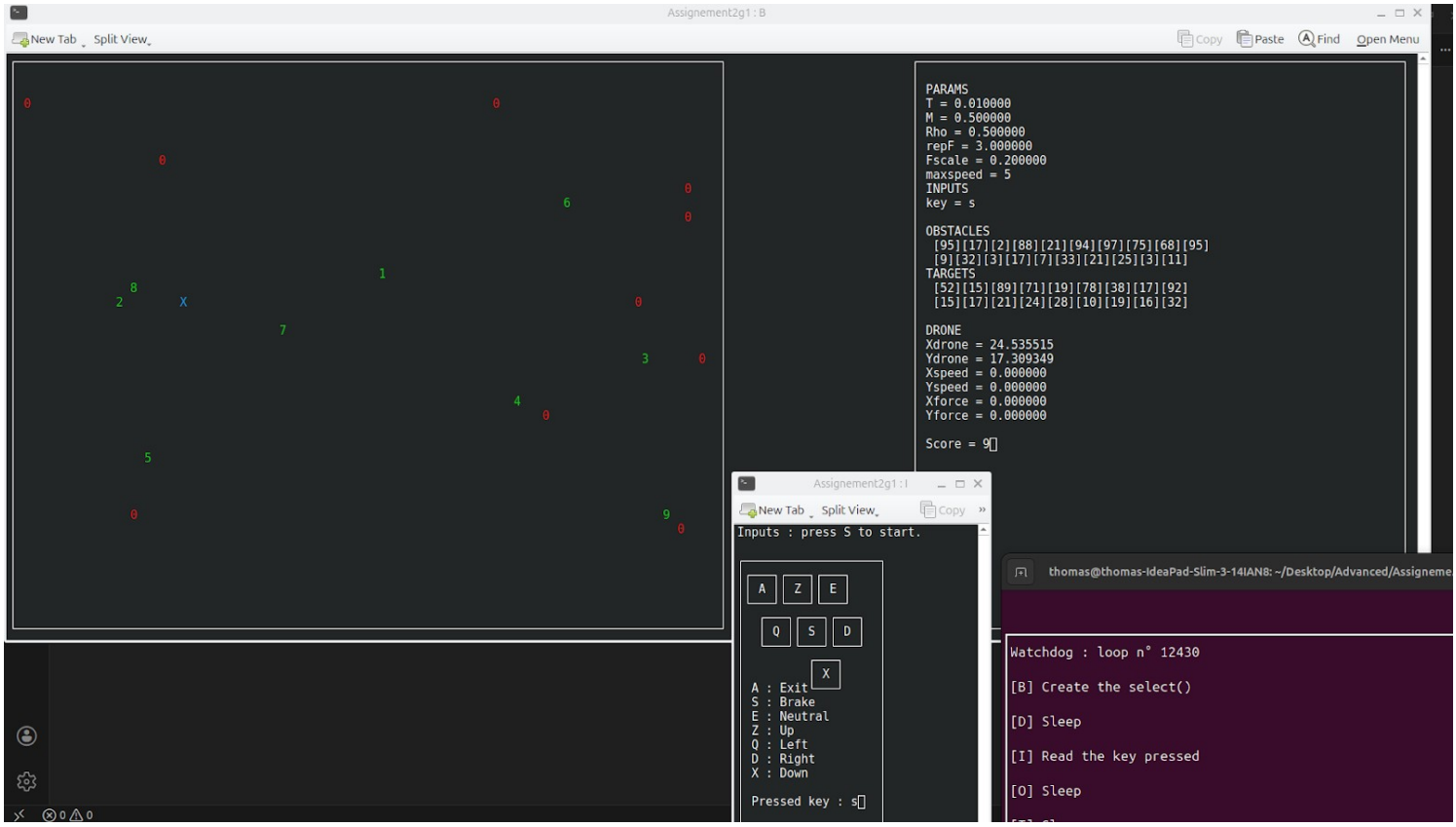
DRONE SIMULATOR PROJECT

Groupe ERASMUS n°1

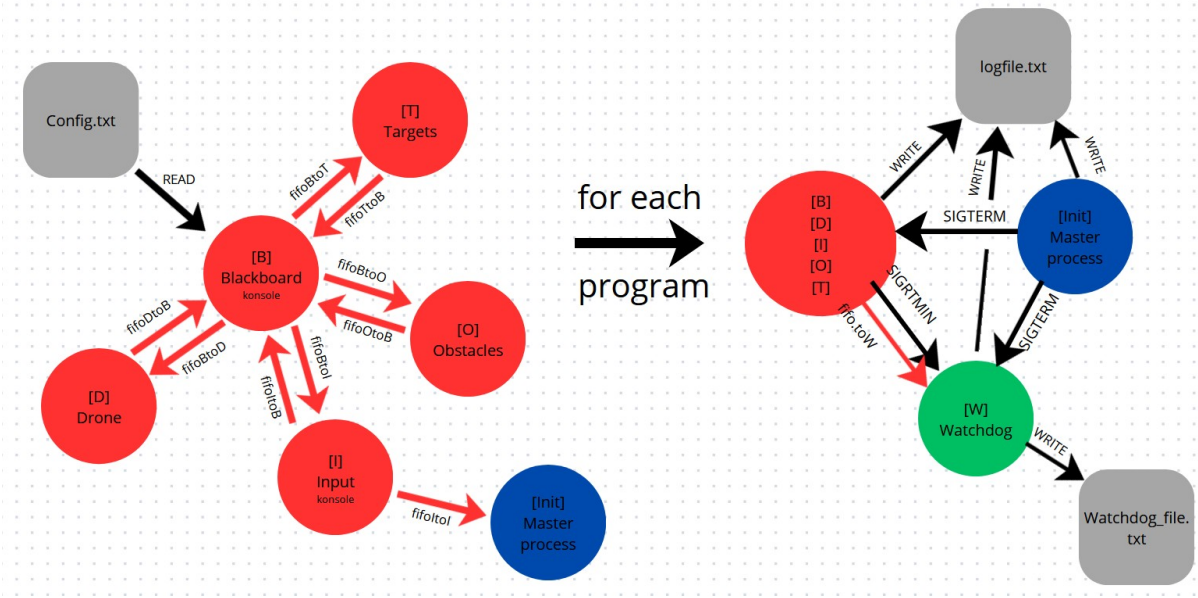
Assignment 2



Project members :

GRESSARD Josselin
PRADINAT Thomas
ROSSIGNOL Victor

1. sketch of the architecture



2. short definitions of all active components :

In bold : New functions of the 2nd assignment

In Italic : New function of the 1st assignment correction

Master process [Init] :

- create all the fifos of this project
- **reset the logfile**
- use the functions fork()+exec() to launch the 5 other programs
- **write all the PID of every program in a pid-file**
- wait a message from [I] when the user want to quit
- send a signal to all the other programs to terminate
- wait for all the programs to terminate
- terminate cleanly
- **write in the logfile (errors, entry into a relevant area of the code)**

Blackboard [B] : Store all the values of the project and communicate it to the different programs.

- endless loop, except if received the terminate signal
- read the content of the file at each iteration and update the values of variables
- use the select() function with all the input fifos
- use a special two-pipe protocol to communicate with the programs
- can send or receive data
- calculate the score of the player
- use ncurses to print the scene *in color* on the screen and additional informations

- terminate cleanly
- **write in the logfile (errors, entry into a relevant area of the code)**
- **read the pid file to find the pid of the Watchdog**
- **periodically send signal and message to watchdog**

Drone [D] : Update the values of the drone representation with interactive dynamic equations.

- endless loop, except if received the terminate signal
- interaction period can be changed through config file
- get value from the blackboard
- use dynamic equations to calculate new values
- send the new values to the blackboard
- terminate cleanly
- **write in the logfile (errors, entry into a relevant area of the code)**
- **read the pid file to find the pid of the Watchdog**
- **periodically send signal and message to watchdog**

Input [I] : Read the key pressed.

- endless loop, except if received the terminate signal
- use ncurses to read the key pressed
- the key must be one of the keys selected for this project (a,z,q,s,..)
- send the key value to the blackboard
- if the key is the one to terminate, send a message to the master process
- terminate cleanly
- **write in the logfile (errors, entry into a relevant area of the code)**
- **read the pid file to find the pid of the Watchdog**
- **periodically send signal and message to watchdog**

Obstacles [O] : Generate obstacles.

- endless loop, except if received the terminate signal
- long period
- generate new coordinates for the obstacles with random values from /dev/urandom
- send the coordinates to the blackboard
- terminate cleanly
- **write in the logfile (errors, entry into a relevant area of the code)**
- **read the pid file to find the pid of the Watchdog**
- **periodically send signal and message to watchdog**

Targets [T] : Generate targets.

