U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Final Report

CFLAP

(Collaborative Formal Languages and Automata Package)

Laboratório de Computadores 2020/2021

T7G01

Filipe Campos **up201905609**@fe.up.pt

Francisco Cerqueira **up201905337**@fe.up.pt

Index

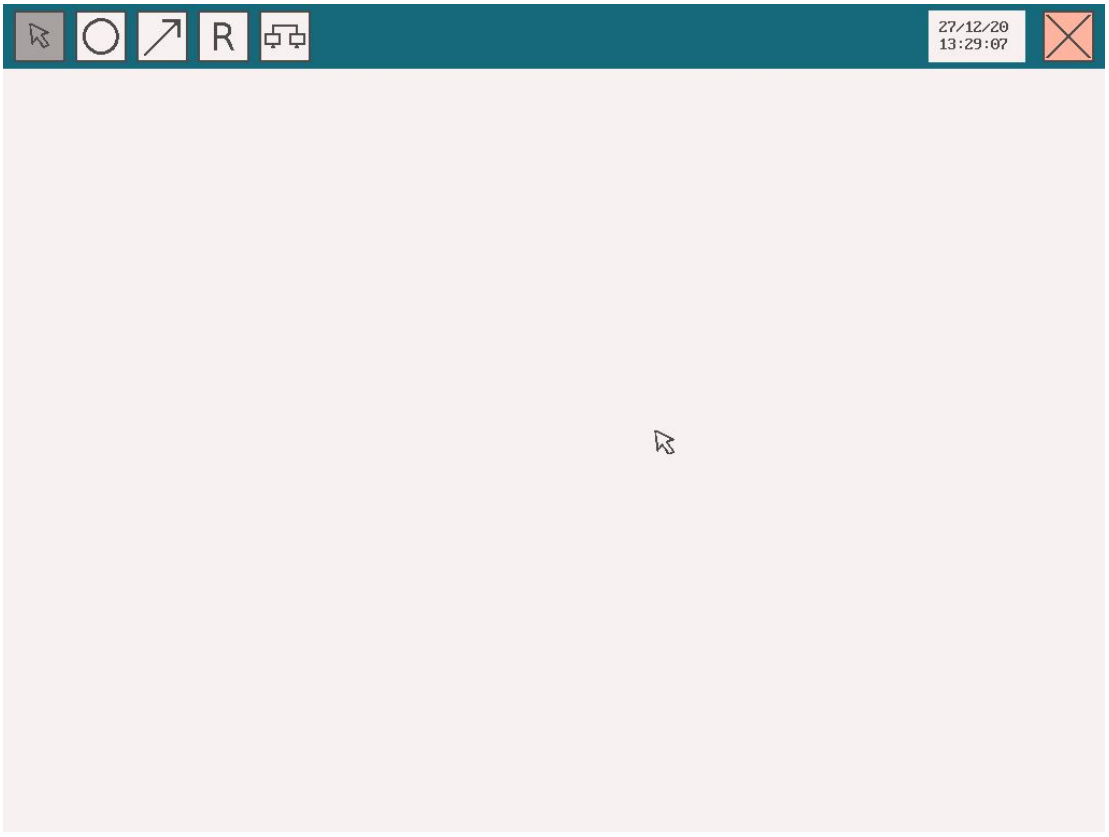
Program Instructions

Title Screen

When starting the program, the user is presented with a title screen, where any key must be pressed in order to continue.

Main Interface

Facing the main interface, the user will have at his disposal an empty canvas and ready to be filled with states and transitions, using buttons or shortcut keys that will be explained later.



