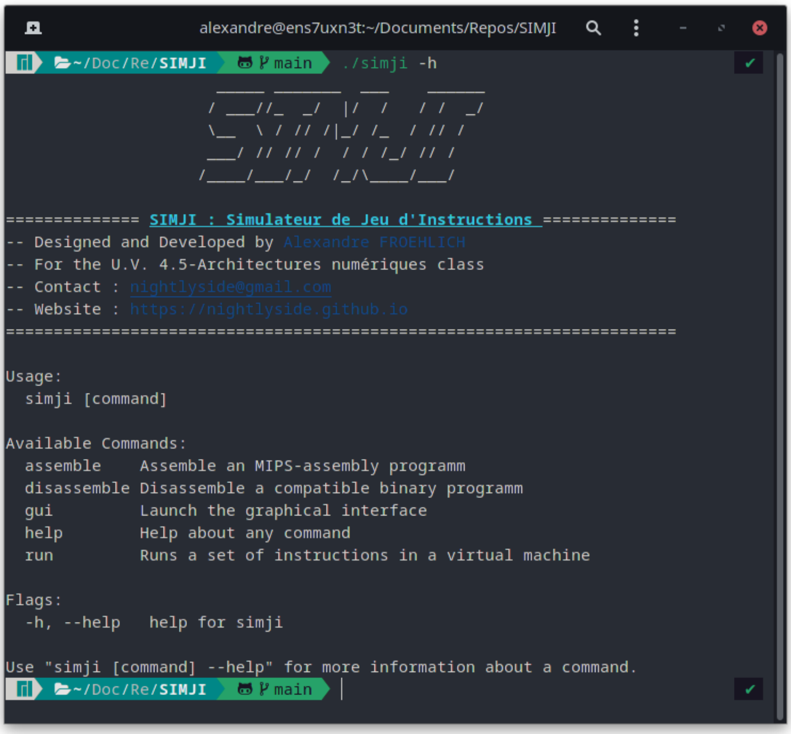
SIMJI - SIMulateur de Jeu d'Instructions

Build passing | codecov 68% | maintainability B | go report A+ | GO reference



```
alexandre@ens7uxn3t:~/Documents/Repos/SIMJI
 ~/Doc/Re/SIMJI  main  ./simji -h
                      _____ _______  ___    ______
                     / ___//_  _/  |/  /   / /  _/
                     \__  \ / // /|_/ /_  / // /
                     ___/ // // /  / / /_/ // /
                    /____/___/_/  /_/\____/___/

============== SIMJI : Simulateur de Jeu d'Instructions ==============
-- Designed and Developed by Alexandre FROEHLICH
-- For the U.V. 4.5-Architectures numériques class
-- Contact : nightlyside@gmail.com
-- Website : https://nightlyside.github.io
======================================================================

Usage:
  simji [command]

Available Commands:
  assemble    Assemble an MIPS-assembly programm
  disassemble Disassemble a compatible binary programm
  gui         Launch the graphical interface
  help        Help about any command
  run         Runs a set of instructions in a virtual machine

Flags:
  -h, --help   help for simji

Use "simji [command] --help" for more information about a command.
 ~/Doc/Re/SIMJI  main 
```

SIMJI is a powerful and fast instruction set simulator (ISS) made in Golang that assemble and run MIPS-assembly programs using a convienient and simple CLI or GUI.

Table of contents

Features

- Fast Virtual Machine (30 000 000 it/sec in average)
- Assemble mini-MIPS-assembly files in binary

- Graphical User Interface integrated to debug and try new code
- Open source project

Installation

From sources

First step you need to install Golang.

Linux

Depending on your distribution :

```
sudo apt install go # Debian based distro
sudo pacman -S go # Arch based distro
```

Windows

Install Go using the graphical installer from the official download page.

