Final Report

Griffin Symans

This is a declaration that the work I'm submitting is my own original work except where explicitly marked otherwise and cited.

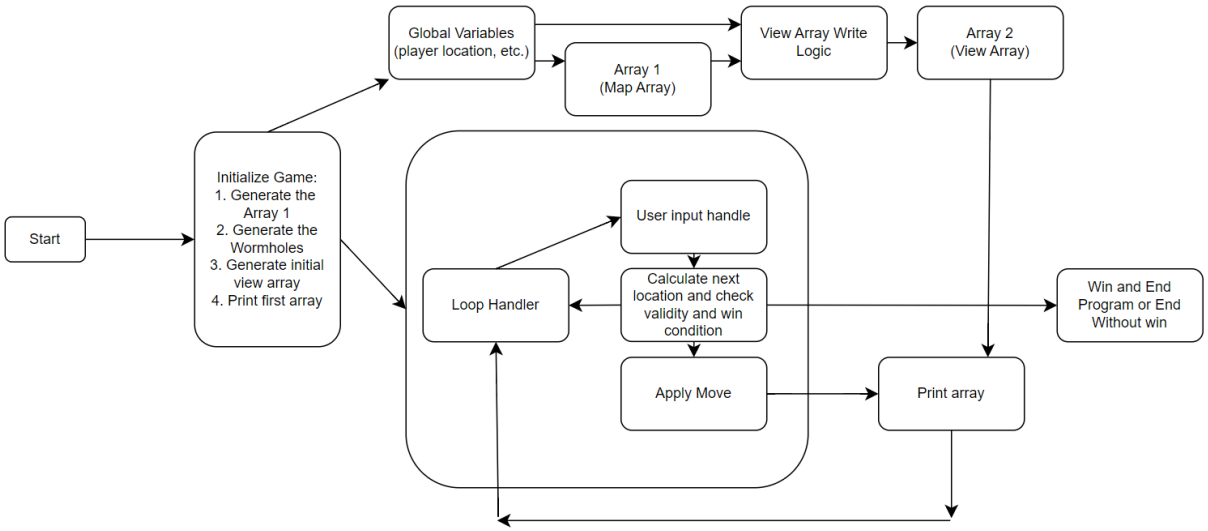
High Level Description

The game that I have created is a blind maze that shows the player what is around them up to one block away. It is then their goal to reach the end of the level using the WASD keys.

