# Bitmap Helper Module Documentation

## Overview

This document provides a detailed explanation of the bitmap helper module implemented in MIPS assembly. The module offers rendering functionality for the A\* pathfinding visualization, providing essential graphics operations for displaying the grid, nodes, and algorithm progress.

## Data Segment

### Display and Grid Constants

.eqv displayWidth, 16 # Width of the display in units (512 / 32 = 16)  
.eqv displayHeight, 16 # Height of the display in units (512 / 32 = 16)  
.eqv gridCellWidth, 2 # Width of each grid cell in display units  
.eqv gridCellHeight, 2 # Height of each grid cell in display units  
.eqv gridWidth, 8 # Width of the grid in cells  
.eqv gridHeight, 8 # Height of the grid in cells  
.eqv bitmapBaseAddress, 0x10040000 # Memory address of the bitmap display

These constants define the dimensions and properties of both the display and the grid: - displayWidth and displayHeight: Define the total dimensions of the bitmap display (16×16 units) - gridCellWidth and gridCellHeight: Define the size of each grid cell (2×2 display units) - gridWidth and gridHeight: Define the dimensions of the grid itself (8×8 cells) - bitmapBaseAddress: The memory address where the bitmap display begins in MIPS memory

### Grid Data

grid:  
 .word 0, 0, 0, 0, 0, 0, 0, 0 # Row 0  
 .word 0, 1, 0, 1, 0, 1, 0, 0 # Row 1  
 .word 0, 0, 0, 0, 0, 0, 0, 0 # Row 2  
 .word 1, 0, 1, 0, 0, 0, 1, 0 # Row 3  
 .word 0, 1, 0, 0, 0, 1, 0, 0 # Row 4  
 .word 0, 0, 1, 0, 0, 1, 0, 1 # Row 5  
 .word 0, 1, 0, 0, 0, 1, 0, 0 # Row 6  
 .word 1, 0, 1, 0, 0, 0, 1, 0 # Row 7

This data structure represents the grid map where: - 0: Represents a free/passable cell - 1: Represents a wall/obstacle

The grid is an 8×8 matrix with strategic placement of obstacles to make the pathfinding problem meaningful.

### Color Table

colorTable:  
 .word 0xFAF9F6 # White (background) 0  
 .word 0x000000 # Black (background) 1  
 .word 0x00ff00 # Green 2  
 .word 0xff0000 # Red 3  
 .word 0xffffff # White 4  
 .word 0xFFFF00 # Yellow 5  
 .word 0x0000FF # Blue 6  
 .word 0xFF00FF # Magenta 7  
 .word 0x00FFFF # Cyan 8  
 .word 0x808080 # Gray 9  
 .word 0xFFA500 # Orange 10  
 .word 0x800080 # Purple 11  
 .word 0xA52A2A # Brown 12

The color table defines a palette of 13 colors (indexed 0-12) that can be used for rendering different elements of the grid. Each color is represented as a 24-bit RGB value.

## Text Segment (Functions)

### clearScreen

clearScreen:  
 li $t0, bitmapBaseAddress  
 li $t1, displayWidth  
 mul $t1, $t1, displayHeight  
 sll $t1, $t1, 2  
 add $t1, $t1, bitmapBaseAddress  
 li $t2, 0x808080 # Gray color  
clearLoop:  
 sw $t2, 0($t0)  
 addi $t0, $t0, 4  
 blt $t0, $t1, clearLoop  
   
 #delay  
 li $a0, 20 # 20 milliseconds delay  
 li $v0, 32 # syscall for sleep  
 syscall  
  
 jr $ra

**Purpose:** Clears the entire bitmap display by filling it with a uniform gray color.

**Steps:** 1. Sets $t0 to the base address of the bitmap 2. Calculates the total display size in bytes: width × height × 4 (each pixel is 4 bytes) 3. Sets $t1 to point just beyond the last displayable pixel 4. Sets each pixel to the gray color (0x808080) 5. Adds a brief delay (20ms) for visual smoothness 6. Returns to the caller

### drawGrid

# $a0 = base address, $a1 = free space color, $a2 = wall color  
drawGrid:  
 li $s0, 0  
 move $s2, $a0  
outer\_loop:  
 bge $s0, gridHeight, drawGridEnd  
 li $s1, 0  
inner\_loop:  
 bge $s1, gridWidth, outer\_loop\_next  
 mul $t0, $s0, gridWidth  
 add $t0, $t0, $s1  
 sll $t0, $t0, 2  
 add $t0, $t0, $s2  
 lw $a2, 0($t0)  
 move $a0, $s1  
 move $a1, $s0  
 addi $sp, $sp, -4  
 sw $ra, 0($sp)  
 jal drawGridNode  
  
 #delay  
 li $a0, 10 # 10 milliseconds delay  
 li $v0, 32 # syscall for sleep  
 syscall  
  
 addi $s1, $s1, 1  
 j inner\_loop  
outer\_loop\_next:  
 addi $s0, $s0, 1  
 j outer\_loop  
drawGridEnd:  
 lw $ra, 0($sp)  
 addi $sp, $sp, 4  
 jr $ra

**Purpose:** Renders the entire grid on the bitmap display, differentiating between passable cells and walls.

**Parameters:** - $a0: Base address of the grid data - $a1: Color index for free/passable cells - $a2: Color index for wall/obstacle cells

**Steps:** 1. Uses nested loops to iterate through each cell in the grid (y in outer loop, x in inner loop) 2. Calculates the memory offset for the current grid cell 3. Loads the cell value (0 for free space, 1 for wall) 4. Uses this value as the color index for drawing the grid node 5. Adds a brief delay (10ms) between drawing each node for visual effect 6. Returns to caller after drawing the entire grid

