy(self.table)

        # removing random numbers to create the puzzle sheet

        while (count != 0):

            row = self.random_generator(self.N) - 1

            col = self.random_generator(self.N) - 1

            if (self.answerable_table[row][col] != 0):

                count -= 1

                self.answerable_table[row][col] = 0
```

The remove_digits() method removes a specified number of random digits from the filled grid to create the puzzle. It also creates a copy of the grid (answerable_table) before removing digits.

```
def puzzle_table(self):

    return self.answerable_table



def puzzle_answers(self):

    return self.table



def print_sudoku(self):

    for row in range(self.N):
            for col in range(self.N):

                print(self.table[row][col], end=" ")

            print()

        print("")

        for row in range(self.N):

            for col in range(self.N):

                print(self.answerable_table[row][col], end=" ")

            print()



if __name__ == "__main__":

    N = 9

    E = (N * N) // 2

    sudoku = Sudoku(N, E)

    sudoku.print_sudoku()
```

The last 3 methods are responsible for returning and printing the puzzle and/or answers. The puzzle_table() returns the answerable table (puzzle with some cells removed). The puzzle_answers() returns the complete Sudoku table. The print_sudoku() prints both the complete Sudoku grid and the answerable grid.

Creating the Game Table

Before making the game grid, let's create our table cells. In cell.py, make the function Cell():

```
# cell.py

import pygame

from settings import convert_list



pygame.font.init()



class Cell:

    def __init__(self, row, col, cell_size, value, is_correct_guess = None):

        self.row = row

        self.col = col

        self.cell_size = cell_size

        self.width = self.cell_size[0]

        self.height = self.cell_size[1]

        self.abs_x = row * self.width

        self.abs_y = col * self.height

        self.value = value

        self.is_correct_guess = is_correct_guess

        self.guesses = None if self.value != 0 else [0 for x in range(9)]

        self.color = pygame.Color("white")

        self.font = pygame.font.SysFont('monospace', self.cell_size[0])
        self.g_font = pygame.font.SysFont('monospace', (cell_size[0] // 3))

        self.rect = pygame.Rect(self.abs_x,self.abs_y,self.width,self.height)



    def update(self, screen, SRN = None):

        pygame.draw.rect(screen, self.color, self.rect)

        if self.value != 0:

            font_color = pygame.Color("black") if self.is_correct_guess else
pygame.Color("red")

            num_val = self.font.render(str(self.value), True, font_color)

            screen.blit(num_val, (self.abs_x, self.abs_y))

        elif self.value == 0 and self.guesses != None:

            cv_list = convert_list(self.guesses, [SRN, SRN, SRN])

            for y in range(SRN):

                for x in range(SRN):

                    num_txt = " "

                    if cv_list[y][x] != 0:

                        num_txt = cv_list[y][x]

                    num_txt = self.g_font.render(str(num_txt), True,
pygame.Color("orange"))

                    abs_x = (self.abs_x + ((self.width // SRN) * x))
                    abs_y = (self.abs_y + ((self.height // SRN) * y))

                    abs_pos = (abs_x, abs_y)

                    screen.blit(num_txt, abs_pos)
```

The Cell() class has attributes such as row and col (cell position in the table), cell_size, width and height, abs_x and abs_y (absolute x and y coordinates of the cell on the screen), value (numerical value, 0 for an empty cell), is_correct_guess (indicating whether the current value is a correct guess), and guesses (list representing possible guesses for an empty cell or None if the cell is filled)

The update() method is responsible for updating the graphical representation of the cell on the screen. It draws a rectangle with the specified color using pygame.draw.rect. Depending on whether the cell is filled (value != 0) or empty (value == 0), it either draws the value in a filled cell or the possible guesses in an empty cell.

If the cell is empty and has possible guesses, it converts the guess list into a 2D list using the convert_list() function. It then iterates through the converted list and draws each guess in the corresponding position within the cell. It renders each guess as text using the small font (g_font). It calculates the absolute position within the cell for each guess based on the position within the 2D list. Then, blits (draws) the text onto the screen at the calculated position.

Now, let's move on to creating the game table. Create a class and name it Table in table.py. It uses the Pygame library to create the Sudoku grid, handle user inputs, and display the puzzle, number choices, buttons, and timer.

