(This is a direct translation of the original, there might be some errors and formatting issues, for clarity, consult the original.)

Faculty of Engineering of the University of Porto

Final project

Computer Laboratory (LCOM)

Class 4 - Group 4

Made by:

Beatriz Bernardo	up202206097
Diana Nunes	up202208247
Sofia Goncalves	up202205020
Teresa Mascarenhas	up202206828

Index

1. Introduction

Lab-rinth is a 2D game where the main objective is to escape from a maze within a stipulated time. The player uses a flashlight to visualize the map, navigating the corridors and trying to find buttons where you can cross, in order to unlock certain doors scattered around the path, while looking for the exit.

The game offers two game modes: singleplayer and cooperative.

In single-player mode, the player must explore the maze alone, using the flashlight to light the way and find the exit before time runs out exhaust.

In co-op mode, two players play simultaneously on the same maze. They must cooperate to find the exit, but with the tension additional to compete to see who gets there first. Both players share the same goal, but must balance cooperation with competition to ensure they can get out of the maze before their opponent.

2. Instructions for using the program

2.1 Menu

Home menu

When the game starts, the initial menu is displayed. It has 2 buttons: one to exit the game (“Quit”) and another to go to the menu that allows you to choose the level. desired (“Start”). These buttons must be pressed with the mouse.

Fig 1. Home menu

Levels menu

When we press “Start” on the main menu, the menu for choosing the desired level is displayed. This menu has 3 buttons: one to select level 1 (“Level 1”), another for level 2 (“Level 2”) and finally for the multiplayer level (“Co-op”). These buttons must be pressed with the mouse, similar to those on the home menu, and direct the user to the chosen level.

Fig 2. Levels menu

2.2 Levels

In this game, it is possible to select which of the 3 existing levels the user want to try. Each of these levels has a different maze with different solutions.

Additionally, the colors of the maze change depending on the time of day. The maze each level has 3 color palettes: the first lasts from 6 am to 2 pm, the second from 2pm to 8pm and the third from 8pm to 6am.

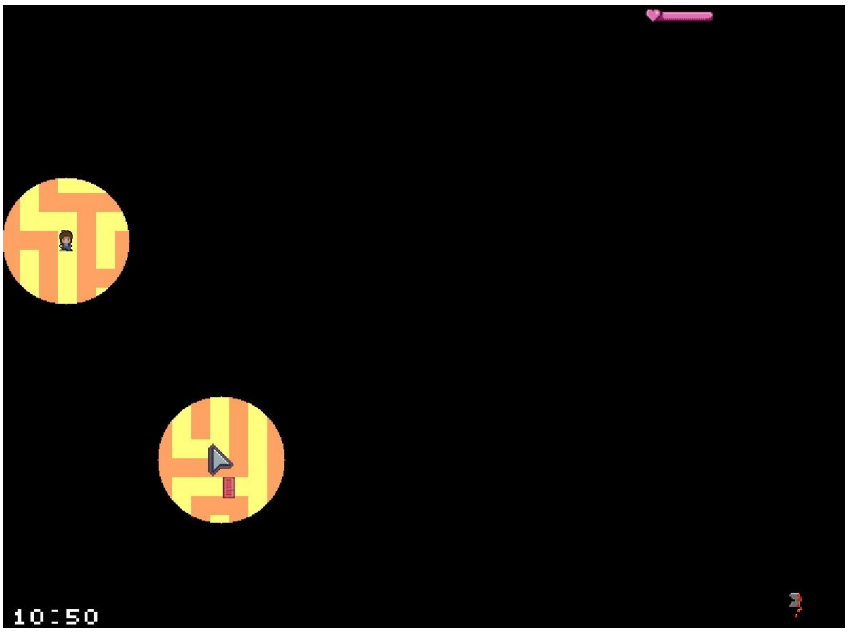


Fig 3. Level 1 (First palette)

