"Always code as if the (person) who ends up maintaining your code will be a violent psychopath who knows where you live." – M. Golding

This is a solo lab!

You will need to re-download QtSpimbot using the instructions from CS233 On Your Own Machine!

Learning Objectives

1. Introduction to memory-mapped I/O
2. Introduction to interrupt handlers
3. Understanding of SPIMBot simulator

Work that needs to be handed in (via github)

1. `part1.s`: Gather resources from around the map and craft an item.
   Run on EWS or locally with:
   `QtSpimbot -part1 -file part1.s`
2. `part2.s`: Pick up resources from around the map, craft an item, and use the puzzle interrupt.
   Run on EWS or locally with:
   `QtSpimbot -part2 -file part2.s`

Be sure to read the entire handout!

Guidelines

- Same procedure as previous labs on MIPS. Use any MIPS instructions or pseudo-instructions you want. We may try to break your code.
- We will post solutions for Lab 7 so that you can use the correct Dominosa implementation tailored for Lab SPIMBot.
- As always, follow all calling and register-saving conventions you've learned.
- **Remember to test your code often!**
- Use the SPIMBot documentation as a reference:
  `https://cs233.github.io/assets/pdfs/labs/spimbot_documentation.pdf`

Stranded

Everyone in CS233 and CS225 got stranded on a remote tropical island! Those pesky CS225 students are taking all the resources for themselves. At this rate, there won't be enough resources for you - yes you - to survive. In this lab we will be our handy MIPS knowledge to use in order deploy a robot to gather the resources for us while we relax on the beach.

Part 1: Learning to gather resources [40 points]

For now, we will setup a small simulation for you to get used the robot environment. The first task we will ask you to do is to gether resources around the map and craft an item with them. The map is split up into discrete tiles. The map consists of 40×40 tiles, with each tile being 8×8 pixels. There are many types of tiles in the map, some of these will be floors, some walls, and others may be resources. Scattered across some of these tiles will resources like stone, water, and wool. You will have to collect one of each resource for this part of the lab to get all the points. It is important to note that the map for this lab is fixed! So, you can **hard code to every point you want to go to and all actions you want to do**.

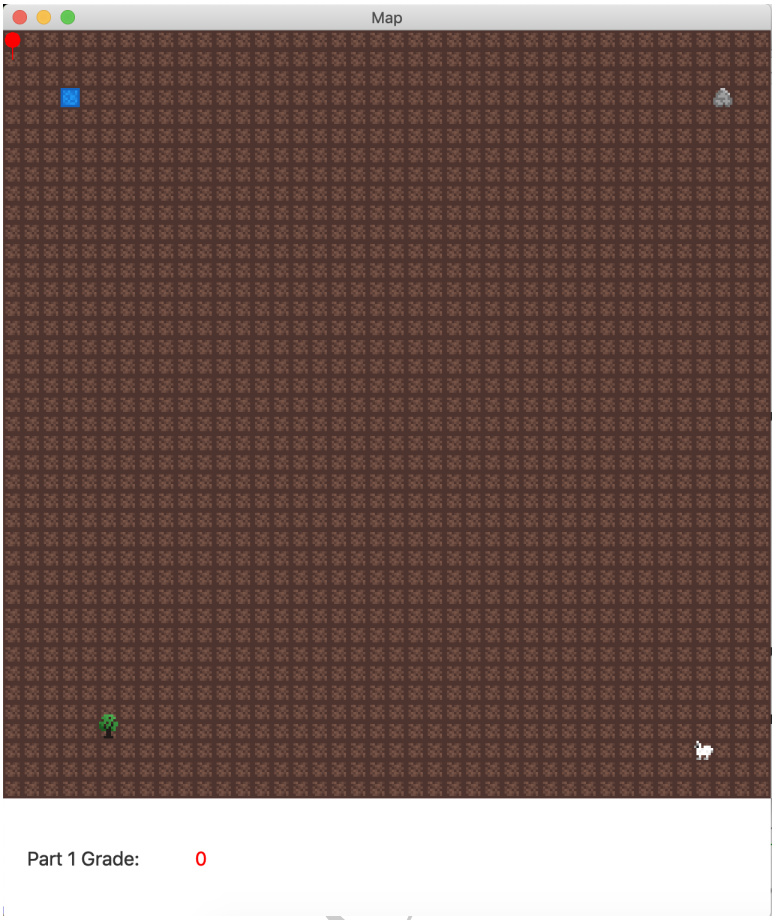
As a brief introduction, there are two main ways of accomplishing this: you can either plan a path to the resource, set your velocity towards your destination, and loop until you reach your destination. So, this method will use the following Memory Mapped I/O: `ANGLE`, `VELOCITY`, `BOT_X`, and `BOT_Y`. Alternatively, you can use calculate the time it would take to get to a points and use the timer interrupt to alert you that you reached the point, so you would likely need: `ANGLE`, `VELOCITY`, and `TIMER`. Some common catches, the SPIMBot travels continuously across the entire map without any jumps in its movement. This means after setting your velocity, your bot will continue to move in that direction until you bonk into a wall. To prevent this, set your velocity to zero when you do not intend to move.

