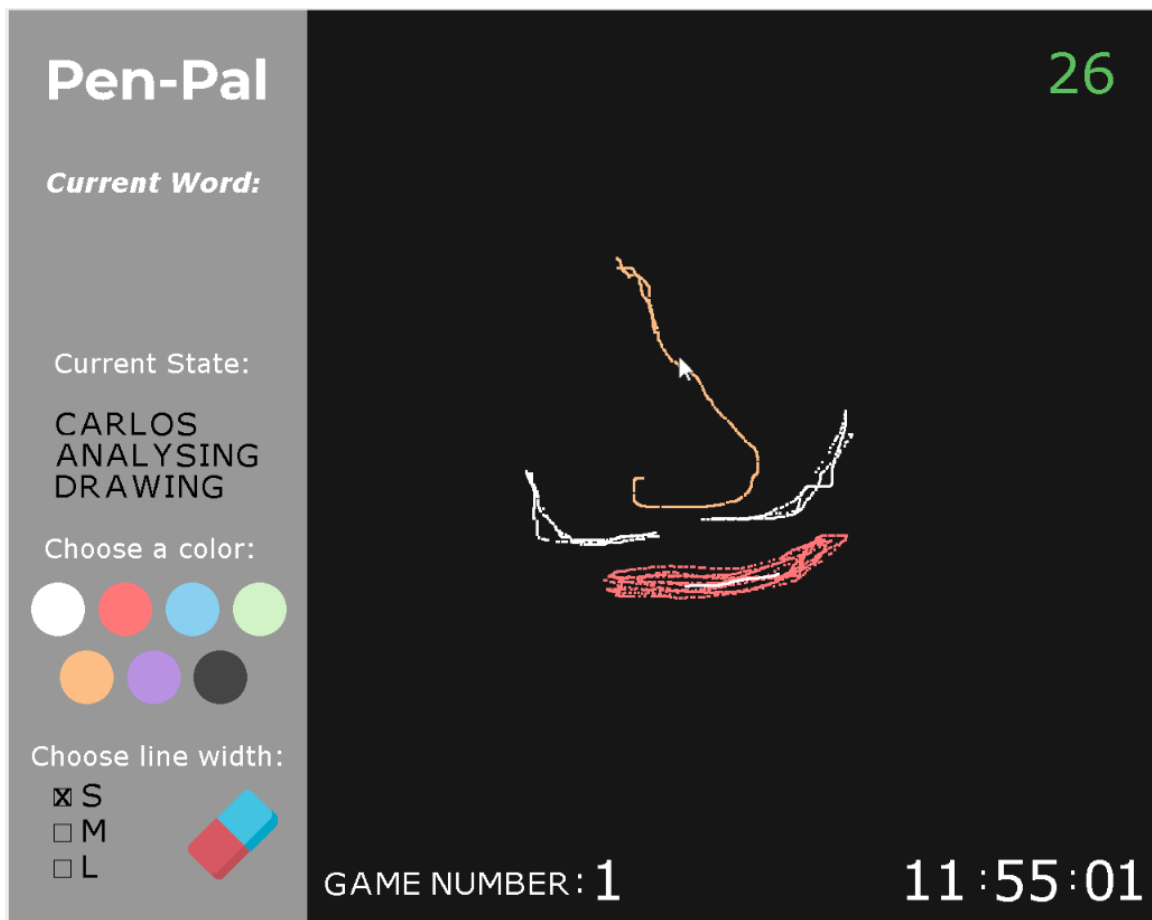


Pen-Pal



Project developed for the 2nd year course of
Computer Laboratory @FEUP

Group 2 Class 6

Carlos Veríssimo, 201907716
Marcos Aires, 202006888
Martim Videira, 202006289
Miguel Silva, 202007972

Index

[Index](#)

[User Instructions](#)

[Project Status](#)

[Code Organization/Structure](#)

[Modules](#)

