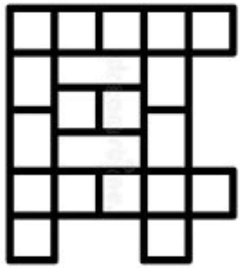
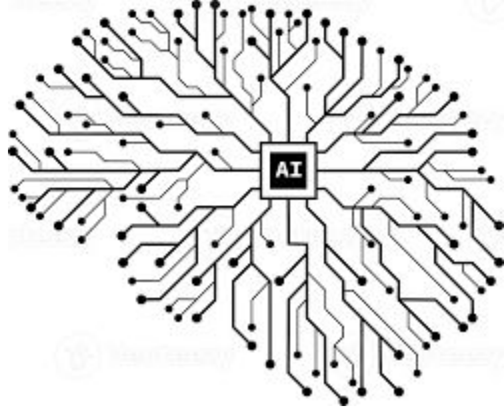
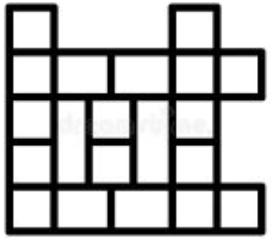
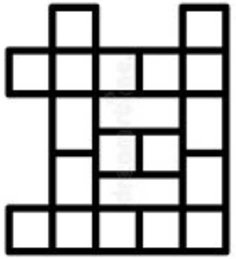




Sudoku Team

Members: *Marc Rodriguez, Aaron Sanchez, Rosa Lisa Silipino, Azza Laz, Hisham Alasadi*





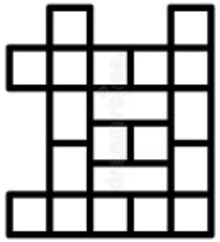
Introduction

Sudoku

- A logic based game, often involves solving a numeric pattern in a puzzle
- Each row, column, quadrant must have a number from 1-9 without instances of repeat
- Sudoku puzzles come in varying levels of difficulty, offering a satisfying yet stimulating experience for players of all skill levels

Our Project:

- Goal is to develop a sudoku puzzle through the usage of artificial intelligence that mimics that of a human approach

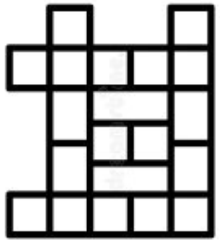


Importance of The Project

Artificial Intelligence has been an integral part of games for several decades. It is revered for ability to mock intelligence and generate entertainment. It aids in creating a more personal experience. AI makes games more fun it's almost like playing with a smart friend who knows just how to match your skill level and challenge you.

This project will highlight some of the key findings discussed in class. Mainly implemented will be the approaches of:

- Constraint Satisfaction Problems
- Backtracking
- Minimum Remaining Value (MRV) Heuristic



Data and Methodology

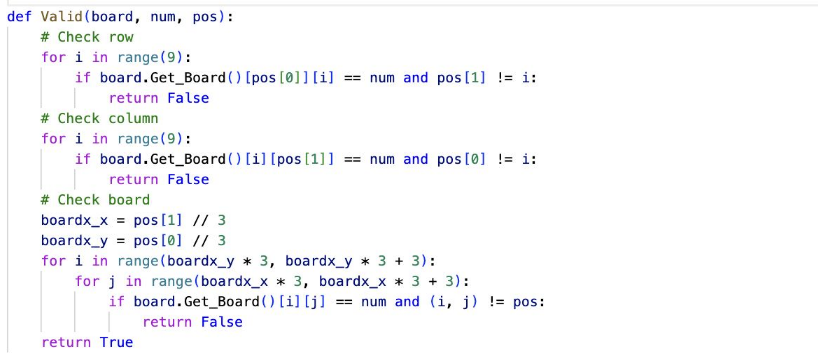
Description of Data

- Integers found within code represent levels, cell blocks, a 0 represents empty cell, Grids represented via usages of arrays.
- Boards are randomly generated, where cells will be deleted to create a solvable puzzle.
- Implementation of functions to solve game.
- Trials held for solving times based upon seconds for various levels.
  - Average time returned for execution of 100 trials.
  - Graphs plot individual execution times for each puzzle.
- Main function returns time to solve each puzzle along with original puzzle and solved puzzle.

AI Related Algorithm

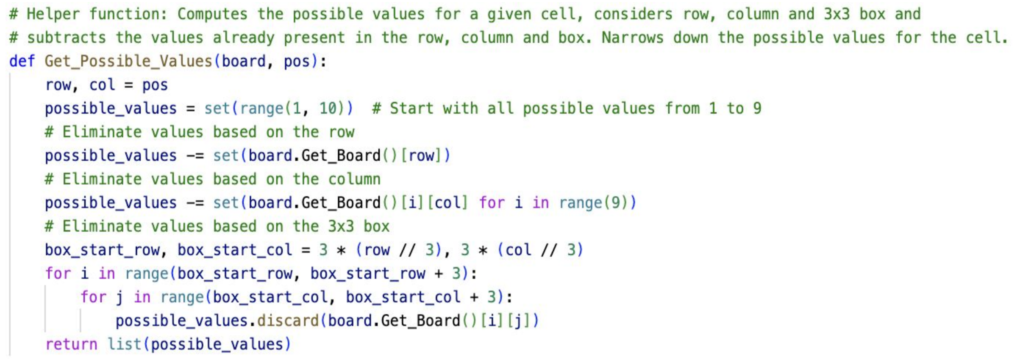
- **Backtracking**

- Code checks rows, columns and box constraints for certain values based off current state of board.
- Backtracks to try new value to solve empty cell.