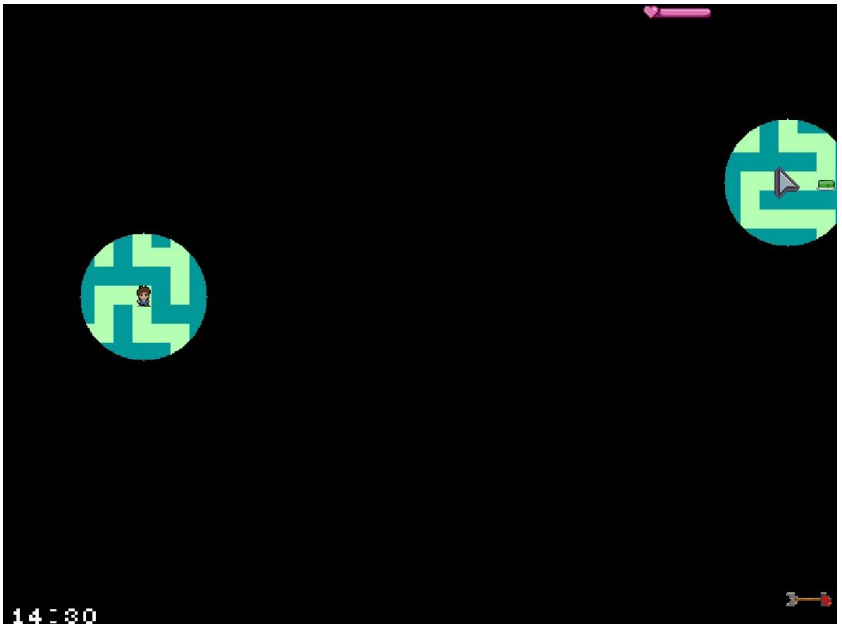


Fig 4. Level 1 (Second palette)

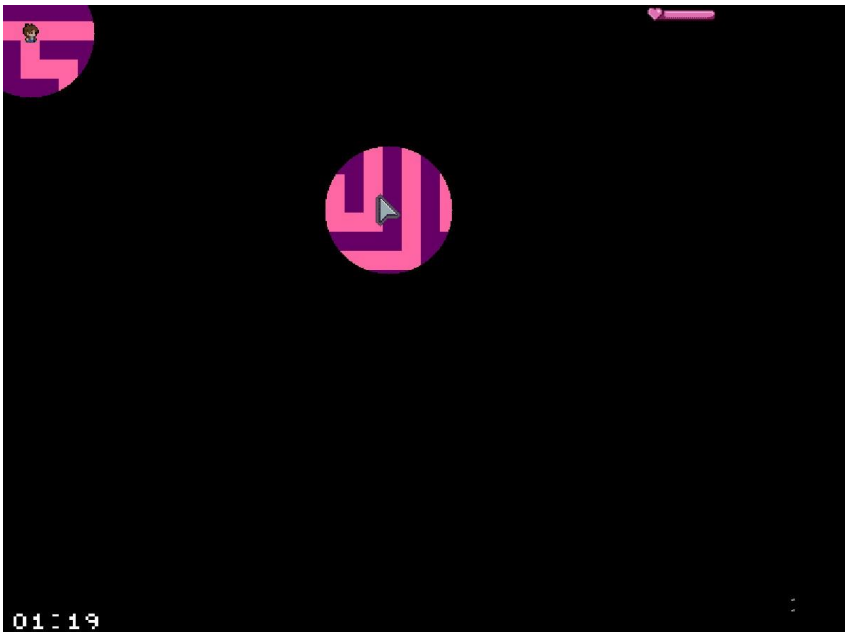


Fig 5. Level 1 (Third palette)

2.3 Maze

In the maze, the player sprite starts out situated in the top left corner and the user must move it with the keyboard to try to reach the exit that is located in the lower right corner. To move up the key used is 'W', to move down 'S', to the left 'A' and to the right 'D'.

The user can only see a small part of the maze, so to to be able to see more you can move the mouse that has a flashlight. In addition, must discover buttons that you can walk through, to unlock certain doors that meet on the way.

Finally, if you fail to reach the exit before time runs out, you are directed to the game over screen but if you succeed you go to the victory screen.

