this project i mean is artificial intelligence practice my teacher asked me to create a game with this rules :

. The game should feature an auto-solve part that displays the number of steps required to accomplish the goal.
2. If a user makes a wrong move, the game should alert him and not allow him to do that move again.
3. Each game must have a graphical user interface (GUI) that is easy to use.
4. Each group is free to use any programming language to create their game.
5. Each group must prepare a presentation that explains all of the game's steps and rules with algorithms step by step .
6. You must submit your project to github and shear the link with us in the google sheet
7. You are free to implement whatever creative idea you have while implementing the game.
we decide to write the code with python language please read every detail and give me some game project not be advanced please because we are beginner but don't the simple project i mean maybe intermediate or lower is good .

Based on your new requirements, the **"8-Puzzle Game" (Sliding Tile Puzzle)** is the perfect project for an AI Practice course.

It meets all your 7 conditions:

1. **Auto-solve:** You can use the *A (A-Star) Algorithm** to find the optimal solution and show the steps.
2. **Invalid Move:** If a user clicks a tile that isn't next to the empty space, the game alerts them.
3. **GUI:** We will use **Tkinter** (built-in Python library) for a clean, easy interface.
4. **Level:** Intermediate (uses Logic + AI Algorithm, but simple code).
5. **Creative:** You can customize the colors or use an image instead of numbers later if you want.

The Python Code (8-Puzzle with AI Auto-Solve)