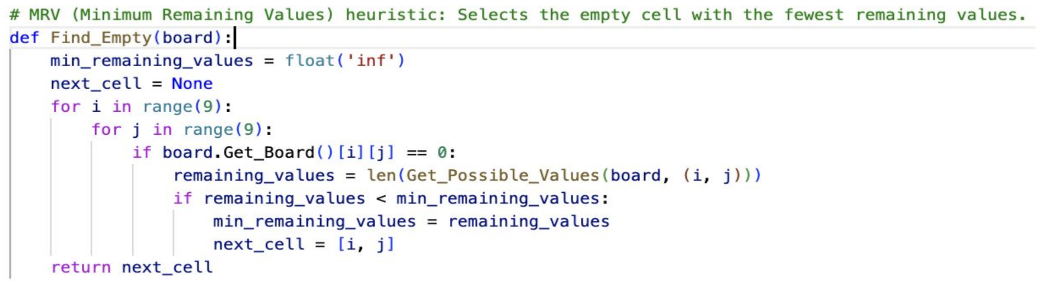


```python
def Valid(board, num, pos):
    # Check row
    for i in range(9):
        if board.Get_Board()[pos[0]][i] == num and pos[1] != i:
            return False
    # Check column
    for i in range(9):
        if board.Get_Board()[i][pos[1]] == num and pos[0] != i:
            return False
    # Check board
    boardx_x = pos[1] // 3
    boardx_y = pos[0] // 3
    for i in range(boardx_y * 3, boardx_y * 3 + 3):
        for j in range(boardx_x * 3, boardx_x * 3 + 3):
            if board.Get_Board()[i][j] == num and (i, j) != pos:
                return False
    return True
```

Data and Methodology (MRV)



```python
# Helper function: Computes the possible values for a given cell, considers row, column and 3x3 box and
# subtracts the values already present in the row, column and box. Narrows down the possible values for the cell.
def Get_Possible_Values(board, pos):
    row, col = pos
    possible_values = set(range(1, 10))  # Start with all possible values from 1 to 9
    # Eliminate values based on the row
    possible_values -= set(board.Get_Board()[row])
    # Eliminate values based on the column
    possible_values -= set(board.Get_Board()[i][col] for i in range(9))
    # Eliminate values based on the 3x3 box
    box_start_row, box_start_col = 3 * (row // 3), 3 * (col // 3)
    for i in range(box_start_row, box_start_row + 3):
        for j in range(box_start_col, box_start_col + 3):
            possible_values.discard(board.Get_Board()[i][j])
    return list(possible_values)
```



```python
# MRV (Minimum Remaining Values) heuristic: Selects the empty cell with the fewest remaining values.
def Find_Empty(board):
    min_remaining_values = float('inf')
    next_cell = None
    for i in range(9):
        for j in range(9):
            if board.Get_Board()[i][j] == 0:
                remaining_values = len(Get_Possible_Values(board, (i, j)))
                if remaining_values < min_remaining_values:
                    min_remaining_values = remaining_values
                    next_cell = [i, j]
    return next_cell
```

AI Related Algorithm

- **MRV Heuristic**

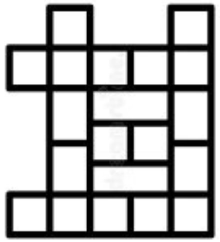
- Helper function computes possible values for a given cell based off constraints of row, column and 3x3 box. Subtracts already present values from possible values.

- Find_Empty function is similar to a human approach in which it finds the next empty cell which will be the easiest to solve, due to the fact that the possible values it could be is the fewest possible in comparison to the the other empty cells.

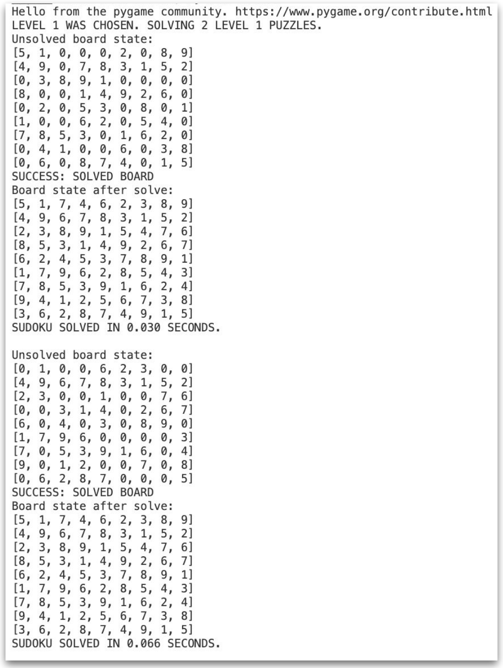
Experimental Results

Performance metrics based upon:

- Effectiveness of implemented backtracking and heuristic implementation.
- Time the puzzle(s) takes to execute upon level difficulty.
- Quality of back tracking vs Minimum Remaining Value (MRV) heuristic.
- Satisfaction or general ability to solve the given puzzle.

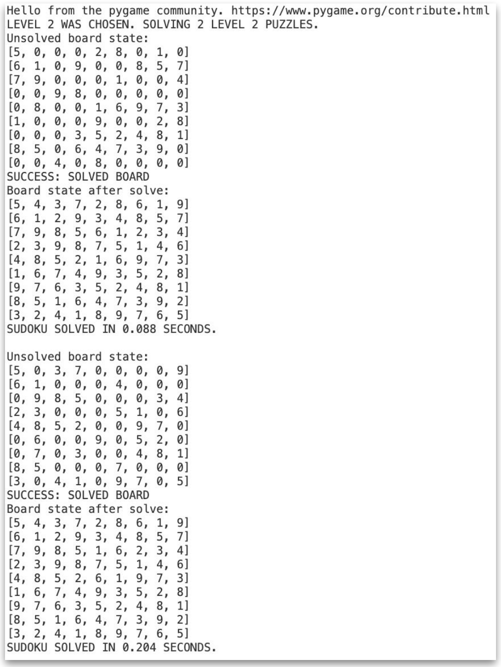


Visualization of Terminal Results (Simple Backing Tracking)