Additionally, the above strategy only takes you to the blocks you want to go to. You need to use additional MMIO in order to interact with the blocks. You use `BREAK_BLOCK` by storing a location (column idx in the bits 8-15 bits, row index in the bottom 8 bits) to the memory address `BREAK_BLOCK` (see documentation for more details). This allows you to break the block at the given location if it is breakable and if it is range of your bot. This is used to gather stone, wool, and wood.

Once the materials have been collected, you can use `CRAFT` in order to craft the recipe. **The craft will succeed if you have 1 wool, 1 stone, and 1 wood and you store the value 7 to the memory address CRAFT** (see Documentation). Once successful, you will obtain the **LEGENDARY WOOL COVERED STONE ON A STICK**.

Points are awarded as follows:

- 10 points for collecting wood
- 10 points for collecting stone
- 10 points for collecting wool
- 10 points for crafting the **LEGENDARY WOOL COVERED STONE ON A STICK**



This is what the map will look like. In the bottom-left, there is a forest where you can get wood by using `BREAK_BLOCK`. In the top right, there is a stone that can be broken to get stone (truly mind boggling) and the bottom right has a sheep that you can break for wool. In the top-left is some water; this is not needed for this part of the lab and it cannot be broken. Please note that these blocks are solid so you cannot enter them. That is why you need to write the position of the block you want to break to `BREAK_BLOCK` (see the documentation for the format). As well, you need to be in range of the block you want to break. The test for if you are in range is if the tile you are on is less than 3 tiles away from the block you want to break, measured using euclidean distance.

Some tips that may be useful for debugging:

- You might find that giving SPIMBot the `-debug` parameter will be useful for your debugging; it prints out lots of information when SPIMBot interacts with the world.
- You can use the `PRINT_INT` memory-mapped I/O address to do print-statement debugging.
- You can use the -drawcycles argument to change the speed the game runs at. This is useful to slow down the game and see what SPIMBot is doing.

Part 2: Solving Puzzles [60 points]

Now that we are done with the previous simulation, it is important to make the simulation a bit more realistic. Crafting something is a very difficult task and requires *Creativity*. So, for this part of the lab you will do everything you did in the previous part of the lab (collect resources, move around, the map) but with one caveat. In order to craft an item, you must have *Creativity*, which is obtained through solving a puzzle.

We will be using the same count disjoint regions puzzle that you solved in Lab 7 as the puzzle for this lab - full details in the spimbot documentation. We will provide MIPS code for the solver that follows the same algorithm implemented in Lab 7. **This means all you need to do is integrate the puzzle solver with the rest of your spimbot and solve a puzzle before crafting**.