```
import pygame

import math

from cell import Cell

from sudoku import Sudoku
from clock import Clock



from settings import WIDTH, HEIGHT, N_CELLS, CELL_SIZE



pygame.font.init()



class Table:

    def __init__(self, screen):

        self.screen = screen

        self.puzzle = Sudoku(N_CELLS, (N_CELLS * N_CELLS) // 2)

        self.clock = Clock()

        self.answers = self.puzzle.puzzle_answers()

        self.answerable_table = self.puzzle.puzzle_table()

        self.SRN = self.puzzle.SRN

        self.table_cells = []

        self.num_choices = []

        self.clicked_cell = None

        self.clicked_num_below = None

        self.cell_to_empty = None

        self.making_move = False
        self.guess_mode = True

        self.lives = 3

        self.game_over = False

        self.delete_button = pygame.Rect(0, (HEIGHT + CELL_SIZE[1]),
(CELL_SIZE[0] * 3), (CELL_SIZE[1]))

        self.guess_button = pygame.Rect((CELL_SIZE[0] * 6), (HEIGHT +
CELL_SIZE[1]), (CELL_SIZE[0] * 3), (CELL_SIZE[1]))

        self.font = pygame.font.SysFont('Bauhaus 93', (CELL_SIZE[0] // 2))

        self.font_color = pygame.Color("white")

        self._generate_game()

        self.clock.start_timer()



    def _generate_game(self):

        # generating sudoku table

        for y in range(N_CELLS):

            for x in range(N_CELLS):

                cell_value = self.answerable_table[y][x]

                is_correct_guess = True if cell_value != 0 else False

                self.table_cells.append(Cell(x, y, CELL_SIZE, cell_value,
is_correct_guess))

        # generating number choices
        for x in range(N_CELLS):

            self.num_choices.append(Cell(x, N_CELLS, CELL_SIZE, x + 1))
```

The Table class' __init__() method (constructor) initializes various attributes such as the Pygame screen, the Sudoku puzzle, the clock, answers, the answerable table, and other game-related variables.

```
    def _draw_grid(self):

        grid_color = (50, 80, 80)

        pygame.draw.rect(self.screen, grid_color, (-3, -3, WIDTH + 6, HEIGHT +
6), 6)

        i = 1

        while (i * CELL_SIZE[0]) < WIDTH:

            line_size = 2 if i % 3 > 0 else 4

            pygame.draw.line(self.screen, grid_color, ((i * CELL_SIZE[0]) -
(line_size // 2), 0), ((i * CELL_SIZE[0]) - (line_size // 2), HEIGHT), line_size)

            pygame.draw.line(self.screen, grid_color, (0, (i * CELL_SIZE[0]) -
(line_size // 2)), (HEIGHT, (i * CELL_SIZE[0]) - (line_size // 2)), line_size)

            i += 1



    def _draw_buttons(self):
        # adding delete button details

        dl_button_color = pygame.Color("red")

        pygame.draw.rect(self.screen, dl_button_color, self.delete_button)

        del_msg = self.font.render("Delete", True, self.font_color)

        self.screen.blit(del_msg, (self.delete_button.x + (CELL_SIZE[0] // 2),
self.delete_button.y + (CELL_SIZE[1] // 4)))

        # adding guess button details

        gss_button_color = pygame.Color("blue") if self.guess_mode else
pygame.Color("purple")

        pygame.draw.rect(self.screen, gss_button_color, self.guess_button)

        gss_msg = self.font.render("Guess: On" if self.guess_mode else "Guess:
Off", True, self.font_color)

        self.screen.blit(gss_msg, (self.guess_button.x + (CELL_SIZE[0] // 3),
self.guess_button.y + (CELL_SIZE[1] // 4)))
```

The _draw_grid() method is responsible for drawing the Sudoku grid; it uses Pygame functions to draw the grid lines based on the size of the cells. The _draw_buttons() method is responsible for drawing the delete and guess buttons; it uses Pygame functions to draw rectangular buttons with appropriate colors and messages.

