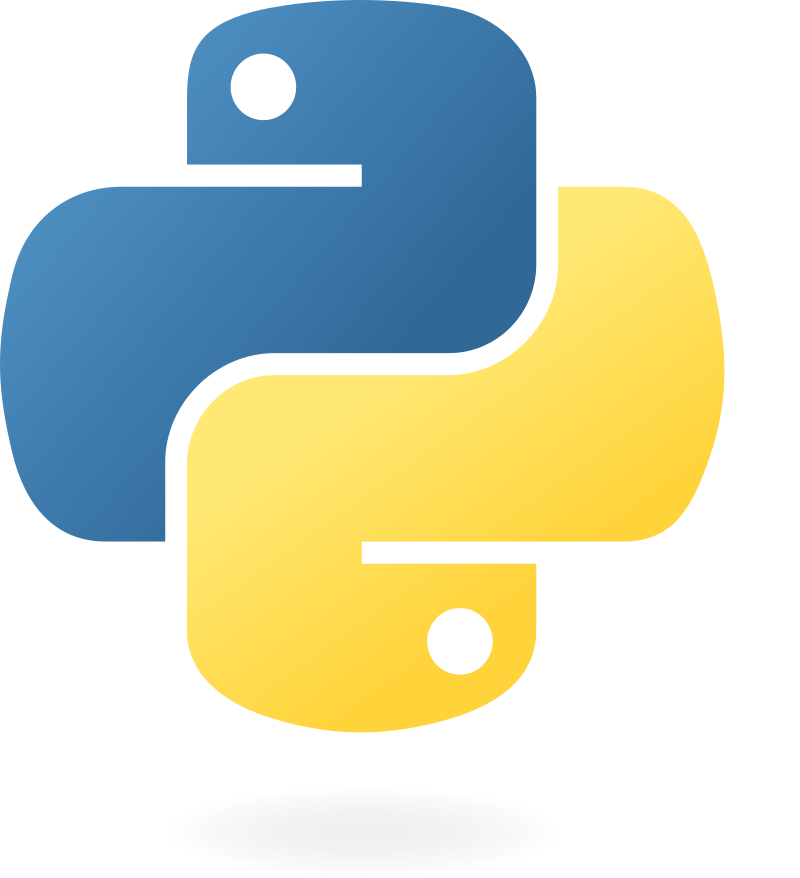
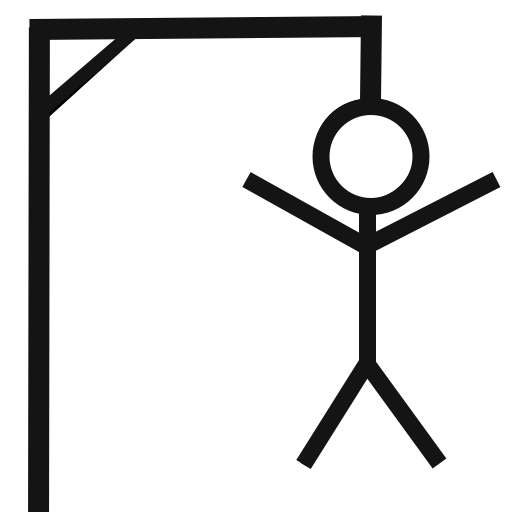
**Software Engineering**

