*"More than the act of testing, the act of designing tests is one of the best bug preventers known. The thinking that must be done to create a useful test can discover and eliminate bugs before they are coded – indeed, test-design thinking can discover and eliminate bugs at every stage in the creation of software, from conception to specification, to design, coding and the rest."* – B. Bezier

*"The speed of a non-working program is irrelevant."* – S. Heller (in "Efficient C/C++ Programming")

- **There is a new version of QtSpimbot in the `_shared` repository. If you are on your own machine, you must do another `git pull` to update the binary.**
- **We cannot update this handout with clarifications or new information. You should consider the online documentation page as the final source of information on SPIMBot. We will announce any game-breaking bugs and patches on Campuswire.**

## Learning Objectives

1. Assembling larger programs from components
2. Creative problem solving

- Part 1 Command: `QtSpimbot -file spimbot.s -mapseed 233`
- Part 2 Command: `QtSpimbot -file spimbot.s -file2 spimbot.s`

## Work that needs to be handed in (via github)

**Only ONE team member should submit. This lab also can not be handed in late!**

1. `spimbot.s`, your SPIMBot tournament entry. **Note that you should only have one SPIMBot file.**
2. `partners.txt`, a list of your and your 1 or 2 contributor's NetIDs with each NetID on it's own line,
3. `writeup.txt`, a few paragraphs (in ASCII) that describe your strategy and any interesting optimizations that you implemented. Explain what your team focused on, why you focused on these goals, and at what points these goals shifted.
4. `teamname.txt`, a name under which your SPIMBot will compete. Team names must be 40 characters or less and should be able to be easily pronounced. Inappropriate names are subject to sanitization. Do not use your real names or netid.

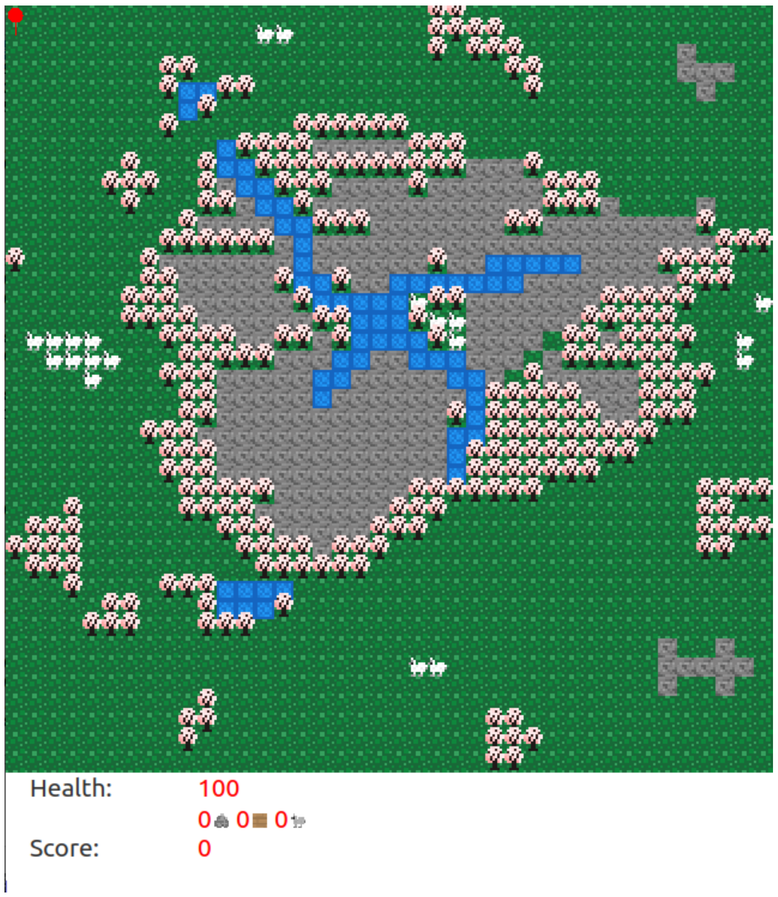
## Guidelines

- **You must do this assignment in groups of 2 or 3 people.** Otherwise, we'll have too many programs for the competition.
- You'll want to use `QtSpimbot`. See Lab 7/9 for directions on running QtSpimbot.
- Use any MIPS instructions or pseudo-instructions you want. In fact, anything that runs is fair game (i.e., you are not required to observe calling conventions, but remember the calling conventions are there to help you avoid bugs). Furthermore, you are welcome to exploit any bugs in SPIMBot or knowledge of its algorithms, as long as you let us know after the contest what you did.
- You may only submit one mips file. All your code must go in that file.
- We will not try to break your code; we will compete it against the other students who might.
- We've provided solution code for Lab 7 in `puzzle.s`. You are free to use any of this code in your SPIMBot contest implementation.

