"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 December 8th 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 Labs 7 and 8 will be provided. You are free to use it in your 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.
- Refer to the SPIMbot documentation for details on its interfaces:
  `https://wiki.cites.illinois.edu/wiki/display/cs233fa15/SPIMbot+documentation`

Problem Statement

In this assignment you are to design a SPIMbot, that will compete with other SPIMbots by trying to smash as many fruits as possible. While smooshing and smashing fruits, your bot will lose energy. In order to compensate and replenish your energy, our bot should also be solving puzzles. On December 9th, we will have a double elimination tournament in class to see which program performs the best. In this handout will provide you with details on both the Game and the Puzzle parts of the tournament.

The Game

Objective

In each round of the tournament, we will compete two robots, to see which can smash the most fruits. Both bots will be placed at the same random starting location, and the fruits will be falling from the sky at random locations. You are thus required to write a SPIMbot that will participate in this game by smooshing and smashing fruits. As in Lab 9, you will be using the `FRUIT_SCAN` memory mapped I/O to request the locations of the fruits. Refer to the Lab 9 handout for more details.

Types of Fruit

The number of fruits has increased since the spimbot lab! Now in addition to lemons, there are also mangos, guavas, loquats, and cherries. Each fruit has unique properties which are described in the table below. Note that for velocity and acceleration, 0 will represent no velocity/acceleration, fixed means that the value will be the same for all fruits of that type, and random means each fruit of that type can have a range of possible values. eg Mangos always have no velocity in the x direction and the same initial y velocity, but the acceleration value can change between mangos.

Fruit Type	X Velocity	Initial Y Velocity	Y Acceleration	Probability	Points
Lemon	0	fixed	0	40%	1
Mango	0	fixed	random	20%	2
Guava	random	random	0	15%	4
Loquat	random	fixed	random	15%	6
Cherry	random	random	0	10%	10

Acceleration: negative values are possible
Cherries: be careful with these! If your bot moves within a radius of 50 pixels of a cherry, the cherry will detect your presence and try to run away. It will change its x velocity and y velocity and you probably won't be able to catch it once this happens. To catch these, you will probably need to calculate its future position so you can go to that position and set your velocity to 0 before the cherry gets within 50 pixels of the location. If your velocity is 0, it will not detect your presence and its velocity will stay constant (unless your opponent upsets it...).

Smooshing/Smashing and Energy

As discussed earlier, your SPIMbot should be Smooshing and Smashing fruits the way it did in Lab 9. One thing that changes for the competition is that your bot will have a certain *energy*, and that energy will deplete when you carry smooshed fruits, and also when you smash fruits. You'll want to make sure you don't run out of energy, otherwise you won't be able to continue smooshing or smashing fruit until you gain more energy. In addition, if you run out of energy while carrying fruits, they will all be lost and you will receive no points for them. In order to replenish your energy, you should solve puzzles. The more puzzles you solve successfully, the more energy you get. We will now give you more details about puzzle solving.

The Puzzles

Objective

The role of the puzzles in this competition is to regenerate your bot's lost energy. As we discussed earlier, your bot will lose energy during the game, and therefore it needs to solve puzzles in order for it to regain energy.

Puzzle Description

The puzzle, as you might have guessed from labs 7 and 8, is a "word search" style puzzle. You are provided with a grid of letters and a word that you have to search for in that grid. Here are some important points that you should know regarding the puzzle:

- The puzzle is in the form of a grid of characters, that you should search in, to find the desired word.
- The number of rows and number of columns in the grid could range anywhere between 16 and 64 each (inclusive).
- The length of the word is at most 80 characters long and the word is NULL terminated
- The word might wrap around in the grid. In other words, if you reached the last row or last column, the next letter might be in the first row or first column respectively.
- When you are solving the puzzle you will need to build a linked list with the locations of the different letters of the word saved in each node. In order to that refer to the Lab 8 handout and code.
  - If you refer to the lab 8 main.s files, you will see that the `allocate_new_node` subroutine we provided you with uses a statically allocated memory area `node_memory` to store the newly allocated nodes and a memory location `new_node_address` to point to the next free memory address in the `node_memory` to allocate the new linked list node there.
  - Make sure you understand the procedure of node 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 `node_memory` after submitting a puzzle solution so that you don't run out of memory after allocating so many nodes. i.e. Every time you want to solve a new puzzle, you should start out with an empty `node_memory`.
  - In order to free your `node_memory`, it suffices to have the `new_node_address` point to `node_memory`, just like it is initialized in the data section.
- Finally, you should make sure you allocate enough space for your puzzle grid, puzzle word, and node memory in case of overflow:
  - Puzzle Grid: 8192 bytes
  - Puzzle Word: 128 bytes
  - Node Memory: 4096 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 regenerate your energy 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 fruit array in Lab 9 will be used. You will need to allocate two static memory locations in your `.data` section, one for the puzzle grid and one for the word. In order to request a puzzle, a memory mapped address called `REQUEST_PUZZLE` is provided. As with `FRUIT_SCAN`, you should store the address of the statically allocated memory for the puzzle into the `REQUEST_PUZZLE` memory mapped I/O. However, as opposed

to the case of requesting the fruits 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 representing the number of rows; followed by
- 32 bit integer representing the number of columns; followed by
- The puzzle grid, the different rows of the grid stored consecutively

After receiving the interrupt, the you should then request the word. In order to do that you have been provided with a memory mapped address called `REQUEST_WORD`. 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 word stored. This time, the word will arrive instantaneously. Remember, when reading the word, that the it is null terminated, i.e. the last byte will be NULL.

Finally, after you solve the puzzle and generate a linked list with all the locations of the different letters of the word in the grid, 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 a pointer to the first node in your linked list, 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 smashing more fruit, but also by smashing the right fruit. Remember, different fruits have different amounts of points.

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:

- Catch enough fruits to gain at least 75 points
- Solve enough puzzles in order to not run out of energy

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 fruit before having to smash the fruit, which can save on travel time
- If you move your bot higher, you may have a chance to get the fruit before your opponent. The drawback is that smashing fruits comes at a higher cost, and if a fruit falls too quickly you may not be able to reach it before it's too long gone.
- Predicting where fruits will be in the future can be a very powerful tool to maximize your catching abilities. Not only does this help minimize wasted time since you won't have to track a fruit the whole way down, but it is also probably one of the only reliable ways to catch some of the fruits which have less easily trackable motions.