**Assessment Task 1:**

**Python Hangman**

**By Henry**

**MODIFIED HANGMAN IN PYTHON**



**TABLE OF CONTENT:**

PAGE 5 UML

PAGE 6-10 PSEUDOCODE

PAGE 11-14 JUSTIFICATION FOR PROGRAMMING LANGUAGE

PAGE 14-21 USER INTERFACE

PAGE 22- FLOWCHART

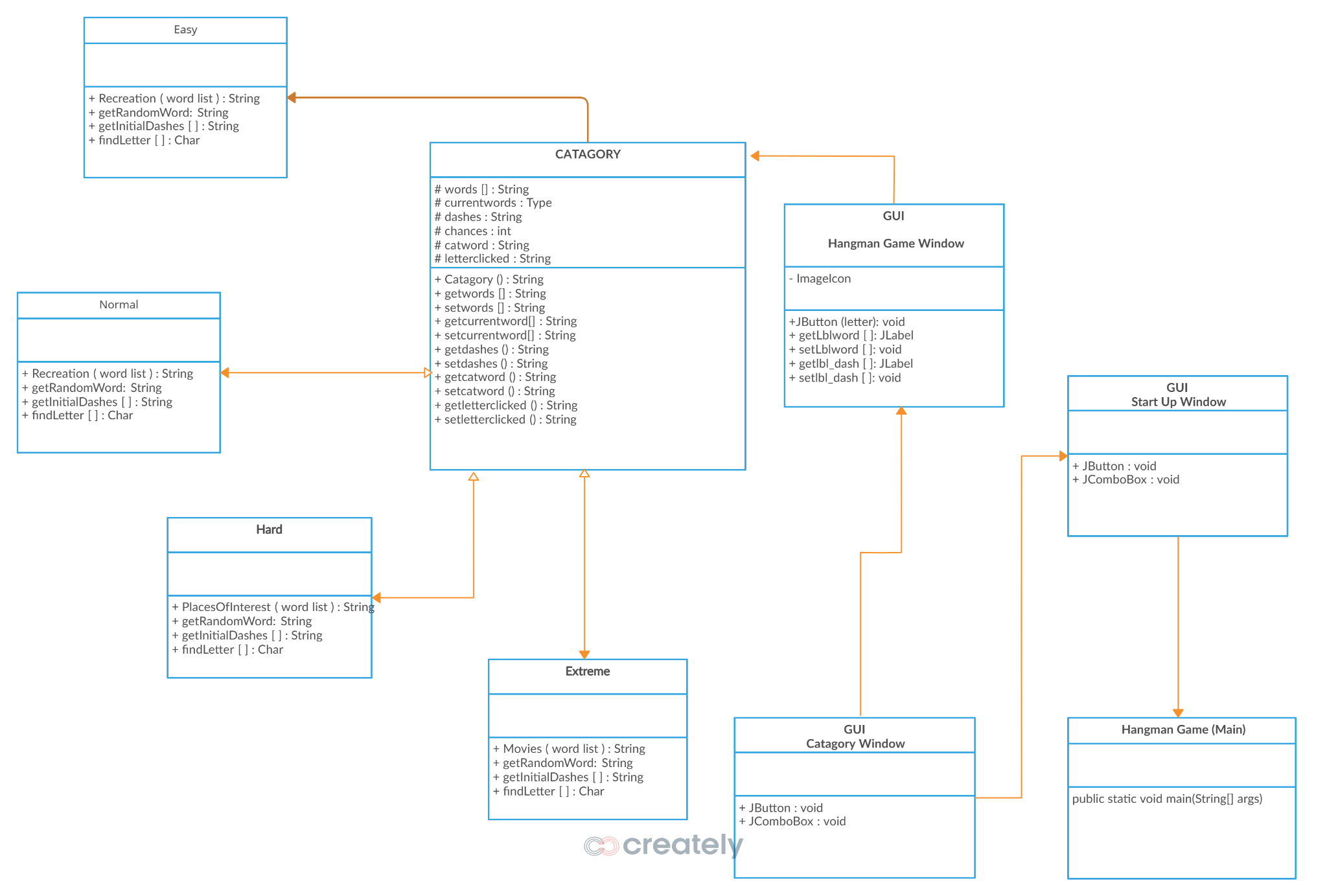
PAGE 23 - IPO CHART

PAGE 24-25 DATA DICTIONARY

PAGE 26-29 VISION CONTROL

PAGE 30-31 LOGBOOK

**UML**



**PSEUDOCODE**

BEGIN HangmanGame

INITIALISE Pygame

TRY

CREATE Game Window (Width: 1200, Height: 800)

SET Title: "Hangman Game!"

