COREWAR
project's organization and usage

CIGLESIA, FGARAULT, RAKROUNA

Compiled October 4, 2020

Core War was inspired by a malicious virus written in the 80's. To deal with the self-replicating virus, a white hat hacker invented Reaper. It was a virus designed to spread and eliminate the malware. He fought fire with fire.
This inspired A. K. Dewdney to coin the idea for Core War.
The idea was simple. You compete by designing viruses to overtake a computer. You win by protecting your own program and overwriting your opponent's programs. This is all happening on a virtual computer. Think, a simple computer within your computer.

https://github.com/fgalar/Corewar

1. INTRODUCTION

This project seeks to create a **virtual machine** (the fight arena) in which *corewar champions* can be executed (fight).
As well as an **assembler** in order to translate the Corewar assembly language into *"Bytecode"* (Bytecode is a machine code, which will be directly interpreted by the virtualmachine).
And finally a **champion** to run on the virtual machine.

There are additional features that will be covered in the bonus section.

2. ORGANIZATION

The tools used for group organization have been the following:

- **github and git** as version-control system.
- **gitkraken** for software project management.
- **LaTeX pdf, org and markdown files** for documentation.

A. Branches

We have implemented a parallel development throughout our project.

Figure 1 shows the basic configuration of the github branches before the integration process (which was an implementation of continuous integration).

- **André** (ciglesia), **fgalar** (fgarault), **jehutyi** (rakrouna) are individual feature development branches.
- **synthesis** is the integration development branch.
- **master** is latest version of the project.

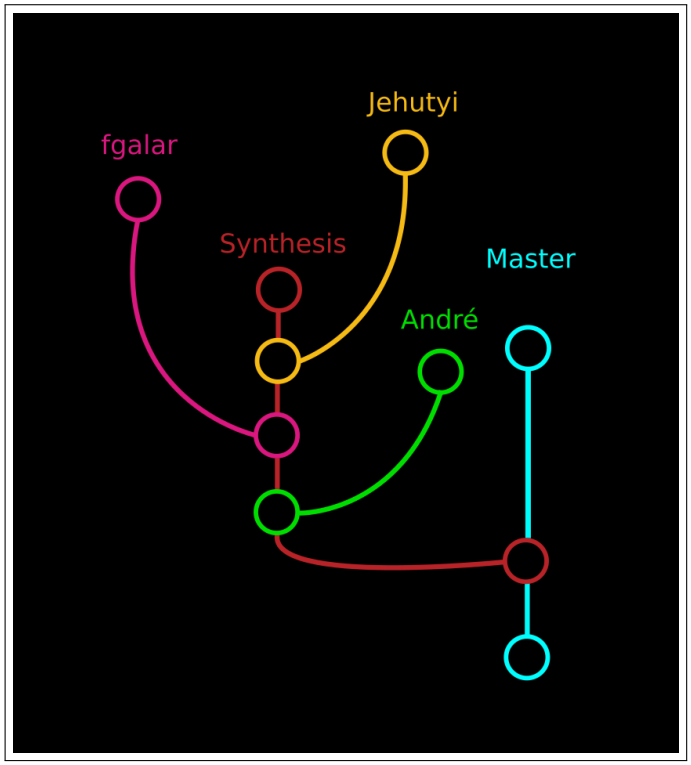
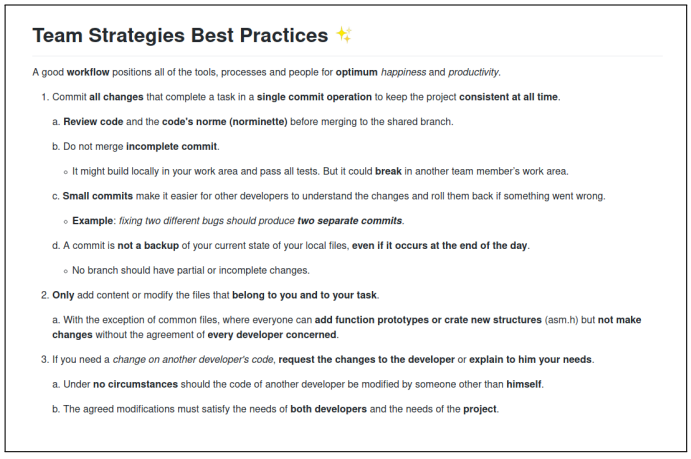


Fig. 1. Github branches (synthesis is the development branch)

B. Teamwork standards

The teamwork standards **greatly facilitated** the development of the project, within the previously mentioned tools (mainly in the control-version system).



