BY: Debasmita Sahu (2320520037)

Project Title : Hangman Game

ABSTRACT

The **Hangman Game** project is an expanded digital adaptation of the classic word-guessing game, combining both fun and educational elements. It’s designed to offer an engaging and interactive experience suitable for players of all ages and skill levels. The aim is to challenge players to guess a hidden word by suggesting letters within a limited number of attempts. With each correct guess, players get closer to unveiling the word, while incorrect guesses gradually form the visual of a hanged man, maintaining the essence of the classic version. Hangman is designed as an engaging and interactive application suitable for players of all ages. The aim of the game is to challenge players to correctly guess a hidden word by suggesting letters within a limited number of attempts.

This project is an excellent demonstration of various programming concepts and techniques, making it a practical and enjoyable exercise for learning and skill development. The game is developed using **Python** programming languages leveraging libraries and frameworks like **Pygame and HTML/CSS** to create an interactive and visually appealing user interface. The game incorporates **random word generation algorithms** that pull from either a pre-defined static dictionary or dynamically sourced APIs, ensuring a diverse and challenging gameplay experience.

The game development utilizes Python programming languages along with libraries to create an interactive and visually appealing user interface. The game incorporates random word generation algorithms that pull from a pre-defined or dynamic dictionary of words, ensuring a wide variety of challenges for players. The user interface displays the hidden word as a series of blank spaces and reveals correct guesses, while incorrect guesses reduce the number of available attempts and gradually complete the visual of the hanged man, maintaining the classic essence of the game.

To enhance user engagement, the game features intuitive controls and responsive design, allowing players to easily input their guesses. The scoring system tracks the number of attempts used and provides feedback for each guess, with messages indicating win or loss at the end of the game. The game also includes options for replay and choosing difficulty levels for a more personalized experience.

This project serves as a platform for exploring fundamental and advanced programming concepts such as loops, conditionals, functions and object-oriented programming. Additionally, it emphasizes the importance of user experience (UX) and graphical user interface (GUI) design, offering insights into how game mechanics and visual elements can be combined for an engaging application. The Hangman Game is not only an enjoyable gaming experience but also a valuable exercise for enhancing programming skills, logical thinking, and problem-solving abilities.

Future developments may include implementing multiplayer modes, online leader boards, or mobile compatibility, expanding the scope and complexity of the game while providing opportunities to explore network programming and cross-platform development.

**INTRODUCTION**

It’s a simple hangman game.

The player has to guess the letter in the word. If you get a wrong word, a body part will be added to the hangman.

The player can view “How to play” by clicking on “How to play” button.

The hangman body will be displayed as an image.

**MODULES**

1. Main Module (main.py)

* This is the entry point for the game.
* Manages the overall game flow, including game initialization, difficulty selection, and handling user input.
* Imports other modules (game engine, UI, word generator) and coordinates them to create a cohesive experience.

2. Game Engine Module (game\_engine.py)

* Handles the core mechanics of the game, such as:
  + Word Selection: Retrieves a word for the player to guess.
  + Tracking Guesses: Keeps track of the letters guessed (correct and incorrect) and the number of remaining attempts.
  + Win/Loss Conditions: Determines if the player has won (all letters guessed) or lost (no attempts left).
* Provides functions like:
  + initialize\_game(): Sets up the initial state of the game.
  + process\_guess(letter): Processes user input and updates the game state.
  + check\_win(): Checks if the player has won.
  + check\_loss(): Checks if the player has lost.

3. User Interface Module (ui.py)

* Manages the display and user input, either through a console interface or a graphical interface using libraries.
* Responsible for:
  + Displaying the current state of the word (with blank spaces and correctly guessed letters).
  + Showing the number of attempts remaining and the visual representation of the hangman.
  + Prompting the player for input and displaying feedback (e.g., messages indicating correct/incorrect guesses, win/loss messages).
* Functions might include:
  + display\_word(current\_word): Displays the word with guessed letters revealed.
  + display\_attempts\_left(attempts): Shows the remaining attempts.
  + display\_hangman(stage): Visual representation of the hangman based on the number of wrong guesses.
  + get\_user\_input(): Collects and validates user input.

4. Word Generator Module (word\_generator.py)

* Manages the selection of random words for the game.
* Can pull words from:
  + Local Dictionary: A list of words stored in a file (words.txt).
  + API Integration: (Optional) Fetches words from an online dictionary API to provide a dynamic and varied set of words.
  + Custom Word List: (Optional) Allows the user to upload or input their custom list of words.
* Key functions include:
  + get\_random\_word(difficulty): Returns a random word based on the difficulty level (easy, medium, hard).
  + load\_words\_from\_file(file\_path): Loads words from a local file.
  + fetch\_word\_from\_api(): (Optional) Fetches words from an online API.

