U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Department of Informatics Enginering

LCOM: BreakPong

Computer Laboratory Report

Authors:

António Rama (up202108801)

José Veiga (up202108753)

Luís Jesus (up202108683)

Pedro Marcelino (up202108754)

May 5, 2023

Contents

1. Game Concept

Our project proposal, BreakPong, consists in a game based on the Atari's best sellers: Breakout and Pong. Both games share some basic common elements: the paddle and the ball. The main difference between them is that the first has walls of bricks that are meant to be destroyed using the ball, and the other one is a typical multiplayer game in which each player has its own paddle and has to score in its oponent goal.

The idea was to join these two games together and make two different modes: singleplayer mode would be based in Breakout, and multiplayer mode would be based in Pong.

Thus, in singleplayer mode the player controls a paddle either with the mouse or with the keyboard. The objective is to destroy the blocks in the screen with a ball without letting it drop below the paddle. The game is made up of three levels and there are three types of blocks, in this order, from higher to lower resistance (hp): red blocks, green blocks and blue blocks. According to their strength, they will successfully lose hp every time th eball touches them. Consequently, their colors and cracks will change throughout the game. The player will have, every 10 seconds, an extremely thirsty opportunity to save his life by catching one of three power ups: na extra life, na extra ball (that can fall below the paddle without losing a life) and a red brick destroyer.

The final score is based on the number of bricks destroyed at the end (one point per block hp).

On the other hand, the multiplayer mode requires 2 players. In this game mode each person controls a padel and has the objective of making the ball touch the other end of the screen and keeping it away of his side.

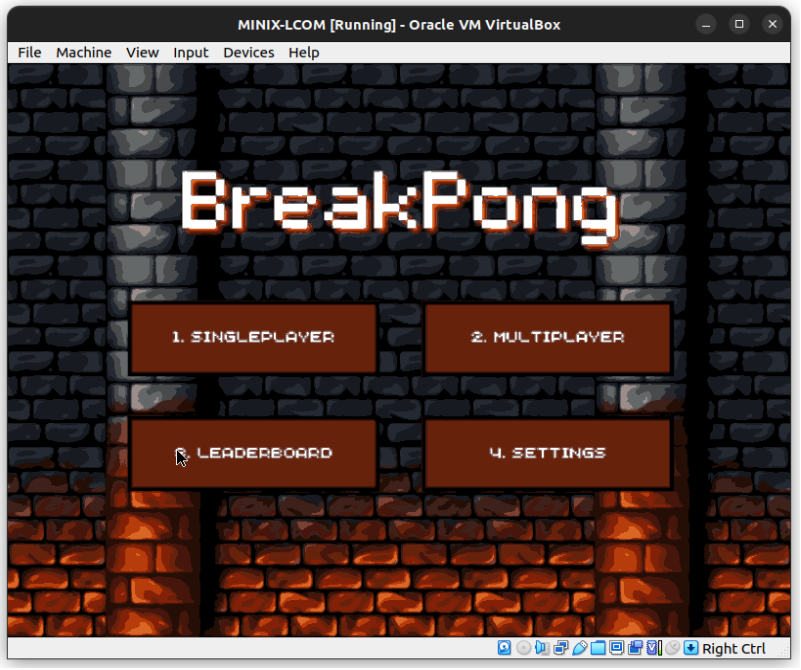
2. User Instructions

2.1. Menu Screen

When the user first opens the game, he is presente with the Menu screen. The user has 4 options to choose:

- SinglePlayer : starts the game (singleplayer mode);
- Multiplayer : starts the game (multiplayer mode);
- Settings : opens the settings screen;
- Leaderboard : opens a table where the last best 5 scores are displayed with information like level, points, hour and date.

The player can always exit the game in every state just by pressing the keyboard key [ESC].