[CodeToCharacter.c](#)

[Color.c](#)

[graphics.c](#)

[keyboard.c](#)

[mouse.c](#)

[number.c](#)

[pen_pal.c](#)

[proj.c](#)

[rtc.c](#)

[sprite.c](#)

[timer.c](#)

[utils.c](#)

[words.c](#)

[Function Call Graph](#)

[Implementation Details](#)

[Keeping Track of the Current Frame](#)

[Event-Driven Code and State Machines](#)

[Game State Machine](#)

[Mouse State Machine](#)

[Keyboard State Machine](#)

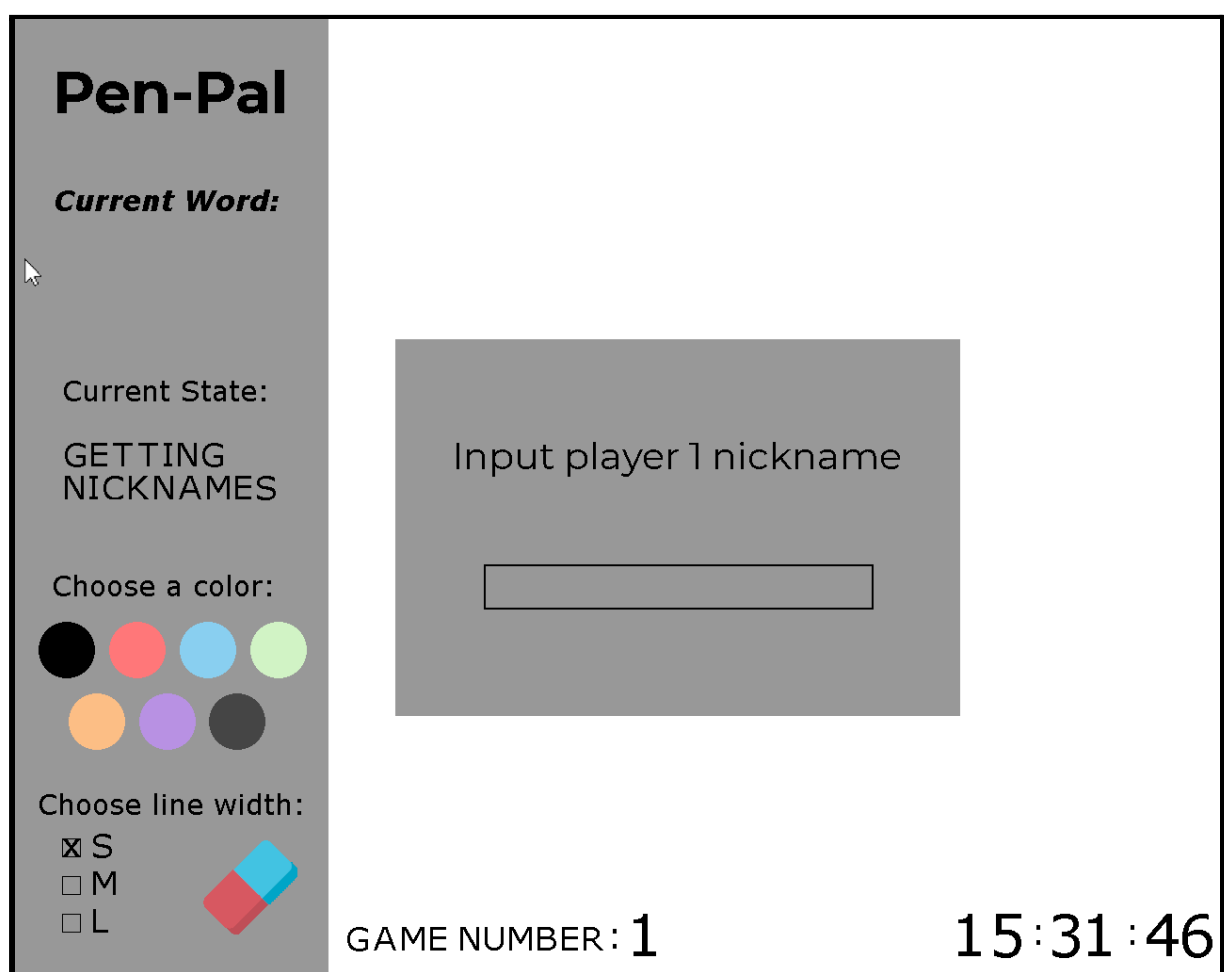
[Conclusions.](#)