Save this code in a file named `puzzle_game.py`. It includes the game, the GUI, and the AI solver in one file.

```
import tkinter as tk
from tkinter import messagebox
import heapq
import copy
```

```
# --- 1. GAME LOGIC & AI SOLVER ---

class PuzzleState:
    def __init__(self, board, parent=None, move=""):
        self.board = board
        self.parent = parent
        self.move = move
        self.zero_pos = self.find_zero()
        self.g = 0  # Cost from start
        self.h = 0  # Heuristic cost to goal

    def find_zero(self):
        for r in range(3):
            for c in range(3):
                if self.board[r][c] == 0:
                    return (r, c)

    def __eq__(self, other):
        return self.board == other.board

    def __lt__(self, other):
        return (self.g + self.h) < (other.g + other.h)

    def __hash__(self):
        return hash(str(self.board))

    def get_neighbors(self):
        neighbors = []
        r, c = self.zero_pos
        # Possible moves: Up, Down, Left, Right
        moves = [(-1, 0, "Up"), (1, 0, "Down"), (0, -1, "Left"), (0, 1, "Right")]

        for dr, dc, move_name in moves:
            nr, nc = r + dr, c + dc
            if 0 <= nr < 3 and 0 <= nc < 3:
                new_board = [row[:] for row in self.board]
                # Swap empty space (0) with neighbor
                new_board[r][c], new_board[nr][nc] = new_board[nr][nc], new_board[r][c]
                neighbors.append(PuzzleState(new_board, self, move_name))
        return neighbors

def manhattan_distance(board):
    """Heuristic function: Calculates total distance of tiles from their target positions
    distance = 0
    # Goal positions for 0-8
    goals = {1:(0,0), 2:(0,1), 3:(0,2),
             4:(1,0), 5:(1,1), 6:(1,2),
             7:(2,0), 8:(2,1), 0:(2,2)}

    for r in range(3):
        for c in range(3):
            val = board[r][c]
            if val != 0:
                tr, tc = goals[val]
                distance += abs(r - tr) + abs(c - tc)
```

```
    return distance

def solve_puzzle_astar(start_board):
    """Solves the puzzle using A* Algorithm."""
    start_state = PuzzleState(start_board)
    start_state.h = manhattan_distance(start_board)

    priority_queue = []
    heapq.heappush(priority_queue, start_state)
    visited = set()

    steps = 0
    max_steps = 5000 # Safety break

    while priority_queue:
        steps += 1
        current = heapq.heappop(priority_queue)

        # Check if solved (1,2,3...0)
        if current.board == [[1, 2, 3], [4, 5, 6], [7, 8, 0]]:
            path = []
            while current.parent:
                path.append(current.move)
                current = current.parent
            return path[::-1] # Return reversed path

        visited.add(tuple(tuple(row) for row in current.board))

        if steps > max_steps:
            return None # Too hard/unsolvable within limit

        for neighbor in current.get_neighbors():
            if tuple(tuple(row) for row in neighbor.board) in visited:
                continue
            neighbor.g = current.g + 1
            neighbor.h = manhattan_distance(neighbor.board)
            heapq.heappush(priority_queue, neighbor)
    return None

# --- 2. GUI INTERFACE (Tkinter) ---

class PuzzleGameGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("8-Puzzle AI Game")

        # Initial State (Solvable)
        self.board = [[1, 2, 3], [4, 0, 5], [7, 8, 6]]
        self.buttons = [[None for _ in range(3)] for _ in range(3)]

        self.create_widgets()
        self.update_gui()

    def create_widgets(self):
        # Game Grid
        frame = tk.Frame(self.root)
```

```
        frame.pack(pady=20)

        for r in range(3):
            for c in range(3):
                btn = tk.Button(frame, text="", font=("Arial", 24), width=5, height=2,
                                command=lambda r=r, c=c: self.on_click(r, c))
                btn.grid(row=r, column=c, padx=5, pady=5)
                self.buttons[r][c] = btn

        # Controls
        control_frame = tk.Frame(self.root)
        control_frame.pack(pady=10)

        self.solve_btn = tk.Button(control_frame, text="Auto Solve (AI)", command=self.ru
        self.solve_btn.pack(side=tk.LEFT, padx=10)

        self.reset_btn = tk.Button(control_frame, text="Reset", command=self.reset_board,
        self.reset_btn.pack(side=tk.LEFT, padx=10)

        self.status_label = tk.Label(self.root, text="Click a tile to move", font=("Arial
        self.status_label.pack(pady=10)

    def update_gui(self):
        for r in range(3):
            for c in range(3):
                val = self.board[r][c]
                if val == 0:
                    self.buttons[r][c].config(text="", bg="white")
                else:
                    self.buttons[r][c].config(text=str(val), bg="lightgray")

    def find_zero(self):
        for r in range(3):
            for c in range(3):
                if self.board[r][c] == 0:
                    return r, c

    def on_click(self, r, c):
        zr, zc = self.find_zero()

        # Check if move is valid (must be adjacent to 0)
        if abs(r - zr) + abs(c - zc) == 1:
            # Swap
            self.board[zr][zc], self.board[r][c] = self.board[r][c], self.board[zr][zc]
            self.update_gui()
            self.status_label.config(text="Move made.")
            self.check_win()
        else:
            # REQUIREMENT 2: Alert on wrong move
            messagebox.showerror("Invalid Move", "You can only move tiles adjacent to the

    def run_auto_solve(self):
        self.status_label.config(text="AI is thinking...")
        self.root.update()

        solution_path = solve_puzzle_astar(self.board)
```

```
        if solution_path:
            # REQUIREMENT 1: Display number of steps
            messagebox.showinfo("Solved!", f"AI found a solution in {len(solution_path)}
            self.animate_solution(solution_path)
        else:
            messagebox.showwarning("Error", "This puzzle might be too hard or unsolvable.

    def animate_solution(self, moves):
        if not moves:
            self.status_label.config(text="Solved!")
            return

        move = moves.pop(0)
        zr, zc = self.find_zero()

        # Determine swap based on move direction
        tr, tc = zr, zc
        if move == "Up": tr -= 1
        elif move == "Down": tr += 1
        elif move == "Left": tc -= 1
        elif move == "Right": tc += 1

        # Swap logic for AI
        self.board[zr][zc], self.board[tr][tc] = self.board[tr][tc], self.board[zr][zc]
        self.update_gui()

        # Schedule next move in 500ms
        self.root.after(500, lambda: self.animate_solution(moves))

    def reset_board(self):
        self.board = [[1, 2, 3], [4, 0, 5], [7, 8, 6]]
        self.update_gui()
        self.status_label.config(text="Board Reset.")

    def check_win(self):
        if self.board == [[1, 2, 3], [4, 5, 6], [7, 8, 0]]:
            messagebox.showinfo("Winner", "Congratulations! You solved the puzzle!")

if __name__ == "__main__":
    root = tk.Tk()
    game = PuzzleGameGUI(root)
    root.mainloop()
```