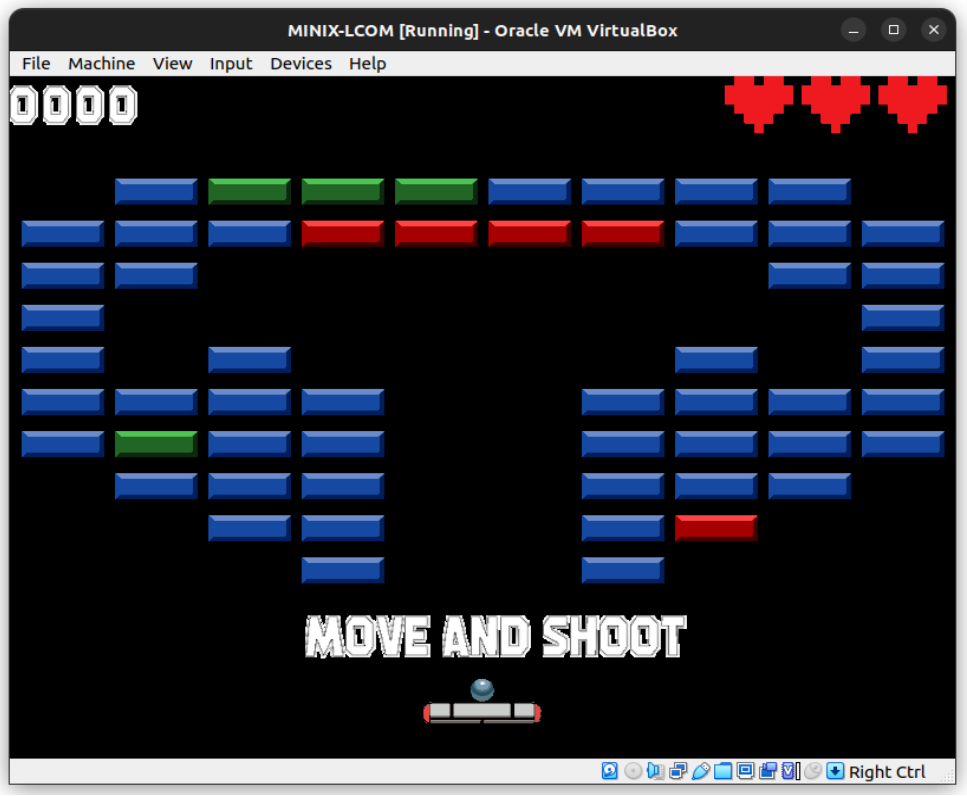
2.2. Singleplayer mode

Once the user starts the game in Singleplayer mode, he will face a bunch of objects/sprites that correspond to the main elements of the game:

- Player's lives;
- Score;
- Paddle (controlled by the user);
- Bricks (with different colors depending on the level);
- Ball.

The user controls the paddle, moving it left and right with the mouse or keyboard, depending on the controller device he has defined in the game settings. However, mouse controller is the default option.

The game starts with a ball above the paddle, as shown in the picture bellow. He can move the paddle in order to choose the place from where he wants to release the ball, by right clicking or pressing [SPACE], according to the chosen device.