Then check that go is working from the command line

```
go version # should print something
```

Then clone the project :

```
git clone https://github.com/NightlySide/SIMJI.git
cd SIMJI
```

Then build the project :

```
make # using the make file
go build . # building only using go
```

> Warning: If you build from sources using only the `go build .` command, the static files for the GUI will not be bundled in the executable
> In order to include those files you need first to package them into a go file using : `go run github.com/markbates/pkger/cmd/pkger`

Once the build is done (should not take more than several seconds) you can use the simulator using the command line :

```
chmod +x simji
./simji -h
```

From a released archive

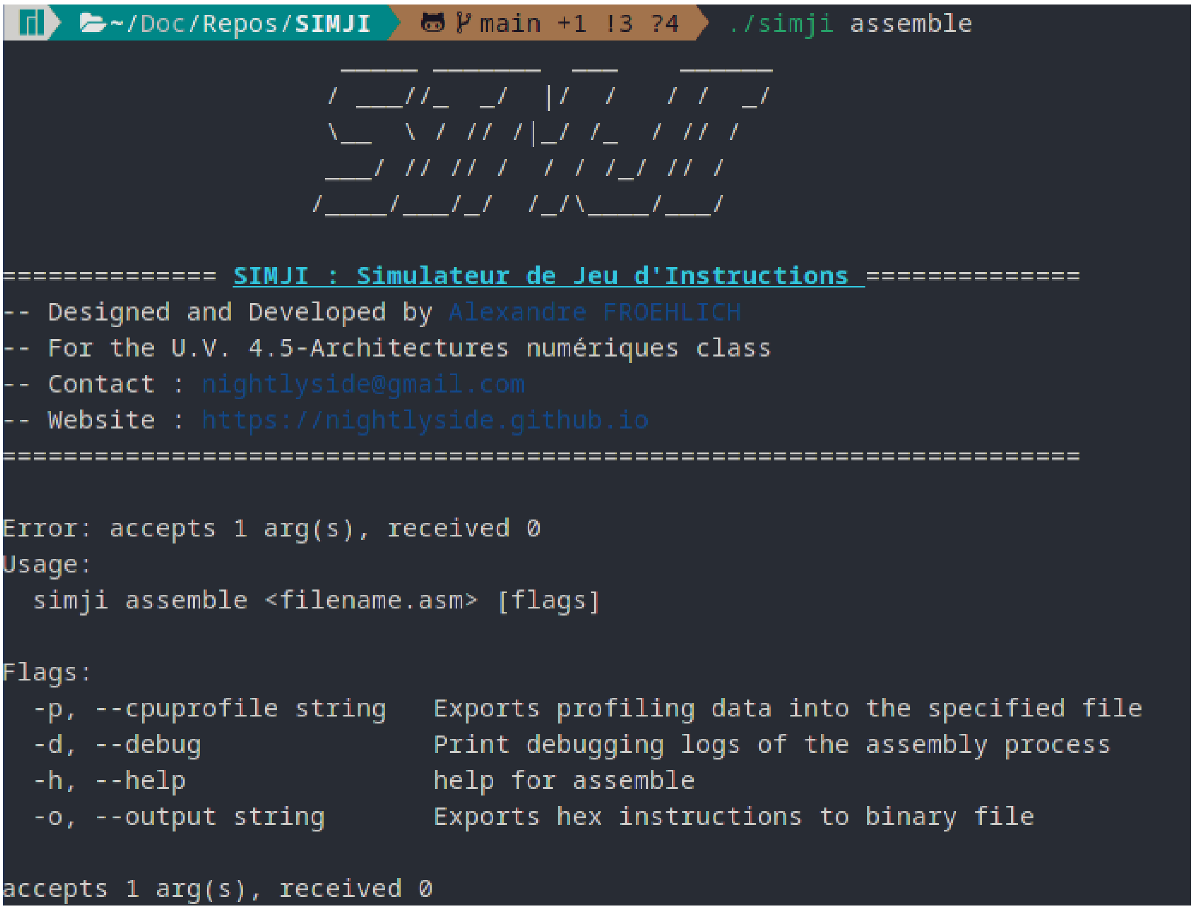
Go to the release page and download the latest release for your operating system.

Then cd into the folder were you placed the binary file :

```
cd Downloads/ # For example
chmod +x simji
./simji -h
```

Usage

Excepting the GUI, you will always need to put a path to a file for the sim to work. Or else you will be facing this message :



```
~/Doc/Repos/SIMJI  main +1 !3 ?4  ./simji assemble
                                   _____ ______  ___    ______
                                  / ___//  _/  |/  /   / /  _/
                                  \__ \ / // /|_/ /_  / // /
                                 ___/ // // /  / / /_/ // /
                                /____/___/_/  /_/\____/___/

============== SIMJI : Simulateur de Jeu d'Instructions ==============
-- Designed and Developed by Alexandre FROEHLICH
-- For the U.V. 4.5-Architectures numériques class
-- Contact : nightlyside@gmail.com
-- Website : https://nightlyside.github.io
======================================================================

Error: accepts 1 arg(s), received 0
Usage:
  simji assemble <filename.asm> [flags]

Flags:
  -p, --cpuprofile string   Exports profiling data into the specified file
  -d, --debug               Print debugging logs of the assembly process
  -h, --help                help for assemble
  -o, --output string       Exports hex instructions to binary file

accepts 1 arg(s), received 0
```