CATCH Error

PRINT "Failed to set up display"

EXIT Program

DEFINE Constants:

- RADIUS = 20

- GAP = 15

- MAX\_ATTEMPTS = 6

- Colors (WHITE, BLACK, LIGHT\_BLUE, RED, GREEN)

DEFINE Fonts (Try loading 'comicsans', else use default)

DEFINE Dictionary WORD\_LISTS containing:

- EASY, NORMAL, HARD, EXTREME difficulty words with hints

CREATE Class `GameState`

METHOD `\_\_init\_\_()`

INITIALIZE:

- `letters` (List of letter buttons)

- `word` (Current word to guess)

- `hint` (Hint for the word)

- `hangman\_status` (Mistakes counter)

- `guessed` (List of guessed letters)

- `hint\_used` (Boolean flag for hint usage)

CALL `generate\_letters()`

METHOD `generate\_letters()`

INITIALIZE `letters` as an empty list

CALCULATE start position for letters

LOOP for each letter A-Z:

- DETERMINE X, Y position

- ADD [X, Y, Letter, Visible=True] to `letters`

METHOD `reset(difficulty)`

RESET:

- `hangman\_status = 0`

- `guessed = []`

- `hint\_used = False`

CALL `generate\_letters()`

SELECT a random `(word, hint)` from `WORD\_LISTS[difficulty]`

FUNCTION `draw\_hangman(status)`

DRAW hangman base structure

IF status >= 1: DRAW Head

IF status >= 2: DRAW Body

IF status >= 3: DRAW Left Arm

IF status >= 4: DRAW Right Arm

IF status >= 5: DRAW Left Leg

IF status >= 6: DRAW Right Leg

FUNCTION `select\_difficulty()`

DISPLAY "Select Difficulty" Screen

SHOW Buttons: EASY, NORMAL, HARD, EXTREME

WAIT for User Input:

IF Mouse Clicks on a Button:

RETURN Selected Difficulty

FUNCTION `draw\_game(state)`

CLEAR Screen

DISPLAY Title "HANGMAN"

SHOW Current Word (Replacing unguessed letters with "\_")

SHOW Letter Buttons

SHOW Hangman based on `state.hangman\_status`

SHOW "Attempts Left"

IF Hint Used:

DISPLAY Hint Text

UPDATE Screen

FUNCTION `show\_message(message, duration)`

CLEAR Screen

DISPLAY `message`

WAIT for `duration` milliseconds

FUNCTION `ask\_to\_play\_again()`

SHOW "Play Again?" Screen

WAIT for User Input:

IF "YES" Clicked:

RETURN True

IF "NO" Clicked:

RETURN False

FUNCTION `main\_game\_loop(state)`

WHILE True:

WAIT for Events:

IF Quit Event:

EXIT Program

IF Mouse Clicked:

IF Hint Button Clicked:

SET `state.hint\_used = True`

CHECK IF Letter Button Clicked:

UPDATE Letter Visibility

ADD Letter to `state.guessed`

IF Incorrect Guess:

INCREMENT `state.hangman\_status`

IF Keyboard Key Pressed:

CHECK IF Key is a Letter:

ADD Letter to `state.guessed`

UPDATE Letter Button Visibility

IF Incorrect Guess:

INCREMENT `state.hangman\_status`

CALL `draw\_game(state)`

IF All Letters in Word Guessed:

CALL `show\_message("YOU WON!")`

RETURN

IF `state.hangman\_status >= MAX\_ATTEMPTS`:

SHOW "YOU LOST!" Message

DISPLAY The Correct Word

WAIT 3 Seconds

RETURN

FUNCTION `main()`

CREATE `GameState` object `state`

WHILE True:

CALL `select\_difficulty()`

CALL `state.reset(difficulty)`

CALL `main\_game\_loop(state)`

IF NOT `ask\_to\_play\_again()`:

EXIT Program

IF \_\_name\_\_ == "\_\_main\_\_":

CALL `main()`

END HangmanGame

JUSTIFICATION FOR PROGRAMMING LANGUAGE

Python offers programming ease to both beginners along with readability standards which lets both well-trained developers as well as novices create and maintain code effectively. Python strikes a balance between easy to read code which generates fewer mistakes and makes programme debugging more simple.

The flexibility of Python extends to multiple applications since developers utilise it for web development as well as data analysis machine learning projects and automation and cybersecurity functions. The programming language fits various industrial requirements which makes it highly beneficial for multiple sectors.

Extensive Libraries and Frameworks – NumPy, Pandas, TensorFlow, along with Django serve as powerful tools that Python developers use for data science, AI, web development and other purposes. Such predefined structures help developers cut down work time.