5. Scoring and Statistics Module (scoring.py)

* Manages the game scoring system and player statistics.
* Keeps track of:
  + The number of attempts used.
  + The player’s performance (wins/losses, highest score, etc.).
  + Storing scores in a local file (scores.json) or using a database for long-term tracking.
* Functions include:
  + calculate\_score(attempts\_left, word\_length): Calculates the score based on attempts left and word length.
  + save\_score(player\_name, score): Saves the score to a file or database.
  + display\_leaderboard(): Displays high scores and player statistics.

6. Graphics and Visuals Module (graphics.py) *(Optional, for GUI versions)*

7. Testing Module (test\_game.py)

8. Additional Features:

* BUTTONS WHICH TIGGERS ON ACTION WHEN CLICKED : the buttons for “How to play” and 26 letters buttons.
* DISPLAY A POP-UP MESSAGE : Basically, a message about wrong or right guess.
* DISPLAY AN IMAGE : The hangman body will be an image.
* PLAY A SOUND FILE : I will play a background music to let the player enjoy my app.

**HOW TO PLAY**

1. There will be an array which will have a collection of words and my app will choose one of those words by a random way.
2. Then, it will be shown on the screen like this( \_ \_ \_ \_ \_ \_ ) according to the number of letters in the word.
3. You (the player) has to guess the word, what will be ?
4. Then, you should enter any letter of the word, you have guessed and then click check.
5. If this letter is correct or if it is present at any position of the word, my app will display a pop-up message (GOOD GUESS !) and if it is wrong or the letter is not present in the word, it will display (BAD GUESS !).
6. If you enter 10 letters wrong, you will be dead means you will lose the game.

**SOURCE CODE**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Hangman Game Interface</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f0f0f0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.container, .game-screen {

background-color: #ffffff;

padding: 20px;

border-radius: 8px;

width: 600px;

box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);

text-align: center;

display: none;

}

.container.active, .game-screen.active {

display: block;

}

h1 {

margin-bottom: 20px;

}

.options {

display: flex;

flex-direction: column;

gap: 15px;

}

.option {

display: flex;

align-items: center;

padding: 10px;

background-color: #f7f7f7;

border: 1px solid #ccc;

border-radius: 5px;

cursor: pointer;

transition: background-color 0.3s;

}

.option:hover {

background-color: #e0e0e0;

}

.icon {

width: 40px;

height: 40px;

background-color: #4CAF50; /\* Default color \*/

border-radius: 50%;

display: flex;

justify-content: center;

align-items: center;

margin-right: 10px;

}

.option:nth-child(2) .icon {

background-color: #9c27b0; /\* Purple for timed \*/

}

.option:nth-child(3) .icon {

background-color: #ff9800; /\* Yellow for untimed \*/

}

.icon img {

width: 20px;

height: 20px;

}

.text-container {

flex-grow: 1;

text-align: left;

}

.main-text {

font-weight: bold;

}

.sub-text {

font-size: 0.9em;

color: #666;

}

/\* Game screen styles \*/

.game-screen {

background-color: #f0f0f0;

border-radius: 10px;

padding: 20px;

width: 600px;

text-align: left;

}

.game-header {

display: flex;

justify-content: space-between;

align-items: center;

margin-bottom: 20px;

}

.hint-button {

background-color: #ffa500;

color: white;

padding: 5px 10px;

border: none;

border-radius: 5px;

cursor: pointer;

}

.word-display {

margin: 20px 0;

font-size: 24px;

letter-spacing: 5px;

text-align: center;

}

.letter-grid {

display: grid;

grid-template-columns: repeat(6, 50px);

gap: 10px;

justify-content: center;

}

.letter {

width: 40px;

height: 40px;

display: flex;

justify-content: center;

align-items: center;

background-color: #3498db;

color: white;

border-radius: 50%;

cursor: pointer;

}

/\* Hangman Styles \*/

.hangman {

font-family: monospace;

font-size: 16px;

white-space: pre;

text-align: center;

margin-top: 20px;

}

</style>

</head>

<body>

<div class="container active" id="mainMenu">

<h1>Welcome to Hangman!</h1>

<div class="options">

<div class="option" onclick="startGame('timed')">

<div class="icon">

<img src="https://img.icons8.com/ios-filled/50/ffffff/hourglass.png" alt="Timed Icon">

</div>

<div class="text-container">

<div class="main-text">Single Player - Timed</div>

<div class="sub-text">Play by yourself, time limit 2 minutes</div>

</div>

</div>

<div class="option" onclick="startGame('untimed')">

<div class="icon">

<img src="https://img.icons8.com/ios-filled/50/ffffff/user.png" alt="Untimed Icon">