- endless loop, except if received the terminate signal
- long period
- generate new coordinates for the targets with random values from /dev/urandom
- send the coordinates to the blackboard
- terminate cleanly
- **write in the logfile (errors, entry into a relevant area of the code)**

- **read the pid file to find the pid of the Watchdog**
- **periodically send signal and message to watchdog**

Watchdog [W] : Watch all the programs.

- **endless loop, except if received the terminate signal**
- **receive signals of the other programs**
- **read the messages from the programs if the message is received**
- **print the messages in the terminal**
- **write the messages in a watchdog-file**
- **terminate cleanly**
- **write in the logfile (errors, entry into relevant area of the code)**

Parameter file [Config.txt] : Store the parameters values for the simulation.

Log-file [logfile.txt] : Get all the log messages of every program of the project.

Watchdog file [Watchdog_file.txt] : Get all the watchdog messages (message received from the other programs and alert messages from the watchdog).

Pid file [PID_file.txt] : Store the Pid numbers of every program.

3. list of components, directories, files

One directory : "Assignement1" :

- *a "bin" directory with all the binary files : Init ; B ; D ; I ; O ; T ; W.*
- *a "config" directory with a text file : Config.txt.*
- ***a "log" directory with three text files : logfile.txt ; Watchdog_file.txt ; PID_file.txt.***
- *a "src" directory with all the code files : Init.c ; B.c ; D.c ; I.c ; O.c ; T.c ; W.c.*
- *a make file : makefile.*
- *the README file : README.pdf.*

4. instructions for installing and running

The project requires the following components to compile and run correctly:

System Requirements

- A Unix/Linux operating system

- Access to the special device file /dev/urandom for random number generation
- POSIX-compliant system calls, including fork(), exec(), and wait()

Libraries

- ncurses (for terminal-based display and input) on Debian/Ubuntu systems, install with: “sudo apt install libncurses5-dev libncursesw5-dev”
- math library (math.h)
- Standard C library (stdio, stdlib, string, etc.)

Build Tools

- gcc
- make

5. operational instructions

Start program :

- make
- *./bin/Init*

Problem with window size :

Sometimes, if the program is started for the first time, it is possible that the konsole window is too small, and the print of the scene is bad. In this case, change the size of the konsole to the maximum (but no full screen), then close the program in the terminal with Ctrl+C and restart it.

Also, sometimes, the obstacles are not generated right -> just re-make the project.