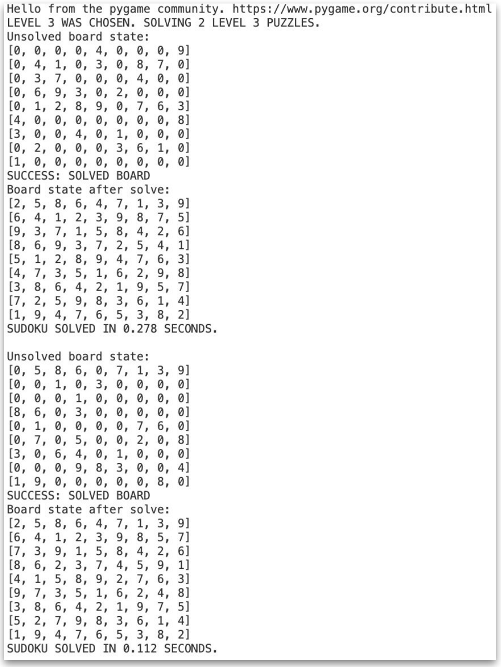
```
Hello from the pygame community. https://www.pygame.org/contribute.html
LEVEL 1 WAS CHOSEN. SOLVING 2 LEVEL 1 PUZZLES.
Unsolved board state:
[5, 1, 0, 0, 0, 2, 0, 8, 9]
[4, 9, 0, 7, 8, 3, 1, 5, 2]
[0, 3, 8, 9, 1, 0, 0, 0, 0]
[8, 0, 0, 1, 4, 9, 2, 6, 0]
[0, 2, 0, 5, 3, 0, 8, 0, 1]
[1, 0, 0, 6, 2, 0, 5, 4, 0]
[7, 8, 5, 3, 0, 1, 6, 2, 0]
[0, 4, 1, 0, 0, 6, 0, 3, 8]
[0, 6, 0, 8, 7, 4, 0, 1, 5]
SUCCESS: SOLVED BOARD
Board state after solve:
[5, 1, 7, 4, 6, 2, 3, 8, 9]
[4, 9, 6, 7, 8, 3, 1, 5, 2]
[2, 3, 8, 9, 1, 5, 4, 7, 6]
[8, 5, 3, 1, 4, 9, 2, 6, 7]
[6, 2, 4, 5, 3, 7, 8, 9, 1]
[1, 7, 9, 6, 2, 8, 5, 4, 3]
[7, 8, 5, 3, 9, 1, 6, 2, 4]
[9, 4, 1, 2, 5, 6, 7, 3, 8]
[3, 6, 2, 8, 7, 4, 9, 1, 5]
SUDOKU SOLVED IN 0.030 SECONDS.

Unsolved board state:
[0, 1, 0, 0, 6, 2, 3, 0, 0]
[4, 9, 6, 7, 8, 3, 1, 5, 2]
[2, 3, 0, 0, 1, 0, 0, 7, 6]
[0, 0, 3, 1, 4, 0, 2, 6, 7]
[6, 0, 4, 0, 3, 0, 8, 9, 0]
[1, 7, 9, 6, 0, 0, 0, 0, 3]
[7, 0, 5, 3, 9, 1, 6, 0, 4]
[9, 0, 1, 2, 0, 0, 7, 0, 8]
[0, 6, 2, 8, 7, 0, 0, 0, 5]
SUCCESS: SOLVED BOARD
Board state after solve:
[5, 1, 7, 4, 6, 2, 3, 8, 9]
[4, 9, 6, 7, 8, 3, 1, 5, 2]
[2, 3, 8, 9, 1, 5, 4, 7, 6]
[8, 5, 3, 1, 4, 9, 2, 6, 7]
[6, 2, 4, 5, 3, 7, 8, 9, 1]
[1, 7, 9, 6, 2, 8, 5, 4, 3]
[7, 8, 5, 3, 9, 1, 6, 2, 4]
[9, 4, 1, 2, 5, 6, 7, 3, 8]
[3, 6, 2, 8, 7, 4, 9, 1, 5]
SUDOKU SOLVED IN 0.066 SECONDS.
```

Easy Difficulty



```
Hello from the pygame community. https://www.pygame.org/contribute.html
LEVEL 2 WAS CHOSEN. SOLVING 2 LEVEL 2 PUZZLES.
Unsolved board state:
[5, 0, 0, 0, 2, 8, 0, 1, 0]
[6, 1, 0, 9, 0, 0, 8, 5, 7]
[7, 9, 0, 0, 0, 1, 0, 0, 4]
[0, 0, 9, 8, 0, 0, 0, 0, 0]
[0, 8, 0, 0, 1, 6, 9, 7, 3]
[1, 0, 0, 0, 9, 0, 0, 2, 8]
[0, 0, 0, 3, 5, 2, 4, 8, 1]
[8, 5, 0, 6, 4, 7, 3, 9, 0]
[0, 0, 4, 0, 8, 0, 0, 0, 0]
SUCCESS: SOLVED BOARD
Board state after solve:
[5, 4, 3, 7, 2, 8, 6, 1, 9]
[6, 1, 2, 9, 3, 4, 8, 5, 7]
[7, 9, 8, 5, 6, 1, 2, 3, 4]
[2, 3, 9, 8, 7, 5, 1, 4, 6]
[4, 8, 5, 2, 1, 6, 9, 7, 3]
[1, 6, 7, 4, 9, 3, 5, 2, 8]
[9, 7, 6, 3, 5, 2, 4, 8, 1]
[8, 5, 1, 6, 4, 7, 3, 9, 2]
[3, 2, 4, 1, 8, 9, 7, 6, 5]
SUDOKU SOLVED IN 0.088 SECONDS.

Unsolved board state:
[5, 0, 3, 7, 0, 0, 0, 0, 9]
[6, 1, 0, 0, 0, 4, 0, 0, 0]
[0, 9, 8, 5, 0, 0, 0, 3, 4]
[2, 3, 0, 0, 0, 5, 1, 0, 6]
[4, 8, 5, 2, 0, 0, 9, 7, 0]
[0, 6, 0, 0, 9, 0, 5, 2, 0]
[0, 7, 0, 3, 0, 0, 4, 8, 1]
[8, 5, 0, 0, 0, 7, 0, 0, 0]
[3, 0, 4, 1, 0, 9, 7, 0, 5]
SUCCESS: SOLVED BOARD
Board state after solve:
[5, 4, 3, 7, 2, 8, 6, 1, 9]
[6, 1, 2, 9, 3, 4, 8, 5, 7]
[7, 9, 8, 5, 1, 6, 2, 3, 4]
[2, 3, 9, 8, 7, 5, 1, 4, 6]
[4, 8, 5, 2, 6, 1, 9, 7, 3]
[1, 6, 7, 4, 9, 3, 5, 2, 8]
[9, 7, 6, 3, 5, 2, 4, 8, 1]
[8, 5, 1, 6, 4, 7, 3, 9, 2]
[3, 2, 4, 1, 8, 9, 7, 6, 5]
SUDOKU SOLVED IN 0.204 SECONDS.
```