```
    def _get_cell_from_pos(self, pos):

        for cell in self.table_cells:

            if (cell.row, cell.col) == (pos[0], pos[1]):

                return cell
```

The _get_cell_from_pos() method returns the Cell object at a given position (row, col) in the Sudoku table.

```
# checking rows, cols, and subgroups for adding guesses on each cell

def _not_in_row(self, row, num):

    for cell in self.table_cells:

        if cell.row == row:

            if cell.value == num:

                return False

    return True



def _not_in_col(self, col, num):

    for cell in self.table_cells:

        if cell.col == col:

            if cell.value == num:

                return False

    return True



def _not_in_subgroup(self, rowstart, colstart, num):

    for x in range(self.SRN):
        for y in range(self.SRN):

            current_cell = self._get_cell_from_pos((rowstart + x, colstart + y))

            if current_cell.value == num:

                return False

    return True



# remove numbers in guess if number already guessed in the same row, col,
subgroup correctly

def _remove_guessed_num(self, row, col, rowstart, colstart, num):

    for cell in self.table_cells:

        if cell.row == row and cell.guesses != None:

            for x_idx,guess_row_val in enumerate(cell.guesses):

                if guess_row_val == num:

                    cell.guesses[x_idx] = 0

        if cell.col == col and cell.guesses != None:

            for y_idx,guess_col_val in enumerate(cell.guesses):

                if guess_col_val == num:

                    cell.guesses[y_idx] = 0

    for x in range(self.SRN):

        for y in range(self.SRN):
```

```
            current_cell = self._get_cell_from_pos((rowstart + x, colstart + y))

            if current_cell.guesses != None:

                for idx,guess_val in enumerate(current_cell.guesses):

                    if guess_val == num:

                        current_cell.guesses[idx] = 0
```

The methods _not_in_row(), _not_in_col(), _not_in_subgroup(), and _remove_guessed_num() are responsible for checking whether a number is valid in a row, column, or subgroup and removing guessed numbers when correctly placed.

```
    def handle_mouse_click(self, pos):

        x, y = pos[0], pos[1]

        # getting table cell clicked

        if x <= WIDTH and y <= HEIGHT:

            x = x // CELL_SIZE[0]

            y = y // CELL_SIZE[1]

            clicked_cell = self._get_cell_from_pos((x, y))

            # if clicked empty cell

            if clicked_cell.value == 0:

                self.clicked_cell = clicked_cell

                self.making_move = True

            # clicked unempty cell but with wrong number guess
            elif clicked_cell.value != 0 and clicked_cell.value != self.answers[y][x]:

                self.cell_to_empty = clicked_cell

        # getting number selected

        elif x <= WIDTH and y >= HEIGHT and y <= (HEIGHT +
CELL_SIZE[1]):

            x = x // CELL_SIZE[0]

            self.clicked_num_below = self.num_choices[x].value

        # deleting numbers

        elif x <= (CELL_SIZE[0] * 3) and y >= (HEIGHT + CELL_SIZE[1]) and
y <= (HEIGHT + CELL_SIZE[1] * 2):

            if self.cell_to_empty:

                self.cell_to_empty.value = 0

                self.cell_to_empty = None

        # selecting modes

        elif x >= (CELL_SIZE[0] * 6) and y >= (HEIGHT + CELL_SIZE[1]) and
y <= (HEIGHT + CELL_SIZE[1] * 2):

            self.guess_mode = True if not self.guess_mode else False

        # if making a move

        if self.clicked_num_below and self.clicked_cell != None and
self.clicked_cell.value == 0:

            current_row = self.clicked_cell.row
            current_col = self.clicked_cell.col

            rowstart = self.clicked_cell.row - self.clicked_cell.row % self.SRN

            colstart = self.clicked_cell.col - self.clicked_cell.col % self.SRN

            if self.guess_mode:

                # checking the vertical group, the horizontal group, and the subgroup

                if self._not_in_row(current_row, self.clicked_num_below) and
self._not_in_col(current_col, self.clicked_num_below):

                if self._not_in_subgroup(rowstart, colstart,
self.clicked_num_below):

                    if self.clicked_cell.guesses != None:

                        self.clicked_cell.guesses[self.clicked_num_below - 1] =
self.clicked_num_below

            else:

                self.clicked_cell.value = self.clicked_num_below

                # if the player guess correctly

                if self.clicked_num_below ==
self.answers[self.clicked_cell.col][self.clicked_cell.row]:

                self.clicked_cell.is_correct_guess = True

                self.clicked_cell.guesses = None

                self._remove_guessed_num(current_row, current_col, rowstart,
colstart, self.clicked_num_below)

                # if guess is wrong
                else:

                self.clicked_cell.is_correct_guess = False

                self.clicked_cell.guesses = [0 for x in range(9)]

                self.lives -= 1

            self.clicked_num_below = None

            self.making_move = False

        else:

            self.clicked_num_below = None
```

The handle_mouse_click() method processes mouse clicks based on the position on the screen. It updates game variables like clicked_cell, clicked_num_below, and cell_to_empty accordingly.

```
def _puzzle_solved(self):

    check = None

    for cell in self.table_cells:

        if cell.value == self.answers[cell.col][cell.row]:

            check = True

        else:

            check = False

            break

    return check
```

The _puzzle_solved() method checks if the Sudoku puzzle is solved by comparing the values in each cell with the correct answers.

```
    def update(self):

        [cell.update(self.screen, self.SRN) for cell in self.table_cells]

        [num.update(self.screen) for num in self.num_choices]

        self._draw_grid()

        self._draw_buttons()

        if self._puzzle_solved() or self.lives == 0:

            self.clock.stop_timer()

            self.game_over = True

        else:

            self.clock.update_timer()

        self.screen.blit(self.clock.display_timer(), (WIDTH // self.SRN,HEIGHT
+ CELL_SIZE[1]))
```

The update method is responsible for updating the display. It updates the graphical representation of cells and numbers, draws the grid and buttons, checks if the puzzle is solved or the game is over, and updates the timer.

Adding a Game Timer

And for the last part of our code, we're making a class for timer. Create Clock class in clock.py:

```
import pygame, time
```

```
from settings import CELL_SIZE


pygame.font.init()


class Clock:
    def __init__(self):
        self.start_time = None
        self.elapsed_time = 0
        self.font = pygame.font.SysFont("monospace", CELL_SIZE[0])
        self.message_color = pygame.Color("black")

    # Start the timer
    def start_timer(self):
        self.start_time = time.time()

    # Update the timer
    def update_timer(self):
        if self.start_time is not None:
            self.elapsed_time = time.time() - self.start_time
    # Display the timer

    def display_timer(self):

        secs = int(self.elapsed_time % 60)

        mins = int(self.elapsed_time / 60)

        my_time = self.font.render(f"{mins:02}:{secs:02}", True,
self.message_color)

        return my_time



    # Stop the timer

    def stop_timer(self):

