CLEAN CODE DEVELOPMENT

Clean code developement is an important step of software development that deals with creating code that is easy to read, understand, maintain and develop without compromising on its performance and integrity.

CCD enables others to easily understand our code and overall makes collaboration easy.

Some of the benefits of Clean Code are:

- Readability
- Maintainability
- Scalability
- Collaboration
- Efficiency
- Reusability

Here are 5 ways I made my code clean:

1. Descriptive Variable Names:

```
class SudokuGame:
    def __init__(self, master, difficulty):
        self.master = master
        self.master.title("Sudoku Game")
        self.difficulty = difficulty
        self.board = self.generate_board()
        self.initial_board = [row[:] for row in self.board]
        self.create_widgets()
```

A well descriptive variable name help to make the code more readable, maintainable and understandable. Here in my code variables like ‘self.master’, ‘self.difficulty’, ‘self.board’ all have unique and descriptive nname which make it easy to understand their functionalities.

2. Modularization:

```
def generate_board(self):
    board = [[0] * 9 for _ in range(9)]
    self.solve_sudoku(board)
    self.remove_numbers(board)
    return board

def solve_sudoku(self, board):
    empty_cell = self.find_empty_cell(board)
```

Here the sudoku solving algorithm is encapsulated inside a function ‘def generate_board(self)’ and the puzzle solving algorithm is placed in a different function called ‘def solve_sudoku(self, board)’. This promotes modularity in the code making it easier to maintain.

3. Consistent Indentation and Readability:

```
def is_valid_move(self, board, row, col, num):
    if num in board[row] or num in [board[i][col] for i in range(9)]:
        return False
```

In this instance, consistent indentation improves code readability. Here the functions ‘self’, ‘board’, ‘row’ are all separated by commas to help us easily identify and understand.

4. Error Handling:

```
def validate_entry(self, event, row, col, widget):
    value = widget.get()
    if value.isdigit() and 1 <= int(value) <= 9:
        if self.is_valid_move(self.board, row, col, int(value)):
            self.board[row][col] = int(value)
        else:
            messagebox.showwarning("Invalid Move", "This number cannot be placed in the selected cell.")
            widget.delete(0, tk.END)
            widget.insert(0, "")
    else:
        widget.delete(0, tk.END)
```

Here an error message is shown to the user when they perform an invalid input. Error handling is important as it includes scenarios in which the user provides incorrect inputs and it helps to prevent unecpected problems or crashes.

5. Decreasing clutter:

Unwanted codes can be commented out or removed to avoid sphagetti code and ensure that our code remains clean and easily understandable.