HANGMAN WORD-GUESSING GAME PROJECT REPORT

By: Aditya Negi (25BCE10324)

---

1. COVER PAGE

HANGMAN WORD-GUESSING GAME PROJECT

An Interactive Python Console Application for Word Puzzle Entertainment

Project Report

Institution: VIT Bhopal University

Date: 18th November 2025

This report documents the design, implementation, testing, and deployment of an interactive Hangman word-guessing game developed as a college project.

---

2. INTRODUCTION

The Hangman Word-Guessing Game is a Python-based interactive console application that brings the classic pen-and-paper word guessing game to digital life. This project demonstrates fundamental concepts of game development, user interaction, string manipulation, random selection, and game state management. The system challenges players to guess a hidden word by suggesting letters within a limited number of attempts, with visual feedback through ASCII art representing the hangman's progression.

The application serves as an engaging educational tool to practice Python programming concepts including loops, conditionals, lists, string operations, and user input validation. It combines entertainment with practical programming principles, making it suitable for both learning and recreational purposes.

---

3. PROBLEM STATEMENT

Objective: Develop an interactive Hangman game that:

- Selects random words from a predefined word bank
- Manages player guesses and tracks game state
- Validates user input (single letters only)
- Prevents duplicate letter guesses
- Displays ASCII art representation of hangman progression
- Provides meaningful feedback for correct and incorrect guesses
- Determines win/loss conditions accurately
- Allows multiple game sessions in a single run

Scope: The system focuses on core Hangman game mechanics with console-based interaction, providing an engaging user experience while demonstrating essential game development concepts. Database persistence and network features are excluded from this scope.

4. FUNCTIONAL REQUIREMENTS

The Hangman Word-Guessing Game must fulfill the following functional requirements:

1. **Word Selection & Management**
    - Maintain a predefined list of words (fruits in current implementation)
    - Randomly select a word from the list for each game
    - Display word representation using underscore placeholders
2. **Game Initialization**
    - Display welcome message
    - Initialize game variables (tries, guessed letters, word completion)
    - Display initial game state with hangman ASCII art
3. **Player Input & Validation**
    - Accept single letter input from player
    - Validate that input is a single alphabetic character
    - Reject non-alphabetic or multi-character inputs
    - Convert input to lowercase for consistent comparison
4. **Guess Processing**
    - Track all previously guessed letters
    - Prevent duplicate guesses with warning message
    - Check if guessed letter exists in the target word
    - Update word completion display for correct guesses
    - Decrement tries counter for incorrect guesses
5. **Game State Management**
    - Maintain current game state (tries remaining, guessed letters, word completion)
    - Update display after each guess
    - Show ASCII art hangman at appropriate progression levels
    - Track win/loss conditions
6. **Visual Feedback**
    - Display 7-stage ASCII hangman art (from empty to complete)
    - Show current word with guessed letters revealed and unknowns as underscores
    - Provide text feedback for correct/incorrect guesses
    - Display tries remaining after incorrect guess
7. **Win/Loss Detection & Handling**
    - Win condition: All letters in word guessed before tries exhaust
    - Loss condition: Tries reduced to zero before word completion
    - Display appropriate end-game message
    - Reveal complete word on loss
8. **Session Management**
    - Allow player to choose whether to play another game
    - Support multiple consecutive games