- The contest will be run on the EWS Linux machines, so those machines should be considered to be the final word on correctness. Be sure to test your code on those machines.
- Make sure your code is well documented with comments explaining the logic and register usage if you expect course staff to help you. **Course staff can also ask to look at your psuedocode and your debugging attempts so far before deciding to help you.**

## Survival Island

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 deploy a robot to gather the resources for us while we relax on the beach.



Unlike in Lab 9, we are no longer in a simulation. It's spring time, cherry blossoms are blooming and you will have to deal with the harsh realities of the world. In SPIMLand, the world operates on a day/night cycle where a full day-night cycle lasts 2,000,000 cycles. And at night time, strange and rabid creatures come out. Rabid squirrels who have been eyeing your resources during the day will come out and try and attack your spimbot. It is your spimbot's job to gather resources and fortify yourself during the day, so that it can survive the night and we can relax.

We will be competing your bots against each other to find the best bot to help us survive on the island.

## Understanding The World

Our survival island consists of a few different item types naturally. The naturally occurring blocks include water, stone, wool, water, trees, and ground. The exact properties of these can be found in the documentation, but keep in mind that these resources will be integral to survival. Trees, stone, and wool can be used to craft other items for survival and other blocks like water will have other purposes like satiating thirst.

## Living A Sedentary Life

In order to survive in such a harsh environment, the first step is to see what we can make with the resources around us. If we look at the documentation, we can see that breaking blocks and storing them into our inventory enables us to create items like walls, beds, doors, and chests. These items will allow us to create forts in order to survive the night.

The ability to build large forts is an excellent sign that your bot is suited to surviving on the island and it will be awarded a lot of points if it can survive the night. A fort is defined as a region enclosed by walls and doors. There are several considerations for building a base. For one, bases with more valuable items inside them are worth more points. As well, it may be worth it to invest in adding thicker walls around your base in order to protect it from rabid squirrels. At night rabid squirrels will appear and will target your SPIMbot to steal its resources. If there is a solid block or door blocking their path to the spimbot, they will break it at a slow rate. If you die without having registered bed, then you lose a significant number of points. So, it is a good idea to either fortify your base and place your bed inside or lure the squirrels away from it.

## Those Darn Squirrels

Every night squirrels, sent by CS225 students, will appear at the edge of the map. Due to their rabidity, squirrels will hunt you down if you are within their seek radius. Fortunately, this also leaves the squirrels with a crucial weakness, they flee at daytime. You have several approaches to dealing with squirrels like attacking them, fortifying yourself with walls, and letting them chase you until daytime.

## Summary

In order to survive, you will build upon your lab 9 code. To begin, your bot needs to solve puzzles to gain creativity. This creativity is used to craft items (new recipes have been added so check out the SPIMbot Documentation). These items can be used to build your fort and help you survive the night in safety. Additionally, you should make a bed so that you lose less points if your bot dies.

Unlike Lab 9, we are no longer in a simulation, so you have to keep track of more things in your code to ensure your spimbot's survival. For one, if you do not drink water frequently enough you will die of thirst. The other key stat to keep track of is you health, which kills your bot if it reaches 0 (it'll respawn don't worry). There is a day/night cycle and squirrels spawn at night.

The rules are as follows:

1. In each round of the tournament, two bots will compete to see which one can collect more points. The final score of each bot will be the sum of points accumulated through crafting items and submitting your fort. You can lose when you bot dies and you will lose additional points if your bot dies without a bed. Your bot will respawn upon death either at its bed or the deafult spawn positions. See the SPIMbot Documentation for details.

2. The map is fixed but refreshes resources on the night to day transition. The map layout is hardcoded and is invariant to the `mapseed`. Mapseed is still relevant as it may be used for other parts of the game like squirrel spawning. This means that you should feel free to hard code a routine that works for your bot. As well, the map resets all blocks that a user is not on and are not part of a fort when it turns to day. This means that you can reuse locations to gather resources.

3. Creativity is obtained by requesting and solving the FillFill puzzle from Lab 8. Solving a puzzle yields 1 puzzle coin and keep in mind that puzzles have an arbitrary delay before reaching your bot, so it is best to use the puzzle interrupt to deal with the puzzles.

4. There are some more fine-grain information like health that we provide through MMIOs. Information about these systems can be found on the SPIMBot documentation page.