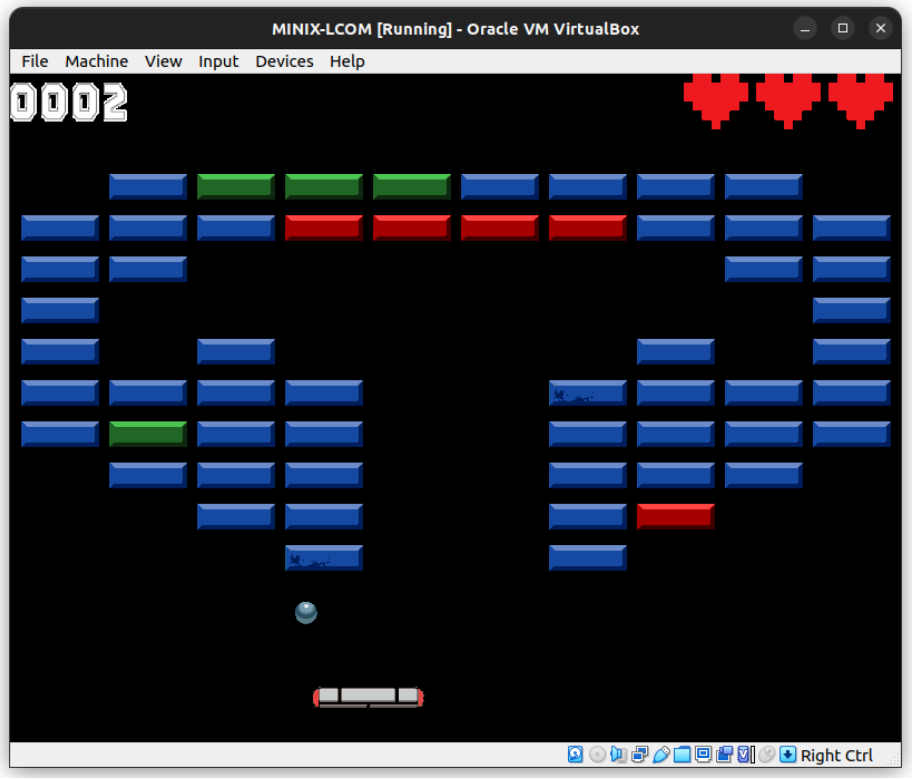


The score is determined by the number of bricks destroyed since the beggining of the game. However, even though red blocks are stronger than the other ones, they have "more points to offer" to the player. Bricks have the following HP (each HP subtracted to the block in each ball touch is added to the player's score):

- Red bricks: 9 HP;
- Green bricks: 6 HP;
- Blue bricks: 2 HP.

The player starts the game with 3 lives. When the ball touches the bottom border of the screen, the user loses a life.

The game ends when the player reaches 0 lives or completes the third level (he is returned to the main menu). When this happens, his score may be added to the leaderboard menu, if it is among the better 5 scores ever. In case the user has not completed the third level, still has lives, and has destroyed the level's bricks, a new set of bricks – of the next level – will be shown and he will continue storing points.



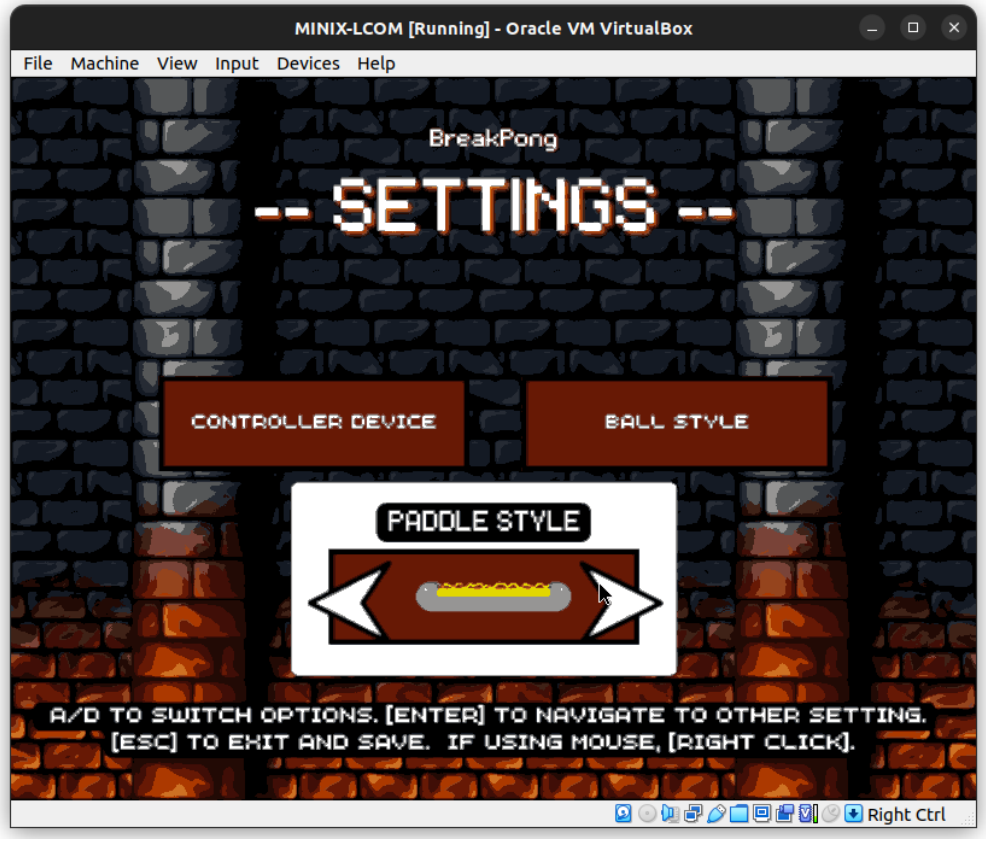
2.3. Settings Screen

When the user gets in the settings' screen, he is able to modify three game parameters:

