Coursework 1: Developing a Sudoku Game

Course: 4COM2015 Problem-Solving and Programming
Total Marks: 100
Submission Deadline: **10th of April 2025 at 23:59**

Overview

In this coursework, you will develop a **Sudoku game** in two stages:

1. **Part 1**: Develop the game using Python **in the terminal (command-line interface)**.
2. **Part 2**: Extend the game by **adding a web interface using Flask and HTML**.

Throughout this coursework, you will apply **problem-solving strategies**, implement algorithms, and use a **conceptual modelling technique** to represent the software design.

Part 1: Developing a Terminal-Based Sudoku Game (50 Marks)

Requirements

1. **Conceptual Modelling Technique (10 Marks)**
    - Choose an appropriate method for representing your software, such as:
        - **Entity-Relationship Diagrams (ERDs)**
        - **Unified Modelling Language (UML) Diagrams**
        - **Activity Diagrams**
        - **Statecharts**
    - Provide a brief **description** and **justification** for your chosen method.

2. **Algorithm and Problem-Solving Strategy (10 Marks)**
    - Describe the **algorithm** used for solving Sudoku.
    - Explain the **strategy** behind **checking valid moves** in the game.

3. **Implementation of the Sudoku Game (20 Marks)**
    - Provide a **function** for printing the Sudoku board in the terminal.
    - Implement an **algorithm** for solving the Sudoku board.
    - Develop a function for **generating new Sudoku boards** using Python's random number generator.
    - Allow **user interaction in the terminal**:
        - The game **displays the board**.

        - The user **inputs numbers** to fill empty spaces (e.g. to change `0` values).
        - A function checks if the number entered is a **valid move**.
        - If needed, the **solve function** should solve the Sudoku board.

4. **Evaluation of the System (10 Marks)**

    - Explain how you will **test** the Sudoku game to ensure it works correctly.
    - Describe methods to verify **correctness and efficiency**.

Expected Workflow for Part 1

1. The game **initialises the board** with predefined numbers and empty spaces.
2. The game **displays the board** in the terminal.
3. The player **enters a number** and its position.
4. The system **checks if the number is a valid move**.
5. If valid, the number is placed; if not, an error message is shown.
6. The **game continues** until the board is completed.
7. The player can choose to **solve the board automatically** using the proposed algorithm.

Part 2: Developing a Web-Based Interface using Flask (50 Marks)

Requirements

1. **Conceptual Modelling Technique (10 Marks)**
    - Choose an appropriate modelling technique (e.g., **ERDs, UML diagrams, Statecharts**).
    - Provide a **brief explanation** of how it represents your system.

2. **Adapting the Sudoku Game to a Web-Based System (20 Marks)**
    - Convert the **terminal-based game** to a **Flask web application**.
    - Use an **HTML table** to display the Sudoku board.
    - Create a **simple web form** to allow users to input numbers.
    - Ensure **minimal modifications** to the existing Python logic.

3. **Implementation of Web-Based Sudoku (10 Marks)**
    - The **Flask app serves the game** to users via a web browser.
    - The board is displayed as a **9x9 HTML table**.
    - Users **enter a row, column, and number** through an HTML form.
    - The Python backend **validates the input and updates the board**.
    - The system **notifies the user of invalid moves**.
    - The game runs until **all cells are filled**.
4. **Evaluation of the System (10 Marks)**
    - Describe how you will **test the Flask-based Sudoku game**.
    - Explain methods to check for **correctness, efficiency, and usability**.

Expected Workflow for Part 2

1. A **Flask web server** runs the application.
2. The **game board is displayed** as an HTML table.
3. Users **input numbers** through a form.
4. The Flask backend **validates moves** and updates the board.
5. The game continues until **all cells are filled**.
6. Invalid moves are rejected with **error messages**.
7. The player can **refresh the page** to restart the game.

Submission Guidelines

- Submit **Python source code** for **Part 1** and **Part 2**.
- Include **screenshots of the working game** in both the terminal and web versions.
- Submit a **short report** (2-3 pages) covering:
    - **Conceptual modelling technique** used.
    - **Algorithm description**.
    - **Evaluation methods** for both versions.
- Package all files in a **ZIP folder** and submit via the course portal.

Assessment Criteria

Criterion	Marks
Part 1: Terminal-Based Game	**50 Marks**
Conceptual Modelling	10
Algorithm & Problem-Solving Strategy	10
Implementation of the Game	20
Evaluation	10
Part 2: Web-Based Game (Flask)	**50 Marks**
Conceptual Modelling	10
Web-Based Adaptation	20
Implementation of Web Sudoku	10
Evaluation	10
Total	**100 Marks**

Final Notes

- Ensure your code is **well-commented** and follows best practices.
- Keep the **Python logic in Part 2 as close as possible** to Part 1.
- You are **encouraged to use GitHub/GitLab** for version control.