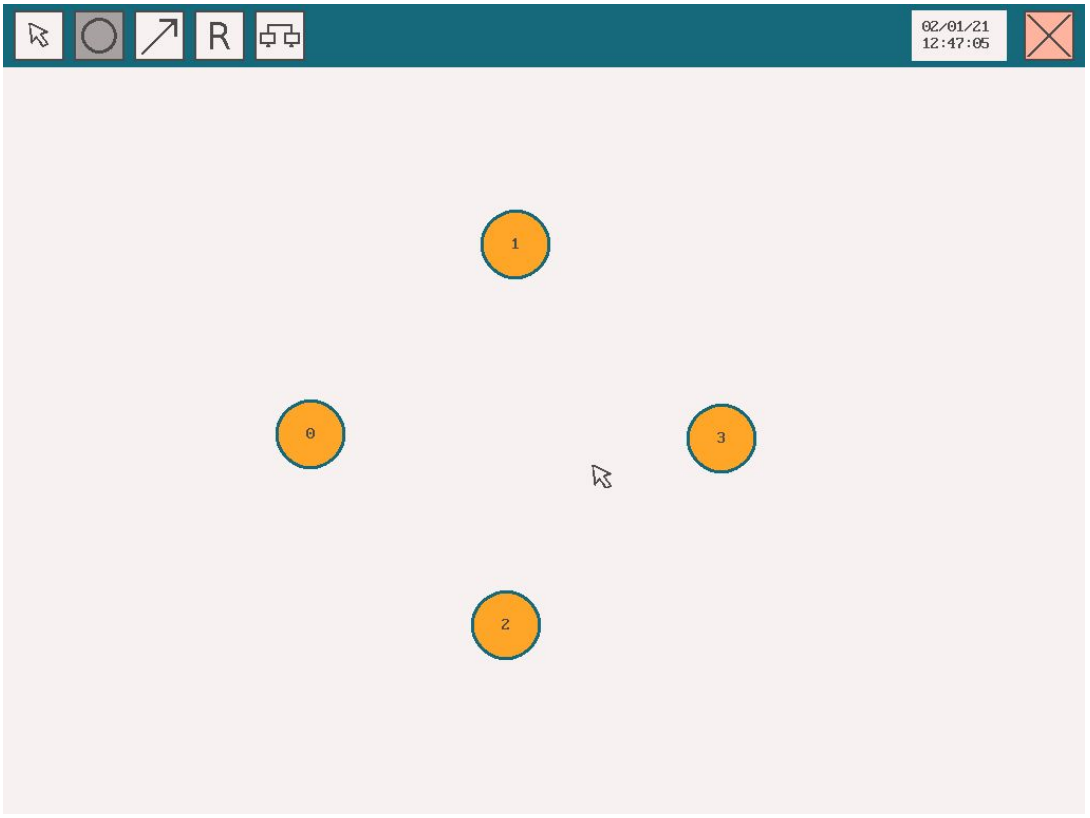
At the top of the screen, a header will contain 5 buttons, a clock and an option to exit. Occasionally and at appropriate times, an input box will also appear. It is in this interface that the drawn state machine will be presented, being as a workspace for the user. The shortcut keys to access the buttons are from 1 to 5, corresponding from leftmost to the rightmost button.

Select Tool

The first button, select button, gives the user the possibility to translocate states and to set / unset initial and final states, using the shortcut keys “I” and “F” respectively, although this can be done in other tools. The user can switch between tools by clicking on them or the corresponding shortcut key.

State Tool

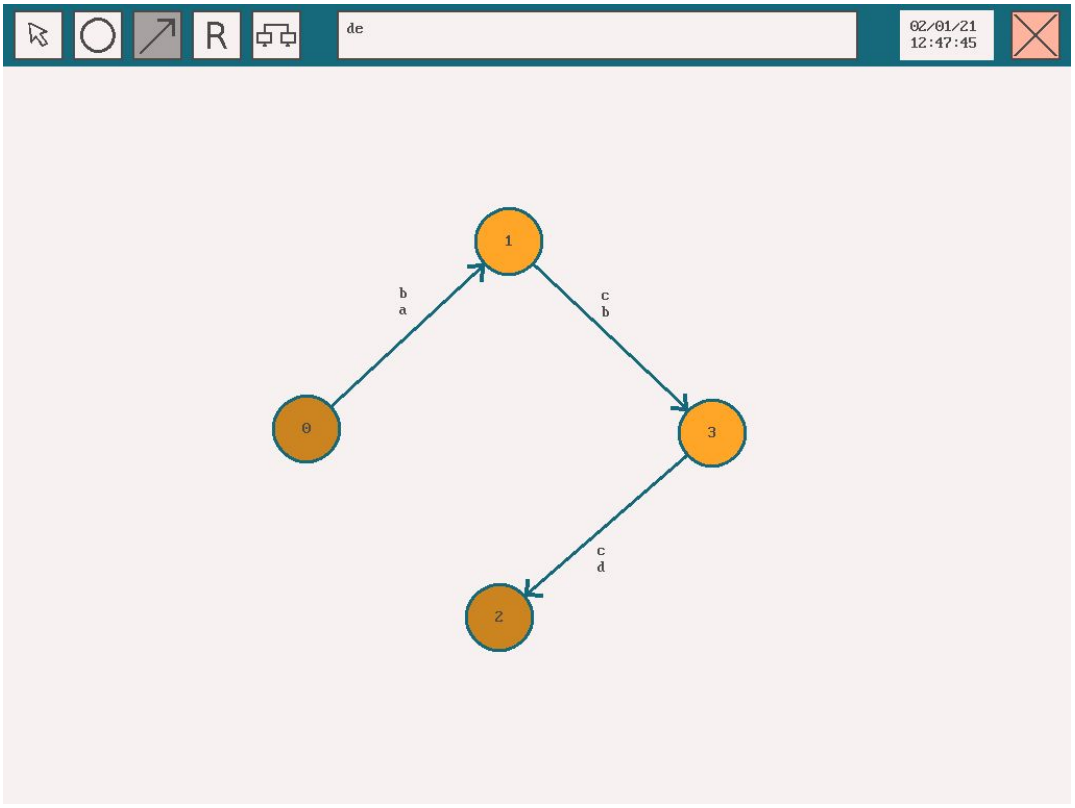
The second button, state button, is used to create and delete states. In order to create a state, the user just needs to click with the left mouse button where he / she wants it to be. The same thing has to be done to delete a state, except the right mouse button should be clicked instead. Set / unset initial and final states is also possible in this tool.