- **Change how to control the paddle:** the player can decide how to control the game paddle. He can either use the keyboard, by pressing A and D to move the paddle continuously, pressing [SPACE] to release the ball, or the mouse, by moving it and right clicking to shoot.
- **The ball style:** there are two ball styles available and the use is presented with the option to choose its favorite.
- **The paddle style:** just like the ball, the paddle i salso a very "stylish" componente in the game and the player can choose one of the two options available.

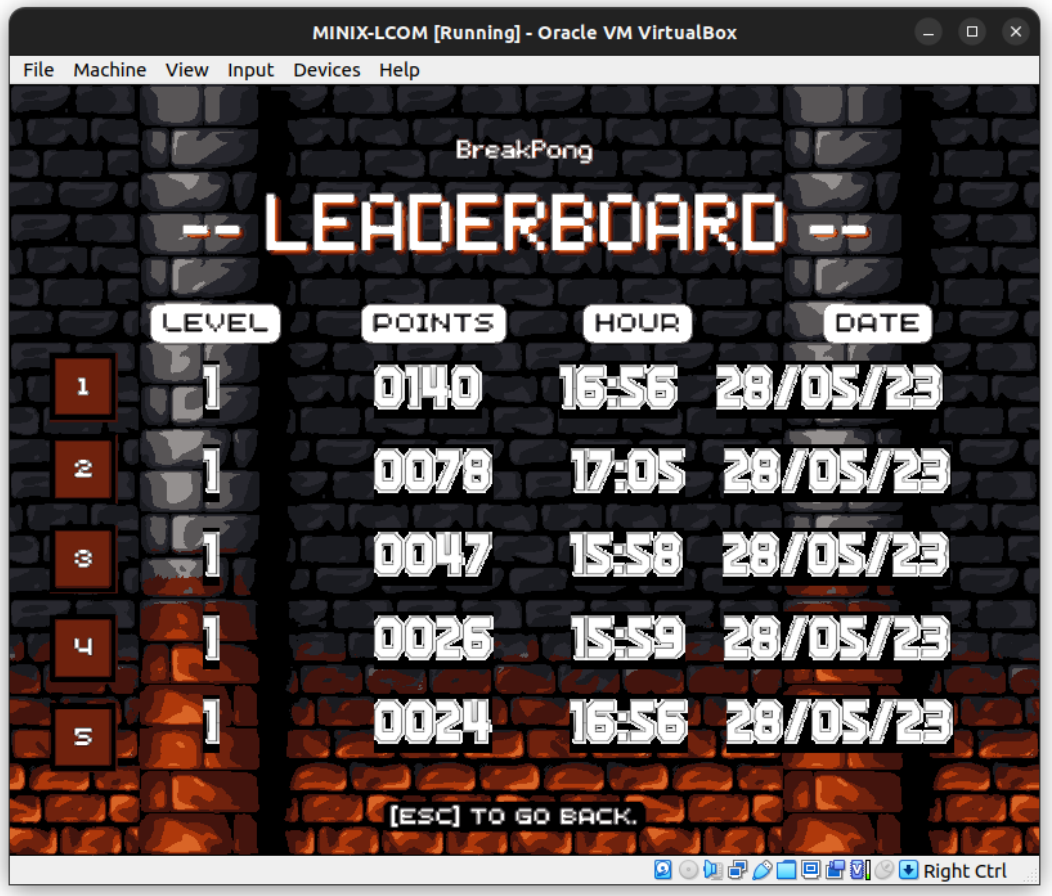
In the bottom of the screen there is message explaining how to use it depending on the controller device. The navigation and interaction with each controller device is pretty much the same in every game screen.

The user can then choose accordingly, from the options presented to him, by reading the instructions in the bottom of this screen.



2.4. Leaderboard

As already explained, after each game ends, if the score is high enough, the player's last score can be part of the Leaderboard TOP 5 table, that has the information displayed in the image below:



2.5. Multiplayer mode

This game mode wasn’t fully implemented due to technical difficulties with the configuration of the Serial Port.