Assembly

For the following examples we will take this simple assembly programs that puts the number 15 in the first register and prints its value on screen :

```
; program.asm
add r0, 15, r1
scall 1
stop
```

In order to assemble a program into binary instructions you need to use the `assemble` command :

```
# Will print the instructions in the terminal
./simji assemble program.asm
```

You may save the instructions in a binary file as well using the `--output` flag :

```
# Will save the content in a file
./simji assemble --output program.bin program.asm
```

The content of the binary file should look like that :

```
0x00000000  0x082001e1
0x00000001  0x90000001
0x00000002  0x00000000
```

Disassembly

The same way you can disassemble a binary file using the `disassemble` command :

```
./simji disassemble program.bin
```

Which should print something like that in the terminal :

```
[+] INFO: No output file specified. Printing binary to console.
add r0, 15, r1
scall 1
stop
```

And likewise you can save the output to an external file using the `--output` flag.

Virtual Machine

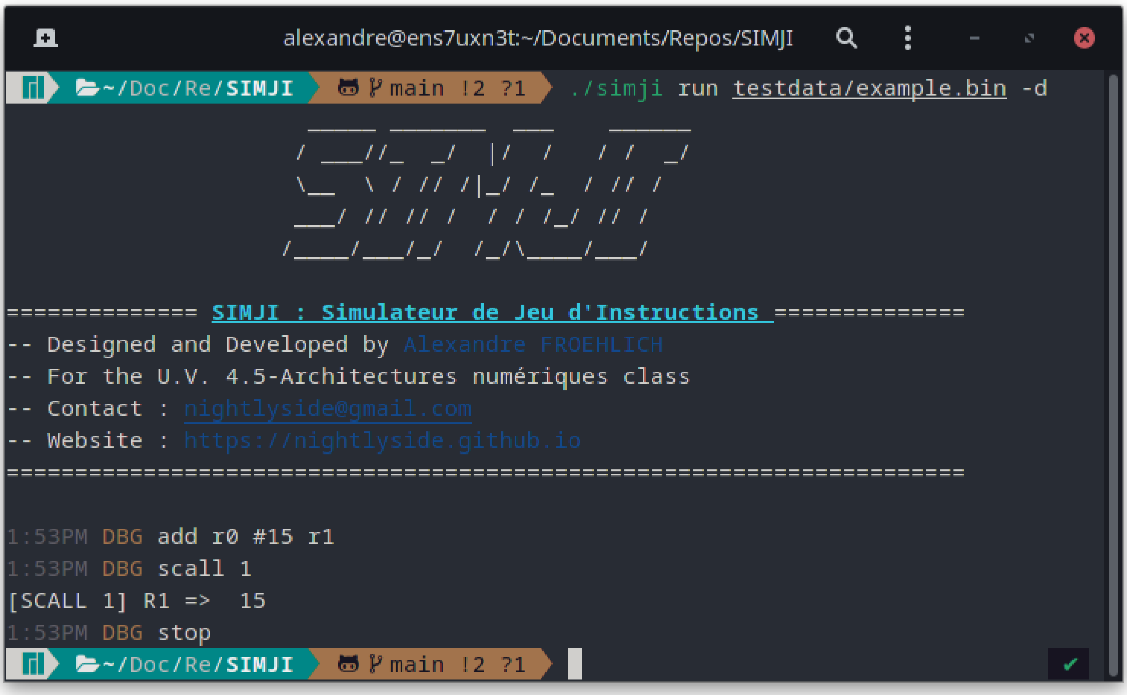
Using the `run` command, you can execute an assembled file:

```
./simji run program.bin
```

Which should print something like that in the terminal:

```
[SCALL 1] R1 =>  15
```

There are several new flags to use with the running process such as the debug `--debug` flag which outputs data about the processes going on like interpreting the commands and so on:

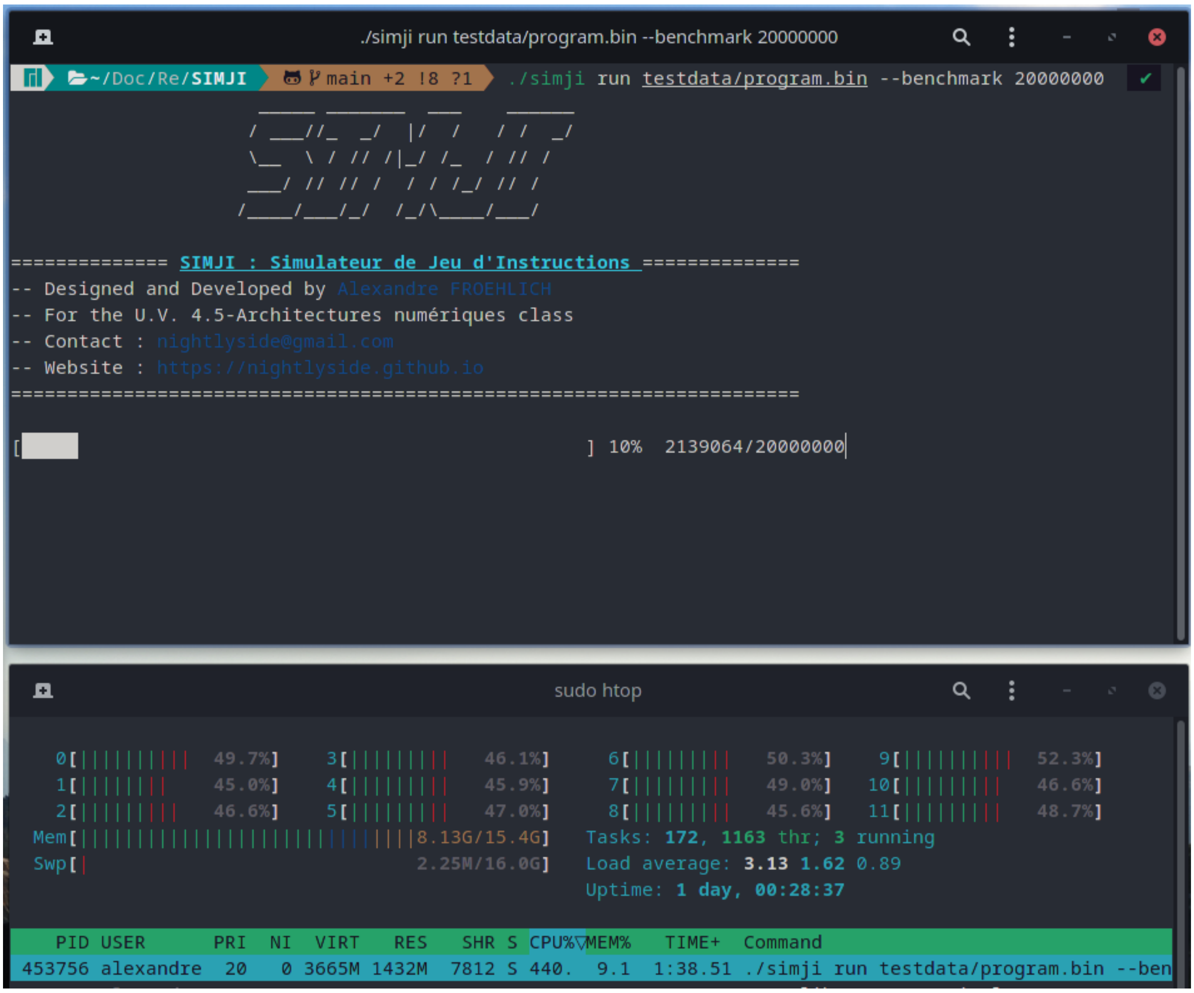


```
~/Doc/Re/SIMJI  main !2 ?1  ./simji run testdata/example.bin -d
           _____ ________  ___       ______
          / ___//  _/  |/  /      / /  _/
          \__ \ / // /|_/ /_  / // /
         ___/ // // /  / / /_/ // /
        /____/___/_/  /_/\____/___/

============== SIMJI : Simulateur de Jeu d'Instructions ==============
-- Designed and Developed by Alexandre FROEHLICH
-- For the U.V. 4.5-Architectures numériques class
-- Contact : nightlyside@gmail.com
-- Website : https://nightlyside.github.io
======================================================================

1:53PM DBG add r0 #15 r1
1:53PM DBG scall 1
[SCALL 1] R1 =>  15
1:53PM DBG stop
~/Doc/Re/SIMJI  main !2 ?1
```