Transition Tool

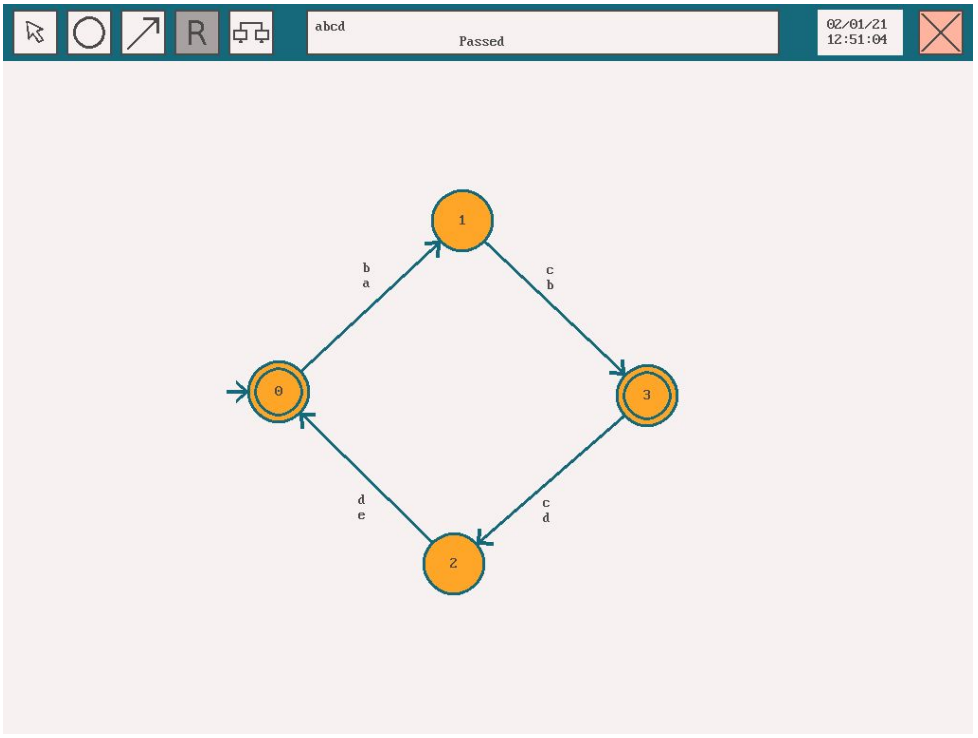
The third button, transition button, is used to create and delete transitions. In order to create a transition, the user needs to drag, with the left mouse button, a line between the states he / she wants the transition to be. The same thing has to be done to delete a state, except the right mouse button should be used instead.

After that, an input box will appear inside the interface header, next to the clock, waiting for the user to write the symbols he / she wants to add, if creating a transition, or erase, if deleting a transition. When the user writes the symbols, the ENTER key must be pressed to submit the input. To leave the box without performing any changes, the ESC key must be used.



Run Tool

The fourth button, run button, allows the user to check if the input string is in the language recognized by the FA. To start the simulation, the user needs to press ENTER after writing the string he / she wants to test.
Then, the program will print one of the following messages: "*Passed*", "*Failed*" or "*Invalid FA*". Finally, the ESC key must be pressed to switch to select tool.



Share Tool

The fifth and final button, share button, serves to establish a serial port connection between two computers. Upon clicking the button, if another computer is connected and running CFLAP, the button will turn yellow to signify that a connection is pending, the other user will also see the button change color on his screen. The second user needs to press the button again to establish the connection, clearing the canvas and turning the share button green. From now on, any change on a given computer will be replicated in the other one. The connection can be terminated by pressing the share button again or by exiting CFLAP.