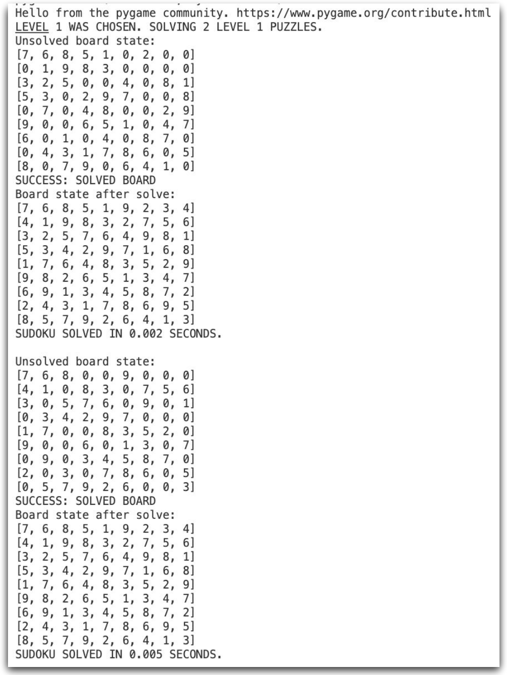
Medium Difficulty



```
Hello from the pygame community. https://www.pygame.org/contribute.html
LEVEL 3 WAS CHOSEN. SOLVING 2 LEVEL 3 PUZZLES.
Unsolved board state:
[0, 0, 0, 0, 4, 0, 0, 0, 9]
[0, 4, 1, 0, 3, 0, 8, 7, 0]
[0, 3, 7, 0, 0, 0, 4, 0, 0]
[0, 6, 9, 3, 0, 2, 0, 0, 0]
[0, 1, 2, 8, 9, 0, 7, 6, 3]
[4, 0, 0, 0, 0, 0, 0, 0, 8]
[3, 0, 0, 4, 0, 1, 0, 0, 0]
[0, 2, 0, 0, 0, 3, 6, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0]
SUCCESS: SOLVED BOARD
Board state after solve:
[2, 5, 8, 6, 4, 7, 1, 3, 9]
[6, 4, 1, 2, 3, 9, 8, 7, 5]
[9, 3, 7, 1, 5, 8, 4, 2, 6]
[8, 6, 9, 3, 7, 2, 5, 4, 1]
[5, 1, 2, 8, 9, 4, 7, 6, 3]
[4, 7, 3, 5, 1, 6, 2, 9, 8]
[3, 8, 6, 4, 2, 1, 9, 5, 7]
[7, 2, 5, 9, 8, 3, 6, 1, 4]
[1, 9, 4, 7, 6, 5, 3, 8, 2]
SUDOKU SOLVED IN 0.278 SECONDS.

Unsolved board state:
[0, 5, 8, 6, 0, 7, 1, 3, 9]
[0, 0, 1, 0, 3, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0]
[8, 6, 0, 3, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 7, 6, 0]
[0, 7, 0, 5, 0, 0, 2, 0, 8]
[3, 0, 6, 4, 0, 1, 0, 0, 0]
[0, 0, 0, 9, 8, 3, 0, 0, 4]
[1, 9, 0, 0, 0, 0, 0, 8, 0]
SUCCESS: SOLVED BOARD
Board state after solve:
[2, 5, 8, 6, 4, 7, 1, 3, 9]
[6, 4, 1, 2, 3, 9, 8, 5, 7]
[7, 3, 9, 1, 5, 8, 4, 2, 6]
[8, 6, 2, 3, 7, 4, 5, 9, 1]
[4, 1, 5, 8, 9, 2, 7, 6, 3]
[9, 7, 3, 5, 1, 6, 2, 4, 8]
[3, 8, 6, 4, 2, 1, 9, 7, 5]
[5, 2, 7, 9, 8, 3, 6, 1, 4]
[1, 9, 4, 7, 6, 5, 3, 8, 2]
SUDOKU SOLVED IN 0.112 SECONDS.
```

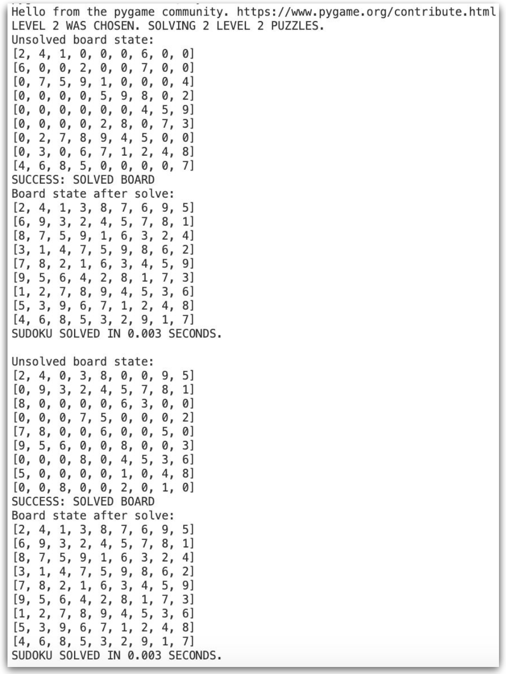
Hard Difficulty

Visualization of Terminal Results (MRV Heuristic)