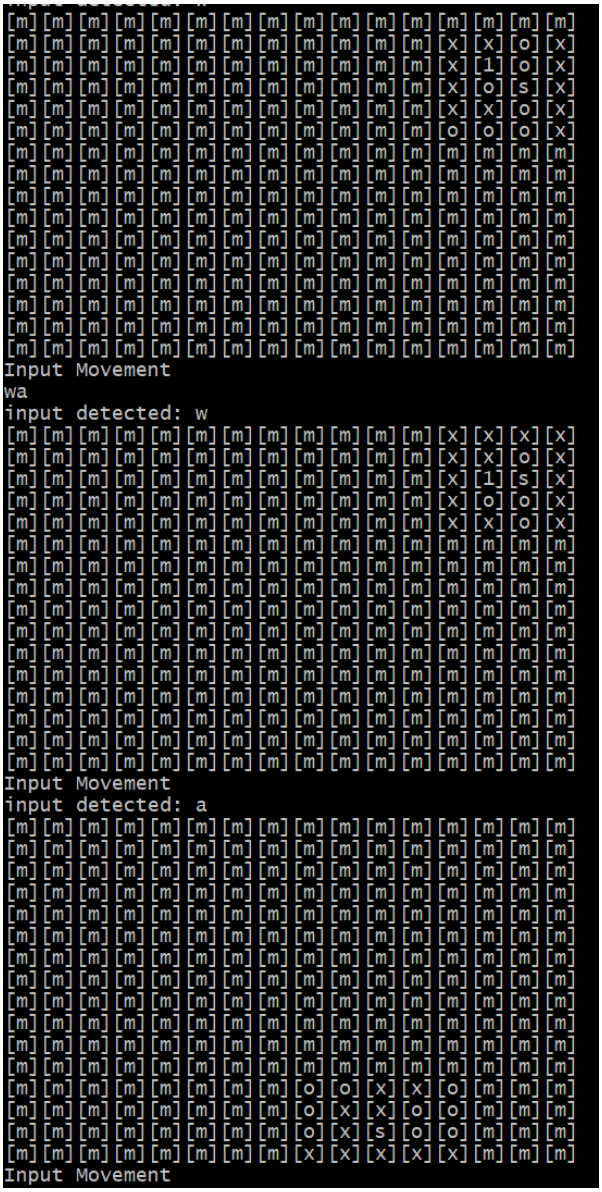
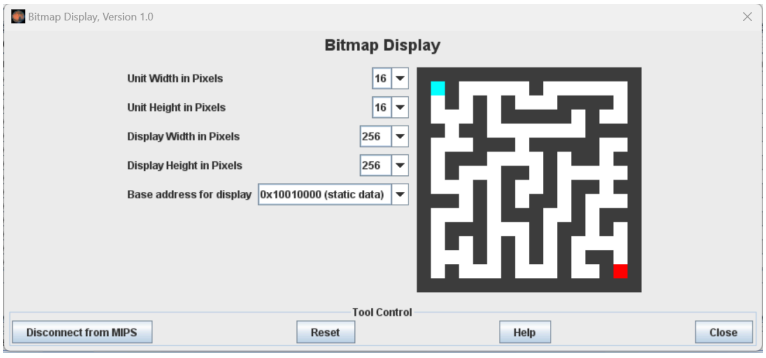
Core Functions and Operations

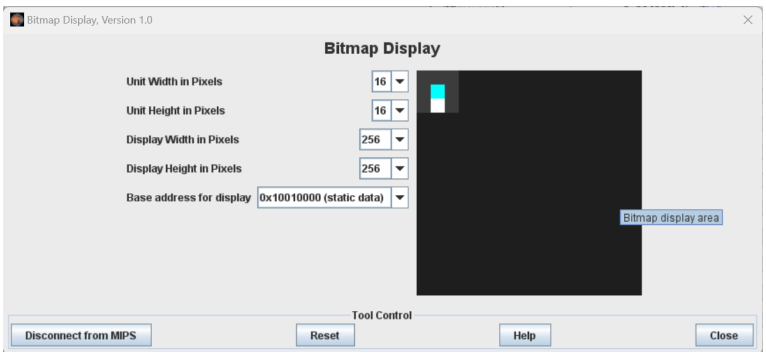
This game holds 2 arrays in memory. One is the game map that has a full view of the map with no "fog" obscuring the vision. This map is used to write to the second map, or the view map. The view map is what the player sees and each time the player moves the view map is updated such that the visible area is exactly 1 block away from the player. This game is most simply implemented using a turn based system where the game simply waits for the player input then runs the basic game loop to calculate the players next location and check any conditions. The game follows the basic format visually described in the next section.

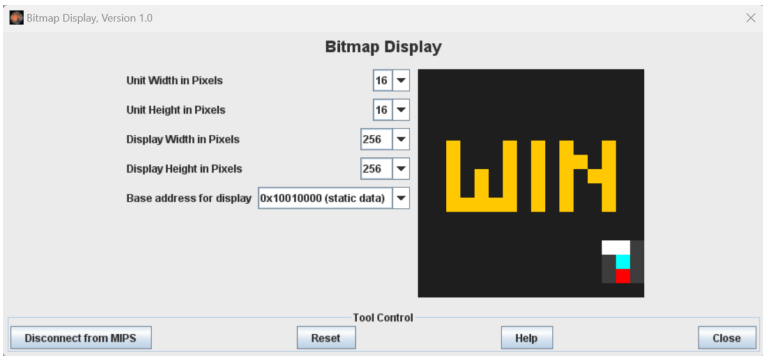
High Level Code Description



Summary of Tests

C tests: While many interactions lead up to this, the final output looked pretty basic with a simple array of chars that represented the different elements of the maze. Functionally, the entire game worked and all logic that was planned worked in the end.	
MIPS tests: While many of the program functionalities are more difficult to implement in MIPS the majority has been successfully implemented with all major functionality working. The pictures on the right show 2 parts of development. The above is the maze before the view array was implemented. The middle is the end product with the player view turned on. The final picture is the winning screen.	