Disconnected

Connection Pending

Connected

Project State

Used Devices

Device	Description	Mode
Timer	Keep constant framerate.	Interrupt
Keyboard	Use shortcuts and write transitions and strings to be tested.	Interrupt
Mouse	Navigation and utilize the providen tools.	Interrupt
Graphics Card	Display the application interface to the user.	VBE Mode 0x105
Real Time Clock	Show the current date and time and change the color palette at a specific time using alarm.	Interrupt
Serial Port	Allows the application to be synced between two computers.	Interrupt

Timer

This device is used to guarantee a fixed framerate (60 frames per second) [*timer.c*].

Keyboard

The keyboard allows the user to access every tool through a shortcut-key and to type strings or transition symbols onto the input box [*keyboard.c*].

Mouse

This is one of the main devices of the program, as it's used for navigation purposes and to take advantage of the tools provided, which can also be done using the keyboard [*mouse.c*].

Graphics Card

Allowing for the exhibition of the screen, this device is arguably the most important. The selected VBE mode is 0x105, featuring a resolution of 1024x768 and 1 byte per pixel (indexed) [*video_gr.c*].

In order to allow colors that better fit our project, we used a BIOS call to load custom colors onto a given index [*palette.c*].

The use of double buffering by page flipping allows for more fluid movement [*vg_show_buffer:video_gr.c, vbe_set_display_start:bios.c*].

To exhibit ASCII characters on the screen, we use a function called *vg_draw_char* [*video_gr.c*], that uses a pixmap previously loaded by the function *bios_get_font_ptr* [*bios.c*] that through a BIOS call reads the font bitmap used by MINIX's text mode and converts it to a pixmap.

Real Time Clock

The RTC is used to read the current time and date. In our project, it's also used to generate alarms, allowing the user to experience "*night mode*" after a certain time of the day. For demonstration purposes, the RTC is programmed to set alarms every minute [*rtc.c*].

Serial Port

The serial port serves as the connection that can keep two computers synced in real time. CFLAP uses an interrupt driven implementation of Duplex communication using FIFOs, where computers send and receive events through the UART [*uart.c, comm.c*].

Parameters used:

Parameter	**Value**
Bit-Rate	57600
Bits per Character	8
Stop Bits	2
Parity	Even Parity
FIFO	Both enabled
Trigger Level	4

Code Structure and Organization

Bios

This module includes functions responsible for managing BIOS calls, such as *bios_call*, *bios_get_font_ptr*, *vbe_mode_get_info*, *vbe_set_mode*, *vbe_map_vram* [*bios.c*].

Button

This module provides a structure that can be used to interact with buttons, storing useful information (position, icon xpm and hover and selected color) and providing functions such as *create_button*, *change_button_colors* and *draw_button* [*button.c*].

Cflap

This module is responsible for handling events generated by each device in the interrupt handler. It also contains the functions responsible for initializing (*cflap_startup*, *cflap_setup_uart_queue*) and terminating (*cflap_cleanup*) the program.

Comm

This module contains all the structures and functions needed to establish a communication protocol between the two PC's.

Communication Protocol			
Event Type	**Char**	**Data Size (bytes)**	**Data**
INIT_COMM	I	0	
INIT_COMM_AKH	K	0	
INIT_COMM_ACCEPT	A	0	
END_COMM	E	0	
LOCK_STATE	L	1	State number
UNLOCK_STATE	U	1	State number
MOVE_STATE	M	4	XY position
CREATE_STATE	C	4	XY position
DELETE_STATE	D	1	State number
CREATE_TRANSITION	T	1	Transition symbol
DELETE_TRANSITION	t	1	Transition symbol
SET_FINAL_STATE	F	1	State number
SET_INITIAL_STATE	i	1	State number

Example event (each rectangle represents a byte):

M	4	X_LSB	X_MSB	Y_LSB	Y_MSB

This module allows us to encode a events and their data into a queue using the functions *comm_add_event_to_queue(), comm_add_event_with_data_to_queue()* Thereafter the bytes in the queue are sent through the UART and the second computer can read and decode the bytes using *comm_decode_event()*.

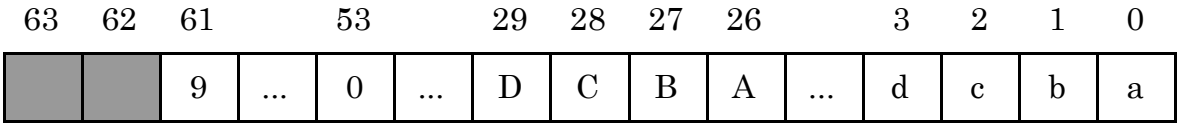
FA

This is probably one of the most important modules, as it's responsible for managing the entire Finite Automata, providing efficient structures to implement it, like a struct to record every state, containing information about if it's active, final or locked and it's position. The module contains also two arrays: the first one used to store all the information about the states and the second one to store the information about the all transitions [*fa.c*].