Team Strategies Best Practices ✨

A good **workflow** positions all of the tools, processes and people for **optimum** *happiness* and *productivity*.

1. Commit **all changes** that complete a task in a **single commit operation** to keep the project **consistent at all time**.
   a. **Review code** and the **code's norme (norminette)** before merging to the shared branch.
   b. Do not merge **incomplete commit**.
      ◦ It might build locally in your work area and pass all tests. But it could **break** in another team member's work area.
   c. **Small commits** make it easier for other developers to understand the changes and roll them back if something went wrong.
      ◦ **Example**: *fixing two different bugs should produce* ***two separate commits***.
   d. A commit is **not a backup** of your current state of your local files, **even if it occurs at the end of the day**.
      ◦ No branch should have partial or incomplete changes.
2. **Only** add content or modify the files that **belong to you and to your task**.
   a. With the exception of common files, where everyone can **add function prototypes or crate new structures** (asm.h) but **not make changes** without the agreement of **every developer concerned**.
3. If you need a *change on another developer's code*, **request the changes to the developer** or **explain to him your needs**.
   a. Under **no circumstances** should the code of another developer be modified by someone other than **himself**.
   b. The agreed modifications must satisfy the needs of **both developers** and the needs of the **project**.

Fig. 2. Our teamwork best practices (found in the directory docs/ of the project as a markdown file)

3. USAGE

A. Corewar's VM

The loading of the **champions** and the **enumeration** of each player happens in the command line (with the possibility of a **dump** and an **interface** parameter).
Every parameter's validity will be verified.
It's important to remember that the last player will have the first process in the order of execution.

```
USAGE: ./corewar [-n | -dump nbr_cycles] [-v] <[-n number] champion.cor> ...

-n :                    Ncurses display

-dump nbr_cycles:       After nbr_cycles dump the memory and quit the game
                        must be dumped in hexadecimal format with 32 octets per line.

-v :                    Verbosity

-n number:              The number of the player (positive int)
                        if non-existent, the player's number will be generated automatically.

                        The last player will have the first process in order of execution.

The champions (max: 4) cannot go over CHAMP_MAX_SIZE (682), otherwise it is an ERROR
```

Fig. 3. Corewar usage

Champions without options

When champions are loaded without any option, the output is simply the **live executions** with a message at the end saying who the **winner** is.

```
A process shows that player -1 (zork) is alive
A process shows that player -1 (zork) is alive
A process shows that player -1 (zork) is alive
A process shows that player -1 (zork) is alive
A process shows that player -1 (zork) is alive
A process shows that player -1 (zork) is alive
A process shows that player -1 (zork) is alive
Player -1 (zork) won
```

Fig. 4. Zork without dumping the memory

Champions with the -dump option

It is the same output but displaying the memory at a given cycle with **32 bytes per line in hexadecimal** as shown in Fig. 5. The winner message isn't displayed if the game has not finished at the given cycle.

Fig. 5. Dumping the memory at a given cycle

Champions with the -v option

The verbosity flag alone displays some extra information; as the **name, number and size** of the contestants.

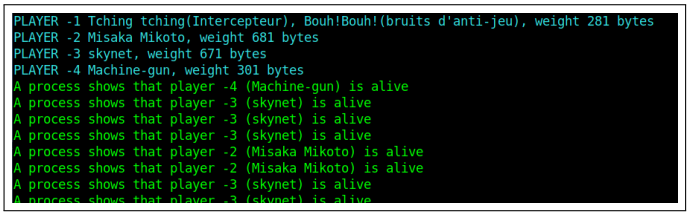


Fig. 6. Contestant's information

As well as the memory at the end of the game with a little color to identify the **actions** of each player. As shown in the figure 7.

If the flag is combined with the **-dump option** we obtain the same result but displaying the memory at the cycle given.

If the flag is combined with the **-n (ncurses) option** we obtain the same result but displaying the memory at the cycle in which we exit the interface.

