"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. Creative problem solving

Work that needs to be handed in (via SVN)

1. `spimbot.s`, your SPIMbot tournament entry,
2. `partners.txt`, a list of your 1 or 2 contributor's NETIDs,
3. `writeup.txt`, answer the 4 questions with a few paragraphs each (in ASCII) that describe your strategy and any interesting optimizations that you implemented, how your team divided the work and operated, (solid responses to these questions will earn 10 points of extra credit for this assignment), 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.** Otherwise, we'll have too many programs for the competition. We will not compete and assignment done solo, so such assignments cannot earn the 40 points from the competition.
- You'll want to use `/class/cs232/Linux/bin/QtSpimbot`. See Lab 9 for directions on running Qt-Spimbot.
- 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 (the full source is provided in the _shared directory), as long as you let us know after the contest what you did.
- We will not try to break your code; we will compete it against the other students.
- We've provided solution code for Labs 7-9. You are free to use any of this code in your SPIMbot contest implementation.
- The contest will be run using the SPIMbot executable installed on the EWS linux machines; so that executable should be considered to be the final word on correctness.
- Documentation on SPIMbot for this semester can be found at:
  `https://wiki.engr.illinois.edu/display/cs398sp13/SPIMbot+documentation`

Problem Statement

In this assignment you are to produce a program for SPIMbot that will grow the longest possible snake. On Wednesday, 5/1, we will have a double elimination tournament to see which program performs the best.

In each round of the tournament, we will compete two robot snakes, to see which can grow their snake to be the longest. You can grow your snake by "eating" apples; an apple is considered to be eaten by the first snake that drives over it. The length of the snake is reduced if its head runs into either a snake body (its own or its opponent's) or a wall (more details below). Whichever snake is longer when the contest ends (after 2,000,000 cycles) will be declared the winner. If both snakes are equal length, a random winner will be chosen.

Each snake will be placed in a fixed starting position based on whether they are the red or blue player. At all times during the game there is one apple, whose location can be queried by both players (by reading the `PUBLIC_APPLE_X` and `PUBLIC_APPLE_Y` memory-mapped I/O (MM-IO) locations). We call this the "public apple". Every time it is eaten by a snake, a new public apple is created in a random location. Snakes can request an "apple eaten" interrupt to be notified each time an apple that is visible to your snake is eaten. The first public apple is always in the middle of the map.

In addition, each snake has the ability to create "private" apples. These apples are private in the sense that only the creating player is informed of where they are, but they can still be picked up by either player. Private apples are created by solving the puzzles from Lab 8. To receive a puzzle, first, create a HexList data structure in memory capable of holding the list for a 4x4 matrix puzzle (e.g., `list2` from `p4_main.s`. Then write the memory address of the HexList structure to the `REQUEST_PUZZLE` MM-IO address. The I/O device will fill in this structure with a new puzzle. Solve this puzzle to create a HexMatrix data structure (*e.g.*, `blank2`) and then write the address of HexMatrix data structure to the `SUBMIT_PUZZLE` MM-IO address. Some time after submission, you will receive a "private apple created" interrupt. For the next 100 cycles after this interrupt, the location of the private apple can be read from the `PRIVATE_APPLE_X` and `PRIVATE_APPLE_Y` MM-IO addresses. SPIMbot will check the solution against the most recently requested puzzle. If an incorrect solution is provided, an interrupt will still be raised but a random map location will be provided by `PRIVATE_APPLE_X` and `PRIVATE_APPLE_Y` and **NO** new private apple will be created. There is no limit to the number of private apples that can be on the screen at the same time.

Snakes are controlled by setting their heads to point in one of the cardinal directions (up, left, down, right). The speed of the snake cannot be controlled; it is constant unless the snake runs into something. If the snake runs into something (either a wall or another snake body), its head stops but its tail keeps going. As a result, during the time period when the head is stopped, the snake starts to shrink. This is bad, because the winner of each match is determined by which snake is longer.

At any time a SPIMbot program can request the positions of its or its opponent's snake. As per Lab 7, the snake positions are described by the series line segments that connect an array of (X,Y) points that specify the snake's head, each point where the snake turned, and the end of the tail. These X and Y locations are stored as two independent arrays. For your own snake location, the arrays of points can be retrieved by writing the memory addresses of empty arrays to the `PIVOT_NODES_X` and `PIVOT_NODES_Y` MM-IO locations. Your opponents locations can retrieved using the `OTHER_PIVOT_NODES_X` and `OTHER_PIVOT_NODES_Y`. As the snake can have at most 256 points, you should provide at least 1024 bytes for these empty arrays. Such arrays can be allocated in your data segment using the `.size` directive as follows:

```
the_name_of_your_array:  .size 1024
```

If you only want to know the location of your or your opponent's snake head, these can be read directly from `HEAD_X` and `HEAD_Y` and `OTHER_BOT_HEAD_X`, `OTHER_BOT_HEAD_Y`, respectively.

You might find that giving SPIMbot the `-debug` parameter will be useful for your debugging; it prints out the puzzle solutions, whether each solution submission is correct, and the coordinates of the public and private apples as they are placed on the map. Note: you can use the `PRINT_INT` memory-mapped I/O address to print out the values you generated if you want to compare your values with ours.

Also, you can pause the contest's execution by clicking on the map with your mouse.

Strategy

In class, we'll discuss the merits of the iterative design process. You are strongly encouraged to, first, get a simple implementation working and then focus on optimizations. Specifically, a simple implementation would:

1. Reads the location of the public apple.
2. Drives toward the location of public apple.
3. Whenever an apple gets eaten, go to step 1.

In Labs 7 through 9, you have already written much of the code necessary to assemble such a contest entry, with very little effort. Such an entry will earn 30 points on this assignment.

To guarantee a score of at least 60 points on this assignment, you must do **at least one of the following**:

1. eat at least one private apple. (This too can be achieved by using your code (or our solutions) to labs 7-9.)
2. or ensure that your snake never runs into itself while it continues to eat public apples. (We will test this by seeing if your snake can eat 20 apples before running into itself. To ensure that you didn't just get lucky, we'll repeat this test a total of 5 times.)

The remaining 40 points will be determined based on your rank in the SPIMbot contest. In the passed, grades have been assigned roughly as follows: the worst performing 10% of the teams earn 0 points, the next 20% of the teams earn 10 points, the middle 40% of the teams earn 20 points, 20% of the teams earn 30 points, and the top 10% of the teams earn the full 40 points from the contest.

To improve your chances of winning the competition, you can optimize your SPIMbot program. While there are many ways that this can be done, we provide a few suggestions.

- SPIMbot can perform computation in parallel with driving. This makes it ideal to try to solve puzzles while picking up apples.
- You can optimize the code that solves puzzles, so you can create more apples for yourself.
- As you solve puzzles, you will know about multiple apples that you could pick up. You can prioritize these to minimize the distance that you need to drive.
- You can try to avoid running into things.
- You can try to force your opponent to run into you.