As this is an important module, it comes with a large amount of functions:

- ➔ Conversion functions, used to work with the transitions information: *fa_get_bit_from_char*, *fa_get_char_from_index*
- ➔ Initialize and terminate the FA: *fa_startup*, *fa_cleanup*
- ➔ Manipulate states: *fa_create_state*, *fa_delete_state*, *fa_set_initial_state*, *fa_toggle_final_state*
- ➔ Manipulate transitions: *fa_create_transition*, *fa_remove_transition*, *fa_remove_all_transitions*
- ➔ Draw on the screen: *fa_draw*, *fa_draw_states*, *fa_draw_transitions*, *fa_draw_self_transition*, *fa_draw_double_transition*, *fa_draw_transition_chars*
- ➔ Manipulate FA: *fa_translocate_state*, *fa_set_state_pos*, *fa_lock_state*, *fa_unlock_state*
- ➔ Simulate strings: *fa_simulate, fa_simulate_from_state*

An entry in the transition table is represented by a 64 bit number, in which each bit denotes a symbol



There's also a macro defining the maximum number of states allowed, which was specifically set to 10.

Kbc

Small interface used to interact with the KBC, only includes two functions *kbc_issue_command* that allows us to write a command to a given KBC port and *kbc_read_outb* that reads the byte present in the out buffer. [*kbc.c*]

Keyboard

This module contains functions and structures that are used to manipulate keyboard keys and to enable / disable interrupts [*keyboard.c*].

Manager

Defined in the file [*manager.c*], this module is the central hub for all the devices being composed of three main functions: *init_devices, interrupt_cycle* and *exit_devices*. It is responsible for setting up every device and subscribing the necessary interrupts, handling every interrupt through the use of device-specific interrupt handlers and channeling the events to the main event handler defined in *cflap.c*. It's also responsible for restoring the proper configuration of each device upon exit.

Mouse

This module contains functions that are used to treat mouse data packets and to enable / disable interrupts [*mouse.c*].

Palette

This module includes all the necessary functions to interface with the indexed mode palette, allowing the application to read/write a color at a given index and set the number of bits per pixel used for each color. Additional functions used to load the specific color palette used by our application are also included (*palette_load_default, palette_set_day_mode, palette_set_night_mode*) [*palette.c*]

Proj

This is the base module of the project, its only purpose is to call the functions *cflap_startup*, *interrupt_cycle* and *cflap_cleanup* [*proj.c*].

Queue

This module provides a structure that can be used to create and manage queues, storing useful information (buffer and its size) and providing functions to interact with it, such as *create_queue*, *queue_add_string*, *queue_add_char*, *queue_read_string* and *queue_read_char* [*queue.c*].

RTC

This module contains functions that are used to manage RTC registers, such as reading / writing bytes (*rtc_read_register*, *rtc_set_register*), read date and time (*rtc_read_date_to_string*, *rtc_read_time_to_string*), set alarms (*rtc_set_alarm*) and to enable / disable interrupts [*rtc.c*].

Sprite

This module includes all the necessary functions to interact with sprites, storing its *position*, *size*, *speed* and *xpm image*. Besides that, contains functions that can be used to create and manage them, such as *create_sprite*, *animate_sprite* and *translocate_sprite* [*sprite.c*].

Timer

This module contains functions and structures that are used to manipulate the timer device and to enable / disable interrupts [*timer.c*].

UART

This module contains the necessary functions to interact with the UART, including but not limited to setup communication parameters, read / write to registers and subscribe / unsubscribe interrupts [*uart.c*].

Utils

This module contains small utility functions that are used along the project and facilitate its organization and readability. Some of these functions are *check_rectangle_collision*, *check_circle_collision* and *count_set_bits*.

We also decided to include a structure that helped us create and manage the devices, storing useful information, such as *bit_no*, *irq_set* and *hook_id*, and a function to create it (*create_device*) [*utils.c*].