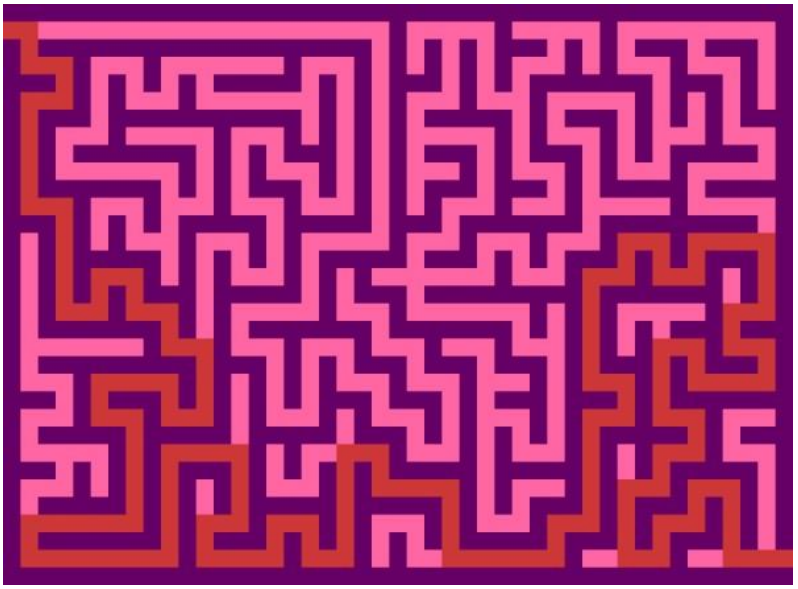


Fig 6. Maze solution 1

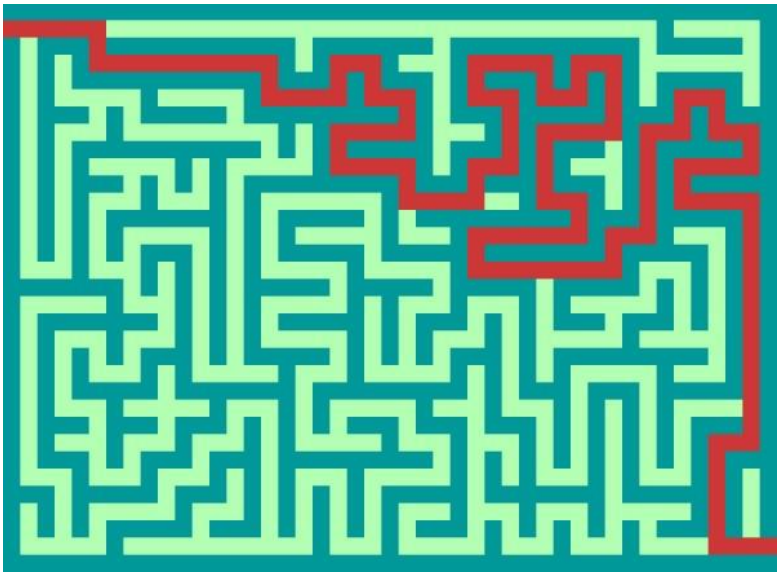


Fig 7. Maze Solution 2

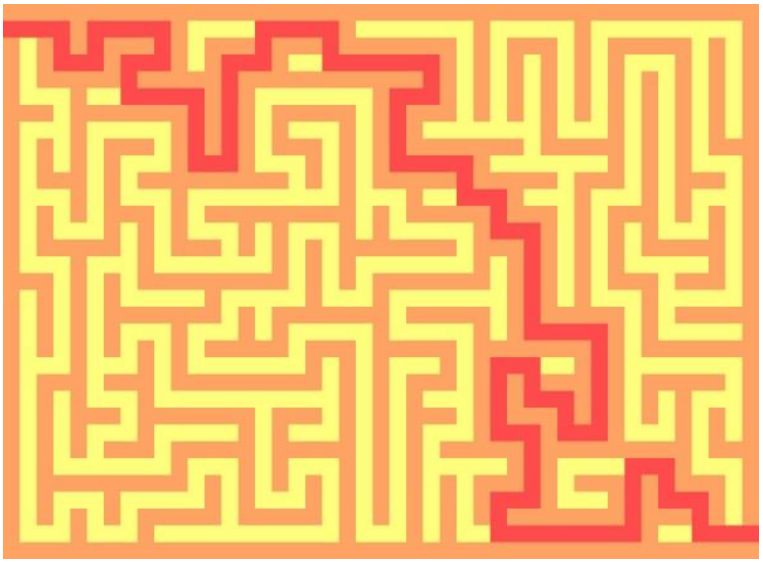


Fig 8. Cooperative Maze Solution

2.4 Victory screen

This screen appears if the user manages to reach the exit before the time runs out. game ends, showing the time it took to complete the maze. It should press the “Esc” key to exit the game.

Fig 9. Victory screen

2.5 Game Over Screen

This screen appears if the game time is up and the user has not yet managed to reach the exit. You must press the “Esc” key to exit the game.

