3/1/2025

# Applied Ai

Report

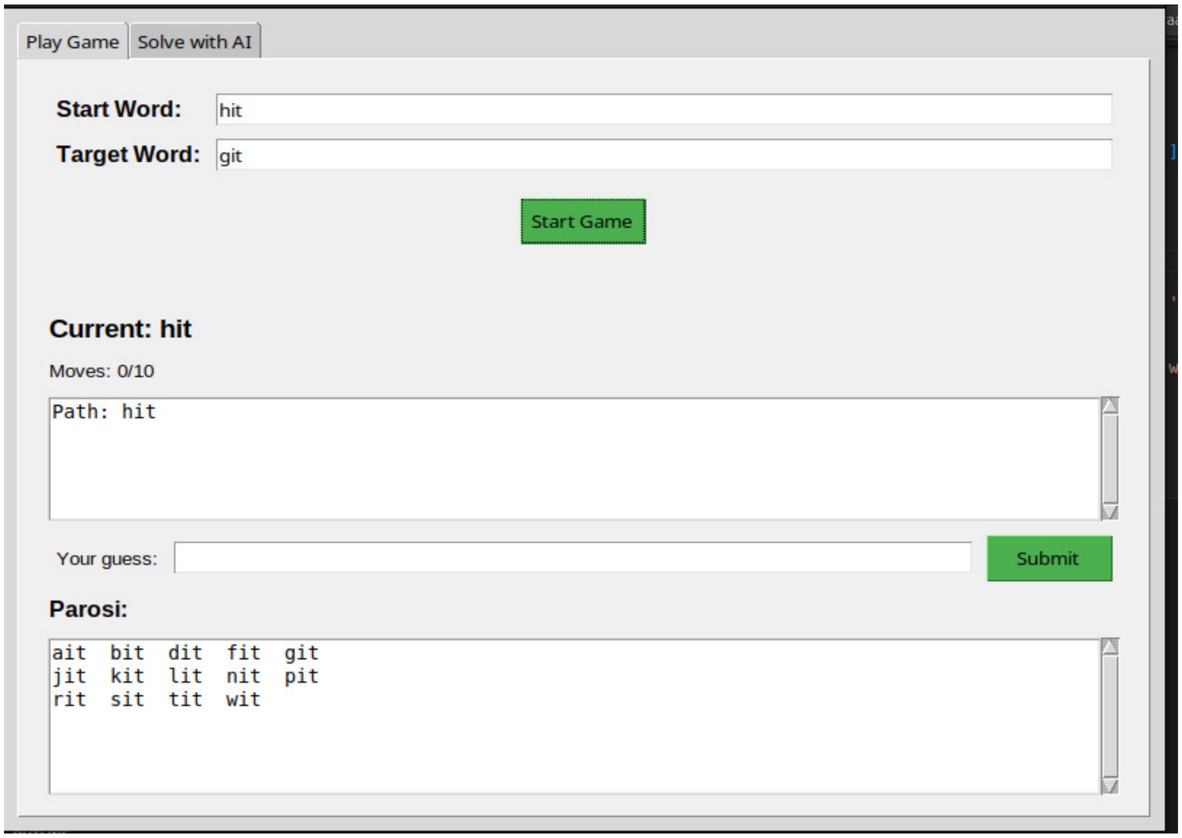
Muhammad Awais Rafique
Hassaan

# Report on Programming with AI Project

## 1. Introduction

The "Programming with AI" project aims to develop an intelligent system that assists in various programming tasks, including code generation, debugging, and optimization. This report details the approach taken, the algorithms implemented, and the challenges faced during the development process.