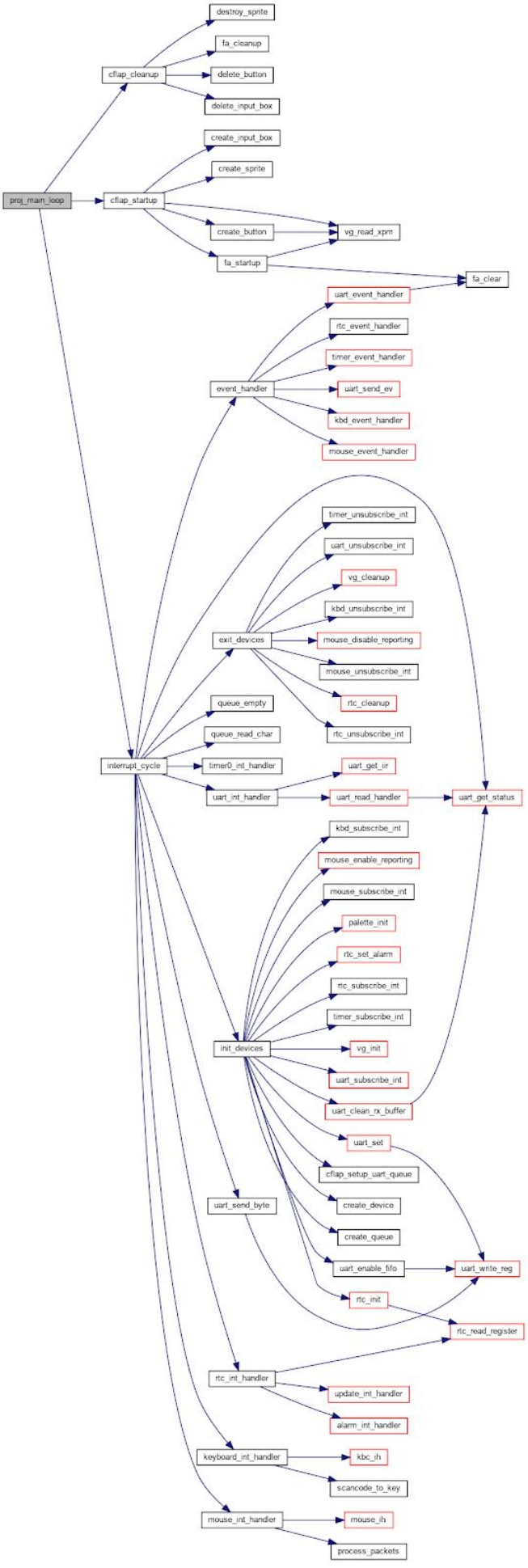
Video Graphics

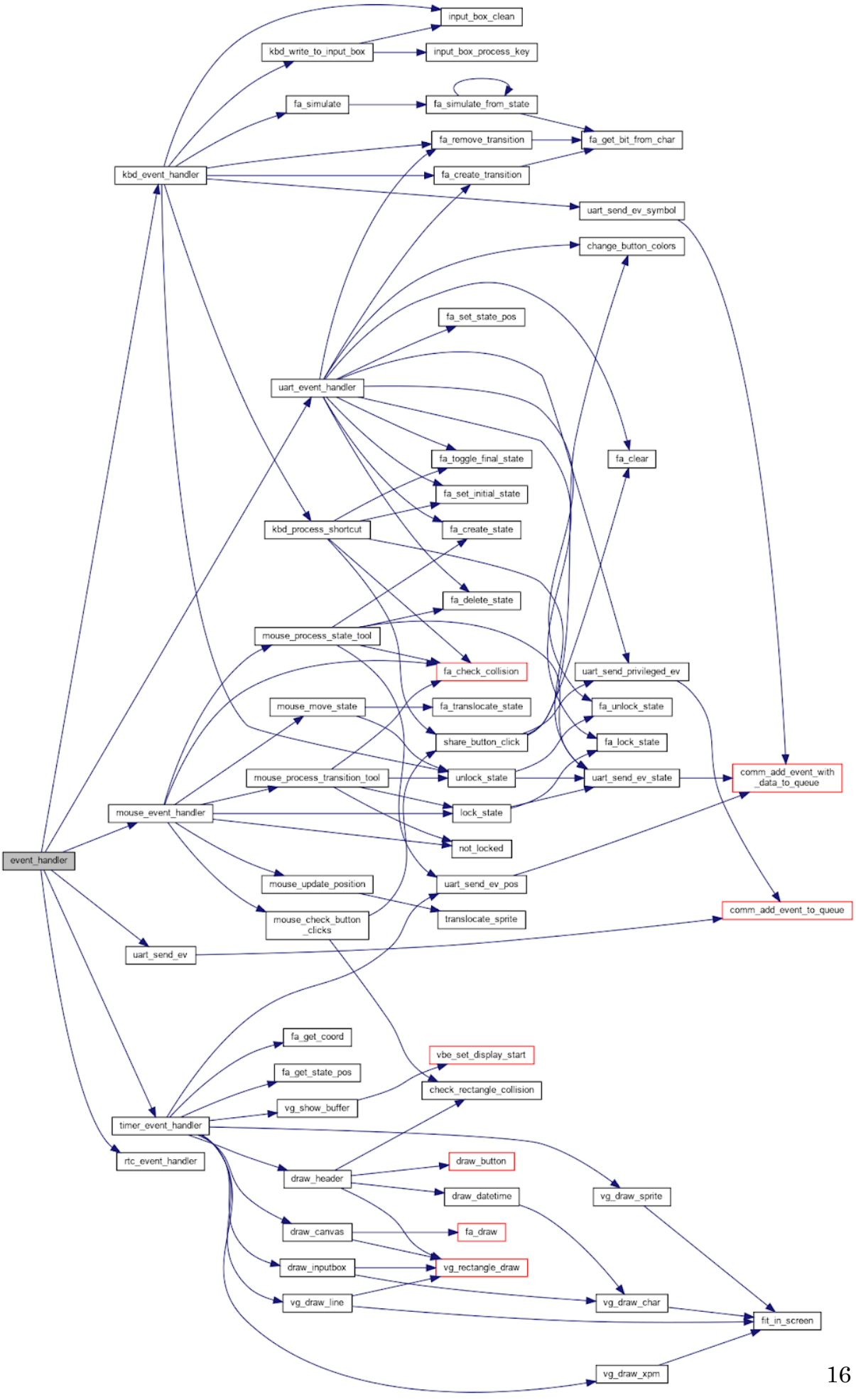
This module contains the necessary functions that are used to display the intended information on the screen. It contains essentially functions to draw lines and pixmaps, such as *vg_draw_xpm*, *vg_draw_line*, *vg_draw_arrow* and some others that were useful in our project [*video_gr.c*].

