**Software Engineering**

Assessment task 1:

Python Hangman

By Peter

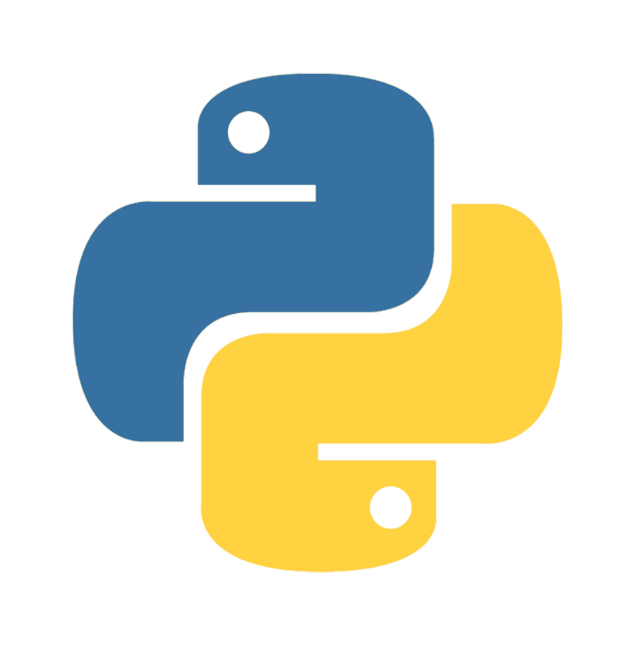
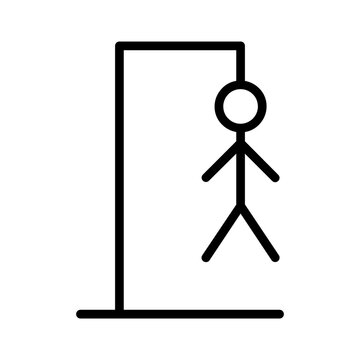


Table of contents

Gantt Chart……………………………………………………pg 3  
Justification for programming language…………..pg 4-5

User Interface (UI)………………………………………..pg 6-9

Storyboard…………………………………………………..pg 10

Pseudocode…………………………………………….pg 11-17

Flowchart…………………………………………………….pg 18

UML Chart……………………………………………………pg 19

Data Dictionary………………………..………………pg 20-25

Version Control………………………………………..pg 26-30

Logbook…………………………………..……………..pg 31-33

Gantt Chart

A blue graph with white text

AI-generated content may be incorrect.

Justification for Programming Language

**1. Readability & Simplicity**

Python's clean and intuitive syntax makes it easy for beginners to understand and write code.

* **Example**: The use of lists (WORDS = ["IDE", "PIP", ...]) and dictionaries for game logic is straightforward.
* **No complex syntax**: Unlike languages like C++ or Java, Python avoids excessive boilerplate (e.g., no semicolons, brackets are minimal).

**2. Rich Library Support (Pygame)**

Python’s **Pygame** library simplifies game development:

* **Easy graphics handling**: Loading images (pygame.image.load()), rendering text (pygame.font), and drawing shapes (pygame.draw) are beginner-friendly.
* **Event-driven programming**: Mouse clicks (pygame.MOUSEBUTTONDOWN) and keyboard inputs are easy to implement without low-level complexity.

**3. Rapid Prototyping & Debugging**

* **No compilation step**: Python runs code immediately, making testing and debugging faster.
* **Interactive mode**: Beginners can test small parts of the game (e.g., button logic) in a Python shell.

**4. Cross-Platform Compatibility**

* The game works on **Windows, macOS, and Linux** without modification, thanks to Python’s portability.

**5. Beginner-Friendly Community & Resources**

* Python has **extensive documentation** and tutorials, making it easier for learners to find help (e.g., Pygame tutorials, Stack Overflow support).

**6. Built-in Data Structures for Game Logic**

* Lists (letters = []), strings, and loops simplify game mechanics like:
  + Tracking guessed letters (guessed = [])
  + Managing button states (buttons = [x, y, letter, visible])

**In conclusion**

Python’s **simplicity, readability, and Pygame integration** make it the best choice for beginners creating a **Hangman Game**. It allows learners to focus on **game logic** rather than complex syntax or low-level details, making the development process **fun and educational**.

