DOKU+

1. Project Overview

This project is a graphical implementation of the classic **Sudoku puzzle game** using Python and Pygame. The goal is to enhance the traditional Sudoku experience with an intuitive interface, interactive controls, visual feedback, and performance tracking. The user completes a 9×9 grid by filling in digits such that each row, column, and 3×3 box contains all digits from 1 to 9 exactly once. The game supports multiple difficulty levels and logs gameplay data for analysis.

Core Gameplay Mechanics:

- Players select difficulty: Easy (30 empty), Medium (40), Hard (50).
- A partially filled Sudoku grid is generated.
- Players click a cell, input a number, and confirm with Enter.
- Real-time validation alerts users to incorrect inputs.
- Timer tracks completion time.
- Game ends when the player completes a valid grid.

Functionality:

- Interactive Pygame GUI with buttons and keyboard input.
- Visual highlights for selected cells and errors.
- Pause/Resume functionality.
- Hint button to autofill a correct value.
- CSV data logging after each session.
- Menu system with difficulty selection.

2. Project Review

This project builds on the traditional Sudoku game by adding modern UX elements, gameplay tracking, and modular OOP structure.

Improvements Over Existing Projects

- **User Feedback:** Highlights errors, completed rows/columns, and win/loss messages.
- **Timer & Hint System:** Tracks how fast users solve puzzles and helps with stuck positions.
- **Data Tracking:** Logs gameplay data (moves, mistakes, duration, difficulty) for statistical analysis.
- **Difficulty Selection:** Dynamic board generation based on chosen level.

---

3. Programming Development

3.1 Game Concept

Objective: Complete a 9×9 grid using logic so that each number from 1–9 appears only once per row, column, and 3×3 subgrid.

Mechanics:

- Player clicks a cell, types a number, and confirms with Enter.
- The number is validated against the solution.
- Mistakes and invalid attempts are recorded.
- The game ends when the board is correctly filled.

Key Selling Points:

- **Valid Puzzle Generator**: Boards are filled using backtracking to ensure validity and solvability.
- **Timer & Mistake Counter**: Players are scored based on time and accuracy.
- **Hint System**: Helps players fill one correct cell at a time.
- **Gameplay Statistics**: Tracks data for analysis via charts or summary tables.
- **Responsive UI**: Cell selection, visual feedback, and clean layout.
- **Difficulty Scaling**: Easy/Medium/Hard

3.2 Object-Oriented Programming Implementation

1. SudokuBoard (Handles grid data, puzzle generation, and validation)

Attributes:

1. **grid:** 9×9 list of current board values
2. **solution:** Solved version of the current puzzle
3. **difficulty:** Difficulty level selected
4. **selected:** Currently selected cell

Methods:

1. **generate_board( ):** Generates a solvable board
2. **is_valid(row, col, val):** Checks rule validity
3. **check_win( ):** Checks if the board is fully correct
4. **get_empty_cell( ):** Locates an empty space

---

2. SudokuGame (Controls game flow, input, state)

Attributes:

1. **board:** Instance of SudokuBoard
2. **state:** playing/paused/gameover
3. **mistakes,** hints_used, moves
4. **start_time:** Timestamp of start

Methods:

1. **handle_input():** Handles mouse/keyboard events
2. **update_state():** Game logic on player actions
3. **check_endgame():** Detects win/loss

---

3. SudokuRenderer (Handles all drawing logic via Pygame)

Attributes:

1. **screen:** Pygame display surface
2. **cell_size**
3. **colors**
4. **fonts**

Methods:

1. **draw_grid():** Draws board with cells
2. **highlight_cell(row, col):** Visual selection
3. **draw_timer_mistakes():** UI elements

---

4. Menu (Handles difficulty selection and game transitions)

Attributes:

1. **buttons**: Dictionary of difficulty options
2. **selected_difficulty:** Current selection

Methods:

1. **draw_menu( ):** Renders menu
2. **get_selection( ):** Returns chosen difficulty

---

5. Timer (Tracks elapsed time)

Attributes:

1. **start_time:** When the game began
2. **paused:** Boolean
3. **elapsed:** Total time played

Methods:

1. **start( )**
2. **pause( )**
3. **resume( )**
4. **get_time( ):** Returns elapsed time in MM:SS

---

6. StatsTracker

Attributes:

1. **data:** Dictionary of tracked values
2. **filename:** CSV file for storage

Methods:

1. **log_stats( ):** Writes session data to file
2. **get_summary( ):** Returns stats summary

3.3 Algorithms Involved

Backtracking (Recursive Search)
Core algorithm used to generate a valid and complete Sudoku board. It recursively attempts to place numbers and backtracks upon encountering invalid states.