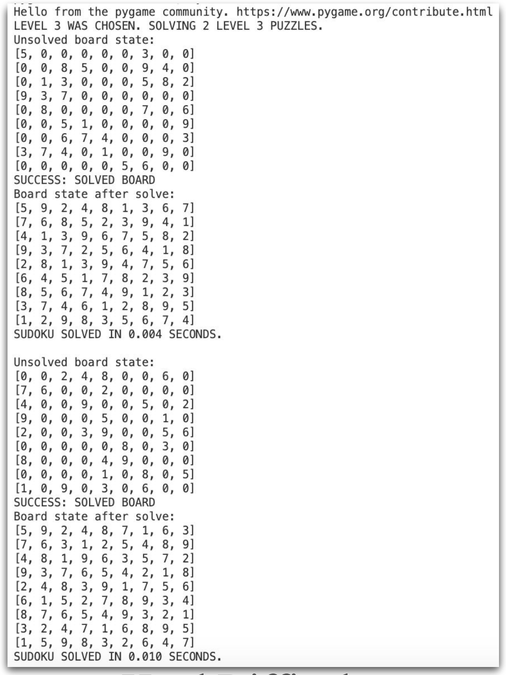
```
Hello from the pygame community. https://www.pygame.org/contribute.html
LEVEL 1 WAS CHOSEN. SOLVING 2 LEVEL 1 PUZZLES.
Unsolved board state:
[7, 6, 8, 5, 1, 0, 2, 0, 0]
[0, 1, 9, 8, 3, 0, 0, 0, 0]
[3, 2, 5, 0, 0, 4, 0, 8, 1]
[5, 3, 0, 2, 9, 7, 0, 0, 8]
[0, 7, 0, 4, 8, 0, 0, 2, 9]
[9, 0, 0, 6, 5, 1, 0, 4, 7]
[6, 0, 1, 0, 4, 0, 8, 7, 0]
[0, 4, 3, 1, 7, 8, 6, 0, 5]
[8, 0, 7, 9, 0, 6, 4, 1, 0]
SUCCESS: SOLVED BOARD
Board state after solve:
[7, 6, 8, 5, 1, 9, 2, 3, 4]
[4, 1, 9, 8, 3, 2, 7, 5, 6]
[3, 2, 5, 7, 6, 4, 9, 8, 1]
[5, 3, 4, 2, 9, 7, 1, 6, 8]
[1, 7, 6, 4, 8, 3, 5, 2, 9]
[9, 8, 2, 6, 5, 1, 3, 4, 7]
[6, 9, 1, 3, 4, 5, 8, 7, 2]
[2, 4, 3, 1, 7, 8, 6, 9, 5]
[8, 5, 7, 9, 2, 6, 4, 1, 3]
SUDOKU SOLVED IN 0.002 SECONDS.

Unsolved board state:
[7, 6, 8, 0, 0, 9, 0, 0, 0]
[4, 1, 0, 8, 3, 0, 7, 5, 6]
[3, 0, 5, 7, 6, 0, 9, 0, 1]
[0, 3, 4, 2, 9, 7, 0, 0, 0]
[1, 7, 0, 0, 8, 3, 5, 2, 0]
[9, 0, 0, 6, 0, 1, 3, 0, 7]
[0, 9, 0, 3, 4, 5, 8, 7, 0]
[2, 0, 3, 0, 7, 8, 6, 0, 5]
[0, 5, 7, 9, 2, 6, 0, 0, 3]
SUCCESS: SOLVED BOARD
Board state after solve:
[7, 6, 8, 5, 1, 9, 2, 3, 4]
[4, 1, 9, 8, 3, 2, 7, 5, 6]
[3, 2, 5, 7, 6, 4, 9, 8, 1]
[5, 3, 4, 2, 9, 7, 1, 6, 8]
[1, 7, 6, 4, 8, 3, 5, 2, 9]
[9, 8, 2, 6, 5, 1, 3, 4, 7]
[6, 9, 1, 3, 4, 5, 8, 7, 2]
[2, 4, 3, 1, 7, 8, 6, 9, 5]
[8, 5, 7, 9, 2, 6, 4, 1, 3]
SUDOKU SOLVED IN 0.005 SECONDS.
```

Easy Difficulty



```
Hello from the pygame community. https://www.pygame.org/contribute.html
LEVEL 2 WAS CHOSEN. SOLVING 2 LEVEL 2 PUZZLES.
Unsolved board state:
[2, 4, 1, 0, 0, 0, 6, 0, 0]
[6, 0, 0, 2, 0, 0, 7, 0, 0]
[0, 7, 5, 9, 1, 0, 0, 0, 4]
[0, 0, 0, 0, 5, 9, 8, 0, 2]
[0, 0, 0, 0, 0, 0, 4, 5, 9]
[0, 0, 0, 0, 2, 8, 0, 7, 3]
[0, 2, 7, 8, 9, 4, 5, 0, 0]
[0, 3, 0, 6, 7, 1, 2, 4, 8]
[4, 6, 8, 5, 0, 0, 0, 0, 7]
SUCCESS: SOLVED BOARD
Board state after solve:
[2, 4, 1, 3, 8, 7, 6, 9, 5]
[6, 9, 3, 2, 4, 5, 7, 8, 1]
[8, 7, 5, 9, 1, 6, 3, 2, 4]
[3, 1, 4, 7, 5, 9, 8, 6, 2]
[7, 8, 2, 1, 6, 3, 4, 5, 9]
[9, 5, 6, 4, 2, 8, 1, 7, 3]
[1, 2, 7, 8, 9, 4, 5, 3, 6]
[5, 3, 9, 6, 7, 1, 2, 4, 8]
[4, 6, 8, 5, 3, 2, 9, 1, 7]
SUDOKU SOLVED IN 0.003 SECONDS.

Unsolved board state:
[2, 4, 0, 3, 8, 0, 0, 9, 5]
[0, 9, 3, 2, 4, 5, 7, 8, 1]
[8, 0, 0, 0, 0, 6, 3, 0, 0]
[0, 0, 0, 7, 5, 0, 0, 0, 2]
[7, 8, 0, 0, 6, 0, 0, 5, 0]
[9, 5, 6, 0, 0, 8, 0, 0, 3]
[0, 0, 0, 8, 0, 4, 5, 3, 6]
[5, 0, 0, 0, 0, 1, 0, 4, 8]
[0, 0, 8, 0, 0, 2, 0, 1, 0]
SUCCESS: SOLVED BOARD
Board state after solve:
[2, 4, 1, 3, 8, 7, 6, 9, 5]
[6, 9, 3, 2, 4, 5, 7, 8, 1]
[8, 7, 5, 9, 1, 6, 3, 2, 4]
[3, 1, 4, 7, 5, 9, 8, 6, 2]
[7, 8, 2, 1, 6, 3, 4, 5, 9]
[9, 5, 6, 4, 2, 8, 1, 7, 3]
[1, 2, 7, 8, 9, 4, 5, 3, 6]
[5, 3, 9, 6, 7, 1, 2, 4, 8]
[4, 6, 8, 5, 3, 2, 9, 1, 7]
SUDOKU SOLVED IN 0.003 SECONDS.
```