Work Distribution

Module	**Relative Weight**	
Button	4.5%	
Bios	6.5%	
Cflap	10.5%	
Comm	7%	
FA	10.5%	
Kbc	1%	
Keyboard	7.5%	
Manager	9.5%	
Mouse	7.5%	
Palette	4.5%	
Proj	0.2%	
Queue	2%	
RTC	4%	
Sprite	2%	
Timer	3%	
UART	7.3%	
Utils	2%	
Video Graphics	10.5%	
Total	100%	
	50%	50%
	Filipe	**Francisco**

Call Graph


destroy_sprite
fa_cleanup
cflap_cleanup
delete_button
delete_input_box
create_input_box
create_sprite
proj_main_loop
cflap_startup
create_button
vg_read_xpm
fa_startup
fa_clear
uart_event_handler
rtc_event_handler
timer_event_handler
event_handler
uart_send_ev
kbd_event_handler
mouse_event_handler
timer_unsubscribe_int
uart_unsubscribe_int
vg_cleanup
kbd_unsubscribe_int
exit_devices
mouse_disable_reporting
mouse_unsubscribe_int
rtc_cleanup
queue_empty
rtc_unsubscribe_int
queue_read_char
interrupt_cycle
timer0_int_handler
uart_get_ir
uart_int_handler
uart_read_handler
uart_get_status
kbd_subscribe_int
mouse_enable_reporting
mouse_subscribe_int
palette_init
rtc_set_alarm
rtc_subscribe_int
timer_subscribe_int
init_devices
vg_init
uart_subscribe_int
uart_clean_rx_buffer
uart_set
cflap_setup_uart_queue
uart_send_byte
create_device
create_queue
uart_enable_fifo
uart_write_reg
rtc_init
rtc_read_register
rtc_int_handler
update_int_handler
alarm_int_handler
keyboard_int_handler
kbc_ih
scancode_to_key
mouse_int_handler
mouse_ih
process_packets