        self.start_time = None
```

The start_timer() method sets the start_time attribute to the current time using time.time() when called. This marks the beginning of the timer.

The update_timer() method calculates the elapsed time since the timer started. If the start_time is not None, it updates the elapsed_time by subtracting the current time from the start_time.

The display_timer() method converts the elapsed time into minutes and seconds. It then creates a text representation of the time in the format "MM:SS" using the Pygame font. The rendered text is returned.

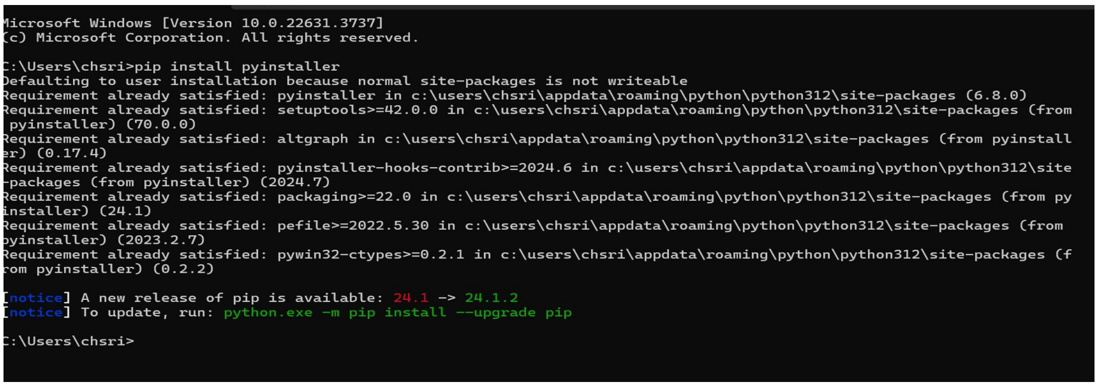
The stop_timer() method resets the start_time to None, effectively stopping the timer.

And now, we are done coding!! To try our game, simply run python main.py or python3 main.py on your terminal once you're inside our project directory.

6.1.1 Converting main.py into .exe file

Step 1: Install PyInstaller

Open a command prompt or terminal and run the following command to install PyInstaller:



```
icrosoft Windows [Version 10.0.22631.3737]
(c) Microsoft Corporation. All rights reserved.

:\Users\chsri>pip install pyinstaller
efaulting to user installation because normal site-packages is not writeable
equirement already satisfied: pyinstaller in c:\users\chsri\appdata\roaming\python\python312\site-packages (6.8.0)
equirement already satisfied: setuptools>=42.0.0 in c:\users\chsri\appdata\roaming\python\python312\site-packages (from
 pyinstaller) (70.0.0)
equirement already satisfied: altgraph in c:\users\chsri\appdata\roaming\python\python312\site-packages (from pyinstall
r) (0.17.4)
equirement already satisfied: pyinstaller-hooks-contrib>=2024.6 in c:\users\chsri\appdata\roaming\python\python312\site
-packages (from pyinstaller) (2024.7)
equirement already satisfied: packaging>=22.0 in c:\users\chsri\appdata\roaming\python\python312\site-packages (from py
installer) (24.1)
equirement already satisfied: pefile>=2022.5.30 in c:\users\chsri\appdata\roaming\python\python312\site-packages (from
yinstaller) (2023.2.7)
equirement already satisfied: pywin32-ctypes>=0.2.1 in c:\users\chsri\appdata\roaming\python\python312\site-packages (f
rom pyinstaller) (0.2.2)

[notice] A new release of pip is available: 24.1 -> 24.1.2
[notice] To update, run: python.exe -m pip install --upgrade pip

:\Users\chsri>
```

Fig 6.1.1.1.Installing pyinstaller

Step 2: Navigate to your script's directory

Use the `cd` command to navigate to the directory where your Python script is located.

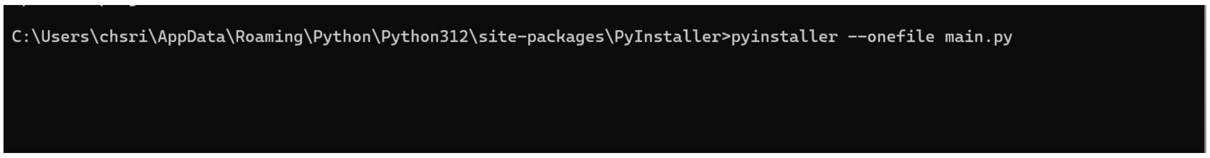
C:\Users\chsri\AppData\Roaming\Python\Python312\site-packages\PyInstaller

Step 3: Run PyInstaller

Run PyInstaller with the following command:

pyinstaller --onefile main.py

The `–onefile` flag indicates that you want a single executable file instead of a bunch of files.