- Graceful exit with thank you message

---

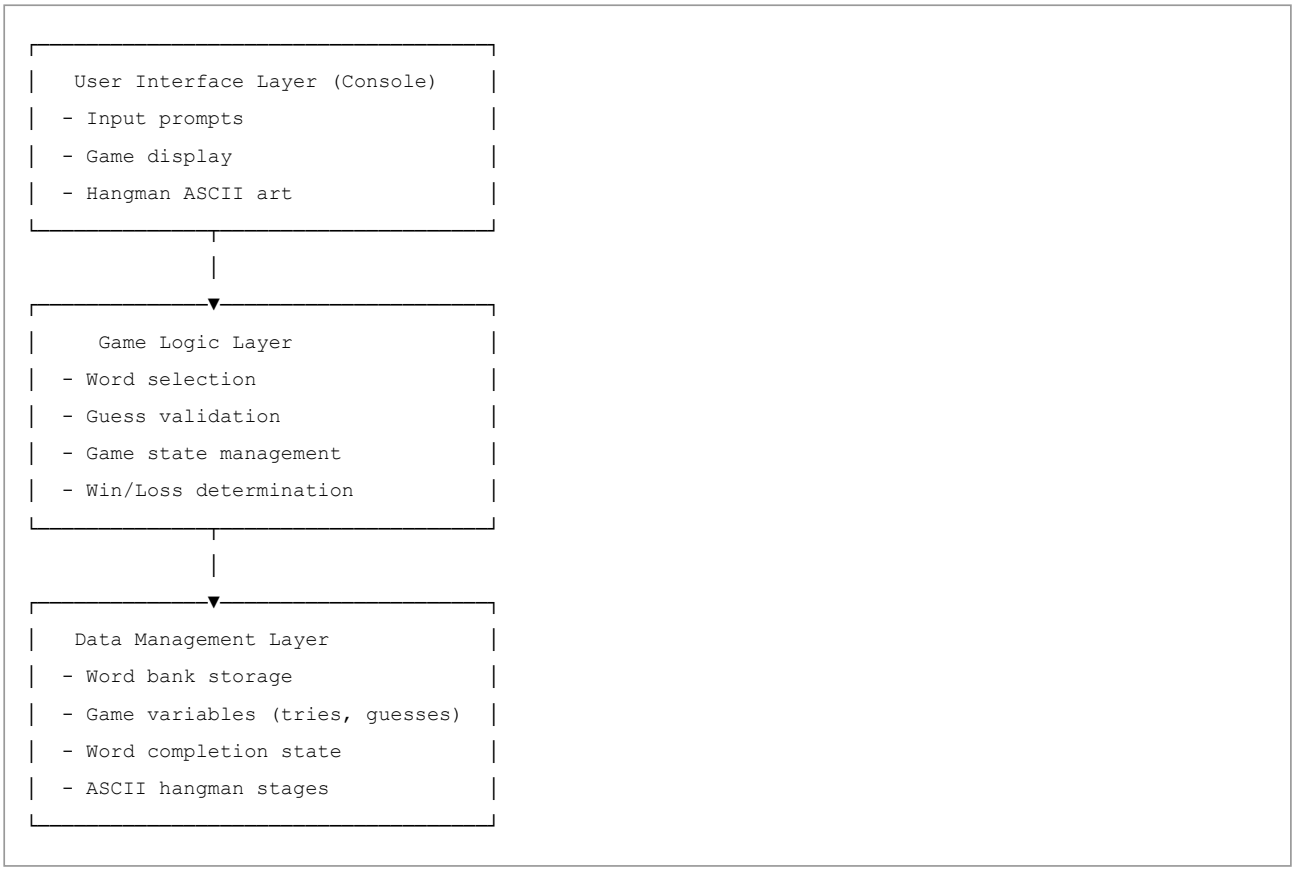
5. NON-FUNCTIONAL REQUIREMENTS

1. **Usability**
    - Clear, intuitive console interface
    - Understandable game rules and flow
    - Immediate visual and textual feedback
    - Logical game progression
2. **Performance**
    - Instant letter validation and processing
    - Immediate display updates
    - Efficient memory usage for word storage
    - Responsive to player input
3. **Reliability**
    - Robust input validation preventing crashes
    - Graceful handling of edge cases (empty input, special characters)
    - Consistent game logic and win/loss conditions
    - Error prevention through input checking