Fig 10. Game Over Screen

3. Project status

3.1 Features

- Menu exploration and button selection - mouse and video card
- Moving a character in the game - keyboard and video card
- Maze Discovery - Keyboard, Mouse and Video Card
- Activating buttons and opening doors in the maze, as well as its graphical representation - keyboard and video card
- Arrow indicating the exit from the maze - timer and video card
- Graphical representation of the time remaining to finish the game - timer and video card

- Changing the time and colors of the maze - rtc and video card
- Communication between two players - serial port
- Display hours and minutes on the game screen - rtc and video card
- Display of the time it took the user to complete the maze - timer and video card.

3.2 Device table

Device	What is it for?	Interrupt / Polling
Timer	Determine the duration of the game and to regulate the change of certain sprites to over time	Interruption
Keyboard	Move the characters that are found in the game with the keys A, W, S and D	Interruption
Mouse	Select options from main menu and levels menu	Interruption
Video Card	Display the entire interface game and menus	None
RTC	Change the game theme according to the time current	Interruption
Serial Port	Game communication between two players (multiplayer)	Interruption

3.2.1 Timer

The timer is used to determine the time of each game.

The functions implemented for the Timer are found in the filetimer.c .

3.2.2 Keyboard

The keyboard is used to move characters in the game. After obtaining a certain**makecode**, if this has relevance in the game (in this case, 'A', 'W', 'S' or 'D'), a movement will be made in the corresponding direction ('A' for left, 'D' for right, 'W' for up and 'S' for down) in the sprite controlled by the player. The sprite is changed whenever a change of direction. When the**breakcode**is triggered, the movement is stopped and the sprite is changed. Additionally, we can use the keyboard to exit the game by pressing the 'ESC' key.

The functions implemented for the Keyboard are found in the files keyboard.c andKBC.c .

3.2.3 Mouse

With each mouse interruption, the in-game mouse cursor is updated accordingly with the movement applied by the player, allowing to select or point to a certain position at any time. During the game, when the option selected from the levels menu is "Level1" or "Level2", the mouse is used as if was a flashlight to make it easier to see the map that is located hidden. We also use the mouse cursor to select any of the main menu and level menu options.

The functions implemented for the Mouse are found in the filemouse.c .

3.2.4 Video Card

The Video Card is used to display all the backgrounds in the game, the buttons needed to navigate the menus, the characters controlled by the players, the time bar (which indicates how much time is left in the game finish), an arrow indicating the exit, doors and buttons that allow you to advance through the labyrinth.

The resolution used in our project is 800x600 pixels, in 0x115 mode. The color model is direct and the bits per pixel (R:G:B) are 24 (8:8:8), so with 2_{24} (about 17 million) possible colors.

For rendering the sprites, we apply the double buffering technique. We have two buffers and a draw buffer pointer that points to the buffer that is not currently being displayed, that is, where we will draw what will be shown below. This ensured that the processing of the keys to move the character and mouse cursor on the screen were displayed in a fluid and playable way.

So, in the menus we directly copy the background to the draw buffer using memcpy in our draw_background function and for all others sprites we perform painting in the draw buffer of the color map obtained by load_xpm of the sprite in the drawing_sprite function. Page flipping is done through the update_flip_frames, which swaps buffers with a VBE function 0x07 (Set Start of Display).

A system has been implemented that checks the color of the pixel where the character wants to move to detect its collisions with the walls
of the maze and with the existing doors by the check_collision() function. The character uses several animated sprites that change depending on the direction of the movement, creating the illusion of animation.

A function was also created to draw the hours in the lower corner.
left of the screen, updating them digit by digit, according to the current time. Finally, the color palette associated with each maze is changed to certain hours of the day.

The functions implemented for the Video Card are found in the file graphics.c.

3.2.5 RTC: Real Time Clock

The RTC is primarily used to determine the theme of the game/maze.
according to the time. Between 6am and 2pm, the game environment is daytime, with of a light-toned palette. Between 8pm and 6am, the atmosphere is nocturnal, with

a palette in shades of pink and purple. During the remaining hours, the maze maintains a color palette in shades of green, simulating the afternoon period.

RTC timestamps are displayed intuitively on the screen of the game, updating the project variables that contain these values every minute through the update interrupt (rtc_update_time_info()).

The functions implemented for the RTC are found in the filertc.c .

3.2.6 Serial Port

In cooperative game mode, communication between virtual machines (VMs) is ensured by the serial port. It consists of a communication protocol that was implemented to transmit data, which in this case are limited to scancodes which indicate the direction of movement of each character.