```
C:\Users\chsri\AppData\Roaming\Python\Python312\site-packages\PyInstaller>pyinstaller --onefile main.py
```

Fig 6.1.1.2.converting .py into .exe

Finally .py file is converted into .exe which is flexible to use anywhere.

TESTING

7.TESTING

Testing is the debugging program is one of the most critical aspects of the computer programming triggers, without programming that works, the system would never produce an output of which it was designed. Testing is best performed when user development is asked to assist in identifying all errors and bugs. The sample data are used for testing. It is not quantity but the quality of the data used the matters of testing. Testing is aimed at ensuring that the system is accurate and efficient before live operation commands

7.1 TEST CASES

Test Case 1 : Unit

- Ensure that the Sudoku grid initializes correctly with either an empty grid or a pre-filled grid.
- Test that values can be set in cells according to Sudoku rules (no duplicates in rows, columns, or 3x3 sub-grids).

Test Case 2 : Integration

- Ensure the puzzle solver integrates correctly with the grid, solving valid puzzles completely and accurately.
- Test that user inputs (e.g., number placements) are processed correctly and update the grid as expected.

Test Case 3 : Functional

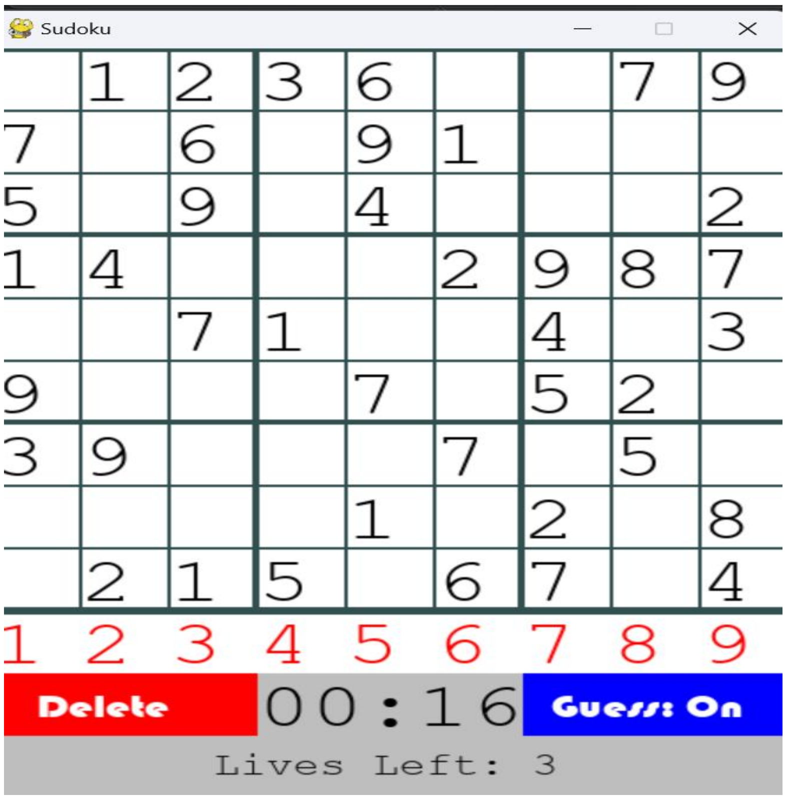
- Test that the game correctly identifies valid and invalid puzzles.

RESULTS

8. RESULTS

8.1. HOME PAGE

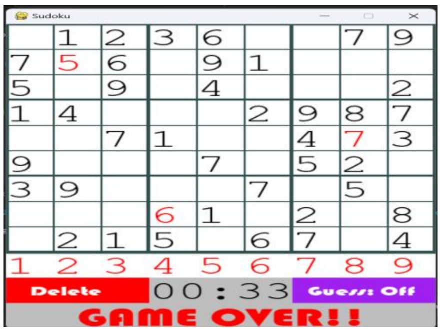
Fig 8.1.1 states that we get the sudoku board by clicking the executable file for playing the game.



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

Fig 8.1.1: HOME PAGE

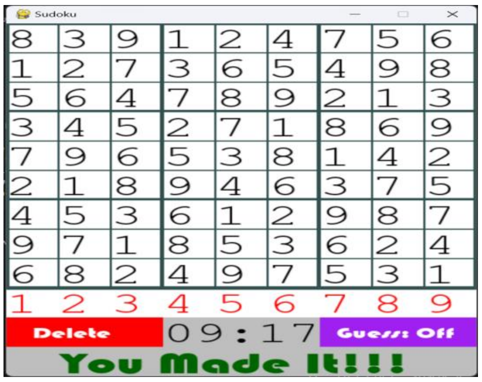
8.1.2 IF WE LOOSE GAME



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

Fig 8.1.2:LOOSE GAME

8.1.3 IF WE WIN GAME



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

1 2 3 4 5 6 7 8 9

09:17

You Made It!!!

Fig 8.1.2:WIN GAME

CONCLUSION

9. CONCLUSION

The Sudoku game implemented in Python has been a successful demonstration of how algorithmic logic can be applied to solve complex puzzles. By leveraging backtracking algorithms, we efficiently filled in the Sudoku grid while adhering to the game's constraints. The game allows users to input their puzzles, providing a flexible and interactive experience. It