A wide global community devoted to Python supports its ongoing development through active participation. Documents and tutorial resources and technical help networks enable beginners together with experienced users to find support.

Python functions as a universal codebase because it operates fully on Windows, macOS and Linux operating systems. The same platform can run applications through this ability so developers create applications which deliver a smooth experience regardless of platform.

Python leads to quicker development cycles because its syntax allows developers to create programmes using fewer lines of code relative to other programming languages. The development process becomes accelerated because of this which suits entrepreneurs and businesses that need fast prototype development.

Python stands as the Vatican City of Data Science and Artificial Intelligence since it maintains its position as the leading language for all three subjects. The use of Python in healthcare and finance sectors together with e-commerce allows these industries to detect patterns and automate operations and generate better decisions.

Through its integration capabilities Python allows developers to work effortlessly with C, C++ and Java code for enhancing programme performance while keeping pace with existing systems.

The automated performance of repetitive tasks through Python includes script programming and file management while network maintenance and data processing and handling functions lead to increased efficiency and decreased operator-based mistakes.

Robust Security Features – Python offers built in security features and frameworks for secure application development. It is widely used in web development, cybersecurity, and ethical hacking to protect systems from vulnerabilities.

**USER INTERFACE (UI)**

## **1. Main Menu (Difficulty Selection)**

The game begins with a **main menu screen** where players choose their **difficulty level**. This screen sets the tone for the game by using **aesthetic visuals, interactive buttons, and a structured layout**.

### **Visual Elements:**

* The background is a **solid white color**, ensuring clarity and readability.
* At the **top center** of the screen, the title **"Select Difficulty"** is displayed in a **bold, large-sized font**.
* Below the title, four **interactive buttons** allow the player to choose a difficulty:  
  + **EASY**
  + **NORMAL**
  + **HARD**
  + **EXTREME**
* These buttons are:  
  + **Rounded rectangles** with a **soft color gradient**.
  + **Labeled in uppercase letters** for clarity.
  + **Responsive**: They change color when hovered over.
* As the player moves the mouse over a button, it highlights in **a bright green color**, providing clear feedback.
* Once a button is clicked, the screen **smoothly transitions** to the main gameplay.

## **2. Gameplay Screen**

Once the difficulty level is selected, the **gameplay screen** appears. This is the **core interactive part** of the game, where the player attempts to guess the word while avoiding too many incorrect guesses.

### **Layout Structure**

* The screen is divided into multiple sections:  
  1. **Title Section** – Displays "HANGMAN" at the top in **large, bold letters**.
  2. **Hangman Drawing Area** – Located on the **left side**, where the gallows and hangman figure are drawn progressively.
  3. **Word Display** – At the **center of the screen**, showing the word with underscores (\_) for unguessed letters.
  4. **Alphabet Buttons** – Arranged in a **keyboard-like layout** at the bottom.
  5. **Hint Button** – Positioned at the **top-right corner**.
  6. **Attempts Counter** – Located in the **top-left corner**, displaying the **remaining guesses**.

### **Interactive Features**

* **The Hangman Drawing:**
  + Starts with just the **gallows**.
  + With each incorrect guess, a new part of the **stick figure** is drawn.
  + The sequence follows: **head → body → arms → legs**.
  + When the figure is fully drawn, the game ends in a loss.
* **Word Display:**
  + The unknown word is represented by **underscores (\_)** for each missing letter.
  + As the player correctly guesses letters, they are revealed in the word.
  + The font is **large and easy to read**, ensuring accessibility.
* **Alphabet Buttons:**
  + The letters **A–Z** are displayed as **circular buttons**.
  + When hovered over, they change to **light green**.
  + If a letter has already been chosen, it becomes **invisible**, preventing repeat guesses.
  + Clicking on a letter checks whether it is in the word.  
    - **Correct guesses**: The letter appears in its place within the word.
    - **Incorrect guesses**: The hangman figure progresses.
* **Hint Button:**
  + The **top-right corner** features a small button labeled **"SHOW HINT"**.
  + Clicking this button displays a **short text hint** below the word display.
  + The hint appears in **smaller font size**, subtly blending into the UI.
* **Attempts Counter:**
  + A **small text box** in the **top-left corner** shows the **number of remaining attempts**.
  + It updates dynamically with each incorrect guess.
  + If only one attempt remains, the text color changes to **red**, signaling danger.