[Video of Game Functioning]

Summary of Incompete Tasks

I wanted to implement various power ups but ran out of time. The two proposed power ups were

- A wormhole that teleports you from one side of the map to the other
- A torch upgrade that increases the player view size to be 2 blocks in every direction

Summary of Complete Tasks

- Completed the implementation of a win screen
- Applied the win screen to the bitmap
- Created a simple timer implementation so the player can see how long it took them
- Bitmap display outputs the initial map
- 16 x 16 map is held in storage and updated whenever necessary
- The player location is held in storage and updated with each move
- The gameloop function is implemented and handles each turn of the game
- The player movement has been implemented including checks for if a move is viable or the win condition
- Initializes a basic array of ints to serve as a map
- Created a basic test fixture to print the base address
- Set the player initial x and y

- Create a function to print the instructions to the console
- Implemeted the basic structure and prototypes of the c version
- The memory is properly allocated now and allows for indexing and making changes to the array
- The player location is now properly stored in memory and updates the array when a valid move is made
- The generate view array method was created and the general logic created, however the function still needs to be implemented
- The map now updates each time the player moves
- First use of the stack to allow me to access a grandchild function
- Implented view array
    - Implemented initially using a check when building the array
    - A more efficient way was obtained by hard coding values to set as the offset to create the view array around the player

Conclusion

In conclusion this project gave me a good insight into the world of assembly level programming. I think if I were to do this project again I would do the following.

- Reduce the use of the ‘a’ registers so that this code is more applicable to a wider range of system architectures (risc, etc…)
- Have a set function to find the offset of a certain point in the array, similar to the first homework assignment
- Utilize the stack more (it's very useful in a few places that when initially coded I didn’t know about the stack)

Estimated Time

Assuming that I worked about 2 hours per week I would estimate my time spent on this project to be about 20-30 hours

