"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")

Learning Objectives

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

Work that needs to be handed in (via SVN)

This lab is due May 3rd at 8 PM, and only ONE team member should submit.

1. `spimbot.s`, your SPIMbot tournament entry,
2. `partners.txt`, a list of you and your 1 or 2 partners' NetIDs,
3. `writeup.txt`, a few paragraphs (in ASCII) that describe your strategy and any interesting optimizations that you implemented, and
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. Any team names deemed inappropriate are subject to sanitization.

Guidelines

- **You must do this assignment in groups of 2 or 3 people.** Teamwork is an essential skill for future courses and in the professional world, so it's good to get some practice. If you do the assignment individually, you won't be entered in the tournament, so you can earn at most 60% of the points for this lab.
- Use any MIPS instructions or pseudo-instructions. In fact, anything that runs is fair game (*i.e.*, you are not required to observe calling conventions, but remember calling conventions will aid debugging). Furthermore, you are welcome to exploit any bugs in SPIMbot or knowledge of its algorithms (the full source is provided in the `_shared/LabSpimbot` directory in SVN), as long as you let us know in your `writeup.txt` what you did.
- All your code must go in `spimbot.s`.
- We will not try to break your code; we will compete it against the other students.
- Solution code for Lab 7, and Lab 8 will be provided. You are free to use it in your contest implementation. We will also provide some useful trig functions in the taylor.s file.
- syscalls will be disabled for this lab
- 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.
- Refer to the SPIMbot documentation for details on its interfaces: `https://wiki.cites.illinois.edu/wiki/display/cs233sp16/SPIMbot+documentation`

Problem Statement

In this assignment, you are to design a SPIMbot that will compete with other SPIMbots in the strange land introduced in Lab10. After completing Lab10, SPIMbot decided to settle in this land, believing it could survive with its newfound agricultural skills. However, as soon as SPIMbot settled in, the seemingly benevolent alien presence showed its true colors. The alien presence has demanded entertainment in the form of a tournament, in which SPIMbot must compete against other *chosen*[1] SPIMbots, in a fight for survival. Your bot must grow as many plants for itself as possible, and may even sabotage the enemy bot. Furthermore, the alien presence will only grant water to the cleverest of bots and will bequeath water to a bot if the bot successfully solves a given puzzle.

On May 4th, the malevolent alien presence will decide the lone *favored* SPIMbot once and for all, in a double elimination tournament held in class. The program that performs the best will win special favor amongst the alien presence and will live a prosperous life in the strange land of plants!

This handout will provide you with details on both the Game and the Puzzle parts of the tournament. May The Force Be With You![2]

The Game

Objective

In each round of the tournament, we will compete two robots to see which can grow the most fruits. Both bots will be placed at the same random starting location, and the three clouds will be placed in the sky at three different heights but at otherwise random locations. Furthermore, 10 plants (5 for you and 5 for the enemy bot) will spawn on the ground periodically at random locations (but never too close to the edge that SPIMbot will be unable to reach for watering). You are thus required to write a SPIMbot that will participate in this tournament by growing plants. As in Lab 10, you will be using the `PLANT_SCAN` and `CLOUD_SCAN` memory mapped I/O to request the locations of the plants and clouds, respectively. Refer to the Lab 10 handout for more details.

Methods of Watering Plants

The methods of watering have expanded since Lab 10! SPIMbot can water plants with three different methods:

1. Watering directly from the bot (Similar to Lab 10.1)
2. Moving clouds directly above plants to water them (Similar to Lab 10.2)
3. Catching water

Watering directly from the bot

In Lab 10.1, the alien presence initally provided a large amount of water to SPIMbot's water tank. They will not be so generous in the competition. In order to instantly retrieve large amounts of water in your bot's tank, you will have to request water from the alien presence. Unfortunately, the alien presence will have to verify that the SPIMbot in question is clever enough to earn the valued resource of water. Therefore, the alien presence will provide a puzzle, which will be described in more detail in the Puzzle section of this handout. After completing the puzzle, SPIMbot will return the solution to the aliens, and, after quick verification of the solution, 40 drops of water will magically appear in SPIMbot's water tank.

[1] SPIMbot realizes too late that *chosen* means chosen for competition

[2] That's not how the Force works!

Moving clouds

In Lab 10.2, the alien presence provided one cloud to SPIMbot. Strangely, they have decided to be more kind, and provide three clouds to SPIMbot in the competition. Similar to Lab 10.2, SPIMbot may enter the cloud and move it horizontally to water its plants.

Catching water

SPIMbot can catch water from clouds (which may be helpful if the cloud or plant are in difficult locations) or from the enemy bot (which may be helpful for sabotage). After catching water, SPIMbot can drive to their plant and water it.

Cloud mechanics

The x,y coordinates we give are the exact middle of the cloud hitbox. The hitbox is a square with sides of length 30. If your bot is inside the hitbox of the cloud, the cloud will take on the bot's horizontal velocity. If two bots are in the same cloud, the cloud adopts the addition of both bot velocities.

Be conscious of how you are entering/exiting the cloud. If you want to enter/exit the cloud without moving it in the x direction, you'll probably have to enter/exit coming straight up/down. If you enter the cloud from the left or right, you will be unable to pull the cloud; you will have to resort to pushing.

Sabotage

SPIMbot has been tasked with growing its own plants, and SPIMbot's goal is to fully grow more plants than the enemy bot. This means that it is advantageous for SPIMbot to stop the enemy bot from successfully watering its plants.