**Screenshots of Project:**



Play Game | Solve with AI

**Start Word:** hit

**Target Word:** git

Start Game

**Current: hit**

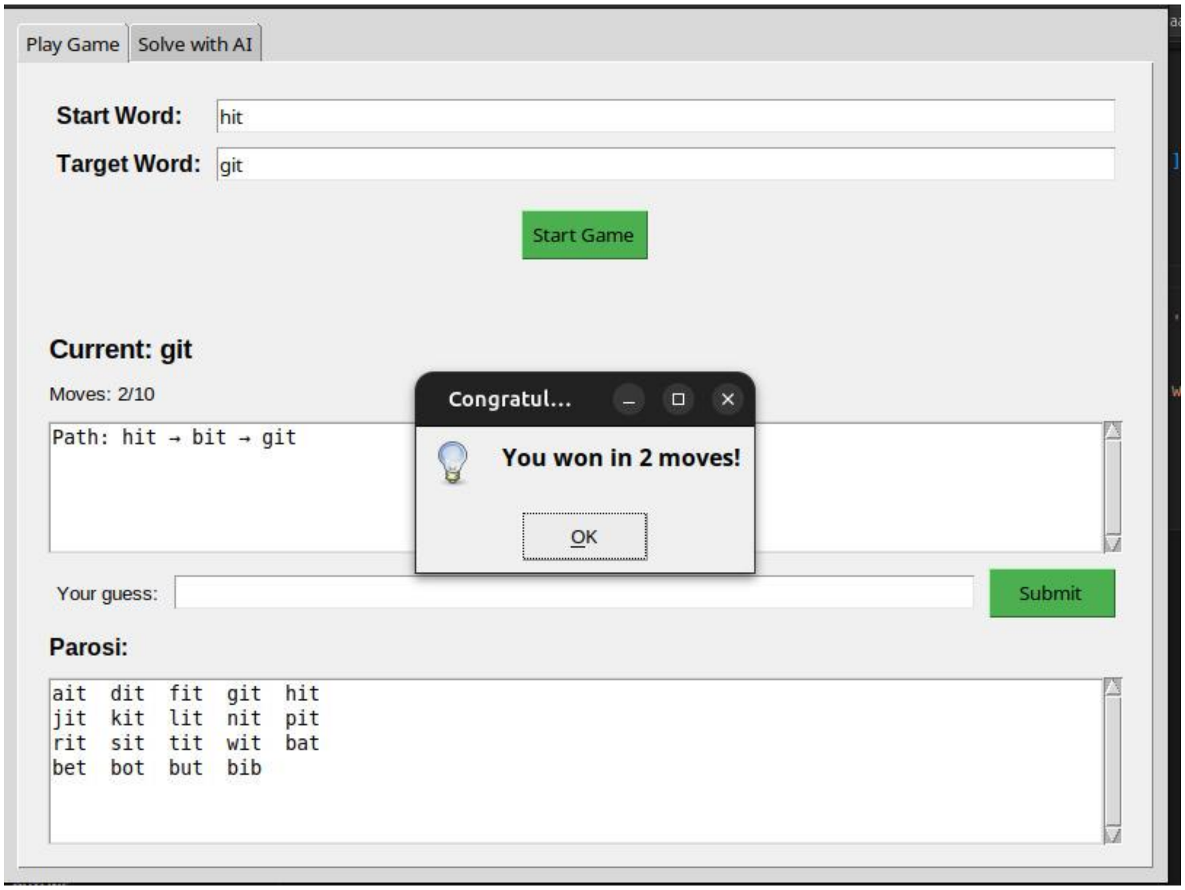
Moves: 0/10

```
Path: hit
```

Your guess: Submit

**Parosi:**

```
ait  bit  dit  fit  git
jit  kit  lit  nit  pit
rit  sit  tit  wit
```



Play Game | Solve with AI

**Start Word:** hit

**Target Word:** git

Start Game

**Current: git**

Moves: 2/10

```
Path: hit → bit → git
```

Congratul...