Code

C code:

```
/*
============================================================================
Name        : project.c
Author      : Griffin Symans
RedID        : 130074994
Version     : 1.0
Description : Blind maze game prototype created in C
============================================================================
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define BOUND 16
#define STARTX 1
#define STARTY 1
int py;
int px;
int vd;
int wx1;
int wy1;
int wx2;
int wy2;
int winner;
void generateArray(char arr[][BOUND]);
void generatePredeterminedArray(char arr[][BOUND]);
void generateViewArray(char arr[][BOUND], char arr2[][BOUND]);
void generateWormhole(char arr[][BOUND]);
void moveWormhole(char arr[][BOUND]);
void printError(char error[]);
void move(char arr[][BOUND], char direction);
void printArray(char arr[][BOUND]);
void win();
/*
//disabled code (replaced with generatePredeterminedArray)
void generateArray(char arr[][BOUND]) {
	//generates the map
	for (int i = 0; i < BOUND; i++) {
		for (int j = 0; j < BOUND; j++) {
			//border generator
			if (i == 0 || i == BOUND - 1 || j == 0 || j == BOUND -1) {
				arr[i][j] = 'x';
			}
			//generate starting pos
			else if (i == STARTX && j == STARTY) {
				arr[i][j] = 's';
			}
			//generate ending pos
			else if (i == BOUND - 2 && j == BOUND - 2) {
				arr[i][j] = 'e';
			}
			//interior generator (random system)
```

```
                  else {
                        if (rand() % 6 == 0) {
                              arr[i][j] = 'x';
                        }
                        else {
                              arr[i][j] = 'o';
                        }
                  }
            }
      }
      py = STARTX;
      px = STARTY;
      winner = 0;
}
*/
void generatePredeterminedArray(char arr[][BOUND]) {
      char temp[BOUND][BOUND] =
{{'x','x','x','x','x','x','x','x','x','x','x','x','x','x','x','x'},

{'x','s','x','o','o','o','o','o','o','o','o','o','o','x','x','o','x'},

{'x','o','x','o','x','o','x','x','o','x','x','o','x','o','o','x'},

{'x','o','o','o','x','o','x','o','t','x','o','o','x','o','o','x'},

{'x','x','x','o','x','o','x','o','x','x','o','x','x','x','o','x'},

{'x','o','o','o','o','o','x','o','x','x','o','o','o','o','o','x'},

{'x','o','x','x','x','x','o','o','x','x','x','x','x','x','o','x'},

{'x','o','x','o','x','o','o','x','x','o','o','o','o','o','o','x'},

{'x','o','o','o','o','o','x','x','o','o','x','x','x','x','x','x'},

{'x','x','o','x','x','x','x','o','o','x','o','o','o','o','o','x'},

{'x','x','o','o','o','o','o','o','x','x','o','o','x','x','o','x'},

{'x','o','o','x','x','x','x','x','x','o','o','x','o','o','o','x'},

{'x','o','x','x','x','x','o','o','o','o','x','x','o','x','x','x'},

{'x','o','x','o','o','o','x','x','o','x','x','o','o','x','x','x'},

{'x','o','o','o','x','o','o','o','o','x','o','o','o','o','e','x'},

{'x','x','x','x','x','x','x','x','x','x','x','x','x','x','x','x'}};
      for (int i = 0; i < BOUND; i++) {
            for (int j = 0; j < BOUND; j++) {
                  arr[i][j] = temp[i][j];
            }
      }
      py = STARTY;
      px = STARTX;
      winner = 0;
```

```
}
void generateWormhole(char arr[][BOUND]) {
	arr[wy1][wx1] = '1';
	arr[wy2][wx2] = '2';
	printf("Generated\n");
}
void generateViewArray(char arr[][BOUND], char arr2[][BOUND]) {
	//generates the players view
	for (int i = 0; i < BOUND; i++) {
		for (int j = 0; j < BOUND; j++) {
			int xdiff = abs(j - px);
			int ydiff = abs(i - py);
			if(xdiff <= vd && ydiff <= vd) {
				arr2[i][j] = arr[i][j];
			}
			else {
				arr2[i][j] = 'm';
			}
		}
	}
}
void printError(char error[]) {
	printf("Error type - %s\n", error);
}
void move(char arr[][BOUND], char direction) {
	switch (direction) {
	//handle left
	case 'a':
		if (arr[py][px-1] == 'o') {
			arr[py][px-1] = 's';
			arr[py][px] = 'o';
			px--;
		}
		else if (arr[py][px-1] == 'e') {
			win();
			break;
		}
		else if (arr[py][px-1] == 't') {