However it was inteded to be an arena where 2 players had the chance to control one paddle each, preventing the ball from touching their border of the screen.

The score would be defined by the number of times they managed to make the ball touch the other player’s border.

3. Project Status

3.1. Used I/O devices

Device	Usage	Interrupts
Timer	Frame Rate. Power ups appearing every 10 seconds.	Yes
Keyboard	Control the paddle / Navigate the menus.	Yes
Mouse	Control the paddle / Navigate the menus.	Yes
Graphics	Display the game graphics.	Yes
RTC	Display the date of the record in the leaderboard.	No
Serial Port	Transfer of information in the multiplayer mode	Yes*

3.2. Timer

Timer's main function is to allow the frames to be displayed at a regular rate (60 Hz), defined at the begining of the project. Since the timer is being used with interrupts, the frequency of the interrupts is set to the desired frequency (60 interrupts every second).

Every screen displaying related funtion is then called whenever a timer interrupt occurs keeping a fixed frame rate.

Other very important feature that uses the timer is the Power Ups. As already refered, the player will have the opportunity to catch a power up that will fall in the screen, every 10 seconds. Timer's interrupt handler (counter incrementer) is used in order to calculate the time that has passed.

3.3. Keyboard

The keyboard is also being used with interrupts. This device's purpose is to read and process user input to be used in game or in the menus.

When the user presses a key, it causes an interrupt to occur, which calls the update_keyboard_state() function that, besides calling the interrupt handler to process the key pressed into usable data, chooses what to do with that information depending on the current App **state**.

It is importante to mention that, sometimes, keyboard is used associated with the timer, for exemple, when it comes to moving the paddle using it. To allow a smooth movement of the paddle if the user keeps the key pressed, we check if there's an A/D key break code being pressed. This happens because the timer interrupts more frequently than the keyboard, and that's why this solution was taken as the perfect symbiosis to solve this little problema that we were facing.

3.4. Mouse

The mouse is used to parse the user input into usable data. In the project, we created a data structure that stores the information relative to the mouse position and buttons state (left button pressed, right button press, x position and y position), to be used later in functions and to display the mouse sprite more easily in the menus.

When there is mouse activity, the generated interrupt calls the interrupt handler. The interrupt handler then processes the information associated to the interrupt parsing the mouse packets and "converting" them in real interactions with the game (by knowing its position in the screen and the position of our game's elements in the screen).

3.5. Graphics

Another crucial I/O device used in this project is the graphics card. It is responsible for displaying all the information in the screen, in a user friendly way, with the highest efficiency possible. The chosen solution was page-flipping (with double buffering):

In the beggining of the program, we set the video mode we thought was more adequate – 0x115 (800px X 600 px). We then mapped double the physical video memory space required by that mode.

Having setup everything, all we were required to do was to change the video memory "start" at every timer interrupt (when a new frame is desired to be drawn in the screen), which already are set to happen at a fixed frame rate.

It is also important to note that, it is crucial to keep track of the current buffer of the video memory being used for displaying, in order to write to the "hidden" one.

3.6. Real-Time Clock (RTC)

Since the RTC is only used to display the date of the records registered in the leaderboard, we decided not to use interrupts in our approach. Therefore, we only request the Real-Time Clock to update its information when we need instead of constantly getting new information that would "go to waste".

Thus, we ask the RTC to update the time everytime the player loses a game, because we need to check if its score is going to be added to the leaderboard, and if so, we will have to display the time as it is one of the features and the purpose of the RTC in this project.

3.7. Serial Port

The objective of the Serial Port in this project was to be able to implement the multiplayer version of the game (based on Pong).

It was implemented using interrupts setup to only occur on **new available data**. To simplify, it was decided that in the multiplayer mode, the only device controller available would be the mouse. Therefore, after a mouse interrupt a packet of bytes containing the host's mouse.

This transfer would then cause an interrupt to occur in the guest's device. The serial port interrupt handler would then read the bytes sent and process them accordingly, enabling the draw_guest_paddle() function to draw the second player's paddle accordingly.

However, due to lack of time, the implementation wasn’t fully implemented leaving many flaws unsolved, making the multiplayer mode unplayable.