Your enemy's bot may decide to move clouds away from your plants or may wait under your cloud, stealing water that would have nourished your plant. To sabotage other bots or to counter sabotage from other bots, you can use the CLOUD_STATUS_INFO provided in the CLOUD_CHANGE_STATUS interrupt. Don't forget that this info is laid out in a 32-bit integer, *not* a struct, so you will have to consider endianness. Refer to the SPIMbot documentation for details on extracting your desired information.

The enemy bot may also steal water by waiting under your SPIMbot. You could try to counter this by watering your plants from a low elevation. However, watering from lower heights will take much longer to fully grow a plant. You will have to decide whether watering from a higher elevation is beneficial given the risk of sabotage!

The Puzzles

Objective

The role of the puzzles in this competition is to acquire larger amounts of water from the alien presence. As discussed earlier, your bot can request the puzzle from the alien, and, upon successful completion of the puzzle, the bot will be granted with 40 drops of water.

Puzzle Description

The puzzle, as you may recall from labs 7 and 8, is a *Word Break* puzzle. You are provided with a Puzzle Dictionary of words and a Puzzle String that you have to *break* into words included in the Puzzle Dictionary. Here are some important points that you should know regarding the puzzle:

- The Puzzle Dictionary is in the form of an array of char pointers, that you should search in, to attempt to match a substring in your Puzzle String.
- The number of words in the Puzzle Dictionary could range anywhere between 160 and 196 each

(inclusive).

- The length of the Puzzle String is at most 128 characters long and the word is NULL terminated.
- When you are solving the puzzle you will need to build *solution*, an array of char pointers with the locations of each solution substring saved in *solution*[0], *solution*[1], ...
  - If you refer to the lab 8 utils.s files, you will see that the `malloc` subroutine we provided you with uses a statically allocated memory area `str_memory` to store the newly allocated strings and a memory location `new_str_address` to point to the next free memory address in the `str_memory` to allocate the new string there.
  - Make sure you understand the procedure of string memory allocation by referring to the mentioned lab, because you will need to do the same.
  - Also, since your bot will be solving multiple puzzles, you need to free out the `str_memory` after submitting a puzzle solution so that you don't run out of memory after allocating so many strings. i.e. Every time you want to solve a new puzzle, you should start out with an empty `str_memory`.
  - In order to free your `str_memory`, it suffices to have the `new_str_address` point to `str_memory`, just like it is initialized in the data section.
- Finally, you should make sure you allocate enough space for your Puzzle Dictionary, Puzzle String, and solution array in case of overflow:
  - Puzzle Dictionary: 13,528 bytes
  - Puzzle String: 129 bytes
  - Solution Array: 516 bytes

We will next discuss the process of requesting and submitting puzzles. You will have find the best way to solve the puzzles in order to acquire water quickly and beat your opponent.

Requesting and Submitting Puzzles

In order to request a puzzle, the same memory mapped I/O scheme that you used to request the plant array in Lab 10 will be used. You will need to allocate two static memory locations in your `.data` section: one for the dictionary and one for the string. In order to request a puzzle, a memory mapped address called `REQUEST_PUZZLE` is provided. As with `PLANT_SCAN`, you should store the address of the memory allocated for the puzzle into the `REQUEST_PUZZLE` memory mapped I/O address. However, as opposed to the case of requesting the plants array, your bot will not receive the puzzle instantaneously; instead, an interrupt will fire when the puzzle is ready. The *request puzzle* interrupt mask and acknowledge address are **`0x800`** and **`0xffff00d8`** respectively. You should use the knowledge you gained in this course about interrupts to handle this interrupt. When you receive the interrupt, the puzzle would have been written into the memory address you provided. The format of the written puzzle would be the following:

- 32 bit integer, *size*, representing the number of words; followed by
- 196 consecutive char pointers, with the first *size* pointers pointing to a different word in the dictionary
- 196×65 chars, for up to 196 words, each with, at most, length 65.

After receiving the interrupt, you should then request the string. In order to do that you have been provided with a memory mapped address called `REQUEST_STRING`. Just like before, you should store the memory address you allocated in your `.data` section in the memory mapped I/O in order to tell SPIMbot where you want the string stored. This time, the string will arrive instantaneously. Remember, when reading the string, that it is null terminated, i.e. the last byte will be NULL.

Finally, after you solve the puzzle and generate an array of char pointers with all the substrings for the solution to the given Word Break problem, your bot should submit the puzzle. In order to do so, a third

memory mapped address has been provided, `SUBMIT_SOLUTION`. You should therefore provide the address of your solution array by storing it's memory address in the provided memory mapped I/O.

Note: For a list of all the new, and old, memory mapped I/O addresses, refer to the SPIMbot documentation on the wiki page.

Winning

In order to win the competition you should get a higher score than your opponent. This can be achieved, not just by watering your plants, but also by making it difficult for your enemy to water their plants.

Strategy

This lab is graded in two parts, 60% for a baseline bot, and 40% based on how the bot fairs in the final tournament. A basic 60% implementation should:

- Grow 10 plants when run by itself.

However, there are many ways to optimize your bot to beat your opponent. Here are a few things you may want to consider examining and optimizing if you want to create a highly competitive bot:

- Solving many puzzles quickly, or solving them while your bot is moving to a new location, will allow you to carry more water before having to water a plant, which can save on travel time
- If you move your bot lower, you have a chance to get to the plant without enemy interference. The drawback is that watering plants at a lower height will take much more water droplets to grow the plant fully.

Good Luck!