			vd++;
			arr[py][px-1] = 's';
			arr[py][px] = 'o';
			px--;
		}
		else if (arr[py][px-1] == '1') {
			moveWormhole(arr);
		}
		else {
			printError("movement-input-failure-a");
		}
		break;
	//handle up
	case 'w':
		if (arr[py - 1][px] == 'o') {
			arr[py - 1][px] = 's';
			arr[py][px] = 'o';
			py--;
```

```
            }
            else if (arr[py - 1][px] == 'e') {
                  win();
                  break;
            }
            else if (arr[py - 1][px] == 't') {
                  vd++;
                  arr[py - 1][px] = 's';
                  arr[py][px] = 'o';
                  py--;
            }
            else if (arr[py - 1][px] == '1') {
                  moveWormhole(arr);
            }
            else {
                  printError("movement-input-failure-w");
            }
            break;
      //handle right
      case 'd':
            if (arr[py][px + 1] == 'o') {
                  arr[py][px + 1] = 's';
                  arr[py][px] = 'o';
                  px++;
            }
            else if (arr[py][px + 1] == 'e') {
                  win();
                  break;
            }
            else if (arr[py][px + 1] == 't') {
                  vd++;
                  arr[py][px + 1] = 's';
                  arr[py][px] = 'o';
                  px++;
            }
            else if (arr[py][px + 1] == '1') {
                  moveWormhole(arr);
            }
            else {
                  printError("movement-input-failure-d");
            }
            break;
      //handle down
      case 's':
            if (arr[py + 1][px] == 'o') {
                  arr[py + 1][px] = 's';
                  arr[py][px] = 'o';
                  py++;
            }
            else if (arr[py + 1][px] == 'e') {
                  win();
                  break;
            }
            else if (arr[py + 1][px] == 't') {
                  arr[py + 1][px] = 's';
                  arr[py][px] = 'o';
```

```
                        py++;
                        vd++;
                }
                else if (arr[py + 1][px] == '1') {
                        moveWormhole(arr);
                }
                else {
                        printError("movement-input-failure-s");
                }
                break;
        //handle reset
        case 'r':
                generatePredeterminedArray(arr);
        //handle input errors
        default:
                printError("movement-input-failure-default");
                break;
        }
}
void printArray(char arr[][BOUND]) {
        for (int i = 0; i < BOUND; i++) {
                for (int j = 0; j < BOUND; j++) {
                        printf("[%c]", arr[i][j]);
                }
                printf("\n");
        }
}
void moveWormhole(char arr[][BOUND]) {
        //set 1 to o and 2 to s
        arr[wy1][wx1] = 'o';
        arr[wy2][wx2] = 's';
        //update player location
        px = wx2;
        py = wy2;
}
void win() {
        printf("=-=-=-=-=-=-=\n  YOU WIN\n=-=-=-=-=-=-=\n");
        winner = 1;
}
int main(void) {
        //initialize game
        char arr[BOUND][BOUND];
        char arr2[BOUND][BOUND];
        char userInput = 'q';
        vd = 1;
        wx1 = 13;
        wy1 = 2;
        wx2 = 10;
        wy2 = 14;
        //generateArray put on pause
        //generateArray(arr);
        generatePredeterminedArray(arr);
        generateWormhole(arr);
        generateViewArray(arr, arr2);
        printArray(arr);
        //default loop
```

```
	while (1) {
		//input detection
		printf("Input Movement\n");
		scanf(" %c", &userInput);
		printf("input detected: %c\n", userInput);
		//check for exit
		if (userInput == 'x') {
			break;
		}
		//process movement
		move(arr, userInput);
		//check win exit condition
		if (winner == 1) {
			break;
		}
		//print the updated array
		generateViewArray(arr, arr2);
		printArray(arr2);
	}
	return 0;
}
```