Rule-Based Logic
Used to validate number placement — ensuring no repetitions in rows, columns, or 3×3 boxes. This drives the game's move-checking and correctness logic.

Sorting Algorithms
Useful in leaderboard ranking (e.g., by time or mistakes) and stat dashboards, using Merge Sort or Quick Sort for efficiency.

Binary Search
When searching for a value in a sorted dataset, such as when retrieving past stats, verifying leaderboard entries, or implementing fast filtering.

4. Statistical Data (Prop Stats)

4.1 Data Features

1. **Time Played**
2. **Mistakes**
3. **Difficulty Level**
4. **Win/Loss**
5. **Hints Used**
6. **Board Completion Rate**

4.2 Data Recording Method

- **Storage Format:** The game data will be stored in a **CSV file** for easy data handling and analysis.
- **File Handling:**
  - ★ A new CSV file will be created if it does not already exist.
  - ★ Each round's data will be **appended** to the file after the game ends.

★ If the file already exists, the new data will be added without overwriting previous data.

4.3 Data Analysis Report

Graphs:

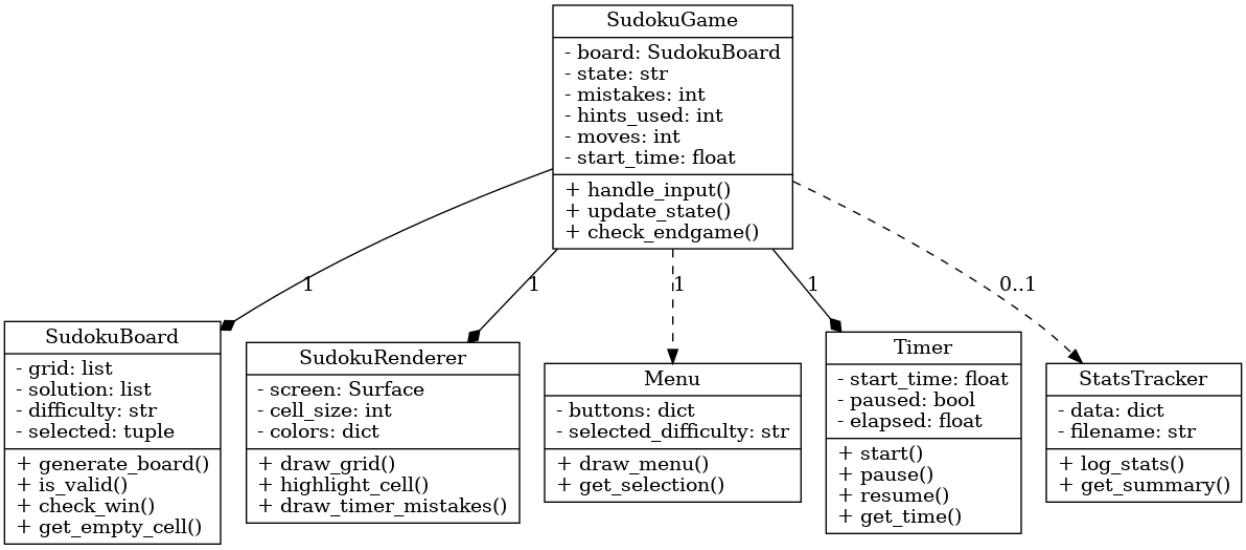
- **Line**: Time per puzzle
- **Bar**: Mistakes per difficulty
- **Pie**: Hint usage distribution
- **Histogram**: Moves per puzzle

Summary Stats:

- Mean/Median time and moves
- Win rate by difficulty
- Max streaks, fastest solve time

5. Project Timeline

Week	Task
1 (10 March)	• Proposal submission / Project initiation
2 (17 March)	• Full proposal submission
3 (24 March)	• Develop game engine and core mechanics
4 (31 March)	• Implement scoring and feedback system
5 (7 April)	• Add special blocks and combo mechanics
6 (14 April)	• Testing, debugging, and final adjustments • Finalize documentation and submit • Submission week (Draft)



