

# CS246 Fall 2024 Project – RAIInet

Godfrey, Lushman, Schmitz

- DO NOT EVER SUBMIT TO MARMOSET WITHOUT COMPILING AND TESTING FIRST. If your final submission doesn't compile, or otherwise doesn't work, you will have nothing to show during your demo. Resist the temptation to make last-minute changes. They probably aren't worth it.
- This project is intended to be doable by three people in two weeks. Because the breadth of students' abilities in this course is quite wide, exactly what constitutes two weeks' worth of work for three students is difficult to nail down. Some groups will finish quickly; others won't finish at all. We will attempt to grade this assignment in a way that addresses both ends of the spectrum. You should be able to pass the assignment with only a modest portion of your program working. Then, if you want a higher mark, it will take more than a proportionally higher effort to achieve, the higher you go. A perfect score will require a complete implementation. If you finish the entire program early, you can add extra features for a few extra marks.
- Above all, MAKE SURE YOUR SUBMITTED PROGRAM RUNS. The markers do not have time to examine source code and give partial correctness marks for non-working programs. So, no matter what, even if your program doesn't work properly, make sure it at least does something.
- **Note:** If you have not already done so for Assignment 4, make sure you are able to use graphical applications from your Unix session. If you are using Linux you should be fine (if making an ssh connection to a campus machine, be sure to pass the `-Y` option). If you are using Windows and PuTTY, you should download and run an X server such as XMin, and be sure that PuTTY is configured to forward X connections. Alert course staff immediately if you are unable to set up your X connection (e.g. if you can't run `xeyes`).

Also (if working on your own machine) make sure you have the necessary libraries to compile graphics. Try executing the following:

```
g++20i window.cc graphicsdemo.cc -o graphicsdemo -lX11
```

## The Game of RAIInet

In this project, your group will work together to produce the video game RAIInet, a game similar in style to Stratego. RAIInet is a game played between two opposing players, who take on the role of computer hackers. Each player controls eight pieces, called links. There are two different kinds of links: viruses and data. Each player has two goals:

- Download four data.
- Make their opponent download four viruses.

Upon achieving either goal, the player wins the game.

A game of RAIInet consists of an  $8 \times 8$  board on which players initially place *links* face-down. In this way, only the player who placed the link knows whether it is *data* or a *virus*.

Play then proceeds in turns. On each player's turn, a player may use a single ability, and then must move one of their links in any cardinal direction.<sup>1</sup>

When a player moves a link off their opponent's *end of the board*, they immediately download it. When a player moves a link onto their opponent's *server ports*, their opponent downloads it. Additionally, when a player's link loses to an opponent's link *in battle*, their opponent immediately downloads it. These mechanics are explained in greater detail below.

As which links are viruses and which links are data is hidden information, you can keep your opponent guessing as to what your game plan is!

<sup>1</sup>A cardinal direction consists of: north, south, east, west.

## Links

There are two kinds of links: data, and viruses. If at any point a player has downloaded four data, they win the game. Conversely, if they have downloaded four viruses, they lose the game. In addition to being viruses or data, links have a strength between 1 and 4, with 1 being the weakest and 4 being the strongest. Initially, players are given viruses and data of each possible strength, for a total of 8 links.

## Movement, Battle, and Capture

Each turn a player must move a single link they control one space in a cardinal direction. (There is a “global compass”, so north is the same direction for both players.) This movement is subject to a few conditions:

- One cannot move a link on top of another link one controls (but can on top of an opponent’s link; see below).
- One cannot move a link on top of one’s own server ports.
- One cannot move a link off the edge of the board, except off the edge upon which one’s opponent’s links started i.e. either the top or the bottom edge, respectively. When one moves a link off the edge in this way, one downloads that link.
- If a link moves into one’s opponent’s server port, it is downloaded by one’s opponent.
- When a link is moved on top of a link controlled by one’s opponent, both links are revealed to all players and they battle; the winner is the player whose link had higher strength. In the event of a tie, the initiating player wins the battle. After the battle, the winner downloads the link of the loser.

After moving one link, one’s turn ends. This is the only way a turn can be ended. Downloading a link must update the appropriate counters. It is up to you as to whether or not the link’s value is revealed.

## Display

The game board is an 8x8 grid, similar to chess or checkers. The middle two squares on the first and last row are the server ports. To set up, players place their links on the first and last rows respectively. The links that would be placed over the server ports are instead placed on the second and second-last rows. Here are two example board setups, one at the start of the game and one halfway through (as seen by player 1).