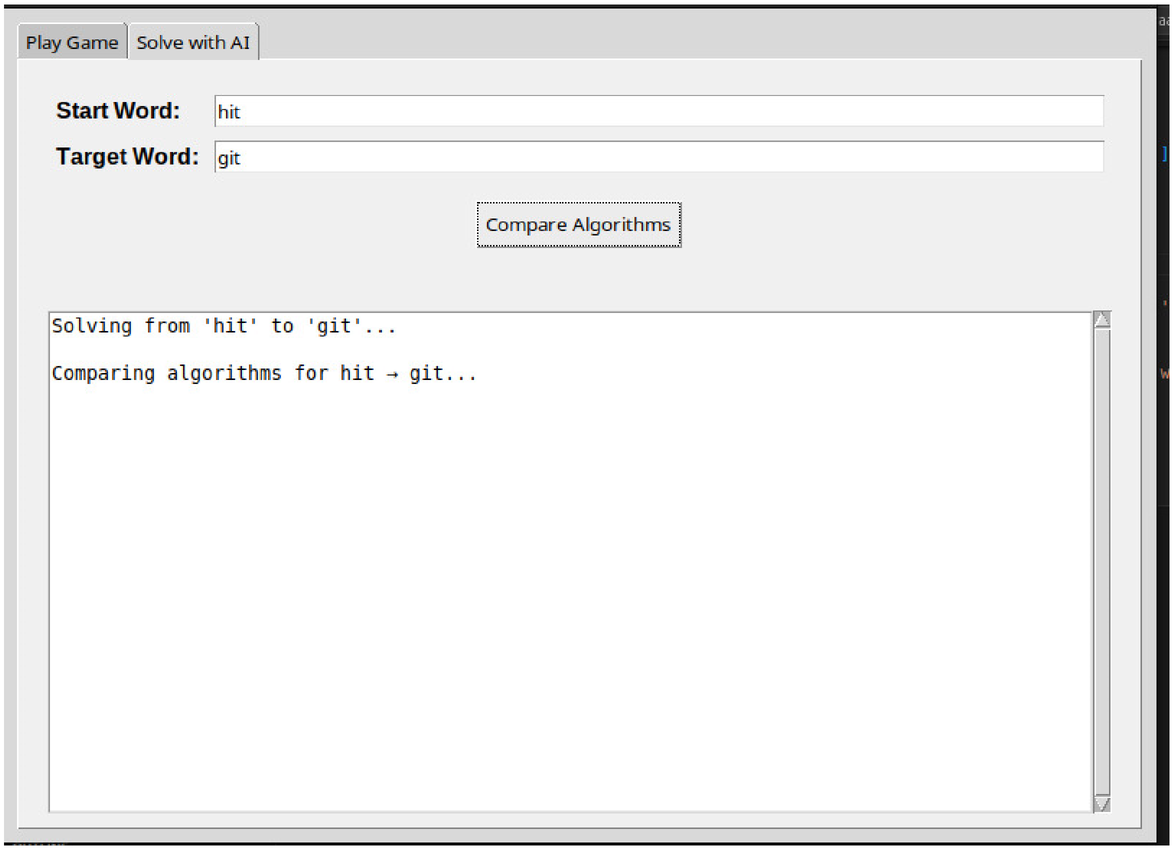
You won in 2 moves!