MIPS code:

```
# Author:       Griffin Symans
# Date:         10/02/2024
# Description:  Project Version 1.0
# Bitmap info: 16x16 256x256 BA=0x10010000

.data
display: .space 1024  # bitmap location definition

array:                # game array location defintion
        .word  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word  0, 2, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0
        .word  0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0
        .word  0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0
        .word  0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0
        .word  0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0
        .word  0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0
        .word  0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0
        .word  0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0
        .word  0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0
        .word  0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0
        .word  0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0
        .word  0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0
        .word  0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0
        .word  0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 3, 0
        .word  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

viewarray: .space 1024        # view array intial

playerpos:
```

```
        .word   1, 1, 0, 0
        # player pos hold more data
winarray:       #this array holds hardcoded values for the win screen letter locations
        .word   264, 280, 288, 296, 308, 328, 344, 352, 360, 364, 372,
        .word   392, 400, 408, 416, 424, 432, 436, 456, 464, 472, 480,
        .word   488, 500, 520, 524, 528, 532, 536, 544, 552, 564, -1

movexp:         .asciiz  "Use WASD to move\n"
time:    .asciiz "\nTime (ms):"
newline: .asciiz "\n"

.text


main:
        # store timer in stack for win conditon
        li      $v0, 30
        syscall
        addi    $sp, $sp, -4
        sw      $a0, 0($sp)

        li      $v0, 1
        la      $a1, array      # load array base address to $a1
        la      $a2, viewarray# load view array base address to $a2
        la      $a3, playerpos          # load misc data base address to $a3

        li      $v0, 's'
        jal     gameloop
        j       done            # end program if ths point is somehow reached because a fatal
error has occured

instructions:                   #print the instructions
        li      $v0, 4
        la      $a0, movexp
        syscall
        j       end     #return to main

gameloop:
        jal     generateviewarray
        jal     printarray      # print array
        jal     testfixture
        li      $v0, 12                 # accept user input
        syscall
        jal     handlemove      # move character
        j       gameloop

handlemove:                     # moves the character and updates the player x and y
        lw      $t1, 0($a3)             # store x at t1
        lw      $t2, 4($a3)             # store y at t2
        li      $t0, 'w'        # store val for comparison
```

```
        beq     $t0, $v0, moveup        # check comparison
        li      $t0, 's'
        beq     $t0, $v0, movedown
        li      $t0, 'a'
        beq     $t0, $v0, moveleft
        li      $t0, 'd'
        beq     $t0, $v0, moveright
        li      $t0, 'x'            # exit case 'x'
        beq     $t0, $v0, done          # jumps to end
        jr      $ra

moveup:                 # decriment py
        addi    $t2, $t2, -1
        j       checkmove

movedown:   # increment py
        addi    $t2, $t2, 1
        j       checkmove

moveleft:       # decriment px
        addi    $t1, $t1, -1
        j       checkmove

moveright:      # increment py
        addi    $t1, $t1, 1
        j       checkmove

checkmove:
        #calculate new address
        mul     $t3, $t2, 64
        mul     $t4, $t1, 4
        add     $t3, $t3, $t4
        add     $t3, $t3, $a1
        lw      $t4, 0($t3)
        beq     $t4, 3, win             # if next address is win slot jump to win
        beq     $t4, 1, applymove       # if next spot is valid and not a win slot jump to apply
move
        j       movedone                # else just go to next round


applymove:
        # change the value at the new address to 2
        li      $t5, 2
        sw      $t5, 0($t3)

        # update the old location
        lw      $t3, 0($a3)
        lw      $t4, 4($a3)
        mul     $t5, $t4, 64
        mul     $t6, $t3, 4
```

```
        add     $t5, $t5, $t6
        add     $t5, $t5, $a1
        li      $t6, 1
        sw      $t6, 0($t5)

        # update player pos values
        sw      $t1, 0($a3)
        sw      $t2, 4($a3)
        j       movedone

movedone:
        jr      $ra

