"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 2nd at 8 PM, and only ONE team member should submit. No late submissions accepted!

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 is provided in the `_shared` directory in svn. You are free to use it in your contest implementation. We have also provided 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/cs233sp17/SPIMbot+documentation`

Problem Statement

In this assignment, you are to design a SPIMbot that will compete with other SPIMbots in the adorable land introduced in Lab10. After completing Lab10, SPIMbot decided to settle in this land, believing it could continue befriending bunnies and using their mystical cuteness to solve the world's biggest problems. However, as soon as SPIMbot settled in this new land, an enemy bot mysteriously appeared in this land. These bots want to capture the bunnies and misuse their cuteness for nefarious schemes. SPIMbot must compete against other SPIMbot species, in a fight to not just solve the world's problems with the aid of bunnies, but also to save the bunnies from being forced to aid in evil. Your bot must place as many bunnies into the protection of your playpen as possible, and may even sabotage the enemy bot. Furthermore, carrots are extremely limited, and SPIMbot will only be able to retrieve new carrots from a nearby garden. Your bot will need to use the CarrotSearch solver that you built in Labs 7 and 8 to find carrots. SPIMbot will only retrieve carrots by successfully solving given CarrotSearch problems.

On May 3rd, SPIMbot and its enemy will battle in a fight to the finish, once and for all, in a double elimination tournament held in class. The program that performs the best will emerge as the sole guardian of the cute bunnies and will live a prosperous life in this adorable land!

This handout will provide you with details on both the Game and the Puzzle parts of the tournament. May The Force Be With You![1]

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

The Game

Objective

In each round of the tournament, we will compete two robots to see which can catch the most bunnies. The bots will be placed at two independent and random starting locations, and each will start with 10 carrots. You are thus required to write a SPIMbot that will participate in this tournament by catching bunnies and placing them in your playpen, and (optionally) sabotaging your opponent's playpen. As in Lab10, you will be using the `SEARCH_BUNNIES` memory mapped I/O to request the locations of the bunnies (in the BunniesInfo struct). Refer to the Lab10 handout or the online SPIMbot documentation for more details.

Carrying Weight Limit

As in Lab10, SPIMbot will need to catch bunnies and place them into the playpen to earn points. However, unlike Lab10, there is a limit on how much weight SPIMbot can carry at any given time (100 Zentners). If SPIMbot's carrying weight *exceeds* that limit, all of the bunnies currently carried by the bot will be lost. :(

To detect if your bot has exceeded the carrying weight limit, SPIMbot can use the `EX_CARRY_LIMIT` interrupt (`EX_CARRY_LIMIT_INT_MASK`). This interrupt will fire when you exceed the carrying weight limit. See the online SPIMbot documentation at the link from the beginning of this handout for more details on the `EX_CARRY_LIMIT` interrupt.

Playpen Locking Mechanism

The bunnies have become very curious after their interactions with SPIMbot. They enjoy exploring and will wander off. Therefore, we have introduced a locking mechanism to the playpen.

When the playpen is unlocked, bunnies will run away at a constant rate (1 bunny per every 100,000 cycles). Bunnies will run away in a LIFO (Last In, First Out) order, that is, the last bunny to be placed in the playpen will be the first to run away.

It is important to keep your playpen locked! You can lock your playpen by writing to the `LOCK_PLAYPEN` memory mapped I/O when you are within 5 pixels of your playpen location. If you are not within 5 pixels of your playpen location, this action will fail and the playpen's lock status will not change. Refer to the online SPIMbot documentation for more details.

Playpen Unlocking Mechanism

Sabotage!

The enemy bot will be collecting bunnies in their own playpen. If you want to sabotage your enemy, you can unlock their playpen and let their bunnies escape. You can unlock *any* playpen by writing to the `UNLOCK_PLAYPEN` memory mapped I/O when you are within 5 pixels of any playpen location. If you are not within 5 pixels of a playpen location, this action will fail and the playpen's lock status will not change. Refer to the online SPIMbot documentation for more details.

Note that you can only help the enemy bot's bunnies escape - you cannot catch them as they are escaping.

Defending from Sabotage

Keep in mind that the enemy bot can sabotage you as well! To detect this treachery, SPIMbot can use the `PLAYPEN_UNLOCK` interrupt (`PLAYPEN_UNLOCK_INT_MASK`). This interrupt will fire when your playpen is unlocked by the opponent bot. To counter this sabotage, you can drive to your playpen to lock it before too many bunnies escape. It may be advantageous for you to stay near the playpen so that you can readily defend from the enemy bot's sabotage. See the online SPIMbot documentation at the link from the beginning of this handout for more details on the `PLAYPEN_UNLOCK` interrupt.

Unlocking Limit

Once the playpen is locked, it will remain locked for 100,000 cycles before anyone can unlock it.

Newly Requestable Information

We have added memory mapped I/O to request query information about your bot:

- `NUM_BUNNIES_CARRIED`: Returns an integer indicating the number of bunnies the bot currently carries.
- `NUM_CARROTS`: Returns an integer indicating the number of carrots the bot currently carries.

Note that there is no memory mapped I/O to request the weight that your bot currently carries. You will have to keep track of this information yourself!

