Project Title: Word Hunt

An AI-Enhanced Word Search Game with Advanced Features

Submitted By:
Syeda Fakhira Saghir [22k-4413]
Aafreen Mughal [22k-4448]
Muhammad Raza [22k-4499]
Course: AI
Instructor: Sir Khalid Khan
Submission Date: May 2025

1 Introduction

Word Hunt is an AI-powered word search game that dynamically generates puzzles. The game features:

- Three distinct game play modes:
  1. Single-player
  2. Two-player
  3. Human vs AI

- 20 progressive difficulty levels with increasing grid sizes (5x5 to 16x16)
- Thematic word categories with visual backgrounds according to the theme
- Intelligent AI opponent mode allows user to play against AI
- Comprehensive scoring system with time bonuses

2 Word Extraction and Processing

2.1 Word Bank Creation

The game builds its vocabulary through multiple methods:

1. **Initial Word Bank is defined**
2. **Web Scraping is used to add words to the list**
3. **Heuristic Selection**:
   - Words scored by length (4-8 letters preferred)
   - Letter diversity (unique letters bonus)
   - Common letter frequency (E,T,A,O,I,N bonus)

2.2 Word Similarity Analysis

The game uses TF-IDF vectorization and k-nearest neighbors to:

- Find semantically related words
- Generate contextual hints
- Enhance AI decision-making

3 Game Flow

3.1 Main Menu Options

Game Mode Selection:

- **Game Modes**: Single-player, Two-player, Human vs AI
- **Progression**: 20 levels (5x5 to 16x16 grids)
- **Themes**: Animals, Fruits, Countries with Pexels API backgrounds
- **AI**: Adaptive difficulty using multiple search strategies

3.2 Core Gameplay Loop

1. **Grid Generation**:
   - Words placed horizontally, vertically or diagonally
   - Valid intersections ensured
   - Remaining spaces filled with random letters
2. **Word Finding**:
   - Players can:
     - Click adjacent letters
     - Type words directly
     - Request hints
   - Validation checks:
     - Word exists in grid
     - Letters are properly connected
     - Word not already found
3. **Scoring**:
   - Base points: word length $\times$ 10
   - Time bonus for fast completion
   - Level completion bonus: grid size $\times$ 20

4. **Progression**:
    - Grid size increases every 3 levels
    - AI difficulty adapts to player level
    - New themes unlock with progress

4 AI Implementation

4.1 Search Strategies

Table 1: AI Search Strategy Comparison

Strategy	Time Complexity	Use Case	Difficulty Level
Random	O(1)	Easy Mode	1-2
Longest Word	O(n log n)	Medium Mode	3-4
BFS	O(n)	Hard Mode	5
Minimax	$O(b^d)$	Expert Mode	5

4.2 AI Algorithms Implementation Search Strategies

Table 2: AI Algorithms and Their Applications

Algorithm	**Purpose**	**Implementation Details**
TF-IDF + KNN	Word similarity	• Vectorizes words • Finds semantic neighbors • Powers hint generation
Breadth-First Search	Word discovery	• Explores all possible words • Guarantees shortest path • Used in Hard difficulty
Minimax	Optimal moves	• With alpha-beta pruning • Evaluates word selections • Used in Expert mode
Hill Climbing	Local optimization	• Random restarts • Scores word choices • Balances exploration/exploitation

4.3 Usage in Game

- **Difficulty Scaling**:
    - *Easy (1-2)*: Random selection (O(1))
    - *Medium (3-4)*: Longest word priority (O(n log n))
    - *Hard (5+)*: BFS/DFS with depth limit
- **Hint Generation**:

  - Uses KNN to find related words
  - Considers word frequency and position
  - Provides contextual clues (first/last letters)

4.4 Adaptive Difficulty

The AI adjusts based on:

- Player level (1-20)
- Average word length in current grid
- Player performance history

5 Technical Specifications

- **Programming Language**: Python 3.8+
- **Libraries**:
  - Tkinter (GUI)
  - Scikit-learn (NLP)
  - BeautifulSoup (Web Scraping)
  - Requests (API Calls)
  - Pillow (Image Processing)
- **System Requirements**:
  - 2GHz processor
  - 4GB RAM
  - 50MB disk space

6 Natural Language Processing (NLP) Integration

6.1 NLP Applications

The game employs NLP techniques in several core functionalities:

1. **Word Similarity Analysis**:
   - Uses `TfidfVectorizer` from scikit-learn to create word embeddings
   - Implements k-nearest neighbors (KNN) to find semantically related words
   - Finds synonyms and related terms for hints
2. **Web Scraping for Vocabulary Expansion**:
   - Parses HTML from dictionary sites using BeautifulSoup

- Extracts synonyms and related terms contextually
- Filters words by length and validity for game use

3. **Word Placement Heuristics**:
   - Scores words based on letter frequency patterns
   - Prioritizes words with common prefixes/suffixes
4. **Hint Generation**:
   - Provides word definitions as contextual clues
   - Shows first/last letters with length indicators
5. **AI Decision Making**:
   - Uses word frequency statistics
   - Considers letter distribution patterns

6.2 Limitations

The implementation has some NLP constraints:

- No deep semantic analysis (word2vec/GPT-style understanding)
- Limited contextual awareness beyond basic similarity
- Dependency on pre-scraped word banks for performance

Table 3: NLP Components vs Traditional Programming

Feature	**NLP Technique Used**	**Traditional Alternative**
Word Similarity	TF-IDF + KNN	Hardcoded synonym lists
Hint Generation	Contextual analysis	Random word selection
AI Word Choice	Frequency analysis	Pure random selection

6.3 Future NLP Enhancements

Potential improvements include:

- Integrating Word2Vec for better semantic relationships
- Adding word sense disambiguation for better hints
- Implementing simple language models for dynamic clue generation

7 Visual Design: Pexels API Integration

7.1 API Implementation

The game dynamically loads thematic background images using the Pexels API

7.2 Image Loading Process

The system follows this workflow:

1. **Theme Selection**:
   - Maps game themes to search queries (e.g., "wild animals aesthetic")
   - Uses HTTPS requests with API key authentication
2. **API Request**:

```
params = {"query": search_term,"per_page": 10,
    "orientation": "landscape","size": "large"
}
headers = {"Authorization": PEXELS_API_KEY}
response = requests.get(PEXELS_BASE_URL,  headers=headers, params=paran
```

3. **Image Processing**:
   - Selects random image from API results
   - Applies preprocessing:
     - Resizing to window dimensions
     - Gaussian blur for readability
     - Brightness adjustment
   - Converts to Tkinter-compatible format

Conclusion

Word Hunt successfully demonstrates:

- Practical NLP applications in game design
- Effective AI opponent implementation
- Engaging progressive difficulty system
- Robust word processing pipeline
- Intuitive user interface

The project showcases how traditional word games can be enhanced with modern AI techniques while maintaining accessibility and fun gameplay.

References

[1] Scikit-learn: Machine Learning in Python, Pedregosa et al.

[2] Tkinter GUI Application Development, Bhaskar Chaudhary

[3] Speech and Language Processing, Jurafsky & Martin