4. **Maintainability**
    - Well-structured modular code
    - Clear function organization
    - Descriptive variable naming
    - Comprehensive code comments
5. **Scalability**
    - Easy to expand word bank
    - Simple to add difficulty levels
    - Straightforward theme variations
    - Support for different word categories
6. **Accessibility**
    - Text-based interface (no graphics required)
    - Works on all systems with Python installed
    - Clear ASCII art representation
    - High contrast text display

---

6. SYSTEM ARCHITECTURE

The Hangman game follows a modular procedural architecture with the following components:



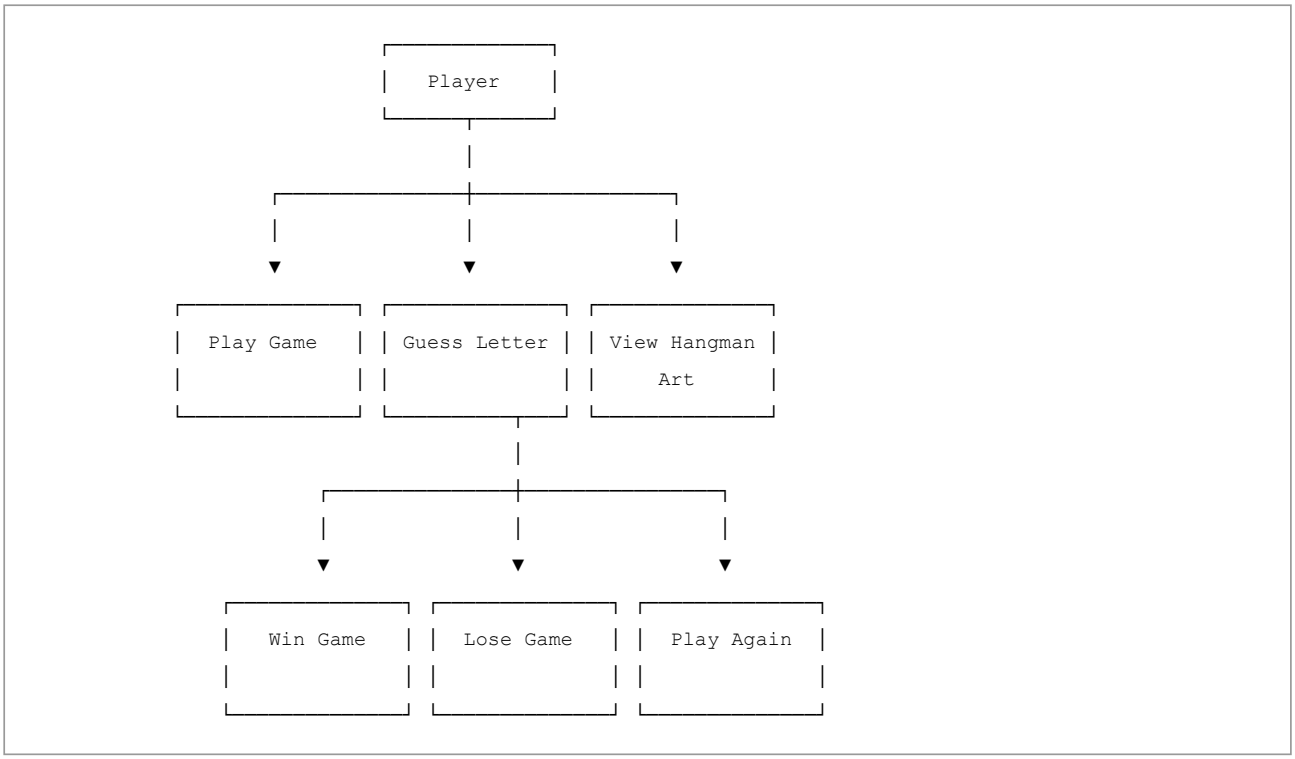
Key Modules:

- **Main Game Module (main()):** Initializes word selection and game instance
- **Game Logic Module (game()):** Core game loop and guess processing
- **Display Module (display_hangman()):** ASCII art rendering based on tries remaining
- **Input Validation Module:** Embedded in game loop for letter validation
- **Session Control Module:** Handles replay decision and program flow

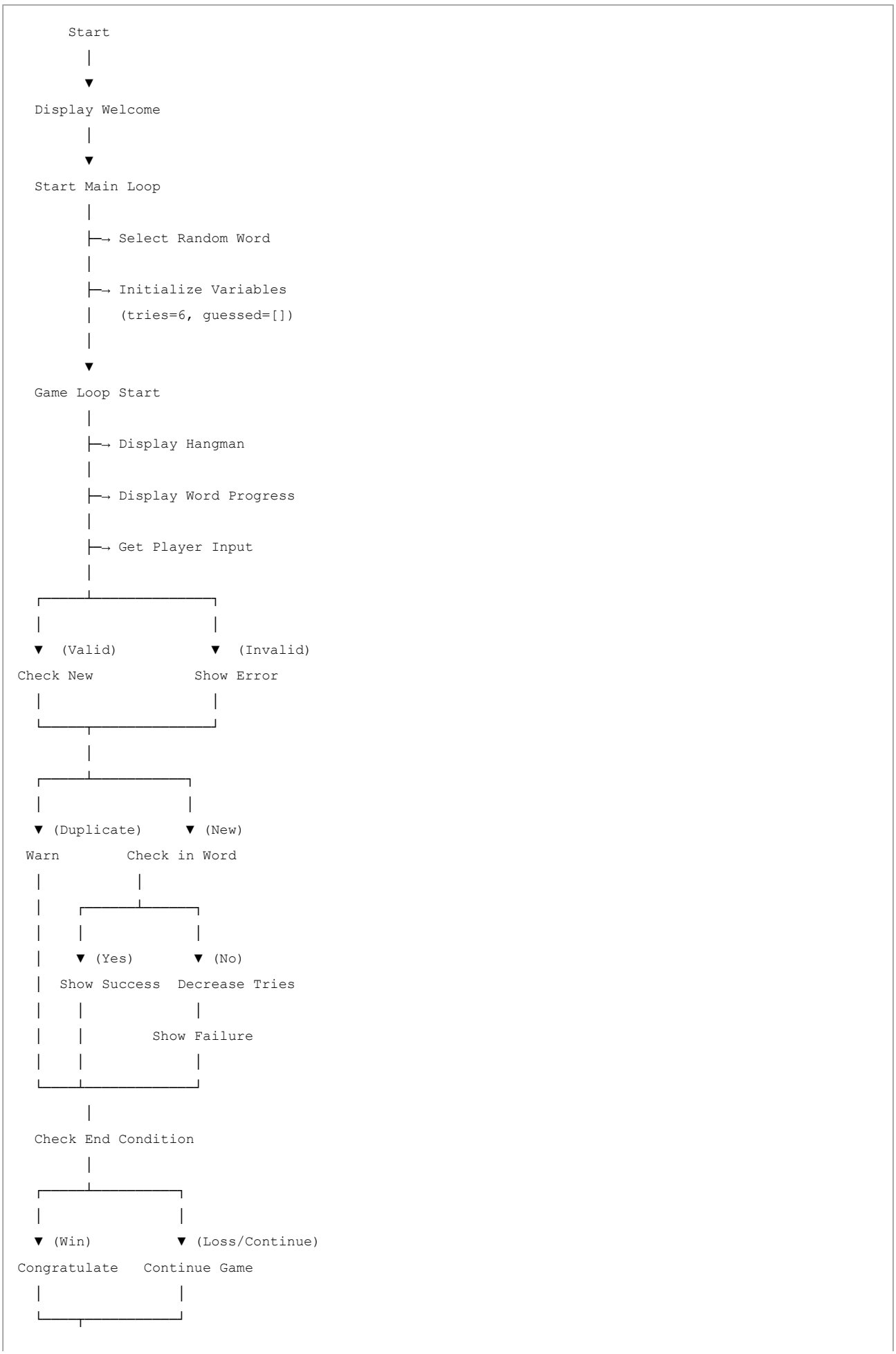
---

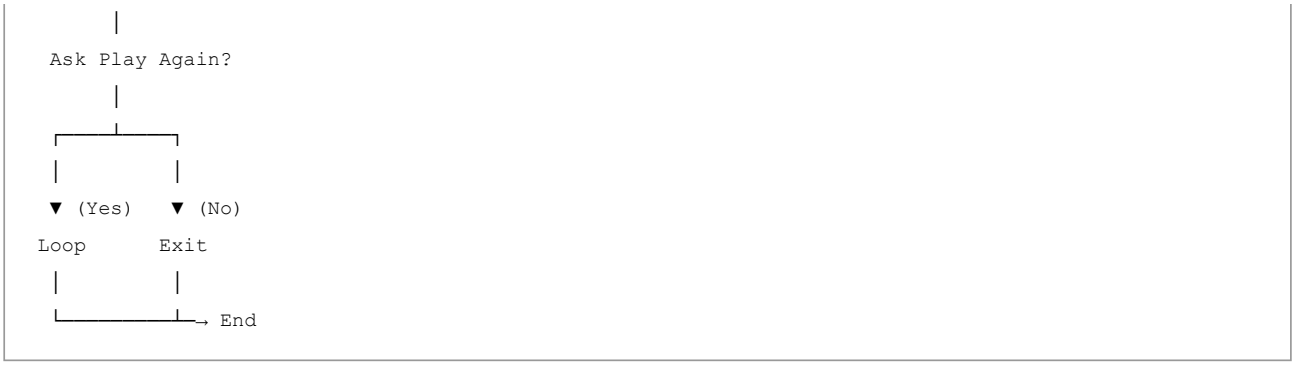
7. DESIGN DIAGRAMS

7.1 Use Case Diagram



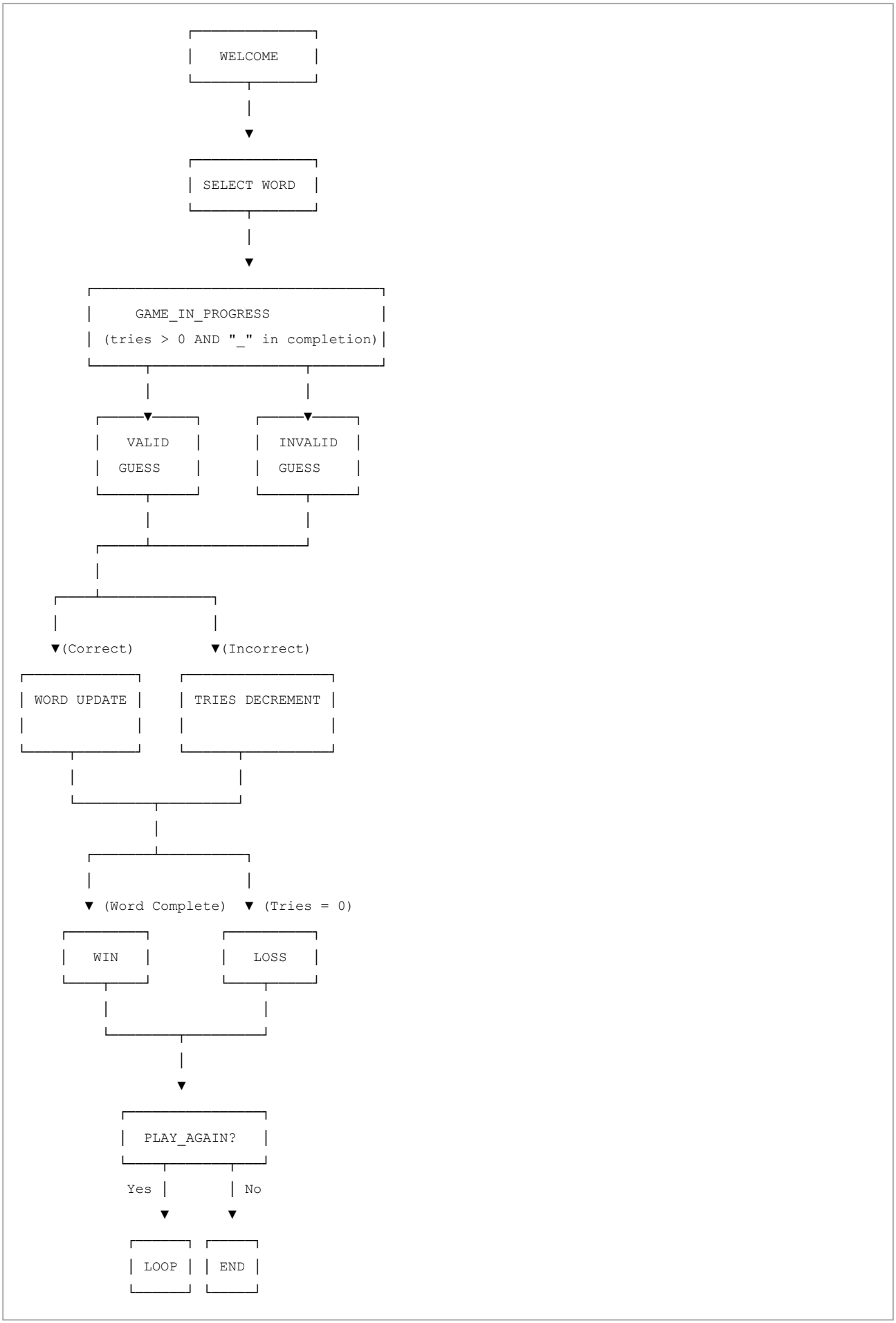
7.2 Workflow Diagram


Start
Display Welcome
Start Main Loop
Select Random Word
Initialize Variables
(tries=6, guessed=[])
Game Loop Start
Display Hangman
Display Word Progress
Get Player Input
(Valid)
(Invalid)
Check New
Show Error
(Duplicate)
(New)
Warn
Check in Word
(Yes)
(No)
Show Success
Decrease Tries
Show Failure
Check End Condition
(Win)
(Loss/Continue)
Congratulate
Continue Game



7.4 Class/Component Diagram

7.5 State Transition Diagram


WELCOME
SELECT WORD
GAME_IN_PROGRESS
(tries > 0 AND "_" in completion)
VALID GUESS
INVALID GUESS
(Correct)
(Incorrect)
WORD UPDATE
TRIES DECREMENT
(Word Complete)
(Tries = 0)
WIN
LOSS
PLAY_AGAIN?
Yes
No
LOOP
END

8. DESIGN DECISIONS & RATIONALE

8.1 Word Bank Storage

Decision: Use a static list of predefined words instead of reading from file

Rationale:

- Simplifies implementation for educational purposes
- Eliminates file I/O complexity and file dependency issues
- Provides predictable, consistent word selection
- Faster execution without file operations
- Easy to modify word list directly in code

8.2 Limited Try System (6 attempts)

Decision: Set game attempts to 6 instead of unlimited or 10

Rationale:

- Aligns with traditional Hangman (7-stage hangman art)
- Provides balanced difficulty for casual play