We have also added memory mapped I/O to query information about the enemy bot.

- `OTHER_BOT_X`: Returns an integer indicating the current x position of the enemy bot.
- `OTHER_BOT_Y`: Returns an integer indicating the current y position of the enemy bot.
- `PLAYPEN_OTHER_LOCATION`: Returns one integer that encodes the x and y location of the enemy bot's playpen (encoded in the same way as `PLAYPEN_LOCATION`). Refer to the `PLAYPEN_LOCATION` documentation in the Lab10 handout and/or the online SPIMbot documentation to review the encoding scheme.

If there is no enemy bot (in the case that you run your bot alone), reading from any of the enemy bot queries will return -1. We will be running your bot alone when grading for the baseline requirements, so make sure your code won't break in the case that there is no enemy bot!

See the online SPIMbot documentation at the link from the beginning of this handout for more details on these new memory mapped I/O.

Competing for the Same Bunnies

Keep in mind that SPIMbot and the enemy bot will be receiving the same BunniesInfo struct when using `SEARCH_BUNNIES`. This means that you will be competing with each other to catch the same bunnies that are currently in the world. If the enemy bot catches the same bunny you are tracking before you do, you may waste a carrot by trying to catch a bunny that isn't there anymore!

The Puzzles

Objective

The role of the puzzles in this competition is to acquire carrots from a nearby garden. As discussed earlier, your bot will be able to restock carrots by completing CarrotSearch puzzles. Upon successful completion of a puzzle, the bot will be granted one carrot. On the other hand, an incorrect puzzle solution will cost you one carrot!

Puzzle Description

The puzzle is the same puzzle described in labs 7 and 8 (CarrotSearch). Not much has changed regarding the puzzle representation, but there was a subtlety in our earlier release of Lab7 and Lab8 solutions that may cause an error with LabSpimbot functionality. The puzzles are bigger now, and rolling the integers around when there is integer overflow is an important component of the CarrotSearch solver. You can use the updated solutions in the `_shared` directory in SVN (make sure you get the latest version), or you can use your own solution. If you choose to use your own solution, make sure you replace all instances of `add` in the C++ code with `addu` to avoid an arithmetic overflow exception.

We will next discuss the process of requesting and submitting puzzles. You will have find the best way to solve the puzzles to acquire carrots 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 BunniesInfo struct in Lab10 will be used, but with a new memory mapped address called `REQUEST_PUZZLE`. First, you will need to allocate one static memory location in your `.data` section for your puzzle. As with `SEARCH_BUNNIES`, you should store the address of the memory allocated for the puzzle into the `REQUEST_PUZZLE` memory mapped I/O address. Below is an example of how to request a puzzle, see the online SPIMbot documentation for more information about requesting puzzles.

```
la  $t0, puzzle_data
sw  $t0, REQUEST_PUZZLE
```

However, 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. 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 (in this order):

- Node array, which makes up the graph that you will need to provide to your CarrotSearch solver. The first element in this array is the root Node (will be an input to your CarrotSearch solver).
- Identity information about the Nodes (will be used within your CarrotSearch solver, but *not* an explicit input).
- k, the number of best baskets that you should extract when using your CarrotSearch solver (will be an input to your CarrotSearch solver). This is the last *word* in the puzzle.

The puzzle (including the Node and Identity elements, and k) will have a fixed size of 9,804 bytes, so make sure you allocate 9,804 bytes of space for the puzzle to be copied into.

After receiving the interrupt, you will need to acknowledge the interrupt and solve the puzzle using much of the code that you wrote in Lab7 and Lab8. Simply call `search_carrots` with the proper arguments:

- `int max_baskets`: Should always be 10, because your Baskets struct will have 10 baskets.
- `int k`: Should be extracted from puzzle.
- `Node *root`: Should be extracted from puzzle.
- `Baskets *baskets`: You will need to allocate and initialize your Baskets struct. Recall from Lab8 that the Baskets struct will keep track of the best baskets found during the CarrotSearch algorithm. You should allocate 44 bytes of space for your Baskets struct. Furthermore, every time you run the CarrotSearch solver, you should initialize `num_found`, the first element of your Baskets struct, to 0.

Make sure you allocate enough space for your puzzle and your Baskets struct:

- Puzzle: 9,804 bytes
- Baskets: 44 bytes

Refer to the Lab8 handout for more details on the `SEARCH_CARROTS` function.

After solving the puzzle, your bot should submit the integer solution returned by the `search_carrots` function. In order to do so, a memory mapped address has been provided, `SUBMIT_SOLUTION`. You should therefore provide the solution by storing it to 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.

Grading

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:

- Earn a score of 250 Zentners when run by itself.
- Solve at least 1 puzzle.

Winning

In order to win the competition you should get a higher score than your opponent. This can be achieved, not just by catching bunnies and placing them into your playpen, but also by sabotaging your enemy's playpen.

Strategy

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 will get you more carrots. Solving puzzles while your bot is moving to a new location could make good use of travel time.
- Being smart about how far you go from your own playpen. When it comes to defending against sabotage, it may be helpful to remain near your playpen to be able to quickly defend your bunnies. On the other hand, this may limit which bunnies you will catch, and you may miss out on heavier bunnies!

Good Luck!