OK

Your guess: Submit

**Parosi:**

```
ait  dit  fit  git  hit
jit  kit  lit  nit  pit
rit  sit  tit  wit  bat
bet  bot  but  bib
```

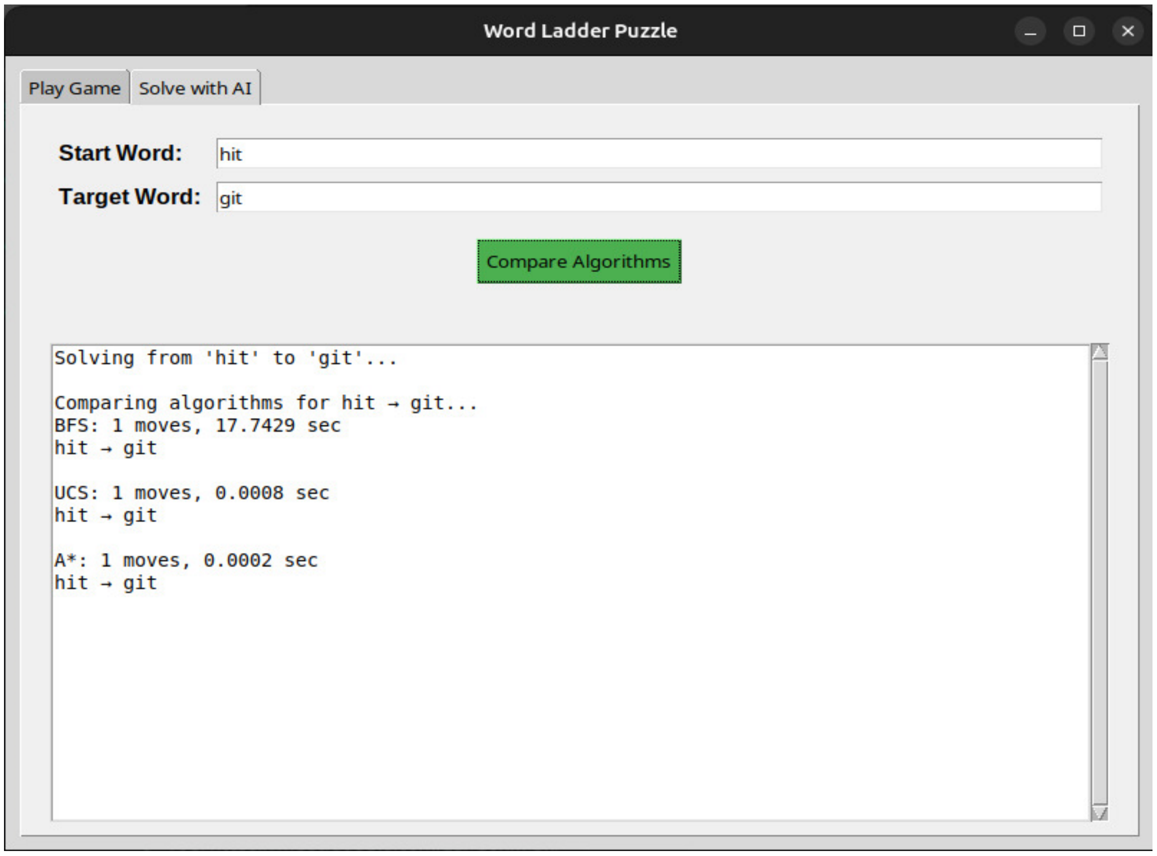


Start Word: hit

Target Word: git

```
Solving from 'hit' to 'git'...

Comparing algorithms for hit → git...
```



Start Word: hit

Target Word: git

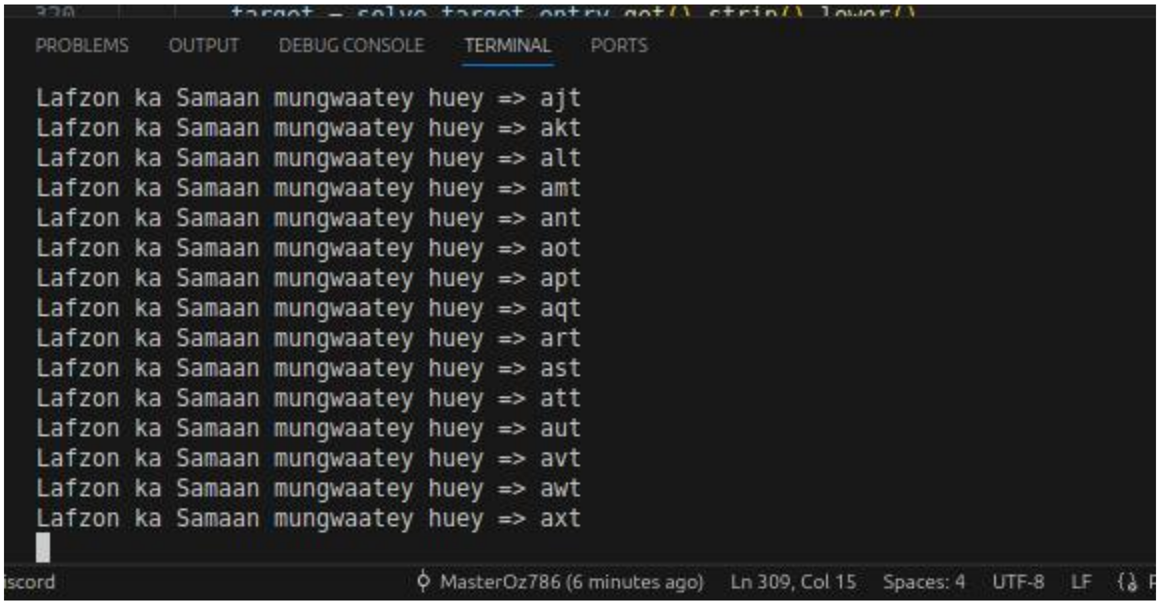
```
Solving from 'hit' to 'git'...

Comparing algorithms for hit → git...
BFS: 1 moves, 17.7429 sec
hit → git

UCS: 1 moves, 0.0008 sec
hit → git

A*: 1 moves, 0.0002 sec
hit → git
```