- Creates reasonable challenge without frustration
- Correlates with hangman ASCII art stages (0-6)
- Encourages strategic letter selection

8.3 Lowercase Letter Processing

Decision: Convert all guesses to lowercase for comparison

Rationale:

- Provides user-friendly experience (uppercase/lowercase accepted)
- Simplifies comparison logic
- Maintains consistency in letter matching
- Prevents case-sensitivity errors
- Improves accessibility for casual players

8.4 Duplicate Guess Prevention

Decision: Track and reject previously guessed letters

Rationale:

- Maintains game integrity
- Prevents accidental repeated guesses
- Teaches list management and state tracking
- Provides meaningful error feedback
- Prevents exploiting system limitations

8.5 ASCII Art Visual Feedback

Decision: Display 7-stage hangman ASCII art based on remaining tries

Rationale:

- Provides immediate visual feedback on game state
- Increases engagement and entertainment value
- Demonstrates string handling and indexing
- Maintains game atmosphere and theme
- Makes game progress visually apparent

8.6 Console-Based Interface

Decision: Single-player console application vs network/GUI

Rationale:

- Simplicity suitable for educational project
- Focuses on core game logic rather than UI complexities
- Platform-independent (works on all systems)
- Easier debugging and testing
- Reduces development time
- Suitable for algorithm and logic demonstration

8.7 Single Game Session Control

Decision: Loop allows multiple games in one program run

Rationale:

- Improves user experience (no restart required)
- Demonstrates loop and control flow concepts
- Enables consecutive game sessions
- Shows state reset and re-initialization
- Maintains player engagement

---

9. IMPLEMENTATION DETAILS

9.1 Code Structure

The implementation follows this organization:

Program Flow:

```
1. Program Start
2. While Loop: Ask to Play
3. If Yes:
   a. Main Function
      - Select random word
      - Initialize tries=6
      - Call game()
   b. Game Function
      - Game loop while tries > 0 AND "_" in word_completion
      - Validate input
      - Check for duplicates
      - Check if letter in word
      - Update state
      - Display progress
   c. Display Function
      - Return appropriate ASCII art based on tries
4. Repeat or Exit
```

9.2 Key Functions & Components

Word Selection:

```
words = ["apple", "banana", "cherry", "date", "elderberry", "fig", "grape", "honeydew", "kiwi", "lemon"]
wrd_selected = random.choice(words)
```

Input Validation:

```
if len(guess) != 1 or not guess.isalpha():
    print("Invalid input. Please enter a single letter.")
    continue
```

Duplicate Detection:

```
if guess in guessed_letters:
    print("You already guessed that letter. Try again.")
    continue
```

Word Completion Update:

```
word_completion = ''.join([letter if letter in guessed_letters else ' _ ' for letter in word])
```

Tries Management:

```
if guess in word:
    print(f"Good job! '{guess}' is in the word.")
else:
    tries -= 1
    print(f"Sorry, '{guess}' is not in the word. You have {tries} tries left.")
```

9.3 Key Algorithms

1. Word Masking Algorithm:

- Compare each character in word against guessed_letters list
- Display letter if guessed, underscore with spaces if not
- Update dynamically after each valid guess

2. Letter Guessing Logic:

- Accept input
- Check validity (single alphabetic character)
- Check if previously guessed
- Check if in target word
- Update tries and display accordingly

3. Win Condition Detection:

- Check if underscore remains in word_completion
- If no underscore: Player wins
- Trigger only when all letters revealed

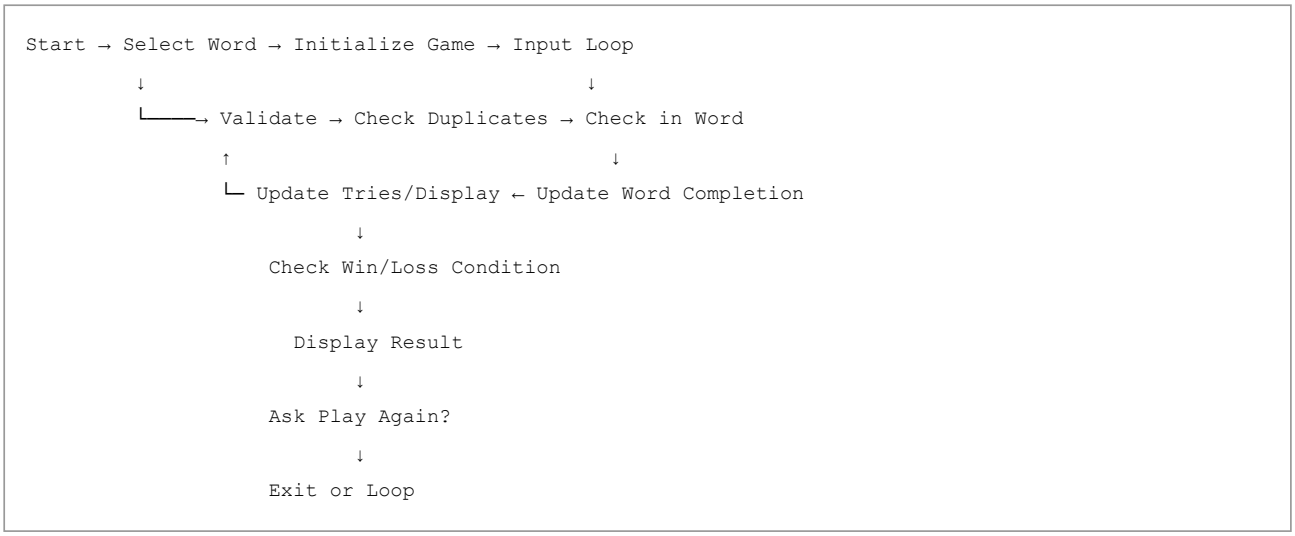
4. Loss Condition Detection:

- Check if tries = 0
- Check if underscore still in word_completion
- If both true: Player loses
- Reveal complete word

5. ASCII Art Progression:

- 7 hangman stages (empty to complete)
- Index by (6 - tries) to get current stage
- Display updates after each incorrect guess

9.4 Data Flow



---

10. SCREENSHOTS / RESULTS

Sample Output 1: Game Start

```
Welcome to Hangman!
Guess the word before the hangman is complete!


--------
|      |
|
|
|

apple:  _  _  _  _  _
Please enter a letter: a
Good job! 'a' is in the word.
a  _  _  _  _
```

Sample Output 2: Correct Guess Sequence

```
--------
|      |
|
|
|


_  _  _  _  _
Please enter a letter: e
Good job! 'e' is in the word.
a  _  _  _  e

--------
|      |
|
|
|

Please enter a letter: p
Good job! 'p' is in the word.
a  p  p  _  e
```