There is the option to make a cpu profile using the `--cpuprofile` flag if you know how to use pprof (from go tools). And finally there is the `--benchmark` flag to run a number of times the program in order to make statistics about the performances of the VM:



```
~/Doc/Re/SIMJI  main +2 !8 ?1  ./simji run testdata/program.bin --benchmark 20000000
           _____ ________  ___       ______
          / ___//  _/  |/  /      / /  _/
          \__ \ / // /|_/ /_  / // /
         ___/ // // /  / / /_/ // /
        /____/___/_/  /_/\____/___/

============== SIMJI : Simulateur de Jeu d'Instructions ==============
-- Designed and Developed by Alexandre FROEHLICH
-- For the U.V. 4.5-Architectures numériques class
-- Contact : nightlyside@gmail.com
-- Website : https://nightlyside.github.io
======================================================================

[████                                      ] 10%  2139064/20000000
```

```
    0[||||||||||   49.7%]   3[|||||||||    46.1%]   6[|||||||||    50.3%]   9[||||||||||   52.3%]
    1[||||||||     45.0%]   4[|||||||||    45.9%]   7[|||||||||    49.0%]  10[|||||||||    46.6%]
    2[|||||||||    46.6%]   5[|||||||||    47.0%]   8[|||||||||    45.6%]  11[|||||||||    48.7%]
  Mem[||||||||||||||||||||||||||||||8.13G/15.4G]   Tasks: 172, 1163 thr; 3 running
  Swp[|                              2.25M/16.0G]   Load average: 3.13 1.62 0.89
                                                   Uptime: 1 day, 00:28:37

    PID USER       PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
 453756 alexandre   20   0 3665M 1432M  7812 S 440.  9.1  1:38.51 ./simji run testdata/program.bin --ben
```

Graphical User Interface

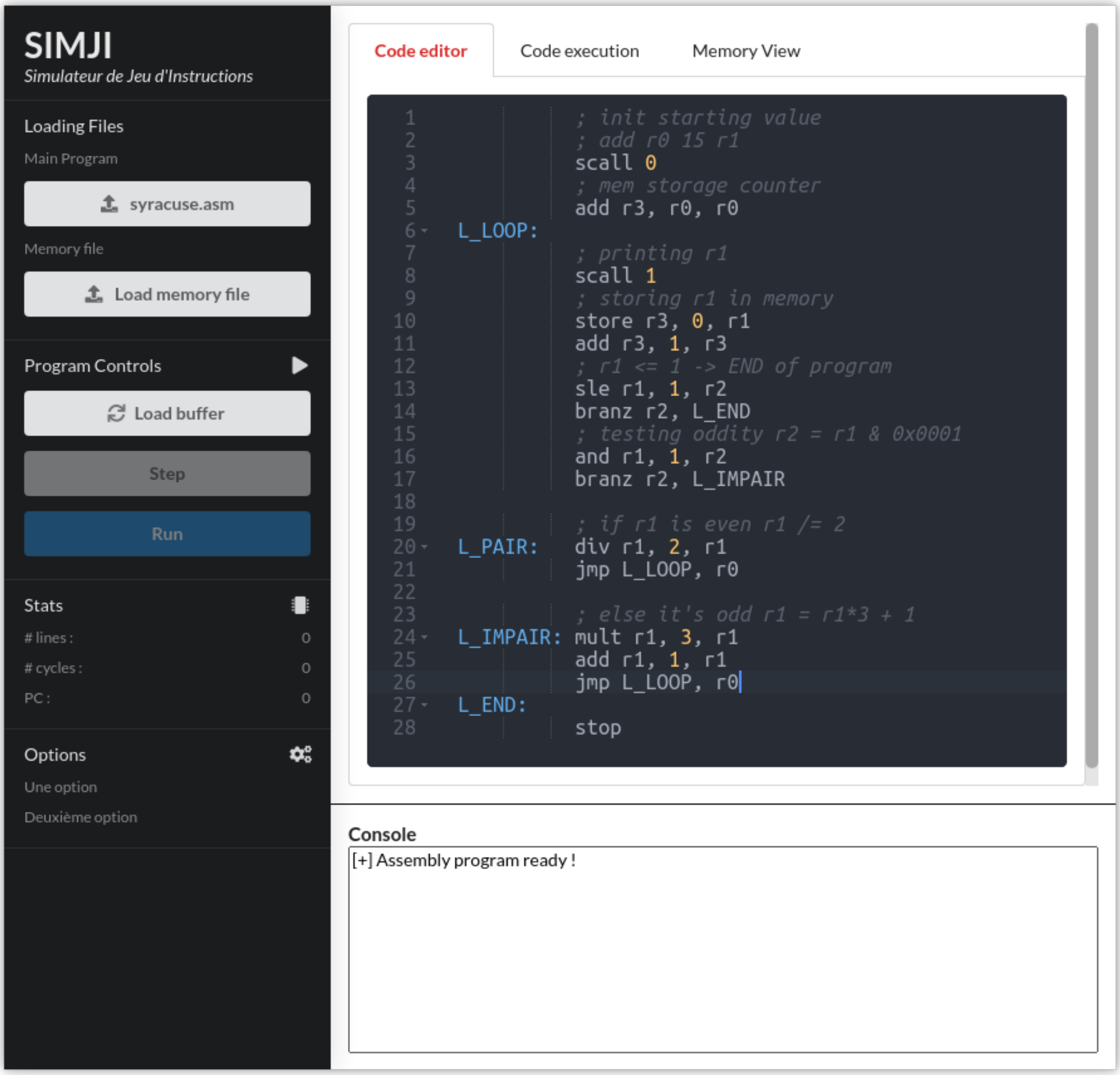
The graphical user interface (GUI) is made using the Lorca library. For this part to work you need to have Chrome or Chromium browser installed. In fact Lorca is working like electron but instead of shipping a whole instance of chromium, it uses the one installed on your system to keep the footprint of the app small.