To ensure the integrity and correct order of the data received, it was developed a waiting queue. This queue prevents data loss and ensures that keyboard information is processed sequentially, without interruptions due to intermediate data.

Additionally, we created a synchronization routine to connect the VMs correctly at game start (sp_handle_start_multi()).

Then, the processing of the information received and the management of the queue waiting are handled by the sp_handle_received_info() function, which decodes the input data and performs the corresponding actions, ensuring a efficient and flawless communication between VMs during gameplay.

The functions implemented for the Serial Port are found in the file serialPort.c .

4. Code organization and structure

4.1 Timer Module

This module includes the functions developed in Lab2 for the Timer device, which cover features related to interruption subscription and increase in playing time. **Percentage:**8 % **Contributors:**

- Beatriz Bernardo
- Diana Nunes
- Sofia Goncalves ● Teresa Mascarenhas

4.2 Keyboard Module

This module includes the functions developed in Lab3 for the device Keyboard, which cover features related to subscription interruptions and reading of keyboard scancodes. **Percentage:**6 % **Contributors:**

- Beatriz Bernardo
- Diana Nunes
- Sofia Goncalves ● Teresa Mascarenhas

4.3 Mouse Module

This module includes the functions developed in Lab4 for the device Mouse, which cover functionalities related to subscription interruptions and reading of packets sent by the mouse. **Percentage:**4 % **Contributors:**

- Beatriz Bernardo
- Diana Nunes
- Sofia Goncalves ● Teresa Mascarenhas

4.4 Graphics Module

This module includes the functions developed in Lab5 for the device Graphic Card, which cover features related to the configuration of the graphics mode, pixel painting and page flipping. **Percentage:**7 % **Contributors:**

- Beatriz Bernardo
- Diana Nunes
- Sofia Goncalves ● Teresa Mascarenhas

4.5 RTC Module

This module includes functions developed for the RTC device, which cover features related to interruption subscription and time update **Percentage:**5 % **Contributors:**

- Teresa Mascarenhas

4.6 Serial Port Module

This module handles both all serial related interrupts port, as well as the functionalities related to data transmission between two VM's. **Percentage:**13 % **Contributors:**

- Teresa Mascarenhas
- Sofia Goncalves

4.7 Sprite Module

In this module we can find functions responsible for building sprites and loading of xpm files. Furthermore, in this class there are still

implemented functions responsible for the design of the lantern (which we will allow you to partially visualize the maze). The structure of this class was inspired by Professor Pedro Souto's slides. **Percentage:**10 %
Contributors:

- Beatriz Bernardo
- Diana Nunes
- Sofia Goncalves ● Teresa Mascarenhas

4.8 Logic Module

In this module you will find all the auxiliary functions that deal with the different types of interruptions. **Percentage:**8 % **Contributors:**

- Diana Nunes
- Sofia Goncalves ● Teresa Mascarenhas

4.9 Game Module

This module, in addition to managing the entire game interface, handles movement of the character, controls keyboard and/or mouse inputs, and detects collisions with the walls.

In addition, he is responsible for the character's animation, updating the animated sprites as the character moves, switching to the next sprite after a certain number of function calls.

Percentage:26 %

Contributors:

- Beatriz Bernardo
- Diana Nunes
- Sofia Goncalves ● Teresa Mascarenhas

4.10 KBC Module

In this module the“controllers”of KBC are treated, encapsulating the keyboard and the mouse.

Percentage:4 %

Contributors:

- Beatriz Bernardo
- Diana Nunes
- Sofia Goncalves ● Teresa Mascarenhas

4.11 Queue Module

This module implements a queue (data structure) in C. For the its implementation, we follow the steps in this tutorial:Creating a Queue in C | DigitalOcean **Percentage:**6 % **Contributors:** ● Teresa Mascarenhas

4.12 Proj Module

This module is responsible for initializing, configuring and controlling the loop. game's main function. It manages the hardware through interrupts, integrating also the logic of the game. **Percentage:**3 % **Contributors:**