## **3. Win/Loss Screen**

At the end of the game, the UI shifts to a **result screen**, depending on whether the player won or lost.

### **Winning Screen**

* If the player successfully guesses the word before running out of attempts:  
  + A **large "YOU WON!" message** appears in bright **green** text.
  + The screen briefly displays a **celebratory animation** (e.g., a flashing effect or color change).
  + A message prompts: **"Play Again?"**, offering two buttons:  
    - **YES** – Restarts the game.
    - **NO** – Closes the game.
  + Hovering over a button **changes its color**, reinforcing interactivity.

### **Losing Screen**

* If the player fails to guess the word in time:  
  + A **large "YOU LOST!" message** appears in **bold red**.
  + The correct word is displayed in **black** below the message.
  + The screen fades slightly to a darker shade, emphasizing failure.
  + A prompt appears: **"Play Again?"**, with the same **YES/NO buttons** as the winning screen.
  + The UI remains interactive, allowing the player to restart or exit.

## **4. General UI Features**

Your UI is designed with **a balance between simplicity and functionality**, ensuring an **engaging user experience**. Here are some key features:

### **Color Scheme**

* **White background** for a **clean and readable layout**.
* **Black text/icons** for high contrast.
* **Light blue and green buttons** for easy identification.
* **Red for losing messages**, creating a visual cue for failure.

### **Typography**

* Uses a **playful yet readable font** (e.g., Comic Sans or similar).
* Different font sizes distinguish **titles, words, and buttons**.
* Text elements are **centered and well-spaced** for clarity.