</div>

<div class="text-container">

<div class="main-text">Single Player - Untimed</div>

<div class="sub-text">Play by yourself, no time limit</div>

</div>

</div>

</div>

</div>

<div class="game-screen" id="gameScreen">

<div class="game-header">

<button class="hint-button">Hint (Alt+L)</button>

<span id="timer">02:00</span>

<div>

<div id="points">0 points</div>

<div id="streak">streak: 0</div>

</div>

</div>

<div class="word-display" id="wordDisplay">\_ \_ \_ \_ \_</div>

<div class="letter-grid" id="letterGrid">

<!-- Alphabet buttons will be inserted here by JavaScript -->

</div>

<div class="hangman" id="hangmanDisplay"></div> <!-- Hangman display -->

</div>

<script>

const words = ["python", "hangman", "challenge", "programming", "developer", "keyboard"];

let selectedWord, wordCompletion, guessedLetters, tries, points, timerInterval;

let timerCount = 120; // 2 minutes

const hangmanStages = [

`

------

| |

| O

| \\|/

| |

| / \\

-

`,

`

------

| |

| O

| \\|/

| |

| /

-

`,

`

------

| |

| O

| \\|/

| |

|

-

`,

`

------

| |

| O

| \\|

| |

|

-

`,

`

------

| |

| O

|

| |

|

-

`,

`

------

| |

| O

|

|

|

-

`,

`

------

| |

|

|

|

|

-

`

];

function startGame(mode) {

document.getElementById('mainMenu').classList.remove('active');

document.getElementById('gameScreen').classList.add('active');

setupGame(mode);

}

function setupGame(mode) {

selectedWord = words[Math.floor(Math.random() \* words.length)];

wordCompletion = "\_ ".repeat(selectedWord.length).trim();

guessedLetters = [];

tries = 6;

points = 0;

document.getElementById('wordDisplay').textContent = wordCompletion;

document.getElementById('points').textContent = `${points} points`;

document.getElementById('hangmanDisplay').textContent = hangmanStages[tries];

generateLetterButtons();

if (mode === 'timed') {

timerCount = 120; // Reset the timer

startTimer();

}

}

function startTimer() {

const timerElement = document.getElementById('timer');

timerInterval = setInterval(() => {

if (timerCount > 0) {

timerCount--;

const minutes = String(Math.floor(timerCount / 60)).padStart(2, '0');

const seconds = String(timerCount % 60).padStart(2, '0');

timerElement.textContent = `${minutes}:${seconds}`;

} else {

clearInterval(timerInterval);

alert(`Time's up! The word was ${selectedWord}. Try again!`);

resetGame();

}

}, 1000);

}

function generateLetterButtons() {

const letterGrid = document.getElementById('letterGrid');

letterGrid.innerHTML = '';

for (let i = 65; i <= 90; i++) {

const letter = String.fromCharCode(i);

const button = document.createElement('div');

button.textContent = letter;

button.className = 'letter';

button.onclick = () => handleGuess(letter.toLowerCase());

letterGrid.appendChild(button);

}

}

function handleGuess(letter) {

if (guessedLetters.includes(letter)) {

alert(`You already guessed the letter "${letter}"`);

return;

}

guessedLetters.push(letter);

if (!selectedWord.includes(letter)) {

tries--;

document.getElementById('hangmanDisplay').textContent = hangmanStages[tries];

if (tries === 0) {

alert(`Game Over! The word was "${selectedWord}".`);

resetGame();

return;

}

} else {

points += 10; // Increment points for correct guess

}

updateWordDisplay();

document.getElementById('points').textContent = `${points} points`;

}

function updateWordDisplay() {

wordCompletion = selectedWord.split('').map(letter => guessedLetters.includes(letter) ? letter : "\_").join(' ');

document.getElementById('wordDisplay').textContent = wordCompletion;

if (!wordCompletion.includes("\_")) {

alert(`Congratulations! You guessed the word "${selectedWord}"!`);

resetGame();

}

}

function resetGame() {

clearInterval(timerInterval);

document.getElementById('mainMenu').classList.add('active');

document.getElementById('gameScreen').classList.remove('active');

}

</script>

</body>

</html>

**SCOPE OF THE PROJECT**

1. The scope of this project is to polish our skills and for entertainment purpose.
2. After developing the proposed features, I will try to enhance some functionalities like
   * 1. Saving the best score for each player (the count of word it guesses)
     2. Add time to know how long the player take to guess the word.
     3. Try to turn it into a beautiful interface.

**SAMPLE USER INTERFACE**