4. Code Organization/Structure

Even though C is not na objected-oriented programming language, we tried to follow a Model-View-Controll Architecture as much as we could. Consequently, this code is organizaed in a way that each module works as independently as possible, also following an event-driven design.

This not only makes the code simple to understand and debug, but also allows further development with relative ease, making it easier to implemente new features or polish already existing ones. It has the following three fundamental packages inside **/src**:

- Controller;
- Model;
- View.

4.1. Controller

This package includes, as its name suggests, all the controllers of the game. Consequently, it includes the device drivers of the used devices (timer, keyboard, mouse, graphics, RTC and Serial Port) as well as files/controllers of game elements (ball.h, paddle.h, powerup.h, etc). Considering this, in order to separate concerns, we decided to create directories inside this package: a *game* directory and another one for each device (*timer, keyboard_mouse, graphics, rtc and serialPort).*

4.1.1. timer.c

This is basically the resolution of the Lab 2 which contains functions (and constants in the header file) such as *timer_subscribe_int(), timer_set_frequency(...), timer_ih()*, etc. As it is widely used to keep a fixed frame rate, as well as allowing power ups spawning, it has signficiant relevance in the project.

Weight: 2%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.1.2. kbc.c

This contains functions that are common to the keyboard and the mouse (because they both use KBC). Here, there are functions that work for both the devices such as: *int read_output(uint8_t port, uint8_t* byte, uint8_t mouse), int write_kbc_command(uint8_t port, uint8_t commandByte),* etc.

Weight: 13%;

Contributors: Luís Jesus and Pedro Marcelino.

4.1.3. mouse.c

This module contains functions that are strictly related with mouse functionality. However, it mostly uses functions already defined in the previous module, which made it easy to implement.

Weight: 7%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.1.4. keyboard.c

This is exactly the same situation as the previous one.

Weight: 7%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.1.5. rtc.c

This module has all the functions related to the RTC and are not very complex, because our use of this module was also very simple (we only needed it to know the current time). There are some functions, being *rtc_config()* and *rtc_update()* the most relevant one.

Weight: 1%;

Contributors: Luís Jesus.

4.1.6. graphics.c

graphics.c includes a bunch of very relevant functions because it is the main device that allows us the interaction with the player (without showing visual elements, that would be very difficult).

Here, we have very relevant functions such as *int (vg_set_start) ()*, for example, that manages the buffers and page flipping, and also other core ones, such as *int (set_frame_buffer)(uint16_t mode)*, to allocate the physical vídeo memory.

Weight: 9%;

Contributors: Luís Jesus e Pedro Marcelino.

4.1.7. serialport.c

serialport.c includes a bunch of functions related to this device, very similar to other devices.

Here, we have very relevant functions such as *int (sp_subscribe_int) (uint8_t *bit_no)* and *void (sp_ih)().* We did not finish the practical implementation of it, so it doesn't have much relevance, even thought the effort.

Weight: 1%;

Contributors: Pedro Marcelino.

4.1.8. queue.c

Contains a queue implementation that is used by the Serial Port.

Weight: 1%;

Contributors: Pedro Marcelino.

4.1.9. utils.c

This one was done during the first lab and has very simple functions that are commonly used in the device drivers, for example, *int (util_sys_inb)(int port, uint8_t *value).*

Weight: 1%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.1.10. ball.c

Here are functions to control the ball upon some user input and other ones related to this object in the game, for example: *void change_ball_pos(Ball* ball), void collision_board(Ball* ball), void collision_brick(Ball* ball, Brick* brick)*, etc.

Basically, where most part of the collisions are handled.

Weight: 7%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.1.11. brick.c

In this one we can see only two functions related to the bricks, because the collisions are in charge of the ball: *void decrease_hp(Brick* brick)* and *void destroy_red_bricks(),* that is used by a power up.

Weight: 1%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.1.12. paddle.c

This module handles all the responsibilites of the paddle such as its movement (either with mouse or keyboard) and boundaries with the walls. Relevant functions, such as: *void mouse_move_paddle(Paddle* paddle)* and *void keyboard_move_paddle(Paddle* paddle)*.

Weight: 6%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.1.13. powerup.c

In this one, all the logic of the power up, such as their random spawning position and it movement along the screen, is implemented in a couple of functions, such as: *void drop_random_powerup(), void disable_powerup(PowerUp *powerup) and void activate_powerup(PowerUp *powerup)*.