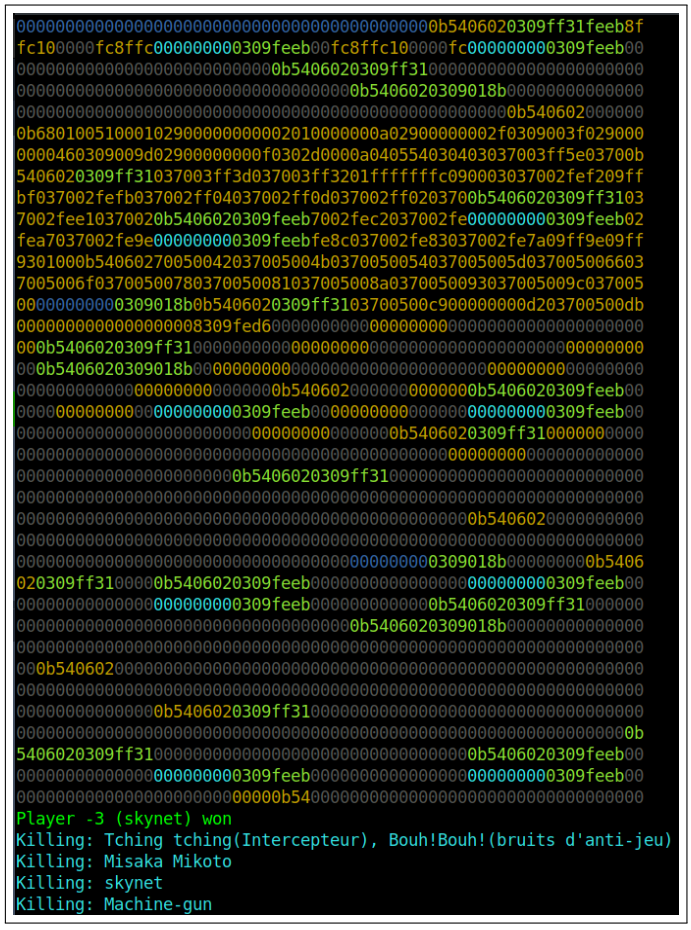


Fig. 7. Dumping the memory with verbosity

Champions with the -n (ncurses) option

Displays in real time the memory and the execution of the champions, with some useful information such as:

- CYCLES
- CYCLES_TO_DIE
- CYCLE_DELTA
- MAX_CHECKS
- CHECKS
- List of players with their "live" count
- List of processes with it's pc, instruction and registers
- Memory with the champions in execution
- The property color for each champion within the memory

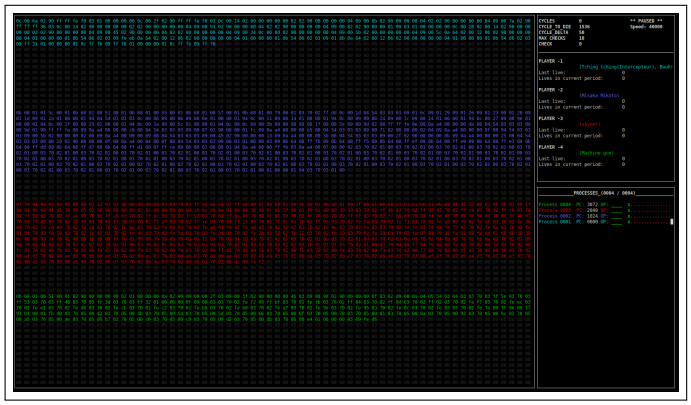


Fig. 8. Ncurse's vm display

Loading champions

The champions are loaded within the memory so that they can space out evenly their entry points.

At startup, each champion have their initial process at their respective entry point. As shown in the figure 9.

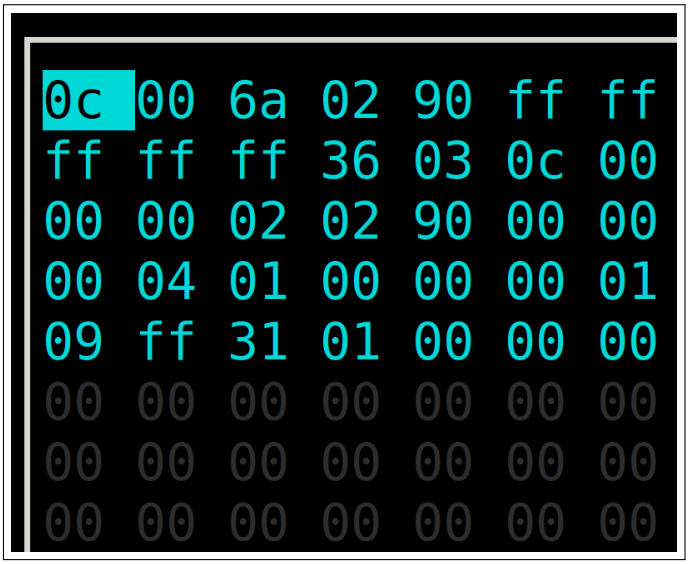


Fig. 9. Initial process

The number (id) of the player is generated by the machine or specified at launch, and is given to the champions via the **r1 registry of their first process** at startup. As shown in the figure 10, where the registry r1 is the green x.