- Diana Nunes
- Sofia Goncalves

4.13 Function call graph

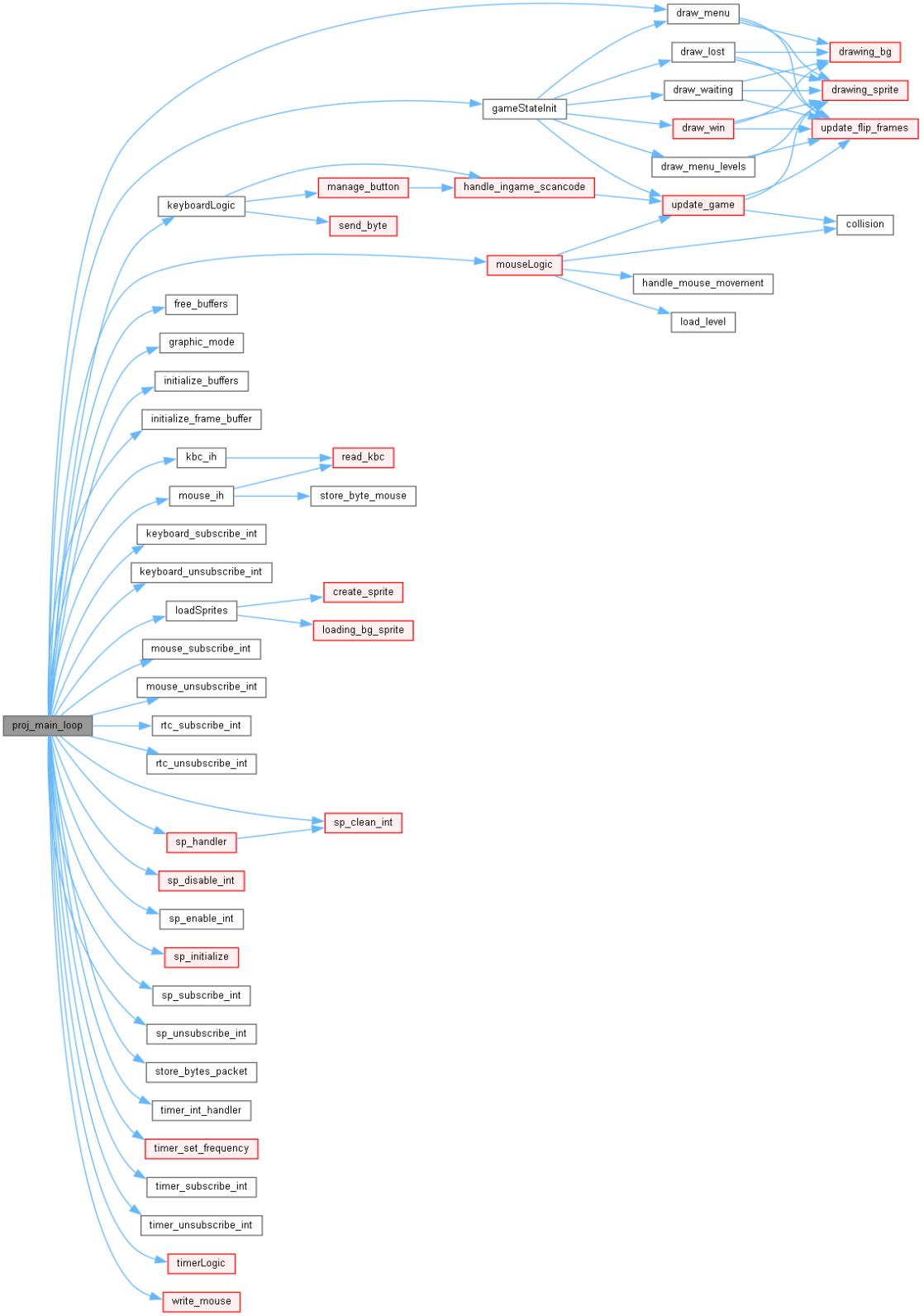


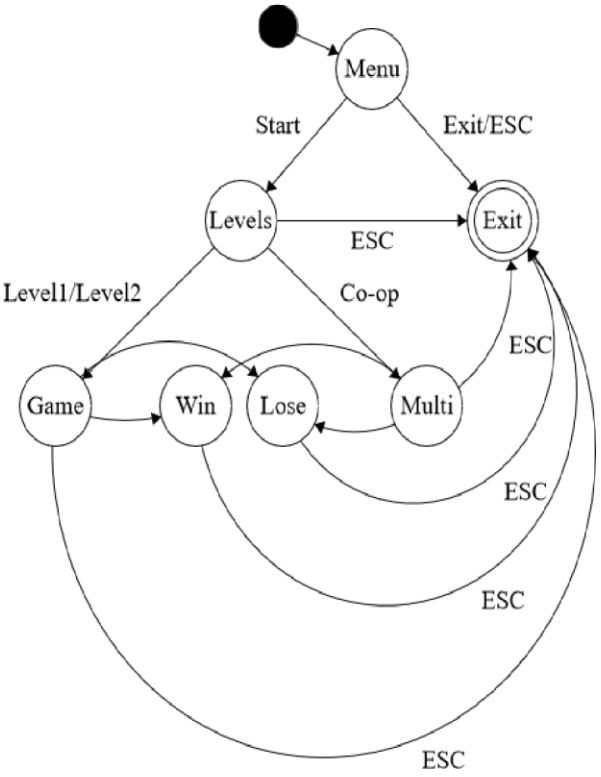
Fig 11. Function call graph

5. Implementation details

5.1 State machine

The game consists of several states: MENU, LEVELS, GAME, MULTI, WIN, LOSE, EXIT. These states are defined in an enumeration called "GameState". The initial state of the game is the MENU defined in the main loop that found in the proj.c.

State transitions occur in the "logic.c" file and are managed by a boolean variable called "gameState_change". When a state change, this variable is set to true. If "gameState_change" is true, the new state is initialized.



- In the MENU state, the user p
  choose between the Star options Exit. Selecting Start will take user to LIGHT state while selecting Exit the lev to the EXIT state, which will end program.

- In the LEVELS state, the uses you can choose the level you want play. Select Level1 or Lev will change the state to GA while selecting co-op change the state to MULTI.
- The WIN and LOSE states are determined based on success**Fig 12.**State machine

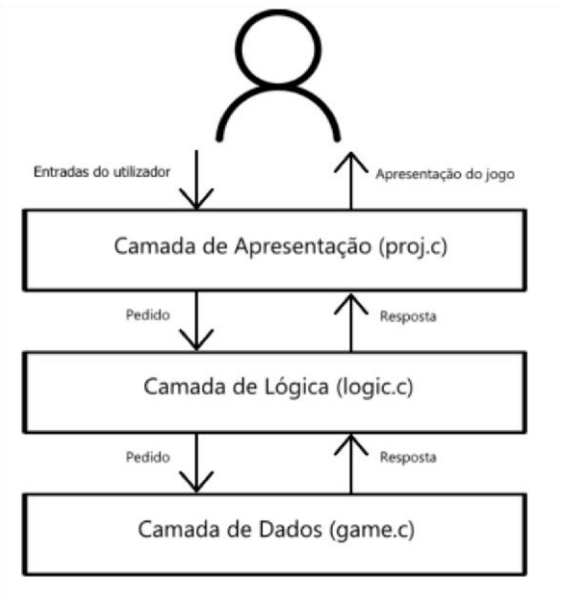
or the user's failure to complete the maze within the allotted time.

5.2 Layering

In this project, the concept of Layering was applied to organize the code in a more structured and modular way. The code was divided into three main layers: Presentation, Logic and Data.

Presentation Layer (proj.c):

- Generate the main game loop;
- Manage hardware interrupts;
- Calls the corresponding function in logic layer.



Logic Layer (logic.c):

- Processes incoming calls from presentation layer, applying the rules of the game when calling functions data layer correspondents.
- Update the game state (“GameState”).**Fig 13.**Code layers

Data Layer (game.c):

- Provides functions to access and modify game elements.

- It is called by the logic layer to get required data and update the game state according to the rules applied.

5.3 Event-driven Code

The code is also "Event-Driven", that is, it responds to events that occur in the system, such as:

- The main loop continues to run until the player presses the ESC key to exit.
- Inside the main loop, the program waits for interrupts to arrive. hardware. When an interrupt is received, the program determines which device generated the interrupt and executes the processing and logic of the matching game.
- When the player decides to exit the game, the code disables all interrupt signatures and performs other necessary cleanup tasks before terminating the execution.

In this way, it is clear that the code follows the "Event-Driven" paradigm, a since its execution is guided by the events that occurred, such as inputs from user, timer signals, among others, instead of following a sequence linear instruction set.

5.4 Sprite

Sprite type objects store all the necessary information about the images we want to draw: their position on the screen with x and y coordinates y, the height and width of the image, the speed (if it moves) and the colormap obtained by load_xpm.

At the beginning of the game, we load xpm and create all the sprites used, except for the maze map, which is only loaded when the level is chosen.