Sample Output 3: Incorrect Guess

```
--------
|      |
|
|
|

a  p  p  _  e
Please enter a letter: z
Sorry, 'z' is not in the word. You have 5 tries left.

--------
|      |
|      O
|
|

a  p  p  _  e
```

Sample Output 4: Win Condition

```
--------
|      |
|
|
|

a  p  p  l  e
Congratulations! You've guessed the word!
```

Sample Output 5: Loss Condition

```
--------
|      |
|      O
|     \|/
|      |
|     / \


Game over! The word was 'banana'.
```

Sample Output 6: Play Again Prompt

```
Do you want to play Hangman? (y/n): y
Welcome to Hangman!
```

---

11. TESTING APPROACH

11.1 Test Cases

Test Case 1: Valid Single Letter Input

- Input: "a"
- Expected: Accept and process
- Result: ✓ Pass

Test Case 2: Invalid Multi-Character Input

- Input: "abc"
- Expected: Reject with error message
- Result: ✓ Pass

Test Case 3: Non-Alphabetic Input

- Input: "1" or "!"
- Expected: Reject with error message
- Result: ✓ Pass

Test Case 4: Duplicate Letter Guess

- Input: "a" (twice in same game)
- Expected: Accept first, reject second with warning
- Result: ✓ Pass

Test Case 5: Correct Letter in Word

- Input: "a" for word "apple"
- Expected: Reveal 'a' positions, maintain tries
- Result: ✓ Pass

Test Case 6: Incorrect Letter in Word

- Input: "z" for word "apple"
- Expected: Decrement tries, show hangman progression
- Result: ✓ Pass

Test Case 7: Complete Word Guess (Win)

- Input: "a", "p", "l", "e" in sequence
- Expected: Display all letters, show congratulations, end game
- Result: ✓ Pass

Test Case 8: Exhausted Tries (Loss)

- Input: 6+ incorrect letters
- Expected: Display complete hangman, show word, end game
- Result: ✓ Pass

Test Case 9: Case Insensitivity

- Input: "A" (uppercase)
- Expected: Convert to lowercase, process as "a"
- Result: ✓ Pass

Test Case 10: Play Again Functionality

- Input: "y" after game complete

- Expected: New game starts with new word
- Result: ✓ Pass

11.2 Edge Cases Tested

- Empty input (just pressing Enter)
- Whitespace input
- Special characters and symbols
- Uppercase vs lowercase letters
- First letter being correct vs incorrect
- All incorrect guesses leading to loss
- All correct guesses on first try (best case)
- Invalid responses to play-again prompt (default to exit)

11.3 Integration Testing

- Complete game flow from start to finish (win path)
- Complete game flow from start to finish (loss path)
- Multiple consecutive games (state reset verification)
- Word bank randomization (verify different words selected)
- Hangman art display consistency across all stages

---

12. CHALLENGES FACED

12.1 Technical Challenges

1. **Word Masking with Variable Length Words**
    - Challenge: Maintaining correct spacing and underscore placement
    - Solution: Used list comprehension with space formatting: `' _ '` for proper alignment
2. **Input Validation Logic**
    - Challenge: Handling multiple invalid input types
    - Solution: Combined length check and isalpha() method for robust validation
3. **Duplicate Prevention Efficiency**
    - Challenge: Checking against previous guesses accurately
    - Solution: Maintained guessed_letters list and checked membership before processing
4. **ASCII Art Integration**
    - Challenge: Proper indexing of hangman stages array
    - Solution: Reversed index calculation (tries value directly maps to stage)
5. **Game State Management**
    - Challenge: Resetting state between multiple games
    - Solution: Reinitialize all variables within main() at each game start

12.2 Conceptual Challenges

1. **String Manipulation**

- Challenge: Updating word display with correct character handling
- Solution: Used list comprehension for elegant character-by-character processing

2. **Game Loop Logic**

- Challenge: Multiple exit conditions (win, loss, or continue)
- Solution: Structured while loop with clear condition checking

3. **User Experience**

- Challenge: Making game intuitive without complex UI
- Solution: Clear prompts and immediate visual/textual feedback

4. **Random Selection**

- Challenge: Ensuring truly random word selection
- Solution: Used random.choice() for unbiased selection from word list

---

13. LEARNINGS & KEY TAKEAWAYS

13.1 Programming Concepts Learned

1. **String Operations**

- String slicing, indexing, and comparison
- List comprehensions for string transformation
- String case conversion (.lower())
- Character-by-character iteration

2. **Control Flow**

- Complex while loop conditions
- Nested if-elif-else statements
- Loop continuation and termination logic
- Conditional branching for game paths

3. **Data Structures**

- List creation and manipulation
- List membership checking (in operator)
- List append operations
- Index-based array access

4. **Input/Output & Validation**

- Console input handling
- Input type checking (isalpha(), len())
- Error messaging for invalid input
- Formatted output with f-strings