By solving puzzles, your bot will get some *Creativity*, which can be used to craft items. To request a puzzle to solve, allocate a space in the `.data` segment for a `Solution` struct and a `PuzzleWrapper` struct that encapsulates (see below for definition of the structs). Then, write the address of the start of this memory to the `PUZZLE_REQUEST` memory-mapped I/O address. When a puzzle is ready, the I/O device will raise a puzzle interrupt. Then, you need to read from `REQUEST_PUZZLE_ACK` to acknowledge the puzzle. When you acknowledge it, the address you wrote to `REQUEST_PUZZLE` will be filled a valid puzzle generated by QtSpimbot. Make sure to acknowledge `REQUEST_PUZZLE_ACK` by writing to it. Check for the interrupt to acknowledge `REQUEST_PUZZLE_ACK` using the `REQUEST_PUZZLE_INT_MASK` interrupt mask. Solve this puzzle using the rules defined in Lab 7 and write back the address to the solution to the `SUBMIT_SOLUTION` address.

Below is sample C++ code to give you a concrete idea of what you need to do for this part of the lab.


```
#define MAX_LINES 12
#define MAX_DIM 12

struct PuzzleWrapper {
    // canvas
    unsigned int height;
    unsigned int width;
    unsigned char pattern;
    char** canvas;

    // lines
    unsigned int num_lines;
    unsigned int* coords[2];

    // data
    int line_data[2 * MAX_LINES];
    char* canvas_row_pointer[MAX_DIM];
    char canvas_data[MAX_DIM * (MAX_DIM + 1)];
};

struct Solution {
    unsigned int length;
    // Number of disjoint regions after drawing each line.
    int * counts;
};

const mem_addr REQUEST_PUZZLE_ACK = 0x.....; // mem_addr is just a pointer (word sized)
const mem_addr SUBMIT_SOLUTION = 0x.....; // mem_addr is just a pointer (word sized)
const mem_addr REQUEST_PUZZLE = 0x.....; // mem_addr is just a pointer (word sized)



// Stuff to put into the data segment
int puzzle_received = 0; // puzzle_received: .word 0
PuzzleWrapper puzzle; // puzzle: .space SIZE_OF_PUZZLE
Solution sol; // solution: .space SIZE_OF_PUZZLE
```

```
void interrupt_handler() {
    ...
    // Puzzle interrupt part
    *REQUEST_PUZZLE_ACK = (int)1; // ACK the interrupt
    puzzle_received = 1;
    // End puzzle part
}

/**
 * Solves two puzzles
 */
void puzzle_part() {

    for (int i = 0; i < 2; i++) {
        puzzle_received = 0;
        *REQUEST_PUZZLE = &puzzle;

        while (puzzle_received == 0) {
            // At some point in time, the we will be interrupted
            // and we will receive the puzzle, but we don't know when.
            // This will initialize the PuzzleWrapper with all the information
            // needed.
        }

        // Received puzzle!
        puzzle_solve(puzzle); // This will store the solution into sol
        *SUBMIT_SOLUTION = &sol; // Send the solution location
    }

}

void part1_code() {
    // Code from part 1
}

int main() {
    puzzle_part();
    part1_code();
}
```


Points are awarded as follows:

- 10 points for collecting wood
- 10 points for collecting stone
- 10 points for collecting wool
- 10 points for crafting the **LEGENDARY WOOL COVERED STONE ON A STICK**
- 10 points for acknowledging the puzzle correctly
- 10 points for a correct solution to the puzzle

Hints

- Keep in mind that the map layout is fixed, so you can hardcode the path that SPIMBot will take.
- You will probably find it very helpful to write functions that move SPIMBot one cell at a time. For example, you could define four functions `move_east`, `move_west`, `move_north`, `move_south`. Then, you could easily express your path as a sequence of moves between cells and not have to worry

about any tiny details. To implement these functions, you can poll the SPIMBot's position. The SPIMBot moves 1 pixel every 1,000 cycles at top speed.

- You will have to switch between moving and solving puzzles. You can choose to either solve all puzzles at the beginning before moving, or use timer interrupts in order to solve puzzles while moving.
- LabSpimbot isn't out yet and we will likely make substantial changes to the game before releasing it.