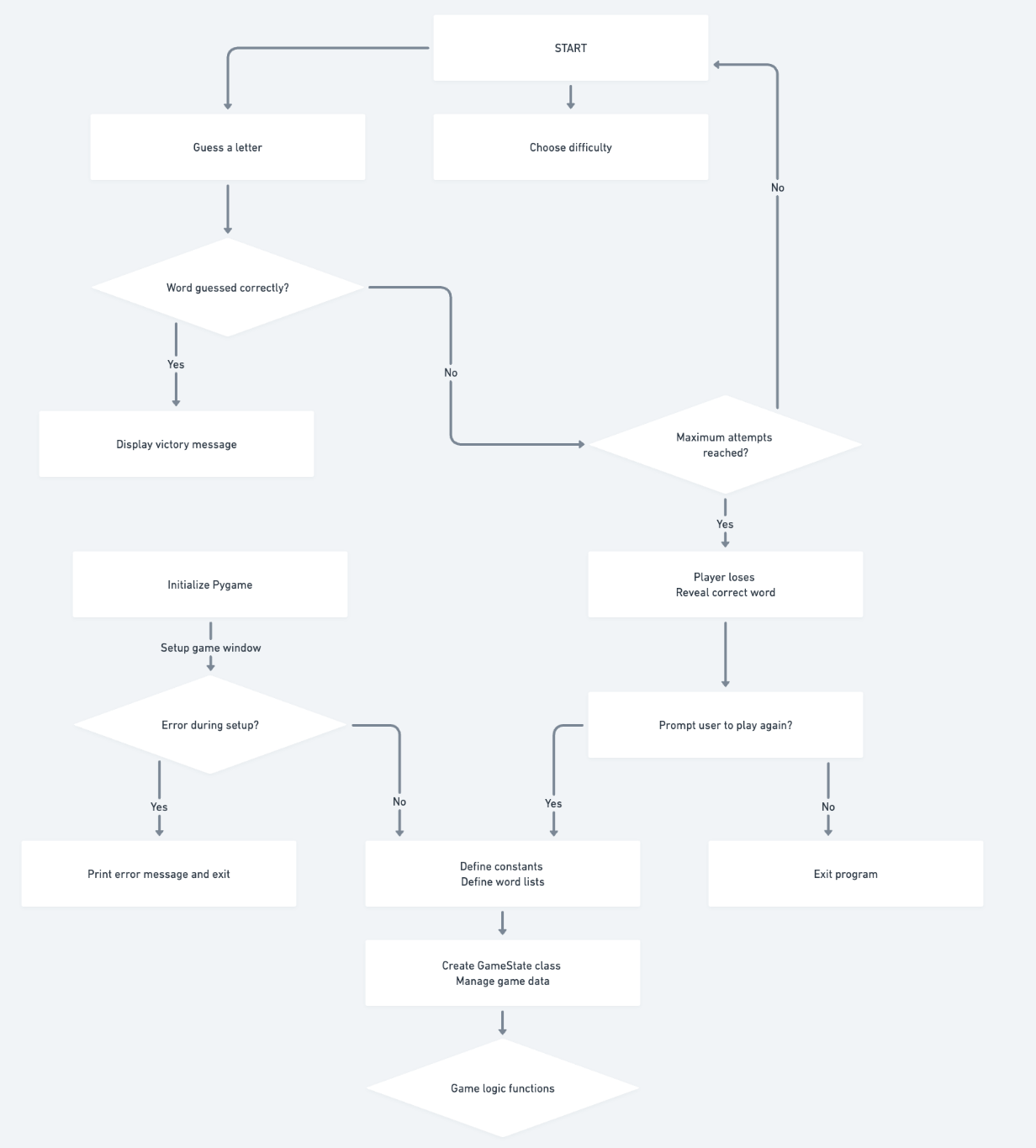
### **Animations & Effects**

* **Hover effects** on buttons (color change).
* **Flashing or fading effect** on the win/loss messages.
* **Dynamic updates** for the word display and hangman drawing.