**Api fetching:**



```
Lafzon ka Samaan mungwaatey huey => ajt
Lafzon ka Samaan mungwaatey huey => akt
Lafzon ka Samaan mungwaatey huey => alt
Lafzon ka Samaan mungwaatey huey => amt
Lafzon ka Samaan mungwaatey huey => ant
Lafzon ka Samaan mungwaatey huey => aot
Lafzon ka Samaan mungwaatey huey => apt
Lafzon ka Samaan mungwaatey huey => aqt
Lafzon ka Samaan mungwaatey huey => art
Lafzon ka Samaan mungwaatey huey => ast
Lafzon ka Samaan mungwaatey huey => att
Lafzon ka Samaan mungwaatey huey => aut
Lafzon ka Samaan mungwaatey huey => avt
Lafzon ka Samaan mungwaatey huey => awt
Lafzon ka Samaan mungwaatey huey => axt
```

**Report on Word Ladder Puzzle Solver**

**1. Introduction**

The "Word Ladder Puzzle Solver" project aims to implement intelligent search algorithms to solve the word ladder puzzle efficiently. This report details the approach taken, the algorithms implemented, and the challenges faced during the development process.

**2. Approach**

**2.1 Project Planning**

The project was planned in multiple phases:

- **Problem Understanding**: Analyzing the structure of the word ladder puzzle and defining its constraints.
- **Algorithm Selection**: Identifying suitable search algorithms for solving the puzzle.
- **Implementation**: Writing the algorithms in a programming language and optimizing their performance.
- **Testing and Evaluation**: Comparing different search algorithms based on efficiency and accuracy.

**2.2 Technology Stack**

The project utilized a combination of technologies:

- **Programming Language**: Python

- **Data Structures**: Graphs, Queues, and Heaps
- **Algorithmic Techniques**: Graph traversal and heuristic search

## 3. Algorithm

The project implements three different search algorithms to find the shortest path in a word ladder puzzle:

### 3.1 Breadth-First Search (BFS)

- **Approach**: Explores all words at the current level before moving to the next level.
- **Data Structure Used**: Queue
- **Time Complexity**: O(N * M^2), where N is the number of words and M is the word length.
- **Pros**: Guarantees the shortest path.
- **Cons**: Can be slow for large dictionaries.

### 3.2 Uniform Cost Search (UCS)

- **Approach**: Uses a priority queue to always expand the lowest-cost path first.
- **Data Structure Used**: Priority Queue (Min-Heap)
- **Time Complexity**: O(E log V), where E is the number of edges and V is the number of nodes.
- **Pros**: Guarantees optimality when all edges have equal cost.
- **Cons**: Slower than BFS if uniform edge costs are used.

### 3.3 A* Search (A*)

- **Approach**: Uses a heuristic function to guide the search towards the goal faster.
- **Data Structure Used**: Priority Queue (Min-Heap)
- **Time Complexity**: O(E log V)
- **Pros**: More efficient than BFS and UCS when a good heuristic is used.
- **Cons**: Requires a well-designed heuristic function.

### 3.4 Heuristic Function for A*

- **Heuristic Used**: Hamming Distance (Number of character mismatches between current word and target word)

- **Purpose**: Helps prioritize words that are closer to the goal state.

## 4. Challenges Faced

### 4.1 Large Word Datasets

- **Memory Constraints**: Storing a large word list required efficient data structures.
- **Performance Optimization**: Searching through a large dictionary efficiently was a challenge.

### 4.2 Algorithm Efficiency

- **BFS Performance**: Slow on large inputs due to exhaustive search.
- *A Heuristic Design**: Choosing the best heuristic function required experimentation.

### 4.3 Edge Cases

- **Disconnected Graphs**: Handling scenarios where no valid transformation path exists.
- **Identical Start and End Words**: Ensuring the algorithm correctly returns a trivial solution.

## 5. Conclusion

The "Word Ladder Puzzle Solver" successfully implemented three graph search algorithms—BFS, UCS, and A* to efficiently solve word ladder puzzles. Each algorithm has trade-offs, and A* proved to be the most efficient with a well-chosen heuristic.

## 6. Future Work

- **Improve Heuristic Functions**: Experimenting with different heuristics for better efficiency.
- **Parallel Processing**: Utilizing multi-threading to improve search speed.
- **Support for More Complex Puzzles**: Extending the solver to handle different variations of word ladder puzzles.