“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 11th 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.** 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.illinois.edu/wiki/display/cs233fa17/SPIMbot+documentation
Problem Statement

It’s time. After all that practice flying around the Cygnus X-1, the competition will begin soon. Conditions will be the same as before: the event horizon will speed you up, and colliding with the black hole will set you back a lap. Hawking Radiation will continue to generate star coins for you to pick up.

However, the competition will take place in the form of a 1v1 double elimination tournament. This means that you will be racing against another Spimbots every round. This is where the star coins will come in handy. They will grant you the energy to use two abilities: space junk, nicknamed “bananas” by the Spimbots, and speed boosts, nicknamed “the mushroom” by the Spimbots, after the mushroom cloud caused by the aftermath of using a speed boost of such explosive power.

On the date of the competition, you and the other Spimbots will compete to see which Spimbots can call themselves the fastest in the galaxy. May the Force be with you!
The Game

Objective
In each round of the tournament, two bots will race each other to see who can fly the most laps. You are thus required to fly circles around the black hole while dodging the obstacles the opposing bot throws at you and (optionally) sabotaging your opponent. As in Lab 10, you can use using the REQUEST_JETSTREAM and REQUEST_RADAR memory mapped I/O to determine the location of the event horizon and the star coins. Refer to the Lab10 handout or the online SPIMbot documentation for more details.

Newly Requestable Information
We have also added memory mapped I/O to query information about your opponent.

- OTHER_BOT_X: Returns an integer indicating the current x position of the opposing bot.
- OTHER_BOT_Y: Returns an integer indicating the current y position of the opposing bot.

If there is no opposing bot (in the case that you run your bot alone), reading from any of the opposing 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 opposing 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.
The Puzzles

Objective

The role of the puzzles in this competition is to unlock certain restricted features on yourself: releasing space junk to slow other bots down, and using nuclear pulse propulsion to speed yourself up. While star coins give you the energy to use the features, server side authentication is necessary to activate them. By solving the puzzles, you crack the authentication and can use a restricted feature.

In summary, this is what you need to do to use powerups:

- Collect at least 4 star coins
- Request the puzzle
- Submit your puzzle solution
- Use your powerup

Puzzle Description

The puzzle is the same puzzle described in labs 7 and 8 (decryption and finding the longest substring of n characters). However, the tasks are now combined and so your code will have to piece together the puzzle solutions to solve them.

You will be given an encrypted string of length 64, but since the decryption function from previous labs only operates on strings of length 16, you will have to call decrypt 4 times. Once the input string is decrypted, you can then find the maximum length substring containing up to n unique characters.

While the solution code for lab7 and lab8 will correctly solve the puzzles, there are a variety of optimizations that could be applied to solve puzzles faster (and therefore get more items). Highly competitive bots will almost certainly have optimized puzzle solvers in order to gain an advantage over opponents. We will next discuss the process of requesting and submitting puzzles.

Requesting and Submitting Puzzles

In order to request a puzzle, you can use the memory mapped IO REQUEST_PUZZLE. First, you will need to allocate one static memory location in your .data section for your puzzle. 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, you 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):

- Encrypted array, which is the 64 character encrypted string.
- Key array, which is a 144 character key.
- Rounds, a character representing the number of decryption rounds needed to be run.
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. See the Spimbot documentation for further details on adapting your Lab 7 and Lab 8 code to this part of the Spimbot competition.

After solving the puzzle, you should submit the solution. In order to do so, a memory mapped address has been provided, SUBMIT_SOLUTION. However, you will need to provide an address to SUBMIT_SOLUTION. You should therefore provide the solution by storing it to memory, and then storing the address of this memory to the SUBMIT_SOLUTION memory mapped I/O.

When you have submitted a correct solution, you will be rewarded with either a banana or a mushroom (randomly chosen, so you will have to use the banana and mushroom mmio to detect what items you currently have).

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

Items

Mushrooms
Mushrooms give you a speed boost, allowing you to travel a maximum speed of 11 pixels (instead of the normal 10 you get in the jetstream) in 10,000 cycles. However, the main powerup from mushrooms is that this speed boost can also act off of jetstream pixels, allowing you to take shortcuts without getting a speed penalty.

Note that some of these speed constants have changed since lab10. The jetstream pixels allow a speed of 1 pixel per 1000 cycles, and non-jetstream pixels are now 8 times slower than jetstream pixels. If you use a mushroom, your boost will last for 60,000 cycles.

Bananas
Once dropped, bananas stay on the map until the end of the game, or until a bot runs into them. Running into a banana causes a penalty of being unable to move for a period of 100,000 cycles. Bananas have a hitbox where each side of the box is 8 pixels long (the coordinate returned by the radar is the center coordinate of the hitbox).

The spimbot radar has been updated to see not only starcoins, but also bananas. The banana locations come after the first terminating word, and have a terminating word of their own (see SPIMbot documentation on the wiki page for more details about the radar).

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:

- Beat our baseline bot (same difficulty as in lab 10) in number of points (20%)
- Solve at least 1 puzzle (20%)
- Not hit bananas (20%)

Since the velocities have been lowered across the board from lab10, you can assume that the baseline bot will be getting at least 4 times fewer number of points as it was for lab10 (ie if you did fine in lab10 for number of points and you get a quarter the amount here, you can assume it will still meet the requirements).
Winning
In order to win the competition you should get a higher score than your opponent. This can be achieved by completing more laps than your opponent as well as slowing them down with the tools at your disposal.

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 powerups. Solving puzzles while your bot is moving to a new location could make good use of travel time.
• Make sure to watch out for space junk! Be sure you can take evasive maneuvers to dodge obstacles.

Good Luck!