Afterwards, to paint the sprites, you only need to access the map. specific sprite, which makes this process more efficient.

5.5 Object orientation

Object orientation was also used in this project. As we can see on topic5.4 , the Sprite structure works like a class, since it encapsulates data related to a sprite (analogous to the attributes of a class), having multiple instances of this structure (analogous to objects).

1. **Encapsulation**: The data and the functions that operate on that data are logically grouped.
2. **Abstraction**: Functions provide a clear interface for creating, destroying, and manipulate sprites without exposing implementation details.
3. **Modularity**: Each sprite and its operations are treated as a separate unit, facilitating maintenance and development of the code.

5.6 Collisions

To handle the character's collisions with the maze walls, we compare the color of specific points in the maze where the character intends to move with the color of the ground. If it is different, it means there is a wall or that is off-screen, and therefore does not move.

The points that are checked depend on the direction in which the character wants to move:

- Upwards, A and B.
- Down, the C and the D.
- To the right, B and D. ● To the left, A and C.

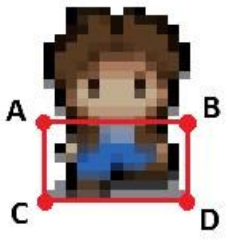


Fig 14. Player hitbox

5.7 Frame Generation

In this project, frame generation is performed mostly by the Graphics (graphics.c) and Sprite (sprite.c) module.

- **Initialization**:
  - ○ Trustholds the video buffers and sets the graphics mode.
  - ○ Allocates memory for the back buffer.
- **Rendering Loop**:
  - ○ In the main loop, (proj_main_loop) the hardware events are processed to update the state, with the sprites that need to be drawn in the current frame are determined in the game logic according to this state.
- **Frame Design**:
  - ○ Copies the background to the drawing buffer.
  - ○ Draws the sprites into the drawing buffer.
- **Page Flipping**:
  - ○ Switches buffers, swapping the drawing buffer and the display buffer.
  - ○ Clears the new drawing buffer to prepare it for the next one frame.

This process ensures that frames are generated and displayed in a efficient, providing a continuous and uninterrupted gaming experience noticeable.

5.8 RTC

The implementation of RTC is done mainly through the function rtc_update_time_info(), which updates the rtc_info time_info structure with time current read from the RTC. This structure is then used to change the colors of the maze and display the time during the game.

The rtc_update_time_info() function reads the RTC registers that contain information about seconds, minutes, hours, day, month and year. It interprets these values according to binary counting mode or BCD (Binary Coded) Decimal) from the RTC and stores them in the time_info structure. Then the values of this structure are used as needed to display game time or to make changes to the maze colors.

5.9 Serial port

The implementation of communication via serial port is done mainly through the sp_initialize() function, which configures the serial port and initializes the receive and send queues. This configuration enables communication between devices during gameplay.

Communication is managed primarily through functions receive_byte() and send_byte(). The receive_byte() function reads data from the receive buffer (RBR) and stores them in the receive queue, while the function send_byte() puts data into the send queue and starts transmission if the queue is full. empty.

During gameplay, the manage_button() function is used to send codes to keys pressed through the serial port. It converts the key codes into transmission codes and sends them using send_byte(). This allows the key events are shared across devices.

Serial port interrupts are managed by the sp_ih() function, which handles both with the reception and transmission of data, calling the appropriate functions based on the type of interrupt detected.

Additionally, the sp_handle_start_multi() function manages the start of the game in multiplayer, synchronizing devices by exchanging codes specific.

6. Conclusions

By carrying out this project we were able to put into practice the knowledge that was transmitted to us, regarding the peripherals of a computer, whether in terms of its operation or its management.

As the project developed is a game, we had to structure our program around the notion of game states. Thus, as discussed in the theoretical classes, we decided to implement a state machine.

In this sense, we were forced to develop a new way of critical thinking and programming, quite different from what we were accustomed. Furthermore, the fact that our game theme is unconventional for this type of projects required us to think more deeply about how to realize ideas that were initially just abstractions.

We faced some difficulties throughout the project, the main ones being the implementation of page flipping and serial port. Therefore, as far as Regarding page flipping, we consider that it was something that arose from the need to make our game more efficient, and to avoid rendering excessive of the entire interface.

In conclusion, despite the adversities faced, we feel that the project was successful, as we were able to implement the functionalities that we set out to do.