# 🌟 Hoshi: A\* Pathfinding Algorithm in MIPS Assembly

## 📝 Overview

**Hoshi** (星, Japanese for “star”) is a complete implementation of the A\* pathfinding algorithm in MIPS assembly language. This project demonstrates low-level programming concepts by implementing a complex algorithmic solution in assembly language, offering valuable insights into computer architecture and optimization techniques.

## 🚀 Features

* **✅ Complete A\* algorithm implementation** in pure MIPS assembly
* **✅ Priority Queue** with efficient insertion and extraction operations
* **✅ Manhattan Distance Heuristic** for optimal path calculation
* **✅ Visual Representation** through bitmap display for step-by-step algorithm progression
* **✅ Obstacle Detection** with robust path planning around barriers
* **✅ Path Reconstruction** with visual highlighting of the optimal route

## 🔍 What is A\*?

A\* (pronounced “A-star”) is an informed search algorithm widely used in pathfinding and graph traversal. It efficiently plots a traversable path between multiple nodes by maintaining a priority queue of paths and choosing the lowest-cost path to expand.

### Core Components

1. **g(n)**: Cost from the start node to the current node
2. **h(n)**: Estimated cost from current node to goal (heuristic)
3. **f(n)**: Total cost, where f(n) = g(n) + h(n)

A\* guarantees an optimal solution (shortest path) when using an admissible heuristic that never overestimates the actual cost.

## 🖥️ Implementation Details

### Data Structures

* **Node Structure**: 32 bytes per node containing:
  + Position (x, y)
  + Wall status
  + g-score, h-score, f-score
  + Parent node reference
* **Priority Queue**: Binary heap implementation for efficient node retrieval
  + O(log n) insertion and extraction operations
  + Stores nodes by f-score for optimal path selection
* **Grid Representation**: 8x8 grid with obstacles (1) and free space (0)

### Algorithm Flow

1. **Initialization**: Create start node, calculate heuristic, add to open set
2. **Main Loop**: Until goal is found or open set is empty
   * Extract node with lowest f-score
   * Check if goal is reached
   * Add current node to closed set
   * Process valid neighbors
3. **Path Reconstruction**: Trace parent nodes from goal to start

### Visualization System

The bitmap display provides real-time visualization of: - ⬜ Free cells and ⬛ obstacles - 🟡 Currently processed nodes - 🟢 Start position - 🟣 Goal position - 🔴 Final path

## 🎮 Demo

[!NOTE] Place your demo video link here. A visual demonstration helps viewers understand the algorithm’s execution.

[Watch the Hoshi A\* Algorithm in Action](your-video-link-here)

## 📊 Performance Analysis

The implementation demonstrates efficient pathfinding in constrained memory environments: - Successfully finds optimal paths in an 8x8 grid - Handles obstacle avoidance effectively - Visual feedback shows the algorithm’s exploration process

## 🧩 Technical Challenges Overcome

* **Memory Management**: Efficiently organizing data structures within limited memory
* **Register Allocation**: Strategic use of registers for critical operations
* **Bitmap Display**: Direct memory manipulation for visualization
* **Complex Algorithm Translation**: Converting high-level algorithm concepts to assembly instructions

## 🛠️ Project Structure

.  
├── .data # Data segment with grid definition, nodes, and heap structures  
│ ├── grid # Map representation (0=walkable, 1=obstacle)  
│ ├── nodes # Node storage for A\* algorithm  
│ └── heap # Priority queue implementation  
│  
├── .text # Code segment  
│ ├── main # Program entry point  
│ ├── a\_star # Core A\* algorithm implementation  
│ ├── visualization # Bitmap display functions  
│ └── helper # Utility functions (heuristics, queue operations)

## 🌟 Contributors

Special thanks to everyone who contributed to this project:

* **Saeed Khalid** (Joyboy)
* **Ahmed Walid**
* **Ahmed Khaled**
* **Hazem Ahmed**

## 📚 Lessons Learned

* **Low-level optimization** techniques for complex algorithms
* **Memory management** in assembly language
* **Efficient data structures** in constrained environments
* **Visual debugging** approaches for algorithm verification

## 🔮 Future Improvements

* Support for larger grid sizes
* Additional heuristic functions (Euclidean, Chebyshev, etc.)
* Dynamic obstacle placement
* Performance optimizations
* Interactive user controls

## 📜 License

This project is open source and available under the [MIT License](LICENSE).

Developed with 💻 and 🧠