To start the GUI you need to provide the `--gui` or `-g` flag.

In that case all other flags are ignored :

```
./simji --gui
./simji -g # does the same thing
```

Using this command will brind the GUI :



# lines : 0
# cycles : 0
PC : 0

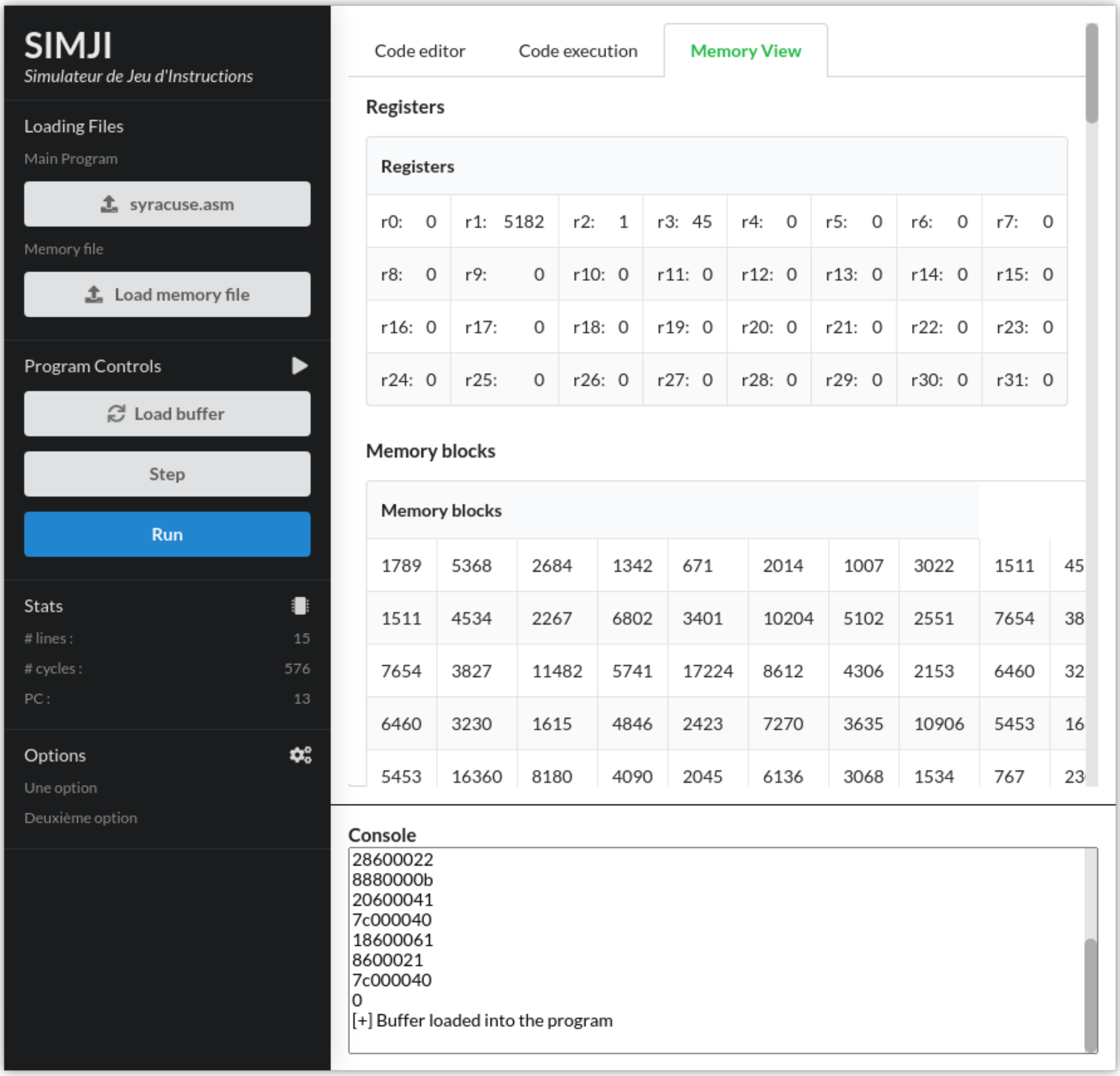
```
          ; init starting value
          ; add r0 15 r1
          scall 0
          ; mem storage counter
          add r3, r0, r0
L_LOOP:
          ; printing r1
          scall 1
          ; storing r1 in memory
          store r3, 0, r1
          add r3, 1, r3
          ; r1 <= 1 -> END of program
          sle r1, 1, r2
          branz r2, L_END
          ; testing oddity r2 = r1 & 0x0001
          and r1, 1, r2
          branz r2, L_IMPAIR

          ; if r1 is even r1 /= 2
L_PAIR:   div r1, 2, r1
          jmp L_LOOP, r0

          ; else it's odd r1 = r1*3 + 1
L_IMPAIR: mult r1, 3, r1
          add r1, 1, r1
          jmp L_LOOP, r0
L_END:
          stop
```

```
[+] Assembly program ready !
```