Player 1:	Player 1:
Downloaded: 0D, 0V	Downloaded: 1D, 2V
Abilities: 5	Abilities: 3
a: V1 b: D4 c: V3 d: V2	a: V1 b: D4 c: V3 d: V2
e: D3 f: V4 g: D2 h: D1	e: D3 f: V4 g: D2 h: D1
=====	=====
abcSSfgh	a.cSSf.h
...de...	...E..g.
.....	.....
.....	.....
.....	.....
.....	A.....
...DE...	...D.F..
ABCSSFGH	...SS.GH
=====	=====
Player 2:	Player 2:
Downloaded: 0D, 0V	Downloaded: 1D, 1V
Abilities: 5	Abilities: 3
A: ? B: ? C: ? D: ?	A: ? B: V2 C: V3 D: ?
E: ? F: ? G: ? H: ?	E: V4 F: ? G: ? H: ?

Notice that all of player two's links appear as question marks; as the game progresses, player 1 may gain information as to what player two's links are. Your display must update itself to reflect this new information as it becomes available. Player 2 would see something similar, but with question marks on the values of player 1's links. Your display should always display all the information available to the active player. Both of the boards above would be how the display is printed on player 1's turn. Note that additionally tracked on the display is each player's number of unused abilities, and their download total—this information must also be reflected on your display. For more information on abilities, read the following section.

In addition to a text display, you are required to create a graphical display. This display must show the board like above, with all of the same information. In addition to a line showing whether links are viruses or data, this should also be reflected in the colour of revealed links in your display (e.g., black for unknown, green for data, red for viruses).

**Question:** In this project, we ask you to implement a single display that maintains the same point of view as turns switch between player 1 and player 2. How would you change your code to instead have two displays, one of which is from player 1's point of view, and the other of which is player 2's point of view?

## Abilities

In addition to the basic movements, each player has a set of five *ability cards* available to them at the start of the game. Each card can only be used once per game; however, players select which cards they wish to use, and can take up to 2 of each kind. For example, player 1 may decide to start the game with two copies of the card Firewall, two copies of the card Download, and one copy of the card Polarize. In this case, player 1 would be able to download two of player 2's links, by using both their download cards. The five abilities your game must support are as follows:

**Link boost** modifies a link you control by making it move one additional square in each direction when it moves. This allows the link to “jump over” obstacles. A link with a link boost equipped may be unable to reach the opponent server port. This effect remains active until the link is downloaded.

**Firewall** is played targeting any empty square on the board other than the server ports. When passing through this square, opponent links are revealed and, if they are viruses, immediately downloaded by their owner. A square with a firewall with it is usually printed to the display with an “m” for a firewall by player 1 and a “w” for a firewall by player 2. If a link is placed on a firewall, print the link rather than the firewall. Links owned by the player who placed firewall are unaffected by this ability. If an opponent attempts to battle a player's link which is stationed on the player's firewall: the firewall effect applies prior to the battle taking place.

**Download** immediately downloads an opponent's targeted link. It does not need to be revealed.

**Polarize** changes a targeted data to a virus of the same strength, or vice versa.

**Scan** reveals the type and strength of any targeted link on the field. It does not reveal the player's own links.

On every turn a player may play one ability card that has not already been used in the current game. After doing so, its effects resolve and the player cannot play any other abilities on that turn. After moving, the turn will end.

**Question:** How can you design a general framework that makes adding abilities easy? To demonstrate your thought process for this question, you are *required* to add in three new abilities to your final game. One of these may be similar to one of the already-present abilities, and the other two should introduce some novel, challenging mechanic. **You are required to actually implement these.**

## Winning and Losing

When a player has downloaded four data, regardless of their strength or who they belonged to originally, the player wins the game. Conversely, if whenever a player has downloaded four viruses, they immediately lose the game. Your program need not have a game-over screen, but it must be clear that the program stopped because a player won the game, and it must indicate which player won.

**Question:** One could conceivably extend the game of RAIInet to be a four-player game by making the board a “plus” (

+

) shape (formed by the union of two 10x8 rectangles) and allowing links to escape off the edge belonging to the opponent directly adjacent to them. Upon being eliminated by downloading four viruses, all links and firewall controlled by that player would be removed, and their server ports would become normal squares. What changes could you make in your code to handle this change to the game? Would it be possible to have this mode in addition to the two-player mode with minimal additional work?