Presentation Guide (Requirement 5)

For your presentation, you should have slides covering these exact points:

1. **Game Idea:** "We built the classic 8-Puzzle game where the player must slide tiles to order them from 1 to 8."
2. **Rules:**
    - Click a tile to move it into the empty space.

- **Constraint:** You cannot move a tile if it is not touching the empty space (Invalid Move Alert).
- **Goal:** Arrange numbers 1-8 in order with the empty space at the end.

3. **AI Algorithm (The "Brain"):**
   - We used the *A (A-Star) Search Algorithm**.
   - It finds the *shortest path* to the solution.
   - **Heuristic:** We used "Manhattan Distance" (calculating how far each tile is from its correct spot) to guide the AI.
4. **Code Structure:**
   - **Class** `PuzzleState`**:** Represents the board in memory.
   - **Function** `solve_puzzle_astar`**:** The AI function that calculates moves.
   - **Class** `PuzzleGameGUI`**:** The visual part using Python's Tkinter.

How to Submit to GitHub (Requirement 6)

1. Create a folder on your computer named `AI_Game_Project`.
2. Put the code above inside a file `puzzle_game.py`.
3. Go to `github.com` → Create New Repository → Name it `8-Puzzle-AI-Game`.
4. Upload your file manually (click "Add file" → "Upload files") or use Git commands.
5. Copy the link (e.g., `github.com/YourName/8-Puzzle-AI-Game`) and paste it into your teacher's Google Sheet.

⁂

1. https://github.com/johnmichaelbacasno/8-Puzzle-Game
2. https://www.askpython.com/python/examples/easy-games-in-python
3. https://www.youtube.com/watch?v=DDdYFmSXrm0
4. https://www.youtube.com/watch?v=5Kzap4DA_Gw
5. https://www.idtech.com/blog/easy-games-to-make-in-python
6. https://www.scribd.com/document/489811497/8-Puzzle-solving-using-the-A-algorithm-using-Python-and-PyGame-CodeProject
7. https://www.youtube.com/watch?v=b1vKhR7ZHzk
8. https://stackoverflow.com/questions/74074041/python-text-based-game-invalid-moves
9. https://pythongeeks.org/python-slide-puzzle-game-project/
10. https://realpython.com/python-maze-solver/
11. https://github.com/asweigart/PythonStdioGames
12. https://www.linkedin.com/posts/ethan-silverthorne_python-computerscience-artificialintelligence-activity-7374481924708503552-8SUQ
13. https://stackoverflow.com/questions/17737669/how-to-begin-coding-a-brain-for-a-8-puzzle-game-to-calculate-least-moves

14. https://stackoverflow.com/questions/8903259/python-idastar-vs-astar-solving-8-puzzle
15. https://www.youtube.com/watch?v=faf82xmvV74
16. https://mostaql.com/portfolio/2238430-developing-an-8-puzzle-solver-using-a-algorithm-with-tkinter-gui
17. https://github.com/davecom/MazeSolvingGUI
18. https://algomap.io/question-bank/check-if-move-is-legal
19. https://www.educative.io/answers/how-to-solve-the-8-puzzle-problem-using-the-a-star-algorithm
20. https://github.com/aarxa/MazeGame-with-Pathfinding-Algorithms