Feature	Why is it good to have this data? What can it be used for?	How will you obtain 50 values of this feature data?	Which variable (and which class) will you collect this from?	How will you display this feature data (via summarization statistics or via graph)?
Time Played	Measures session duration, skill improvement, and engagement. Helps identify difficulty-performance trends.	Tracked for every game played	Timer.get_time()	Summary stats (mean, median, mode) and line graph
Mistakes	Indicates player accuracy and difficulty of puzzles. High mistake rates suggest harder boards or riskier play.	Incremented on incorrect inputs	SudokuGame.mistakes	Bar graph by difficulty
Difficulty Level	Used to compare performance (time, accuracy) across Easy, Medium, Hard.	Stored per session	Menu.selected_difficulty	Grouped analysis tables
Win/Loss Outcome	Measures how often players complete puzzles vs. give up or lose. Useful for balancing and UX review.	Recorded at end of each session	SudokuGame.state	Pie chart or percentages
Hints Used	Reflects puzzle complexity and player reliance on support. Can suggest difficulty tuning needs.	Tracked each time the player clicks “Hint”	SudokuGame.hints_used	Pie chart or bar graph

Board Completion Rate	Percent of cells filled before exiting or winning. Shows user effort, especially on loss.	Tracked on exit or game over	SudokuBoard.filled_ratio()	Bar graph

Statistical Data Revision

1.1 (A) Feature for Statistical Table

Feature	**Statistical Values to Show in the Table**
Time Played	Mean, Median, Min, Max, Standard Deviation

1.2 (B) Graphs Specification Table

Feature	**Graph Objective**	**Graph Type**	**X-axis**	**Y-axis**
Time Played	Analyze player completion speed trends	Line Graph (Time Series)	Session Number	Time (seconds)
Mistakes	Compare mistake rates across difficulties	Bar Graph (Distribution)	Difficulty Level	Average Mistakes
Win/Loss Outcome	Show percentage of completed vs. failed sessions	Pie Chart (Proportion)	-	-
Hints Used	Show frequency of hint usage by players	Pie Chart (Proportion)	-	-

Time Played per Difficulty	Compare how puzzle completion time varies between Easy, Medium, and Hard difficulties	Boxplot (Distribution)	Difficulty Level	Time Played (in seconds)
Time vs Mistakes	Explore correlation between time taken and mistakes made	Scatter Plot (Relation)	Time Played (seconds)	Mistake Count

2. Project Planning

2.1 Weekly Planning

Week	Planned Tasks
26 Mar – 2 Apr	Finalize core game logic, validate board generator, ensure working prototype
3 Apr – 9 Apr	Add scoring, timer, mistake tracking, win/loss validation
10 Apr – 16 Apr	Add data logging (CSV), implement difficulty selection and stat collection
17 Apr – 23 Apr	Implement graph analysis scripts (matplotlib), build Tkinter analytics UI
24 Apr – 11 May	Final UI polish, export graphs, test documentation, and finalize submission

2.2 Milestone: 50% Completed by 16 April

- Fully playable Sudoku game
- Difficulty selection implemented
- Timer, mistakes, win/loss tracking
- Data logging to CSV

2.3 Milestone: 75% Completed by 23 April

- Graph generation working (3 types)
- Data analysis with pandas, matplotlib
- GUI preview of statistics with Tkinter

2.4 Milestone: 100% Completed by 11 May

- Final graphs and table polished
- Hand-drawn graph sketch submitted
- Final report completed and submitted
- All features bug-tested and working

Project Change Information

During the development of the original Wordris project, I encountered significant technical limitations that affected the feasibility and performance of the game. The core mechanic—clearing rows by forming valid English words—required real-time word validation against a very large vocabulary. This created major challenges, especially when running the game on different systems or environments. The validation process became slow and unpredictable, often causing delays that made it difficult or impossible for the game to register cleared rows in time. Additionally, implementing a robust and efficient word-checking system across platforms proved to be more complex than expected. As a result, I decided to pivot to a **DOKU+** project, which offers a more structured, logic-based challenge that still allows for statistical tracking, interactive gameplay, and algorithmic depth—while being more stable and consistent across systems.

Since shifting to the new **DOKU+** project, significant progress has been made. The core gameplay is already fully implemented, including a working puzzle generator, board validation, timer, mistake tracking, and difficulty selection. The game runs smoothly with an interactive Pygame interface and supports all essential player interactions such as cell selection, number input, and win/loss detection. Additionally, the codebase has been restructured into multiple object-oriented classes (SudokuGame, SudokuBoard, SudokuRenderer, Timer, Menu, and StatsTracker) to align with best practices. The data tracking system is also underway, logging session data into a CSV file and preparing for statistical analysis using both tables and graphs. We are well on track to meet the project milestones, with over 50% of the work already completed.

6. Document version

Version: *4.0*
Date: *31 March 2025*