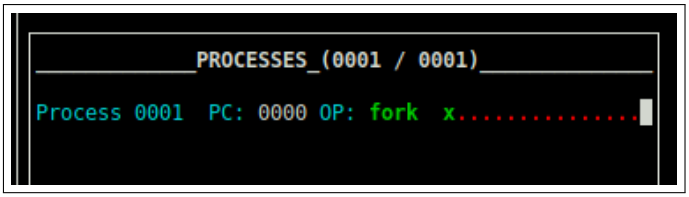


Fig. 10. Process table with one initial process

So even if the champion has no instructions, it will be able to participate as a contestant with the help of the **initial process** and the registry r1 which has its player's id.

B. Assembler

The assembler **(asm)** translates the champion (assembly code figure 12) to bytecode (figure 13 and 14).

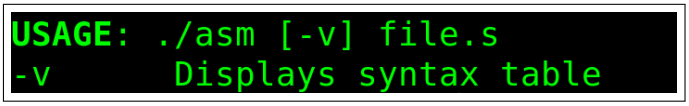


Fig. 11. assembler usage

```
.name "zork"
.comment "just a basic living prog"

l2: sti r1, %:live, %1
    and r1, %0,r2
live:
    live %0
    zjmp %:live
```

Fig. 12. zork.s

```
00000000  00 ea 83 f3 7a 6f 72 6b  00 00 00 00 00 00 00 00  |....zork........|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  00 00 00 00 00 00 00 00  00 00 00 17 6a 75 73 74  |............just|
00000090  20 61 20 62 61 73 69 63  20 6c 69 76 69 6e 67 20  | a basic living |
000000a0  70 72 6f 67 00 00 00 00  00 00 00 00 00 00 00 00  |prog............|
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Fig. 13. zork's header

```
00000880  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000890  0b 68 01 00 0f 00 01 06  64 01 00 00 00 00 02 01
000008a0  00 00 00 00 09 ff fb 09  ff f8
000008aa
```

Fig. 14. zork's instructions in bytecode

asm with the -v option

The asm translates the champion into a .cor bytecode file, as usual, but also displays in the standard output, the **syntax table** with **translation of each instruction** to the right.

```
└─ ./asm -v tmp/zork.s
.name "zork"
.comment "just a basic living prog"
l2:
        sti [r1 %:live %1 ]             0b 68 01 00 0f 00 01
        and [r1 %0 r2 ]         06 64 01 00 00 00 00 02
live:
        live [%0 ]              01 00 00 00 00
        zjmp [%:live ]          09 ff fb
```

Fig. 15. assembler -v with zork

C. Disassembler

The disassembler takes a .cor bytecode file and translates into a .s assembly file.

```
└─ ./asm -v tmp/zork-dis.s
.name "zork"
.comment "just a basic living prog"
MAIN_LABEL:
        sti [r1 %15 %1 ]                0b 68 01 00 0f 00 01
        and [r1 %0 r2 ]         06 64 01 00 00 00 00 02
        live [%0 ]              01 00 00 00 00
        zjmp [%-5 ]             09 ff fb
```

Fig. 16. assembling a disassembled zork.cor

4. BONUS

1. (vm) **ncurse's interface visualizer**
2. (vm) **verbose and formatted output** (-v)
3. (asm) a **detailed syntax table display** (-v)
4. (asm) **beautiful error message** when assembling (with line numbers, and a **cursor** to show where is the error) **differentiating between lexical and syntactical error**.

```
ERROR: syntax: invalid parameter: in line 13: %-487
                                                ^
```

Fig. 17. syntactical error

```
ERROR: lexicon: invalid number of quotes: in line 1: destructor""
                                                            ^
```

Fig. 18. lexical error

5. (other) **disassembler**
6. (other) a **detailed documentation** (also check corewar.org)