User Interface (UI)

**1. Main Game Window**

* **Dimensions**: 1200x800 pixels (width × height).
* **Background**: White (WHITE = (255, 255, 255)).
* **Title**: "DEVELOPER HANGMAN" displayed at the top in a large, bold font.

**2. Hangman Visual (Left Side)**

* **Image Progression**:
  + Shows **7 stages** of the hangman (from hangman0.png to hangman6.png).
  + Updates with each incorrect guess.
* **Position**: Centered vertically, aligned to the left (x=200, y=150).

**3. Word Display (Center-Top)**

* **Format**:
  + **Underscores (\_)** for unguessed letters.
  + **Revealed letters** when correctly guessed (e.g., \_ \_ E \_ for "CODE" if 'E' is guessed).
* **Font**: Large and bold (WORD\_FONT = comicsans, size 60).
* **Position**: Centered horizontally (x=500, y=300).

**4. Letter Buttons (Bottom Section)**

* **Layout**:
  + **Two rows** (A-M on top, N-Z on bottom).
  + **Circular buttons** with a radius of 20 pixels.
* **Interaction**:
  + **Clickable** (disables after selection).
  + **Correct guess**: Letter appears in the word.
  + **Wrong guess**: Hangman progresses, button turns inactive.
* **Position**: Centered horizontally (startx = calculated, starty = 600).

**5. Hint System (Top-Right)**

* **Hint Button**:
  + **Appears** when at least **1 hint is available**.
  + **Green color** (GREEN = (100, 255, 100)).
  + **Label**: "Use Hint" (clicking reveals a random missing letter).
* **Hint Counter**:
  + Displays remaining hints (e.g., Hints: 2).
  + **Earned** after every **2 incorrect guesses**.

**6. Endgame Screens (Win/Lose)**

* **Message Display**:
  + **"You WON!"** (if all letters are guessed).
  + **"You LOST!"** (if hangman completes).
* **Action Buttons**:
  + **Retry Button** (Gray, restarts the game).
  + **Quit Button** (Gray, exits the game).

**UI Flow Summary**

1. **Start Screen** → Hangman image (stage 0) + word blanks.
2. **Gameplay** → Click letters, see feedback (correct/wrong).
3. **Hint System** → Use hints to reveal letters.
4. **Endgame** → Win/lose message + retry/quit options.

**Why This UI Works Well for Juniors?**

✅ **Simple, clean layout** (no clutter).  
✅ **Visual feedback** (hangman progression, button states).  
✅ **Interactive elements** (buttons, hints).  
✅ **Clear win/lose conditions**.

Storyboard

A screenshot of a computer

AI-generated content may be incorrect.

Pseudocode

# ===== PSEUDOCODE FOR DEVELOPER HANGMAN GAME =====

# INITIALIZATION

Initialize pygame

Set window dimensions (WIDTH, HEIGHT)

Create game window with title "Developer Hangman"

# CONSTANTS

Define RADIUS = 20 (button size)

Define GAP = 15 (spacing between buttons)

Define A = 65 (ASCII for 'A')

Define word list WORDS with developer terms

Define colors (WHITE, BLACK, GRAY, GREEN)

# FUNCTION: initialize\_letter\_buttons()

Create empty buttons list

Calculate starting x position to center buttons

Set starting y position

For each letter A-Z:

Calculate x,y position in 2 rows (13 letters per row)

Add button to list with [x, y, letter, visible=True]

Return buttons list

# FUNCTION: reset\_game()

Reset global variables:

hangman\_status = 0

word = random choice from WORDS

guessed = empty list

hint\_available = 0

incorrect\_guesses = 0

Set all letter buttons to visible

# FUNCTION: draw\_end\_buttons()

Create Retry and Quit buttons:

Calculate centered positions

Draw gray rectangles with black borders

Add "Retry" and "Quit" text

Return button rectangles for collision detection

# FUNCTION: draw()

Fill screen with white

Display "DEVELOPER HANGMAN" title at top center

# Display word with blanks/guessed letters:

For each letter in target word:

If guessed, show letter

Else show underscore

Display this at center

# Draw letter buttons:

For each button in letters:

If visible and not guessed:

Draw circle outline

Draw letter centered in circle

# Draw hint system:

If hints available > 0:

Draw green hint button at bottom right

Display remaining hint count

# Draw current hangman image (0-6)

Update display

# FUNCTION: draw\_hint\_button()

Create rectangle at bottom right

Draw green button with black border

Add "Use Hint" text

Return button rectangle for collision

# FUNCTION: use\_hint()

Get list of unguessed letters in word

If any available:

Choose random unguessed letter

Add to guessed list

Decrement hint count

Disable corresponding letter button

# FUNCTION: display\_message(message)

Pause briefly

Clear screen

Display win/lose message at center

Draw Retry and Quit buttons

Wait for player choice:

If Retry: return True

If Quit: return False

# MAIN GAME LOOP

Initialize game (reset\_game())

Set up game clock

Set run = True

While run:

Cap at 60 FPS

# EVENT HANDLING

For each event:

If QUIT event:

Set run = False

If MOUSEBUTTONDOWN:

Get mouse position

# Check hint button click

If hints available and clicked:

Call use\_hint()

# Check letter button clicks

For each visible, unguessed letter button:

If mouse click within button radius:

Disable button

Add letter to guessed list

If letter not in word:

Increment hangman\_status

Increment incorrect\_guesses

Grant hint every 2 incorrect guesses

Call draw() to update display

# WIN/LOSE CHECK

If all letters in word guessed:

Show win message

If player chooses Retry: reset\_game()

Else: quit

If hangman\_status == 6 (full hangman):

Show lose message

If player chooses Retry: reset\_game()

Else: quit

# GAME RESTART LOOP

While True:

If main() returns False:

Break

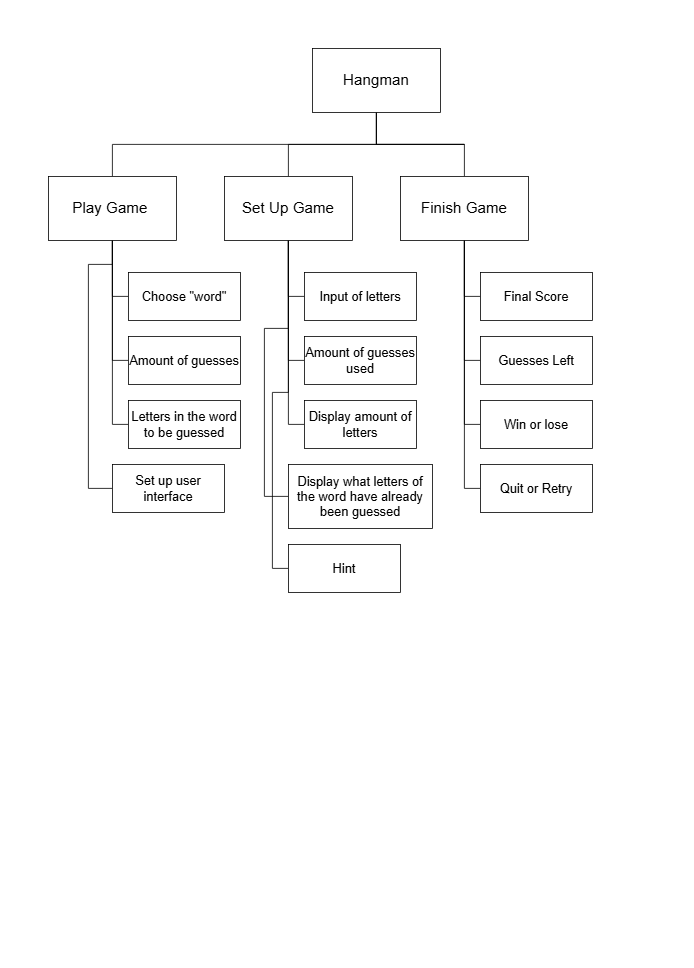
Quit pygame

Flowchart

A diagram of a company

AI-generated content may be incorrect.