### drawGridNode

drawGridNode:  
 # save return address  
 addi $sp, $sp, -4  
 sw $ra, 0($sp)  
  
 # baseX = a0 \* gridCellWidth  
 li $t5, gridCellWidth  
 mul $t5, $a0, $t5  
 move $s7, $t5  
  
 # baseY = a1 \* gridCellHeight  
 li $t6, gridCellHeight  
 mul $t6, $a1, $t6  
  
 addi $t7, $t5, gridCellWidth  
 addi $t8, $t6, gridCellHeight  
  
row\_loop\_bitmap:  
 bge $t6, $t8, finish  
 move $t5, $s7  
col\_loop\_bitmap:  
 bge $t5, $t7, next\_row\_bitmap  
 move $a0, $t5  
 move $a1, $t6  
 jal drawPixel  
 addi $t5, $t5, 1  
 j col\_loop\_bitmap  
  
next\_row\_bitmap:  
 addi $t6, $t6, 1  
 j row\_loop\_bitmap  
  
finish:  
 lw $ra, 0($sp)  
 addi $sp, $sp, 4  
 jr $ra

**Purpose:** Draws a single grid node (cell) at the specified grid coordinates with the specified color.

**Parameters:** - $a0: Grid x-coordinate - $a1: Grid y-coordinate - $a2: Color index for the node

**Steps:** 1. Converts grid coordinates to display coordinates by multiplying by cell dimensions 2. Calculates the boundaries of the cell in display coordinates 3. Uses nested loops to fill the entire cell area with pixels of the specified color 4. Returns to caller after the entire cell is drawn

### calculateAddress

calculateAddress:  
 li $v0, bitmapBaseAddress  
  
 mul $t0, $a1, displayWidth  
 add $t0, $t0, $a0  
 sll $t0, $t0, 2  
 add $v0, $v0, $t0  
 jr $ra

**Purpose:** Calculates the memory address for a given display coordinate.

**Parameters:** - $a0: Display x-coordinate - $a1: Display y-coordinate

**Returns:** - $v0: Memory address corresponding to the specified coordinates

**Steps:** 1. Starts with the base address of the bitmap display 2. Calculates the linear offset: (y \* displayWidth + x) \* 4 bytes per pixel 3. Adds this offset to the base address 4. Returns the resulting memory address

### getColor

getColor:  
 la $t0, colorTable  
 sll $t1, $a2, 2  
 add $t0, $t0, $t1  
 lw $v1, 0($t0)  
 jr $ra

**Purpose:** Retrieves an RGB color value from the color table.

**Parameters:** - $a2: Color index (0-12)

**Returns:** - $v1: 24-bit RGB color value

**Steps:** 1. Loads the address of the color table 2. Multiplies the color index by 4 (since each color is 4 bytes) 3. Adds this offset to the color table address 4. Loads the RGB value from the calculated address 5. Returns this color value

### drawPixel

drawPixel:  
 bltz $a0, drawPixelexit  
 bltz $a1, drawPixelexit  
 bge $a0, displayWidth, drawPixelexit  
 bge $a1, displayHeight, drawPixelexit  
 addi $sp, $sp, -4  
 sw $ra, 0($sp)  
 jal calculateAddress  
 jal getColor  
 sw $v1, 0($v0)  
 lw $ra, 0($sp)  
 addi $sp, $sp, 4  
 jr $ra  
drawPixelexit:  
 jr $ra

**Purpose:** Draws a single pixel at the specified display coordinates with the specified color.

**Parameters:** - $a0: Display x-coordinate - $a1: Display y-coordinate - $a2: Color index

**Steps:** 1. Validates that the coordinates are within display bounds 2. Calculates the memory address for the pixel 3. Gets the RGB color value for the specified color index 4. Stores the color value at the calculated address 5. Returns to caller

## Integration with A\* Algorithm

The bitmap helper module integrates with the A\* pathfinding algorithm to visualize:

1. **The Grid**: Shows the layout of passable areas and obstacles
2. **Current Node**: Highlights the node currently being explored (typically in yellow)
3. **Open Set**: Shows nodes in the frontier (typically in gray)
4. **Closed Set**: Shows already explored nodes
5. **Final Path**: Once found, highlights the path from start to goal (typically in green)

The visualization is enhanced with strategic delays to make the algorithm’s progress visible and comprehensible to human observers.

## Color Code Usage

The implementation uses various colors to represent different elements: - **Color 0** (White): Background/free space - **Color 1** (Black): Walls/obstacles - **Color 2** (Green): Goal node and final path - **Color 5** (Yellow): Current node being explored - **Color 8** (Cyan): Start node - **Color 9** (Gray): Nodes in the open set/frontier

This color coding makes it easy to distinguish between different elements of the visualization and understand the algorithm’s progress.

## Performance Considerations

1. **Display Buffer**: Direct memory access to the bitmap display provides efficient rendering
2. **Strategic Delays**: Small delays are added to make the visualization comprehensible
3. **Pixel-perfect Rendering**: Each grid cell consists of multiple pixels for better visibility
4. **Boundary Checking**: The drawPixel function includes bounds checking to prevent memory access violations

## Conclusion

The bitmap helper module provides a robust foundation for visualizing the A\* pathfinding algorithm. It handles all aspects of rendering from individual pixels to complete grid cells, with careful consideration for memory management, color coding, and visual clarity. This module is essential for transforming the abstract A\* algorithm into a comprehensible visual demonstration that helps users understand how the algorithm explores the grid and finds the optimal path.