## Puzzle Hints

Many students have trouble getting a grasp for puzzle solving, so here is the general puzzle-solving pipeline:

1. Allocate a space in the .data segment capable of holding an entire puzzle struct.
2. Enable puzzle interrupts.
3. Write a pointer to this space into `REQUEST_PUZZLE`.
4. Your code can continue as normal; the puzzle is generated in the background.
5. A puzzle interrupt will trigger.
6. Acknowledging the puzzle interrupt will fill this space with the puzzle struct.
7. Solve the puzzle by any method.
8. Write the pointer to the space to `SUBMIT_SOLUTION`. The I/O device will verify your solution and reward you with creativity if it is correct.

If you suspect you have an issue with setting up your puzzles, make sure that your code properly follows the pipeline above.

You will find that running SPIMBot with the `-debug` flag will be useful for your debugging; it prints out useful interactions that SPIMBot makes with the world. Note: you can use the `PRINT_INT` memory-mapped I/O address to print out relevant values for debugging purposes. **We have noticed that some bots do not perform correctly without the debug flag enabled. It can change how puzzle timing works.** Your SPIMBot will be run without the `-debug` flag during the tournament, so be sure that your code works without it.

For more information, read the documentation at `https://cs233.github.io/assets/pdfs/labs/spimbot_documentation.pdf`.

## Scoring

Your SPIMBot will be scored out of 100 points.

### Qualifying [70 points]

The first 70 points will be awarded when you qualify for the tournament. Qualifying matches will be done by running your bot over some fixed map seed; your score will be the average over 4 runs.
The command to run a qualifying match is:

```
QtSpimbot -file spimbot.s -mapseed 233
```

To qualify, your bot need to score at least 70 points (visually confirmed through the GUI or from the printout when you close the program) at least 3 times out of 4 runs.

For example, if you score 0 points on the first run, but 70 points on the rest, you will qualify. However, if you score 60 points on every time then you will not qualify. The is a lenient qualification and you can expect to earn over 200 points on every map.

### Competing [30 points]

The command to run a qualifying match is:

```
QtSpimbot -file spimbot.s -file2 spimbot.s
```

The remaining 30 points will be earned during the tournament and will be based on your placement. Note that **bots that do not successfully complete the above section will not qualify for the tournament**. You cannot earn these points unless you meet the above baseline.

The tournament will be held in a standard double elimination tournament format that we will run multiple times. Your highest placement will determine how much of the remaining 30 points you receive. The exact scoring here is subject to change depending on the distribution of results.

## Helpful SPIMBot and MIPS Tips

- When naming functions and memory addresses, use long and meaningful names. This prevents you from accidentally reusing a name and also makes debugging easier.
- Minimize the time spent in interrupt handlers. While you are executing your interrupt handler, you can't respond to other interrupts like bonks and timers.
- Refer to the SPIMBot documentation frequently. If there's a clarification we make or a bug that we patch, we can't update this handout, but we will update Campuswire and the documentation page.
- While you don't need to exactly follow the calling conventions we've used before, there's a reason why almost every language works in terms of functions and procedures. Organizing your code according to calling conventions makes it far easier to debug and extend.
- Consider writing your program in C/C++, and then hand-compiling it to MIPS. C is infinitely easier to comprehend than assembly, and it's much quicker to debug and error-check C code than MIPS.
- Although the FillFill algorithm we provide you can solve the puzzle, you can optimize this so that you can greatly speed up crafting.
- 80% of your code will consume 20% of your time. 20% of your code will take up 80% of your time. Learn to recognize when your code is wasting cycles doing nothing and when optimizing a section code is a waste of your time.
- Donald Knuth said "Premature optimization is the root of all evil." Don't blindly optimize! Other than a few obvious tasks (pathfinding, puzzle solving), it's hard to figure out where your code will spend its time. QtSpim has a profiler built in; use it to see where you code spends its time before trying to optimize it!
- Shaving off a few instructions here and there is unlikely to result in speedups most of the time (unless it's in a frequently-run loop). Don't try to be clever with saving cycles - it makes your code harder to read.
- Learn how to use git branching. Knowing that you have a version of your code that you are happy with leaves you to freely experiment with new additions. Gitkraken (free for students) may be helpful with this.
- Consider using polling loops instead of interrupts. For movement, it is much easier to use `BOT_X` and `BOT_Y` instead of the timer interrupt, and it allows you to write more modular code. However, this is a tradeoff because constantly checking `BOT_X` and `BOT_Y` is more expensive.