Weight: 2%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.1.14. game.c

This is basically the main cluster of the game where all the dynamics of the game is managed (level, points, etc). It has very importante functions such as *update_timer_state(), update_keyboard_state()* and *update_mouse_state(),* that call other functions defined in the modules refered above. It plays a very importante role in the game logic.

Weight: 16%;

Contributors: Luís Jesus, Pedro Marcelino.

4.1.15. leaderboard.c

This module handles all the necessary treatment of data that has to be done in order to read the leaderboard.txt file and insert new records. This is importante for the leaderboard menu that is accessible through the menu. Contains functions such as: *void leaderboard_fill()* and *void add_leaderboard_record(int level, int score, rtc_time_info *time_info)*.

Weight: 2%;

Contributors: Luís Jesus.

4.2. Model

This package basically contains classes (structs) and types definitions, such as Ball, Paddle, Brick, etc. Contains other folders, for example, with the different XPMs.

4.2.1. Controller

Contains setup and teardown functions, such as *setup_bricks(char* map[12]), setup_ball(Ball* ball), destroy_sprites(),* etc. It basically defines and sets up the main entities of the game.

Weight: 2%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.2.2. sprite.c

It has two sprite related functions such as: *Sprite *create_sprite_xpm(xpm_map_t sprite)* and *void destroy_sprite(Sprite *sprite)*.

Weight: 1%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.2.3. font.c

It has some sprites of the alphabet and numbers, as well as functions to set up and destroy these specific sprites that are used, for example, in the leaderboard (to write text).

Weight: 1%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.3. View: view.c

This package (that also contains this single file), is responsible for drawing everything taking into account the App State and the contexts, as well as the order of drawing and screen clearing, for example. Some relevant functions are: *int draw_sprite_xpm(Sprite *sprite, int x, int y), draw_mouse(), draw_ball()*, etc.

Weight: 15%;

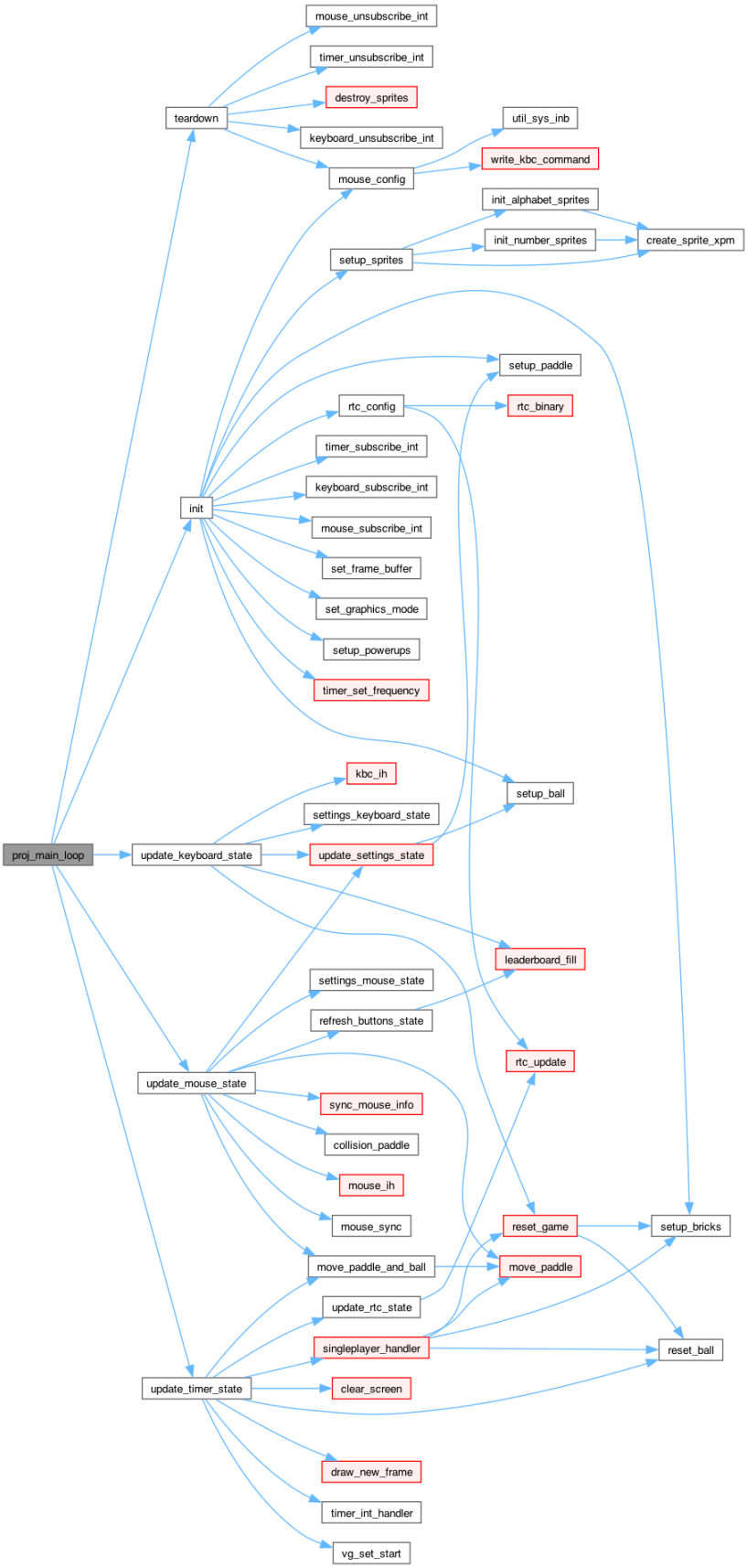
Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.