Medium Difficulty

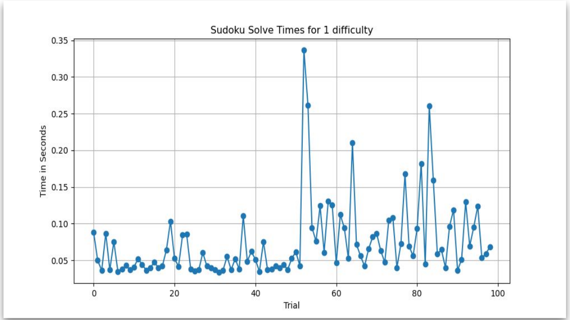


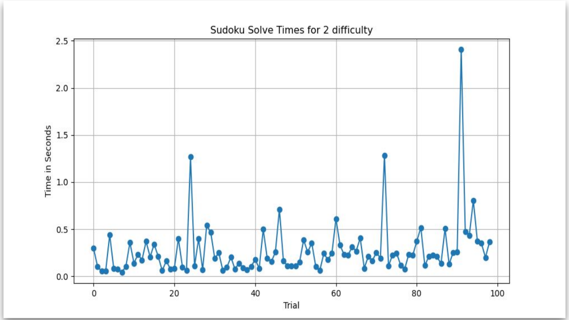
```
Hello from the pygame community. https://www.pygame.org/contribute.html
LEVEL 3 WAS CHOSEN. SOLVING 2 LEVEL 3 PUZZLES.
Unsolved board state:
[5, 0, 0, 0, 0, 0, 3, 0, 0]
[0, 0, 8, 5, 0, 0, 9, 4, 0]
[0, 1, 3, 0, 0, 0, 5, 8, 2]
[9, 3, 7, 0, 0, 0, 0, 0, 0]
[0, 8, 0, 0, 0, 0, 7, 0, 6]
[0, 0, 5, 1, 0, 0, 0, 0, 9]
[0, 0, 6, 7, 4, 0, 0, 0, 3]
[3, 7, 4, 0, 1, 0, 0, 9, 0]
[0, 0, 0, 0, 0, 5, 6, 0, 0]
SUCCESS: SOLVED BOARD
Board state after solve:
[5, 9, 2, 4, 8, 1, 3, 6, 7]
[7, 6, 8, 5, 2, 3, 9, 4, 1]
[4, 1, 3, 9, 6, 7, 5, 8, 2]
[9, 3, 7, 2, 5, 6, 4, 1, 8]
[2, 8, 1, 3, 9, 4, 7, 5, 6]
[6, 4, 5, 1, 7, 8, 2, 3, 9]
[8, 5, 6, 7, 4, 9, 1, 2, 3]
[3, 7, 4, 6, 1, 2, 8, 9, 5]
[1, 2, 9, 8, 3, 5, 6, 7, 4]
SUDOKU SOLVED IN 0.004 SECONDS.

Unsolved board state:
[0, 0, 2, 4, 8, 0, 0, 6, 0]
[7, 6, 0, 0, 2, 0, 0, 0, 0]
[4, 0, 0, 9, 0, 0, 5, 0, 2]
[9, 0, 0, 0, 5, 0, 0, 1, 0]
[2, 0, 0, 3, 9, 0, 0, 5, 6]
[0, 0, 0, 0, 0, 8, 0, 3, 0]
[8, 0, 0, 0, 4, 9, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 8, 0, 5]
[1, 0, 9, 0, 3, 0, 6, 0, 0]
SUCCESS: SOLVED BOARD
Board state after solve:
[5, 9, 2, 4, 8, 7, 1, 6, 3]
[7, 6, 3, 1, 2, 5, 4, 8, 9]
[4, 8, 1, 9, 6, 3, 5, 7, 2]
[9, 3, 7, 6, 5, 4, 2, 1, 8]
[2, 4, 8, 3, 9, 1, 7, 5, 6]
[6, 1, 5, 2, 7, 8, 9, 3, 4]
[8, 7, 6, 5, 4, 9, 3, 2, 1]
[3, 2, 4, 7, 1, 6, 8, 9, 5]
[1, 5, 9, 8, 3, 2, 6, 4, 7]
SUDOKU SOLVED IN 0.010 SECONDS.
```