On the other tabs you can follow the execution of the program step by step and on the last tab you can monitor how each register and each memory block is used.



lines : 15

cycles : 576

PC : 13

Registers

Registers

r0: 0	r1: 5182	r2: 1	r3: 45	r4: 0	r5: 0	r6: 0	r7: 0
r8: 0	r9: 0	r10: 0	r11: 0	r12: 0	r13: 0	r14: 0	r15: 0
r16: 0	r17: 0	r18: 0	r19: 0	r20: 0	r21: 0	r22: 0	r23: 0
r24: 0	r25: 0	r26: 0	r27: 0	r28: 0	r29: 0	r30: 0	r31: 0

Memory blocks

Memory blocks

1789	5368	2684	1342	671	2014	1007	3022	1511	45
1511	4534	2267	6802	3401	10204	5102	2551	7654	38
7654	3827	11482	5741	17224	8612	4306	2153	6460	32
6460	3230	1615	4846	2423	7270	3635	10906	5453	16
5453	16360	8180	4090	2045	6136	3068	1534	767	23

Console

```
28600022
8880000b
20600041
7c000040
18600061
8600021
7c000040
0
[+] Buffer loaded into the program
```

From this interface you can load the program into the buffer (the editor in the GUI) where you can edit the code with syntax highlighting.

Once you are happy with the program click the "Load Buffer" button which will send the program contents to the simulator for execution.

The you can execute the program step by step or do a full run.

Documentation

The documentation is available here: https://pkg.go.dev/github.com/Nightlyside/simji.

You might need to expand the directories in order to follow the documentation for each package making this project.

Project motivations

At first this project was due for a class on Numeric Architectures at the ENSTA Bretagne Engineering School. I then took it further than what was required to pass the topic.

I chose to develop this project using Golang because I wanted to try something new, to be able to put another new tech or language to my range of skills and tools.

I wanted to insist on test coverage for this project as it always seems to be something little worked in a project.

I am quite proud of the result for a first project made in Golang and I will be happy to make something different with this language.