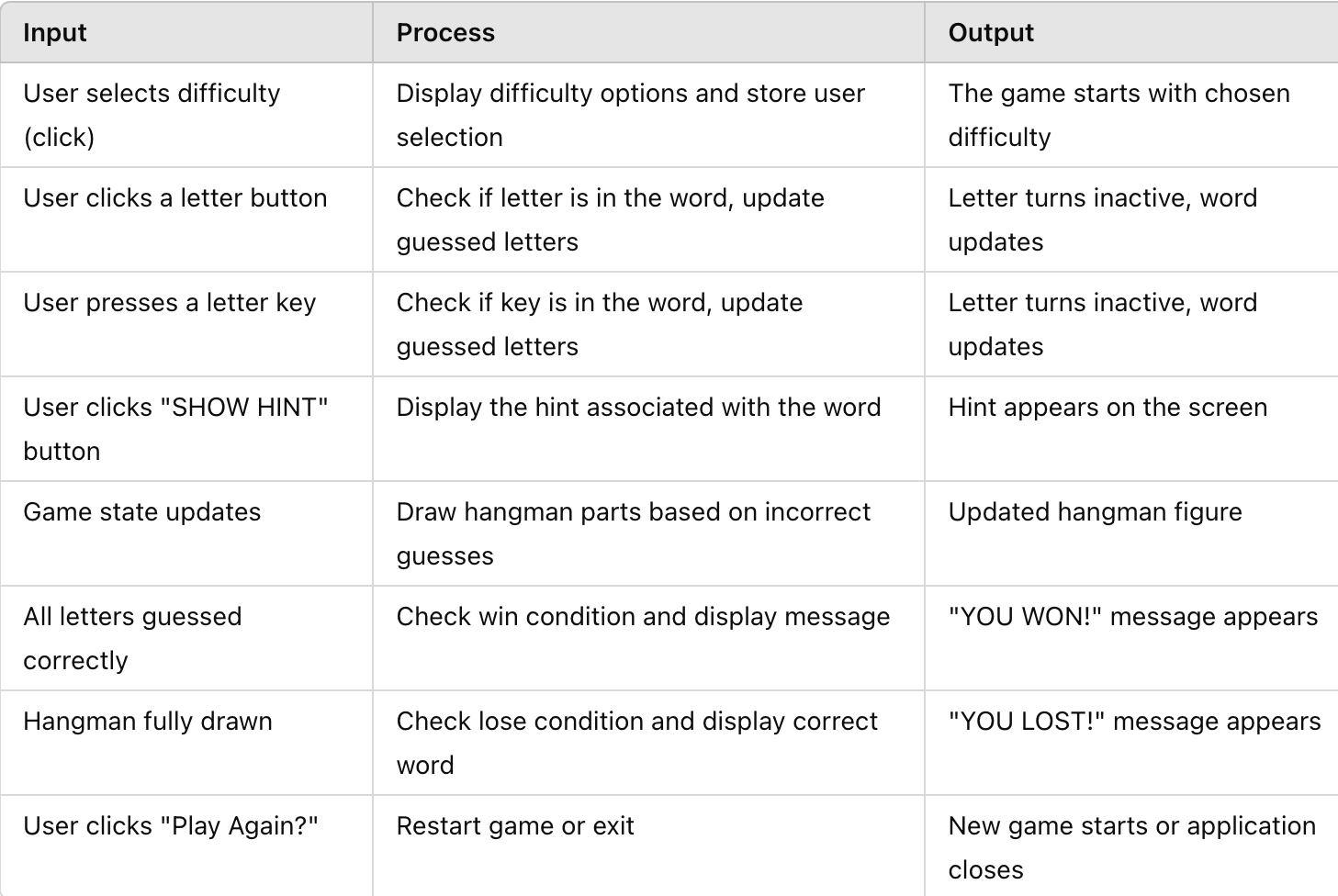
### **Input & Interaction**

* The game is controlled using **both the mouse and keyboard**:  
  + Clicking on letters or buttons with the mouse.
  + Pressing **A-Z on the keyboard** to select letters.

**FLOWCHART**

****

**IPO TABLE**



**DATA DICTIONARY**

| **Variable Name** | **Data Type** | **Description** |
| --- | --- | --- |
| **word** | **str** | **Stores the target word that the player must guess. This is the main puzzle.** |
| **guessed\_letters** | **list** | **A list that tracks all the letters the player has guessed so far, correct or incorrect.** |
| **incorrect\_guesses** | **int** | **Tracks the number of incorrect guesses made by the player during the game. Used to determine when the game ends.** |
| **attempts\_left** | **int** | **The number of guesses the player has left before the game ends, typically based on difficulty.** |
| **difficulty** | **str** | **Represents the difficulty level selected by the player (e.g., "EASY", "NORMAL", "HARD"). This impacts the number of attempts allowed.** |
| **hint** | **str** | **A string that contains a hint related to the word, displayed when the player clicks the "SHOW HINT" button.** |
| **alphabet\_buttons** | **list** | **A list of Pygame button objects representing all the letters of the alphabet (A-Z) on the screen for the player to click.** |
| **game\_state** | **str** | **A string indicating the current state of the game (e.g., "playing", "won", "lost", "paused"). It determines the behavior of the game loop.** |
| **max\_attempts** | **int** | **The maximum number of incorrect guesses allowed based on the selected difficulty. For example, 8 for "EASY", 6 for "NORMAL", and 4 for "HARD".** |
| **game\_message** | **str** | **Stores a message shown to the player after the game ends, indicating whether they won or lost.** |
| **screen** | **Pygame.Surface** | **The main Pygame surface on which all game elements (background, buttons, text, etc.) are drawn.** |
| **score** | **int** | **Keeps track of the player's score based on how quickly they guess the word and whether they complete it within the allowed attempts.** |
| **letter\_buttons\_state** | **list** | **A list of boolean values (True/False) representing the state of each alphabet button. True means the button is active (not guessed yet), and False means it is inactive (already guessed).** |
| **current\_word\_display** | **str** | **The word to be guessed, with correctly guessed letters shown and unguessed letters as underscores.** |
| **button\_font** | **Pygame.font.Font** | **The font used to render the text on the buttons (letters), affecting their appearance on screen.** |
| **game\_start\_time** | **float** | **The timestamp when the game starts, used for calculating the elapsed time for scoring.** |
| **sound\_correct** | **Pygame.mixer.Sound** | **The sound effect is played when the player guesses a letter correctly.** |
| **sound\_incorrect** | **Pygame.mixer.Sound** | **The sound effect is played when the player guesses a letter incorrectly.** |
| **sound\_game\_over** | **Pygame.mixer.Sound** | **The sound effect is played when the game ends (either win or loss).** |
| **player\_name** | **str** | **A string representing the player's name, used for personalizing the game experience.** |
| **previous\_score** | **int** | **Stores the previous score from a completed game, used for showing a high score or comparing performances.** |
| **leaderboard** | **list** | **A list of player names and scores, storing the highest scores to create a leaderboard.** |