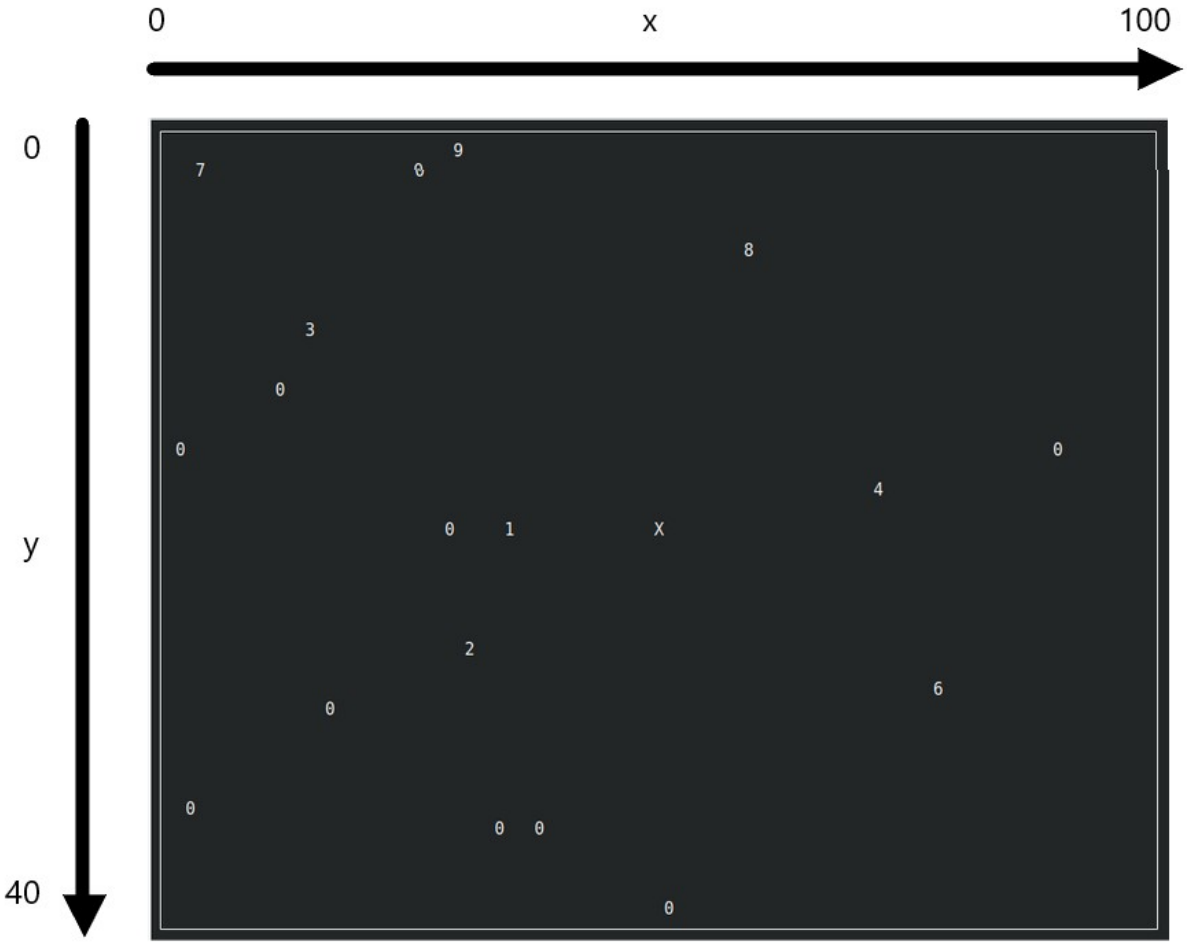
6. useful things

Protocol to communicate with the blackboard :

- The first message is always addressed to the blackboard. It indicates if the other program want to Read or write in the blackboard : ‘r’ or ‘w’
- Then the blackboard answer :
    - the data if the program want to read
    - “ok” if the program wants to write, indicating the blackboard is blocked and is waiting for the values.
- The program then read or write the values

Each program asks to read / asks to write different variables, the buffers are very different between the pipes.

Scene coordinates :

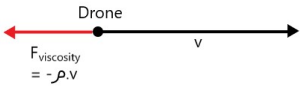


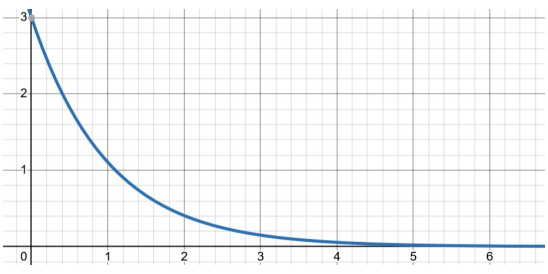
Dynamic calculation of the new coordinates :

2nd Newton law : $\sum F = m.a \quad so \quad a = \frac{1}{m}\sum F$

$\sum F = F_{motor} + F_{viscosity} + F_{obstacles}$

F_{motor} is determined by the keys ; $F_{viscosity} = -\rho.v$





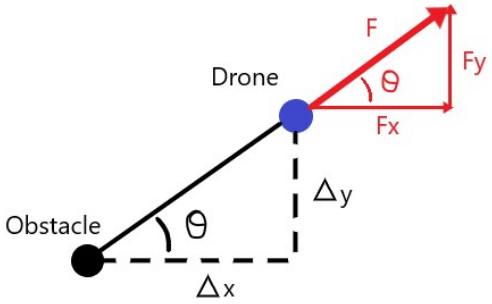
$$F_{obstacle} = \ . \ e^{-\frac{distance}{repulsion\ radius}}$$

$$distance = \sqrt{(x_{drone} - x_{obstacle})^2 + (y_{drone} - y_{obstacle})^2}$$

$$F_x = F_{obstacle} . \cos\theta$$

$$F_y = F_{obstacle} . \sin\theta$$

$$\theta = atan((x_{drone} - x_{obstacle})/(y_{drone} - y_{obstacle}))$$



Correction from the 1st assignment :

- scene printed in color
- 'S' key allows to brake (no more movements) whereas the 'E' key turn the motor force to 0 (but the drone is still moving with inertia and the obstacle forces)
- the different files (code, executables, text) are sorted in different repositories (bin, usr, log, config)