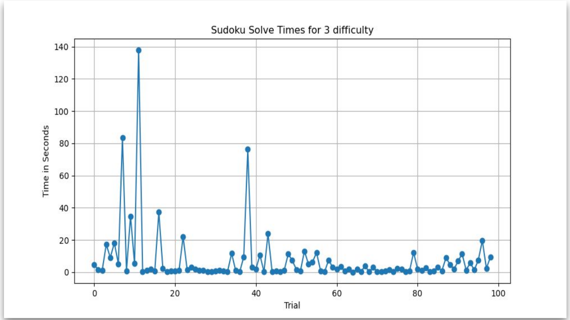
Hard Difficulty

Plot Visualization Per Difficulty Old vs. New

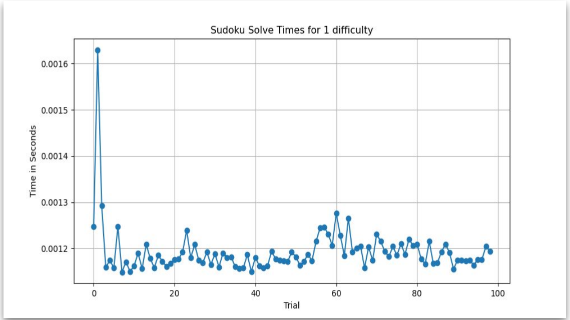
Back Tracking

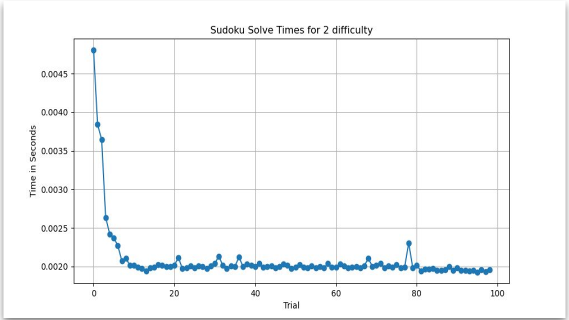


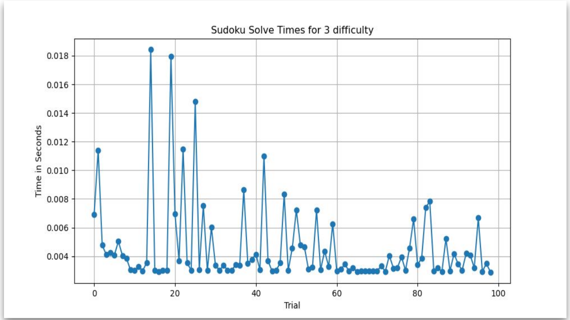




MRV Heuristic

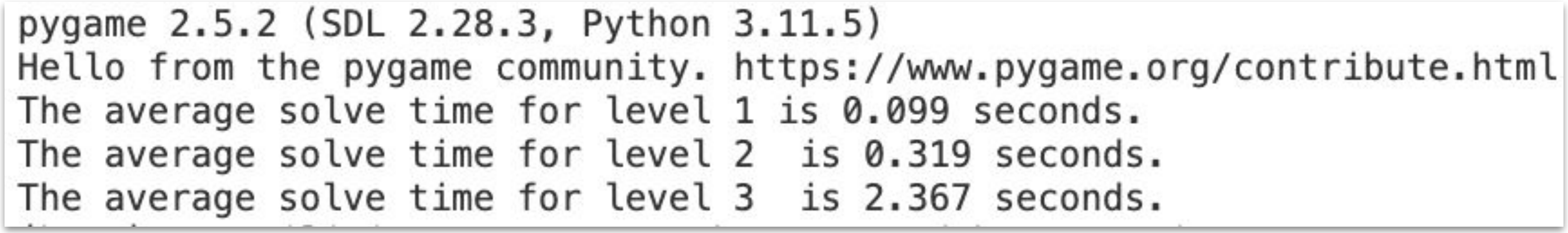




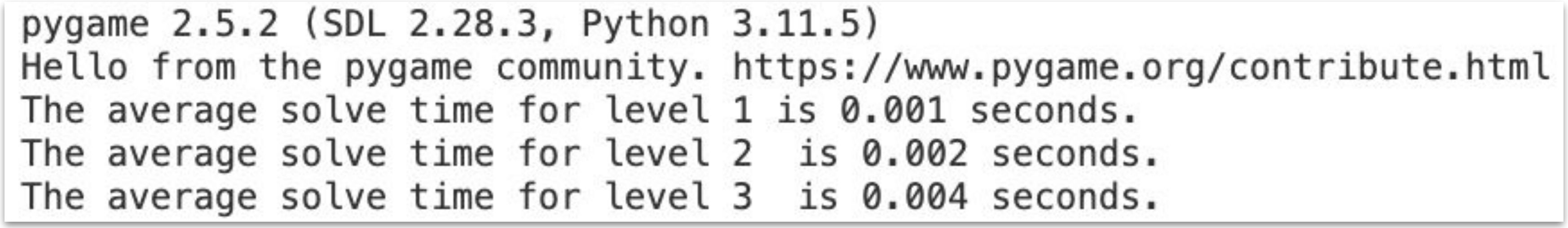


Experiment Data Results Backtracking vs MRV

- Utilizing experimental.py file data shown was collected from running 100 trials of each difficulty, displaying average execution time in terminal.



```
pygame 2.5.2 (SDL 2.28.3, Python 3.11.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
The average solve time for level 1 is 0.099 seconds.
The average solve time for level 2  is 0.319 seconds.
The average solve time for level 3  is 2.367 seconds.
```



```
pygame 2.5.2 (SDL 2.28.3, Python 3.11.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
The average solve time for level 1 is 0.001 seconds.
The average solve time for level 2  is 0.002 seconds.
The average solve time for level 3  is 0.004 seconds.
```

Discussion

Interpretation of Results

- Through trial implementation time of solving increases along with difficulty of level
- Backtracking occurs more frequently in more difficult levels, adds to increase of time

Future Directions

- Check for any final errors, likely continue trial testing, evaluate any performance expectations

Challenges

- Error handling for events, and pygame closure where system doesn't completely exit.
  - Issue resolved by utilizing sys.exit().
- On key commands such as user hitting space bar multiple times, program will execute solving subsequent puzzles before visualization.
  - Implementation of keyup instead of keydown, may prevent user from accidental double pressing space.
- Memory issues in terms of space needed to be ran by IDE

Conclusion

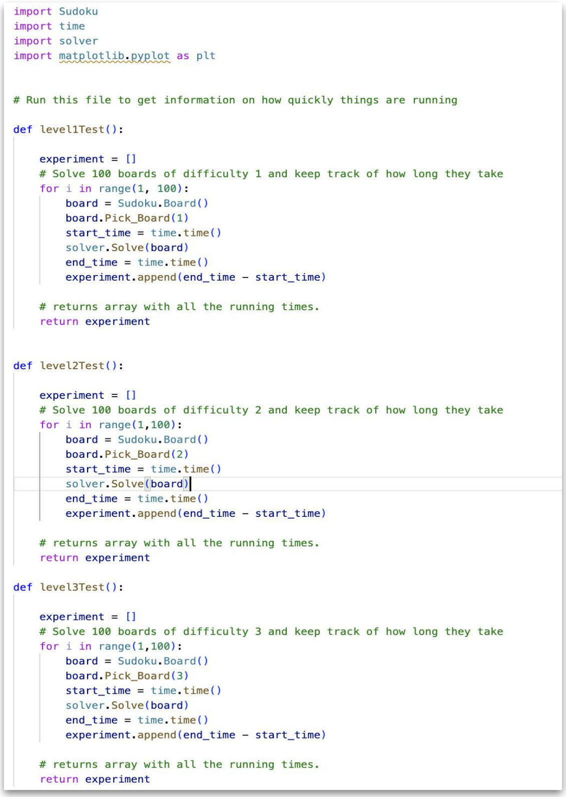
Key Findings: Back tracking is not an optimal solution for this project, with the assistance of MRV it significantly improves solving times.
Contributions:

- Rosa Lisa Silipino: Edited GUI for levelSelect and created GUI runIterations and event handling implementations. Collaborated on Sudoku.py and Solver.py for implementing the changes for utilizing MRV heuristic.
- Marc Rodriguez: General development of slides. Made sure to follow development guidelines on canvas for optimal flow.
- Aaron Sanchez: Focused mainly on the aspects of gathering data for analysis. Successfully integrated a sudoku generator, and small collaborations on other files. Created experiment.py file
- Hisham Alasadi: Added some minor changes to the slides and contributed to the solver.py in regards to the Geting possible values, provided code for heuristic method
- Azza Laz: Through collaborative efforts, ensured that the Sudoku implementation integrated seamlessly with the project's focus on optimizing solving times through the utilization of MRV heuristic, and assisted in refining the initial concept.