UML Chart



Data Dictionary

|  |  |  |
| --- | --- | --- |
| Variable | Type | Description |
| WIDTH, HEIGHT | int | Game window dimensions (1200x800). |
| RADIUS | int | Radius of letter buttons (20 pixels). |
| GAP | int | Spacing between buttons (15 pixels). |
| A (ASCII value) | int | Starting point for generating letters (65 = 'A'). |
| WORDS | list | List of words for the game (e.g., ["PYTHON", "DEVELOPER"]). |
| WHITE, BLACK,  GRAY, GREEN | tuple (RGB) | Color definitions for UI elements. |
| hangman\_status | int | Tracks hangman image state (0 to 6). |
| word | str | Current word to guess (randomly selected from WORDS). |
| guessed | list | Letters guessed by the player (e.g., ['P', 'Y']). |
| hint\_available | int | Number of hints remaining (increases every **2 wrong guesses**). |
| incorrect\_guesses | int | Counts wrong guesses (triggers hints & hangman progression). |
| letters | list of lists | Stores letter buttons: [x, y, char, visible]. |
| images | list | Hangman images (hangman0.png  to hangman6.png). |
| LETTER\_FONT | Font | Font for letter buttons (comicsans, 40). |
| WORD\_FONT | Font | Font for word display (comicsans, 60). |
| TITLE\_FONT | Font | Font for game title (comicsans, 70). |
| BUTTON\_FONT | Font | Font for action buttons (comicsans, 50). |
| initialize\_letter  \_buttons() | startx,  starty, buttons | Creates letter buttons in 2 rows. |
| reset\_game() | hangman\_status,  word, guessed, hint\_available | Resets game state. |
| draw() | display\_word,  word\_text, letter\_text | Renders UI (buttons, hangman, word). |
| draw\_hint\_button() | hint\_rect, button\_width, button\_height | Draws the hint button (green). |
| use\_hint() | available\_letters,  hint\_letter | Reveals a random unguessed letter. |
| display\_message() | retry\_button, quit\_button | Shows win/lose screen with buttons. |

**Summary of Data Flow**

1. **Initialization** → WORDS, letters, images load.
2. **Game Loop** → Updates hangman\_status, guessed, hint\_available.
3. **UI Rendering** → Uses fonts, colors, and buttons to display the game.
4. **Endgame** → Checks guessed vs. word to determine win/lose.

Version Control

**Version 1: Base Game (No Hints or Quit/Retry)**

**Features**:  
✅ Basic Hangman mechanics  
✅ Letter buttons (A-Z)  
✅ Hangman visual progression  
✅ Word display with underscores

**Missing**:  
❌ No **hint system**  
❌ No **quit/retry options** (game exits on win/lose)

**Key Code Differences**:

* No hint\_available or incorrect\_guesses variables.
* No draw\_hint\_button() or use\_hint() functions.
* Game loop ends immediately after win/lose (no display\_message()).

**Use Case**:

* Simplest version for **beginners** to understand core mechanics.

**Version 2: Added Quit/Retry Functions**

**New Features**:  
✅ **Endgame screen** with "You WON/LOST!" message.  
✅ **Retry button** (resets game).  
✅ **Quit button** (exits application).

**Missing**:  
❌ Still **no hints**.

**Key Code Additions**:

1. **display\_message() function**:
   * Renders win/lose text + buttons.
   * Returns True (retry) or False (quit).
2. **draw\_end\_buttons() function**:
   * Draws "Retry" and "Quit" buttons.
3. Modified **main loop** to handle retries:

if won or hangman\_status == 6:

if display\_message("You WON!" if won else "You LOST!"):

reset\_game() # Retry

else:

run = False # Quit

**Version 3: Full Game (All Features)**

**New Features**:  
✅ **Hint system**:

* Earn hints after **2 incorrect guesses**.
* Click "Use Hint" to reveal a random letter.  
  ✅ **Hint counter** displayed (top-right).