**VISION CONTROL**

#### **Initial Commit (v1.0)**

* **Base game structure implemented**
* **Core functionality:**
  + **Pygame initialisation and setup**
  + **Basic game loop**
  + **Keyboard letter input**
  + **Simple hangman illustration**
  + **Win/lose conditions**

#### **Feature Update (v1.1)**

* **Added difficulty levels:**
  + **Easy, Normal, Hard, Extreme**
  + **Distinct word lists for each difficulty tier**
* **Improved user interface:**
  + **Difficulty selection screen**
  + **"Play again?" prompt**
  + **Message display system**
  + **Integrated hints feature**

#### **Refactor (v1.2)**

* **Restructured code into classes:**
  + **GameState class to manage game state**
  + **Separated rendering functions**
  + **Enhanced event handling**
* **Added error management:**
  + **Pygame initialisation checks**
  + **Font loading fallbacks**
* **Code organisation improvements:**
  + **Constants defined at top**
  + **Helper functions modularised**
  + **Main game loop isolated**

#### **UI Enhancements (v1.3)**

* **Introduced hover effects for buttons**
* **Enhanced text rendering:**
  + **Word wrapping for lengthy terms**
  + **Multi-line hint presentation**
* **Visual upgrades:**
  + **Rounded button edges**
  + **Refined colour scheme**
  + **Improved hangman illustration**

#### **Quality of Life Improvements (v1.4)**

* **Added keyboard support alongside mouse input**
* **Attempts counter displayed**
* **Enhanced word display with underscores**
* **Responsive design for varied screen elements**

#### **Current Version (v1.5)**

* **Final polishing:**
  + **Comprehensive code commentary**
  + **Robust error handling**
  + **Edge case protection**
  + **Performance optimisation**

### **Key Components Under Version Control**

#### **Game Architecture**

* **Primary game loop**
* **State management**
* **Event handling**

#### **User Interface System**

* **Difficulty selection mechanism**
* **Game display rendering**
* **Message system**
* **Button interaction logic**

#### **Game Logic**

* **Word selection algorithm**
* **Guess validation system**
* **Win/lose condition checks**
* **Hint functionality**

#### **Assets**

* **Curated word lists with associated hints**
* **Font management**
* **Colour palette configuration**

**LOGBOOK**

| **Date** | **Development Focus** | **Key Implementations** | **Technical Notes** |
| --- | --- | --- | --- |
| **February 27, 2025** | **Core Framework** | - Initialised Pygame with error handling  - Set 1200x800 display window  - Created basic game loop structure | Added system fallbacks for display initialisation failures |
| **Mar 1, 2025** | **Game Architecture** | - Developed GameState class  - Implemented letter grid generation  - Established difficulty framework | Used mathematical grid calculation for letter positioning |
| **Mar 5, 2025** | **Visual Systems** | - Built hangman drawing function  - Created word display with wrapping  - Added hint text rendering | Special handling for extreme length words |
| **Mar 10, 2025** | **User Interaction** | - Implemented mouse/keyboard input  - Added button hover effects  - Developed hint button logic | Dual input system required careful event handling |
| **Mar 13, 2025** | **Game Logic** | - Created win/lose conditions  - Implemented attempt counter  - Developed letter tracking system | Used list comprehensions for efficient letter checks |
| **Mar 15, 2025** | **Word Database** | - Built 4-tier word bank  - Added hints for all words  - Implemented difficulty selection | Extreme words tested rendering limits |
| **Mar 19, 2025** | **UI Polish** | - Added color-coded feedback  - Improved text rendering  - Balanced visual spacing | Fallback fonts for cross-platform compatibility |
| **Mar 21, 2025** | **Game Flow** | - Created play-again prompt  - Implemented clean reset system  - Added proper exit handling | State management critical for smooth resets |
| **Mar 26, 2025** | **Optimisation** | - Fixed memory leaks  - Improved rendering efficiency  - Stabilised frame rate | Used pygame.time.Clock() for consistent FPS |
| **April 1st, 2025** | **Final Testing** | - Input stress testing  -Pep8 standard | Added comments |