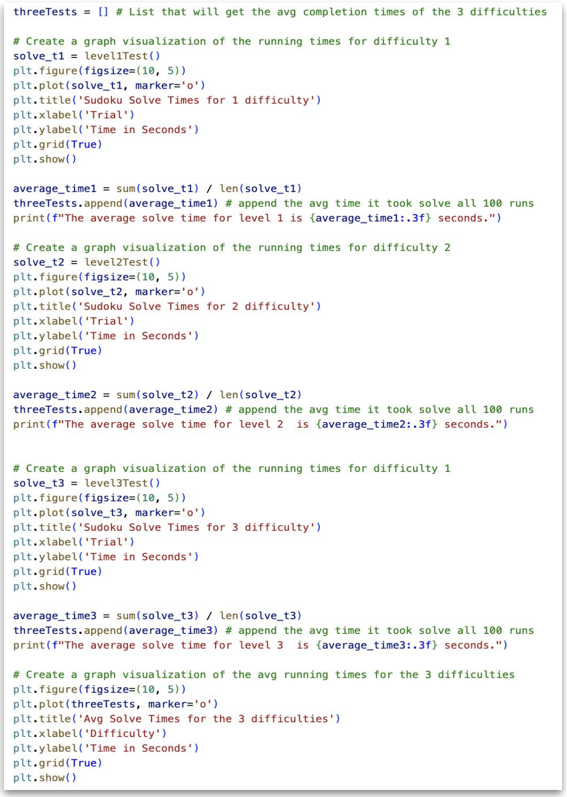
Reflections: Learned to utilize and code for pygame programs. Increased literacy for coding in Python. Gained a better understanding on how to collaborate on GitHub. Gained a better understanding of the Sudoku game. Collaborative teamwork played a pivotal role, with each member's contributions refining our approach.

Final Remarks: Our project's exploration of AI-driven Sudoku solving unveils promising insights applicable beyond gaming reals. By integrating traditional methods like backtracking with advanced strategies like MRV heuristic, we've notably enhanced efficiency.

Appendices



```
import Sudoku
import time
import solver
import matplotlib.pyplot as plt


# Run this file to get information on how quickly things are running

def level1Test():

    experiment = []
    # Solve 100 boards of difficulty 1 and keep track of how long they take
    for i in range(1, 100):
        board = Sudoku.Board()
        board.Pick_Board(1)
        start_time = time.time()
        solver.Solve(board)
        end_time = time.time()
        experiment.append(end_time - start_time)

    # returns array with all the running times.
    return experiment


def level2Test():

    experiment = []
    # Solve 100 boards of difficulty 2 and keep track of how long they take
    for i in range(1,100):
        board = Sudoku.Board()
        board.Pick_Board(2)
        start_time = time.time()
        solver.Solve(board)
        end_time = time.time()
        experiment.append(end_time - start_time)

    # returns array with all the running times.
    return experiment

def level3Test():

    experiment = []
    # Solve 100 boards of difficulty 3 and keep track of how long they take
    for i in range(1,100):
        board = Sudoku.Board()
        board.Pick_Board(3)
        start_time = time.time()
        solver.Solve(board)
        end_time = time.time()
        experiment.append(end_time - start_time)

    # returns array with all the running times.
    return experiment
```



```
threeTests = [] # List that will get the avg completion times of the 3 difficulties

# Create a graph visualization of the running times for difficulty 1
solve_t1 = level1Test()
plt.figure(figsize=(10, 5))
plt.plot(solve_t1, marker='o')
plt.title('Sudoku Solve Times for 1 difficulty')
plt.xlabel('Trial')
plt.ylabel('Time in Seconds')
plt.grid(True)
plt.show()

average_time1 = sum(solve_t1) / len(solve_t1)
threeTests.append(average_time1) # append the avg time it took solve all 100 runs
print(f"The average solve time for level 1 is {average_time1:.3f} seconds.")

# Create a graph visualization of the running times for difficulty 2
solve_t2 = level2Test()
plt.figure(figsize=(10, 5))
plt.plot(solve_t2, marker='o')
plt.title('Sudoku Solve Times for 2 difficulty')
plt.xlabel('Trial')
plt.ylabel('Time in Seconds')
plt.grid(True)
plt.show()

average_time2 = sum(solve_t2) / len(solve_t2)
threeTests.append(average_time2) # append the avg time it took solve all 100 runs
print(f"The average solve time for level 2  is {average_time2:.3f} seconds.")


# Create a graph visualization of the running times for difficulty 1
solve_t3 = level3Test()
plt.figure(figsize=(10, 5))
plt.plot(solve_t3, marker='o')
plt.title('Sudoku Solve Times for 3 difficulty')
plt.xlabel('Trial')
plt.ylabel('Time in Seconds')
plt.grid(True)
plt.show()

average_time3 = sum(solve_t3) / len(solve_t3)
threeTests.append(average_time3) # append the avg time it took solve all 100 runs
print(f"The average solve time for level 3  is {average_time3:.3f} seconds.")

# Create a graph visualization of the avg running times for the 3 difficulties
plt.figure(figsize=(10, 5))
plt.plot(threeTests, marker='o')
plt.title('Avg Solve Times for the 3 difficulties')
plt.xlabel('Difficulty')
plt.ylabel('Time in Seconds')
plt.grid(True)
plt.show()
```

Git Repository:

https://github.com/rosasilipino/AISudokuTeam

Experimental.py

Execution of this file runs 100 trials of each difficulty of different randomly generated Sudoku puzzles.

Calculates the total solving time for each puzzle and plots said data.

Returns graphs for the time completions of each trial and average time for each set of trials per difficulty.