**Key Code Additions**:

1. **New Variables**:
   * hint\_available (int): Tracks remaining hints.
   * incorrect\_guesses (int): Counts wrong attempts.
2. **draw\_hint\_button()**:
   * Renders the green hint button.
3. **use\_hint()**:
   * Selects a random unguessed letter from word.
4. **Logic in Main Loop**:

if incorrect\_guesses % 2 == 0: # Grant hint every 2 wrong guesses

hint\_available += 1

**Use Case**:

* **Balanced difficulty** with assistive hints.
* **Complete gameplay** loop.

**Comparison Summary**

| **Feature** | **Version 1** | **Version 2** | **Version 3** |
| --- | --- | --- | --- |
| Core Hangman | ✅ | ✅ | ✅ |
| Quit/Retry Buttons | ❌ | ✅ | ✅ |
| Hint System | ❌ | ❌ | ✅ |
| Code Complexity | Low | Medium | High |

**Why This Progression?**

1. **Beginner-Friendly**: Version 1 focuses on **fundamentals** (loops, event handling).
2. **UX Improvement**: Version 2 adds **polish** (restartability).
3. **Enhanced Gameplay**: Version 3 introduces **strategy** (hints).

**Git Workflow Example**:

git checkout -b v1\_base\_game # Version 1

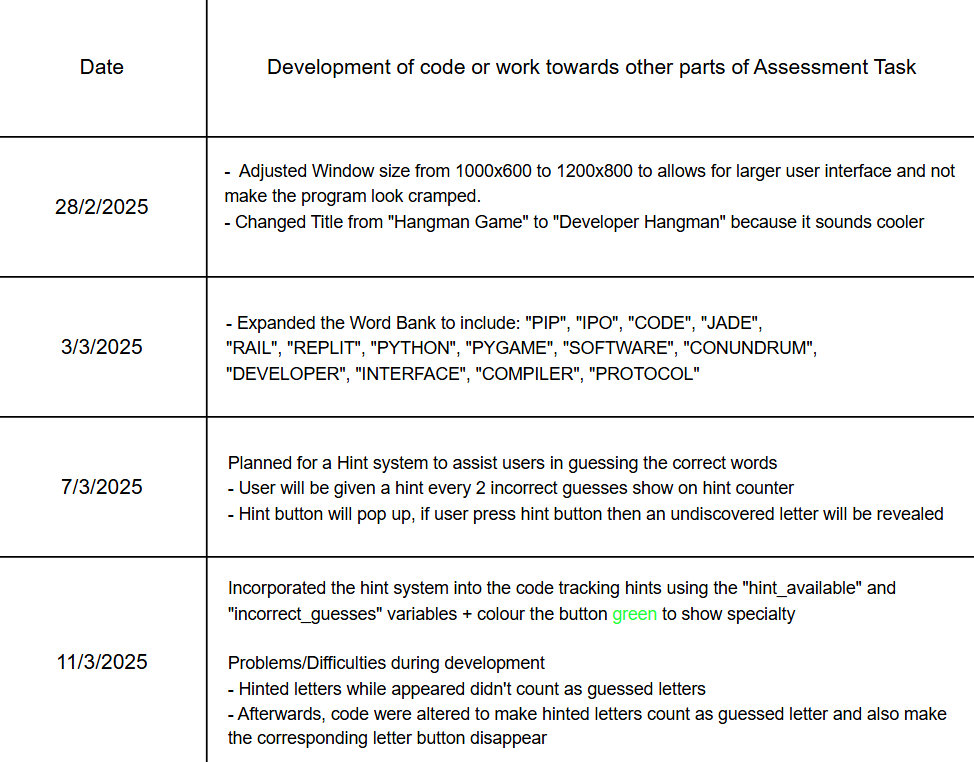
git checkout -b v2\_quit\_retry # Version 2

git checkout -b v3\_hints # Version 3 (final)

**Lessons Learned**:

* **Modularity**: Each version added features **without breaking** existing code.
* **User Feedback**: Quit/retry and hints were added based on **playtesting**.

Logbook



A list of text on a white background

AI-generated content may be incorrect. A white text on a white background

AI-generated content may be incorrect.