can solve puzzles of varying difficulty levels, showcasing the robustness of the implemented algorithm. The Python code was organized into modular functions, making it easy to understand, maintain, and extend. Each function had a clear responsibility, from validating the grid to finding solutions, promoting code reusability. The implementation includes error handling to manage invalid inputs and edge cases gracefully. This ensures that the program remains stable and user-friendly. This project serves as a great educational tool, illustrating the practical application of recursive algorithms, constraint satisfaction problems, and the importance of algorithmic efficiency.

FUTURE ENHANCEMENT :

- **Graphical User Interface (GUI):** Use libraries like Tkinter or Pygame to create an intuitive and visually appealing interface.
- **Timer and Scoring System:** Implement a timer and a scoring system to make the game competitive.
- **Constraint Propagation:** Implement techniques like backtracking, naked pairs, hidden pairs, and other logical techniques.
- **Machine Learning:** Use machine learning to improve the puzzle generation and solving process.
- **Optimization:** Optimize the solving algorithms for better performance.

Themes and Skins: Provide various themes and skins to customize the look of the game.

- **Multiplayer Mode:** Implement a multiplayer mode where players can compete against each other.

BIBLIOGRAPHY

10. BIBLIOGRAPHY

1. Kwon, H. J. (2006). Sudoku: The World's Greatest Brain Game. Sterling Publishing.

2. Berkel, K. (2005). Sudoku Puzzle Book: 200 Puzzles with Solutions. Sterling Publishing.

3. Kraitchik, M. (2006). Mathematical Recreations. Dover Publications.

4. Kendall, M. (2005). Mastering Sudoku Week by Week: 52 Steps to Becoming a Sudoku Wizard. Bantam Books.

5. Yato, T., & Seta, T. (2003). Complexity and Completeness of Finding Another Solution and Its Application to Puzzles. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E86-A(5), 1052-1060.

6. Lewis, R. (2007). Metaheuristics Can Solve Sudoku Puzzles. Journal of Heuristics, 13(4), 387-401.

7. Lewis, R. (2007). Metaheuristics Can Solve Sudoku Puzzles. Journal of Heuristics, 13(4), 387-401.

8. Simonis, H. (2005). Sudoku as a Constraint Problem. Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems (ModRef 2005).