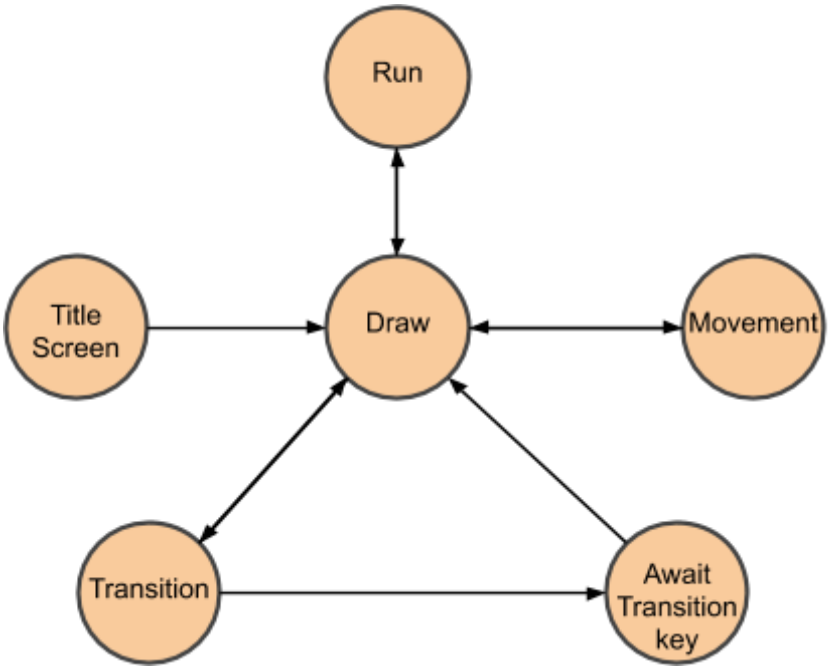

input_box_clean
kbd_write_to_input_box
input_box_process_key
fa_simulate
fa_simulate_from_state
fa_remove_transition
fa_get_bit_from_char
kbd_event_handler
fa_create_transition
uart_send_ev_symbol
change_button_colors
fa_set_state_pos
uart_event_handler
fa_toggle_final_state
fa_clear
fa_set_initial_state
kbd_process_shortcut
fa_create_state
fa_delete_state
mouse_process_state_tool
fa_check_collision
uart_send_privileged_ev
mouse_move_state
fa_translocate_state
fa_unlock_state
share_button_click
fa_lock_state
mouse_process_transition_tool
unlock_state
uart_send_ev_state
comm_add_event_with_data_to_queue
mouse_event_handler
lock_state
not_locked
event_handler
mouse_update_position
uart_send_ev_pos
mouse_check_button_clicks
translocate_sprite
comm_add_event_to_queue
uart_send_ev
fa_get_coord
vbe_set_display_start
fa_get_state_pos
check_rectangle_collision
vg_show_buffer
timer_event_handler
rtc_event_handler
draw_header
draw_button
draw_datetime
vg_draw_sprite
draw_canvas
fa_draw
draw_inputbox
vg_rectangle_draw
vg_draw_line
vg_draw_char
fit_in_screen
vg_draw_xpm

Implementation Details

As requested, we used several layers of code, so we could separate high-level from low-level code, leading to a more organized, readable and reusable code.

We designed our application to be event driven, where a device interrupt produces an event that's sent to the main event handler and from there distributed to more specific event handlers [*cflap.c*]

We also used a small state machine to better organize our application [*cflap.c*].



The function:

void bios_get_font_ptr(uint32_t foreground_color, uint32_t background_color, unsigned bytes_per_pixel)*

allows us to obtain the font bitmap used by MINIX (or more precisely, by MINIX's BIOS) through the use of the BIOS function 0x1130, and then convert it to a pixel map of our choice. This required a lot of tinkering to get working since it's not covered by the course. Because of that, we were required to use parentheses around the *sys_int86* call (*"(sys_int86)(r)")* to bypass a lcf safety measure and also a lot of time was spent trudging through MINIX's documentation (which is quite lacking) to discover a function capable of reading contents of the physical address returned by the bios call (*sys_readbios()*) [*bios.c*].

Additionally, the BIOS calls used to read/write palette colors [*palette.c*] required additional research because they're not mentioned in the class material.

We would like to give emphasis to the function:

vg_draw_line(int16_t xi, int16_t yi, int16_t xf, int16_t yf, uint32_t color, uint16_t thickness)

which draws a line between two given points, with the color and thickness given as parameters. This function allowed us to draw oblique lines, which turned out to be a difficulty at the beginning of the project. We would like to emphasize that we do not resort to any algorithm or research, and any similarity with any existing algorithm is pure coincidence.

Last but not least, we want to clarify that our program makes only one call to the function *driver_receiver*, which happens to be in the file *manager.c*, as we were told that this part would be important in the evaluation.

Conclusion

With regard to our opinion, we think that LCOM is an interesting curricular unity that goes over captivating topics although it suffers due to badly organized study and reference material, like handouts.

Additionally, LCF could benefit greatly from more explicit error messages, that would ease the frustration of dealing with seemingly random bugs.

In conclusion, we would like to say that both members of the group committed themselves equally, having completed all personal goals and foreseen by the course, since the project served as a great way of learning.

Appendix

To run our project simply type

make clean && make

lcom_run proj

inside the **src** folder