Assignment 1

Hangman Game

In this assignment, you will create a terminal-based Hangman game using a modular Python project structure. The goal is to practice organizing code across files and folders, reading data from files, input validation, simple game-state management, and producing clean, readable code. The game must support categories, a large wordlist (≥1000 words), scoring, persistent statistics, and a simple ASCII-art hangman. The program must run by executing main.py only.

Project Structure:

Create a project folder named hangman_game with this structure:

```
hangman_game/
 ├── main.py
 ├── words/
 │   ├── words.txt          # at least 1000 words, one per line
 │   └── categories/        # optional: separate files per category
 ├── game/
 │   ├── engine.py          # core gameplay logic
 │   ├── wordlist.py        # word loading and random selection
 │   └── ascii_art.py       # hangman drawing (bonus)
 ├── ui/
 │   └── display.py         # printing progress and game messages
 ├── game_log/
 │   └── game1/
 │       └── log.txt        # created automatically each game
 ├── README.md
```

The main.py file should serve as the **entry point** of your program, containing only the top-level logic. All helper functions should be kept in their respective modules inside the folders.

Implementation Tasks:

1. Basic word guessing functionality
   - Player guesses letters, game reveals correct letters and keeps unknowns as underscores.
   - Repeated guessing of a revealed letter should be handled (inform the player; do not penalize).
2. Random word selection
   - Randomly pick words from the provided wordlist.
   - The project must include at least 1000 words in words/words.txt or across category files.
   - Support selecting by category. If no category is chosen, select from all words.
3. Simple text-based interface
   - Clear prompts for user input.
   - Show the current word progress (e.g., _ e _ _ o _), letters guessed, and remaining wrong guesses.
4. Win / lose conditions
   - Player wins when all letters are revealed.
   - Player loses when they reach 6 wrong guesses (maximum allowed wrong guesses = 6).
5. Letter validation
   - Accept only single alphabetic characters for letter guesses.
   - Accept an option to guess the full word (optional feature) — if implemented, wrong full-word guess counts as 1 wrong guess.
   - Ignore case (treat 'A' and 'a' the same).

- Disallow non-letter inputs and multi-character inputs except the special full-word guess command (if implemented).

6. Progress tracking
    - Show list of guessed letters (correct and incorrect).
    - Update and display remaining attempts (6 → 0).
    - Display current ASCII-art hangman stage.
7. Clean, readable code
    - Use functions and small modules; keep main.py as the program entry point only.
    - Use docstrings and comments where helpful.
    - Follow PEP8 where reasonable.
8. Categories
    - Provide at least these categories: Animals, Countries, Programming, Science.
    - Words must be reasonably matched to their category.
9. Scoring
    - Score a round when the player wins; losing yields 0 points for that round (e.g. guessed 'python' (length=6) with 2 wrong guesses → 6*10 - 2*5 = 60 - 10 = 50 points).
    - Display the score gained for each win and add it to total_score.
    - Track and display statistics across runs (games_played, wins, losses, total_score, win_rate (%), average_score_per_game)
10. Logging
    - Create a separate folder inside game_log/ for every new game (e.g. game_log/game1/, game_log/game2/, etc)
    - Each new game must create a new folder automatically without overwriting previous logs. Inside each folder, write a log.txt file containing:
        - Selected category and word (hidden during gameplay)
        - Player’s guesses (correct and incorrect)

    - Number of wrong guesses
    - Final result (Win/Loss)
    - Score for the game
    - Timestamp

11. Bonus

    - ASCII-art hangman drawing that updates with each wrong guess (0–6 states). Include a simple drawing for each state.

```
 +---+
 |  |
 O  |
/|\ |
/ \ |
    |
=========
```

Sample Gameplay:

```
Welcome to Hangman!

Choose a category (Animals, Countries, Programming, Science):
Programming

New word selected from 'Programming' (length 6)

_ _ _ _ _ _

Guessed letters:

Remaining attempts: 6


Enter a letter (or type 'guess' to guess full word, 'quit' to exit): p

Correct! Progress: p _ _ _ _ _

Guessed letters: p
```

```
Remaining attempts: 6

[ASCII drawing hangman state 0]


Enter a letter: y

Correct! Progress: p y _ _ _ _

Guessed letters: p, y

Remaining attempts: 6


Enter a letter: z

Wrong! Progress: p y _ _ _ _

Guessed letters: p, y

Remaining attempts: 5

[ASCII drawing hangman state 1]


Enter a letter: t

Correct! Progress: p y t _ _ _

Guessed letters: p, y, t

Remaining attempts: 5


Enter a letter: h

Correct! Progress: p y t h _ _

Guessed letters: p, y, t, h

Remaining attempts: 5


Enter a letter: o

Correct! Progress: p y t h o _

Guessed letters: p, y, t, h, o
```

```
Remaining attempts: 5

Enter a letter: n
Correct! Progress: p y t h o n
Guessed letters: p, y, t, h, o, n
Remaining attempts: 5



You win! Word: python
Points earned this round: 50
Total score: 150

Games played: 3  Wins: 2  Losses: 1  Win rate: 66.67%
```

Example log.txt for this session, content to save in game_log/game3/log.txt

```
Game 3 Log
Category: Programming
Word: python
Word Length: 6

Guesses (in order):
1. p  → Correct
2. y  → Correct
3. z  → Wrong
4. t  → Correct
5. h  → Correct
```

```
6. o  → Correct

7. n  → Correct


Wrong Guesses List: z

Wrong Guesses Count: 1

Remaining Attempts at End: 5


Result: Win

Points Earned: 55

Total Score (after this round): <previous_total> + 55


Games Played: 3

Wins: 2

Losses: 1

Win Rate: 66.67%


Date & Time: 2025-10-18 16:45:00

---------------------------------------

Session Notes:

- ASCII hangman reached state 1 after wrong guess 'z'.

- Progress trace:

  _ _ _ _ _ _

  -> p _ _ _ _ _

  -> p y _ _ _ _

  -> p y _ _ _ _   (z wrong — no progress change)

  -> p y t _ _ _

  -> p y t h _ _
  -> p y t h o _
  -> p y t h o n
--------------------------------------
```

Submission Instructions

- Upload your full hangman_game/ folder to a GitHub repository.
- Only working repositories will be graded.
- Ensure main.py runs correctly without manual execution of internal files.
- Include a basic README.md explaining:
    - How to run the game
    - Wordlist format and categories used
    - Scoring method
    - How logs are saved

Important Reminders

- Use **pathlib** for all directory and file operations, not manual string paths.
- Keep your main.py focused only on controlling game flow — avoid writing direct logic inside it.
- Create all new folders dynamically using mkdir(parents=True, exist_ok=True) so the program does not crash if folders already exist.
- After each move, make sure both the console output and the log file are updated.
- Always follow clean modular programming principles: each .py file must have a clear responsibility.