5. **Functions and Modularity**

- Function definition and calling
- Parameter passing
- Return values
- Function separation of concerns

13.2 Game Development Concepts

1. **Game Loop Architecture**
    - State management in game loops
    - Update-render pattern
    - Event processing (user input)
    - Win/loss condition evaluation
2. **Game State**
    - Tracking multiple game variables
    - State transitions and updates
    - Resetting state between sessions
    - Maintaining consistency
3. **Player Feedback**
    - Immediate visual feedback (hangman art)
    - Textual feedback (messages)
    - Clear game progress indication
    - Meaningful error messages
4. **Game Balance**
    - Difficulty calibration (6 tries)
    - Fair game mechanics
    - Balanced word selection
    - Player engagement factors

13.3 Software Engineering Practices

1. **Code Organization**
    - Modular function design
    - Separation of concerns
    - Clear code structure
    - Logical program flow
2. **Validation & Error Handling**
    - Input validation importance
    - Edge case consideration
    - Graceful error handling
    - Prevention of invalid states
3. **User Experience Design**
    - Clear instructions
    - Intuitive interface
    - Meaningful feedback
    - Accessibility considerations

13.4 Debugging & Problem-Solving

1. **Systematic Debugging**

- Identifying logic errors
- Tracing program flow
- Testing edge cases
- Verifying assumptions

2. **Algorithm Development**

- Breaking problems into steps
- Implementing solutions
- Testing algorithms thoroughly
- Optimizing for clarity

---

14. FUTURE ENHANCEMENTS

14.1 Immediate Enhancements

1. **Expanded Word Bank**

- Add multiple word categories (animals, countries, foods)
- Implement difficulty levels (easy/medium/hard words)
- Load words from external file for scalability

2. **Difficulty Levels**

- Easy: 8 tries, common words
- Medium: 6 tries, moderate difficulty
- Hard: 4 tries, complex words
- Adjustable word lengths

3. **Scoring System**

- Points based on guesses remaining
- Bonus points for perfect wins
- Running score across multiple games
- Leaderboard functionality

14.2 Advanced Features

1. **Multiplayer Support**

- Two-player mode with word choosing
- Turn-based gameplay
- Competitive scoring
- Network-based multiplayer

2. **Graphical User Interface (GUI)**

- Python tkinter implementation
- Visual hangman drawing
- Click-based letter selection
- Game statistics dashboard

3. **Hint System**

- Category hints (e.g., "This is a fruit")

- Letter position hints
- Definition-based hints
- Limited hint usage per game

4. **Statistics & Analytics**

- Win/loss ratio tracking
- Average guesses per game
- Most frequently guessed letters
- Performance analytics dashboard

14.3 Content and Configuration

1. **Word Customization**

- User-provided word lists
- Theme-based word selection
- Dictionary integration
- Custom difficulty parameters

2. **Localization**

- Multiple language support
- Unicode character handling
- Language-specific word banks
- Internationalization framework

3. **Game Variations**

- Speed mode (time limit)
- Hangman with categories
- Phrase-based hangman
- Custom word selection by player

14.4 Advanced Capabilities

1. **Artificial Intelligence**

- AI opponent implementation
- Optimal letter guessing strategy
- Dynamic difficulty adjustment
- Machine learning for word selection

2. **Web Integration**

- Web-based implementation (Flask/Django)
- Online multiplayer
- Cloud-based leaderboard
- Cross-platform accessibility

3. **Mobile Application**

- Native mobile apps (iOS/Android)
- Touch-based interface
- Push notifications
- Offline play capability

---

15. REFERENCES

15.1 Game Design & Theory

[1] McShaffry, M., & Novak, D. (2012). *Game Coding Complete* (4th ed.). Course Technology.

[2] Schell, J. (2019). *The Art of Game Design: A Book of Lenses* (3rd ed.). CRC Press.

[3] Swink, S. (2008). *Game Feel: A Game Programmer's Guide to Virtual Sensation*. CRC Press.

15.2 Python Programming References

[4] Matthes, E. (2019). *Python Crash Course* (2nd ed.). No Starch Press.

[5] Sweigart, A. (2019). *Automate the Boring Stuff with Python* (2nd ed.). No Starch Press.

[6] Python Software Foundation. (2025). Python Official Documentation. Retrieved from https://docs.python.org/3/ (https://docs.python.org/3/)

15.3 Algorithm & Data Structure References

[7] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[8] Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.

15.4 Educational Resources

[9] GeeksforGeeks. (2024). Python Tutorials. Retrieved from https://www.geeksforgeeks.org/python/ (https://www.geeksforgeeks.org/python/)

[10] Real Python. (2024). Python Tutorials and Resources. Retrieved from https://realpython.com/ (https://realpython.com/)

END OF REPORT

This project report documents the complete lifecycle of the Hangman Word-Guessing Game application, from conceptualization through implementation and identified future enhancement opportunities.