printarray:                     # prints out the array
        li      $t0, 0          #initialize the counter to zero
        addi    $t3, $a2, 1024 #set stop condition
        j       printloop

printloop:
        add     $t1, $a2, $t0   # load array base address and offset to t1
        beq     $t1, $t3, end   # end if done
        lw      $t4, 0($t1)     # load current value stored in the point in question
        beq     $t4, 0, printgrey       #jump if the value is array is equal to 0 color
        beq     $t4, 1, printwhite
        beq     $t4, 2, printcyan
        beq     $t4, 3, printred
        beq     $t4, 4, printdarkgrey
        addi    $t0, $t0, 4     #iterate loop
        j       printloop

printgrey:
        li      $t2, 0x404040 #set color to grey
        sw      $t2, display($t0) #set color of map
        addi    $t0, $t0, 4     #iterate loop
        j       printloop

printwhite:
        li      $t2, 0xffffff   #set color to white
        sw      $t2, display($t0) #set color of map
        addi    $t0, $t0, 4     #iterate loop
        j       printloop

printcyan:
        li      $t2, 0x00ffff   #set color to white
        sw      $t2, display($t0) #set color of map
        addi    $t0, $t0, 4     #iterate loop
        j       printloop

printred:
        li      $t2, 0xff0000   #set color to white
```

```
        sw      $t2, display($t0) #set color of map
        addi    $t0, $t0, 4       #iterate loop
        j       printloop

printdarkgrey:
        li      $t2, 0x202020 #set color to grey
        sw      $t2, display($t0) #set color of map
        addi    $t0, $t0, 4       #iterate loop
        j       printloop

generateviewarray: # generates the view array
        li      $t0, 0   # set offset to 0
        li      $t2, 4   # set fog color to be added to array

        addi    $sp, $sp, -4    # stack use stuff
        sw      $ra, 0($sp)

        jal     viewloop        # jump to grandchild

        lw      $ra, 0($sp)
        addi    $sp, $sp, 4

        jr      $ra

viewloop:
        beq     $t0, 1024, addview    # exit to the printing visible locations
        add     $t1, $t0, $a2         # calculate address of viewarray
        sw      $t2, 0($t1)           # store word into viewarray
        addi    $t0, $t0, 4           # interate the loop
        j       viewloop

addview:
        li      $t3, 64
        lw      $t4, 4($a3)     # load y to $t4
        mul     $t3, $t3, $t4   # multiply y and 64
        lw      $t4, 0($a3)     # load x to $t4
        sll     $t4, $t4, 2     # multiply x by 4
        add     $t3, $t3, $t4   # final player pos address offset stored at $t3
        # for each of the following offsets load the map data from viewarray
        # -68 -64 -60 -4 0 4 60 64 68
        # this method is awful but this is techincally the most efficient way I can think of without
doing something similar to the win array method
        add     $t5, $t3, $a1
        add     $t6, $t3, $a2
        lw      $t4, -68($t5)
        sw      $t4, -68($t6)
        lw      $t4, -64($t5)
        sw      $t4, -64($t6)
        lw      $t4, -60($t5)
        sw      $t4, -60($t6)
```

```
        lw      $t4, -4($t5)
        sw      $t4, -4($t6)
        lw      $t4, 0($t5)
        sw      $t4, 0($t6)
        lw      $t4, 4($t5)
        sw      $t4, 4($t6)
        lw      $t4, 60($t5)
        sw      $t4, 60($t6)
        lw      $t4, 64($t5)
        sw      $t4, 64($t6)
        lw      $t4, 68($t5)
        sw      $t4, 68($t6)
        jr      $ra

end:
        jr      $ra

testfixture:    # you can ignore this, it was only used during the development of the
movement system
                # i keep it in case something breaks
        li      $v0, 1
        lw      $a0, 0($a3)     # print px
        syscall
        li      $v0, 4
        la      $a0, newline
        syscall
        li      $v0, 1
        lw      $a0, 4($a3)     # print py
        syscall
        li      $v0, 4
        la      $a0, newline
        syscall
        jr      $ra

win:
        #handle timer
        li      $v0, 30
        syscall #a0 is now current time
        lw      $t0, 0($sp)     #t0 is now old time
        sub     $t0, $a0, $t0
        li      $v0, 4  #print time string
        la      $a0, time
        syscall
        li      $v0, 1  #print total time
        addi    $a0, $t0, 0
        syscall

        #handle win screen
        li      $t0, 0xFFCC00           #load yellow for win text
        la      $t1, winarray
```

```
        j       winloop

        j       done

winloop:
        lw      $t2, 0($t1)     # load new address
        beq     $t2, -1, done   # end if last digit reached
        addi    $t2, $t2, 64
        sw      $t0, display($t2)       #store color to bitmap
        addi    $t1, $t1, 4     #iterate loop
        j       winloop

done:
        li      $v0, 10                 # syscall for exit program
        syscall
```