## Interactions

You interact with the system by issuing text-based commands. The following commands are to be supported:

- `move a <dir>` moves the link `a` (where this can be any link one controls) in the direction `dir`, which can be any of up, down, left, right. The link `a` can be any value from `a-g` (or `A-G` for player 2).
- `abilities` displays the ability cards one has, together with an ID (from 1-5) and some indication if the ability card has been used or not.
- `ability <N>` uses the ability card with ID `N`. Some ability cards will require additional information, such as a target for Link Boost, or a square of the board for Firewall. Your program must support giving links as targets using the alphabetical code shown in the display section; so if `ability 1` is Link Boost, `ability 1 b` attaches a Link Boost to the link `b`. Squares, such as for Firewall, must be specified by row and then column, where the top left corner is (0, 0). For example, to place a firewall directly below one of player 1's server ports, one would use `ability 1 1 3` or `ability 1 1 4`, assuming that ability 1 is Firewall.
- `board` displays the board in the format shown in the Display section.
- `sequence <file>` Executes the sequence of commands found in `file`, where commands are given as described in this section. This is to facilitate the construction of test cases.
- `quit` or end-of-file (Ctrl-d) exits the game.

It would be in your best interest (and will help during your demo) to make sure that your program does not break down if a command is misspelled.

The board should be redrawn, both in text and graphically, each time a move command is issued. For the graphic display, redraw as little of the screen as is necessary to make the needed changes (within reason).

## Setup

Your program should support the following options on the command line:

- `-ability1 <order>` specifies the abilities for player 1 (this is a list of the 5 abilities player 1 will use). If not specified, use the default set of abilities (Link boost, Firewall, Download, Scan, Polarize in that order). The abilities are given by a string consisting of the first letter of each ability. For example, the default order is `-ability1 LFDSP`. Ensure that when you add new abilities, they start with different letters.
- `-ability2 <order>` is as above but for the second player.
- `-link1 <placement-file>`. The contents of `<placement-file>` should list what each link should be for player 1. The first token of `<placement-file>` specifies what link a is, the second, b, and so on and so forth. For example, the example board in the display section can be made with passing `-link1` a file containing `"V1 D4 V3 V2 D3 V4 D2 D1"`.

If `-link1` is not passed to your program, then you should *randomize* the order of the links that player 1 receives.

- `-link2 <order>` is as above but for the second player.
- `-graphics` should enable your graphical interface.

## Grading

Your project will be graded as following the project guidelines document. Even if your program doesn't work at all, you can still earn a lot of marks through good documentation and design, (in the latter case, there needs to be enough code present to make a reasonable assessment).

## If Things Go Wrong

If you run into trouble and find yourself unable to complete the entire game, please do your best to submit something that works, even if it doesn't solve the entire game requirements. For example:

- don't have abilities
- program only produces text output; no graphics
- server ports don't work
- no battles

You will get a higher mark for fully implementing some of the requirements than for a program that attempts all of the requirements, but doesn't run.

A well-documented, well-designed program that has all of the deficiencies listed above, but still runs, can still potentially earn a passing grade.

## Plan for the Worst

Even the best programmers have bad days, and the simplest pointer error can take hours to track down. So be sure to have a plan of attack in place that maximizes your chances of always at least having a working program to submit. Prioritize your goals to maximize what you can demonstrate at any given time. We suggest: save the graphics for last, and first implement the game in pure text. One of the first things you should probably do is write a routine to draw the game board (probably a `Board` class with an overloaded friend `operator<<>`). It can start out blank, and become more sophisticated as you add features. You should also write the command interpreter early, so that you can interact with your program. You can then add commands one-by-one, and separately work on supporting the full command syntax. Work on your pieces one at a time. Once you figure out the right interface and implementation for one piece type, the remaining piece types will be easier to build. Take the time

to work on a test suite at the same time as you are writing your project. Although we are not asking you to submit a test suite, having one on hand will speed up the process of verifying your implementation.

You will be asked to submit a plan, with projected completion dates and divided responsibilities, as part of your documentation for Due Date 1.

## **If Things Go Well**

If you complete the entire project, you can earn up to 10% extra credit for implementing extra features; see the project guidelines document for details.

## **Due Dates and Deliverables**

See the project guidelines for due dates and submission requirements.