4.4. main.c

This is where the project main loop is and calls functions across the entire directory structure. It is very important, essentially for interrupts enabling and disabling, to allow the interaction between the devices (by using interruptions) with the user.

Weight: 5%;

Contributors: Luís Jesus, Pedro Marcelino, António Rama e José Veiga.


mouse_unsubscribe_int
timer_unsubscribe_int
destroy_sprites
teardown
keyboard_unsubscribe_int
util_sys_inb
write_kbc_command
mouse_config
init_alphabet_sprites
init_number_sprites
create_sprite_xpm
setup_sprites
setup_paddle
rtc_config
rtc_binary
timer_subscribe_int
keyboard_subscribe_int
init
mouse_subscribe_int
set_frame_buffer
set_graphics_mode
setup_powerups
timer_set_frequency
kbc_ih
setup_ball
settings_keyboard_state
proj_main_loop
update_keyboard_state
update_settings_state
leaderboard_fill
settings_mouse_state
refresh_buttons_state
rtc_update
update_mouse_state
sync_mouse_info
collision_paddle
mouse_ih
mouse_sync
reset_game
setup_bricks
move_paddle_and_ball
move_paddle
update_rtc_state
singleplayer_handler
reset_ball
update_timer_state
clear_screen
draw_new_frame
timer_int_handler
vg_set_start

5. Implementation Details

During this project we tried as much as we could to separate functions and files in the most logical way possible. Also, one of our main goals was to implement a Model-View-Controller approach.

Considering this, we built the game around the logical of states, such as *enum AppState* (START, INIT, SINGLEPLAYER, etc). Also, it's relevant to mention that all the state changer functions are in the controller package, in order to separate concerns and have a more object-oriented perspective.

The implementation of the device drivers was quite straightforward because it did not involve a big effort as they were already done during the labs. However, we still had to adjust some things, for example, to speed up the sprites drawing and clearing screen, for example, by using C functions *memcpy* and *memset.*

Of course, the RTC was not as easy because it was not taught in the lessons, however, it was not a big deal. The big deal was the Serial Port that we couldn't manage to finish, even though we put some days on it.

6. Conclusion

This project was product of several weeks of work towards learning the basics of I/O devices programming, which not only helped us improve our coding skills in C but also gave us a tour around how the peripherals work.

Far from being an easy task, it forced us to rethink ways of approaching such large project. Besides, it also imposed a way of working with the bare minimum of information making us mostly rely on documentation, which is most commonly the last resource.

It's not possible to go through such a challenging project without noting the hardest parts like the uart whose implementation took several days, and the mouse and video memory optimization process.

Overall it was great project and we learned a lot from it as it showed us a whole different perspective about the interface between hardware and software.