User Instructions

Pen-pal is a drawing and guessing game in which players use their creative skills to draw a visual representation of a word on a screen within a predefined time limit. A second player will then try to guess which word the first player tried to draw.

The game is meant to be played with 2 people, a drawer and a guesser. However, nothing stops you from inviting your friends to play along and help you uncover what word the drawer tried to sketch.

After loading up the game, you are asked to input 2 nicknames: one for the drawer and another for the guesser. The prompt is the following:



The screenshot shows the Pen-Pal game interface. On the left is a grey sidebar with the title "Pen-Pal" at the top. Below it is the label "Current Word:" followed by a blank space. Further down is "Current State:" followed by the text "GETTING NICKNAMES". Below that is "Choose a color:" followed by seven colored circles (black, red, blue, green, orange, purple, grey). At the bottom of the sidebar is "Choose line width:" followed by three checkboxes labeled "S", "M", and "L", and a small icon of a red and blue cube. The main area of the screen is white. In the center, there is a grey rectangular box containing the text "Input player 1 nickname" and a white rectangular input field below it. At the bottom of the screen, there is a black bar. On the left side of this bar, it says "GAME NUMBER: 1". On the right side, it shows a timer "15:31:46".

Figure 1: Nickname input

You can use your keyboard just like you would in any other game.

After writing both names, the guesser (player 2) should step away from the computer so that the drawer (player 1) can pick an option from 3 randomly generated words that are taken from a dictionary:

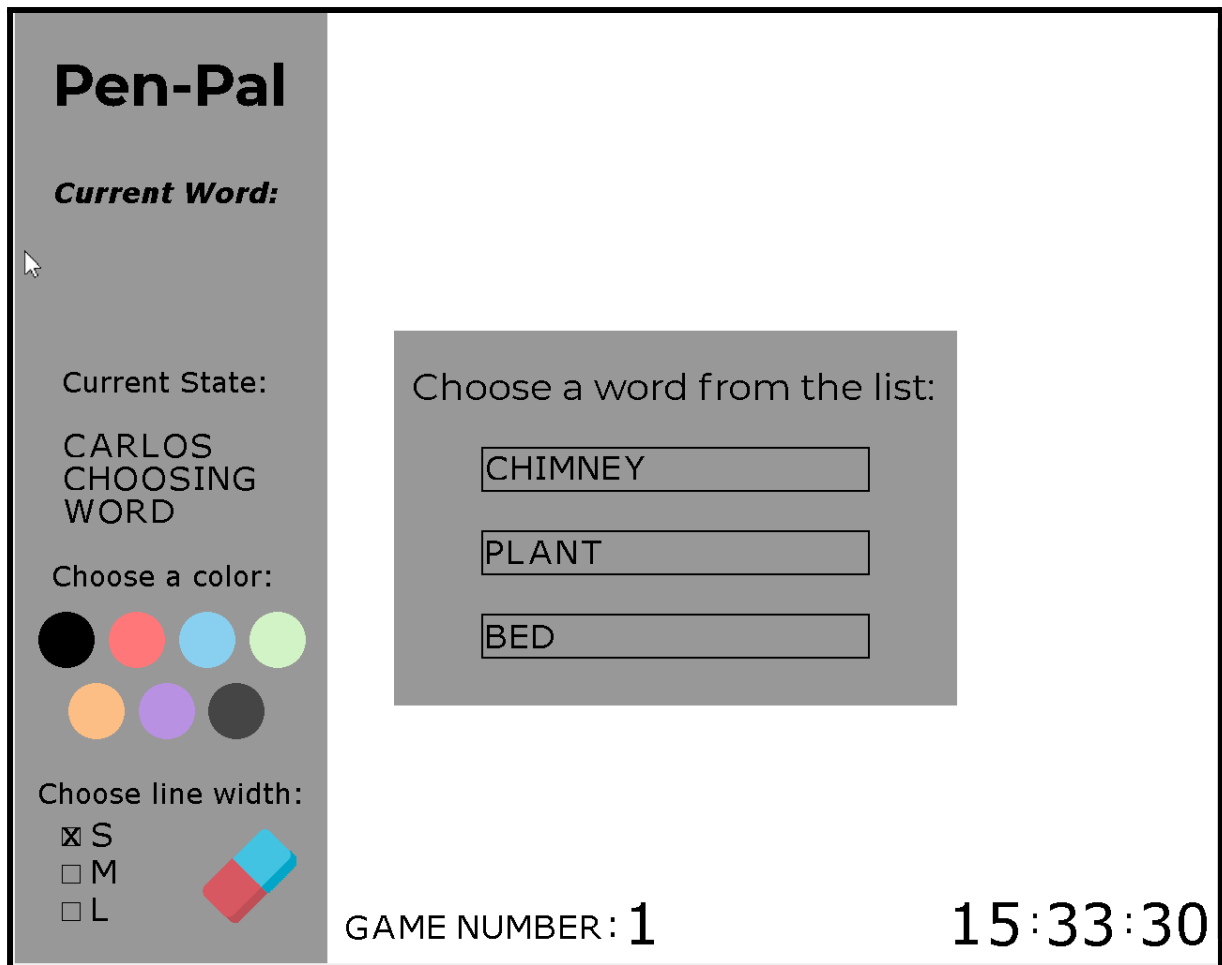


Figure 2: Word selection

Player 1 should then try to sketch the visual representation of the word within 60 seconds (note: this value can be changed, just head over to the **pen_pal.h** file and change this defined value): `#define DRAWING_TIME MINUTES(1)`

The value is defined to be in minutes, e.g. MINUTES(3) will change the drawing time to 180 seconds. However, if you want to only have 30 seconds of drawing time, you can just do **DRAWING_TIME 30** and it works as well. The same can be said for the guessing time, which was defined to be half of what the drawing time is.

The game gives you 7 colours to choose from. Also, the player can, at any time, erase what they drew, since a rubber (of three different sizes) was implemented. There are also 3 checkboxes which allow you to choose which size your drawing should be.

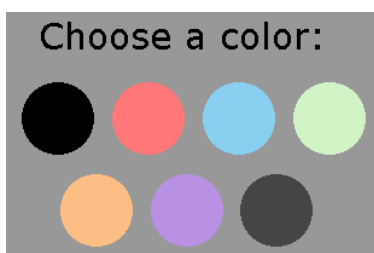


Figure 3a: Colour selection (light-mode)



Figure 3b: Colour selection (dark-mode)

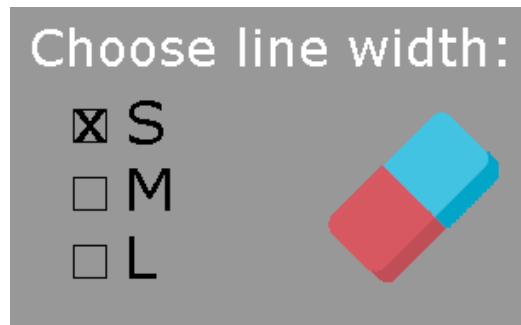


Figure 4: Pixel size selection and eraser

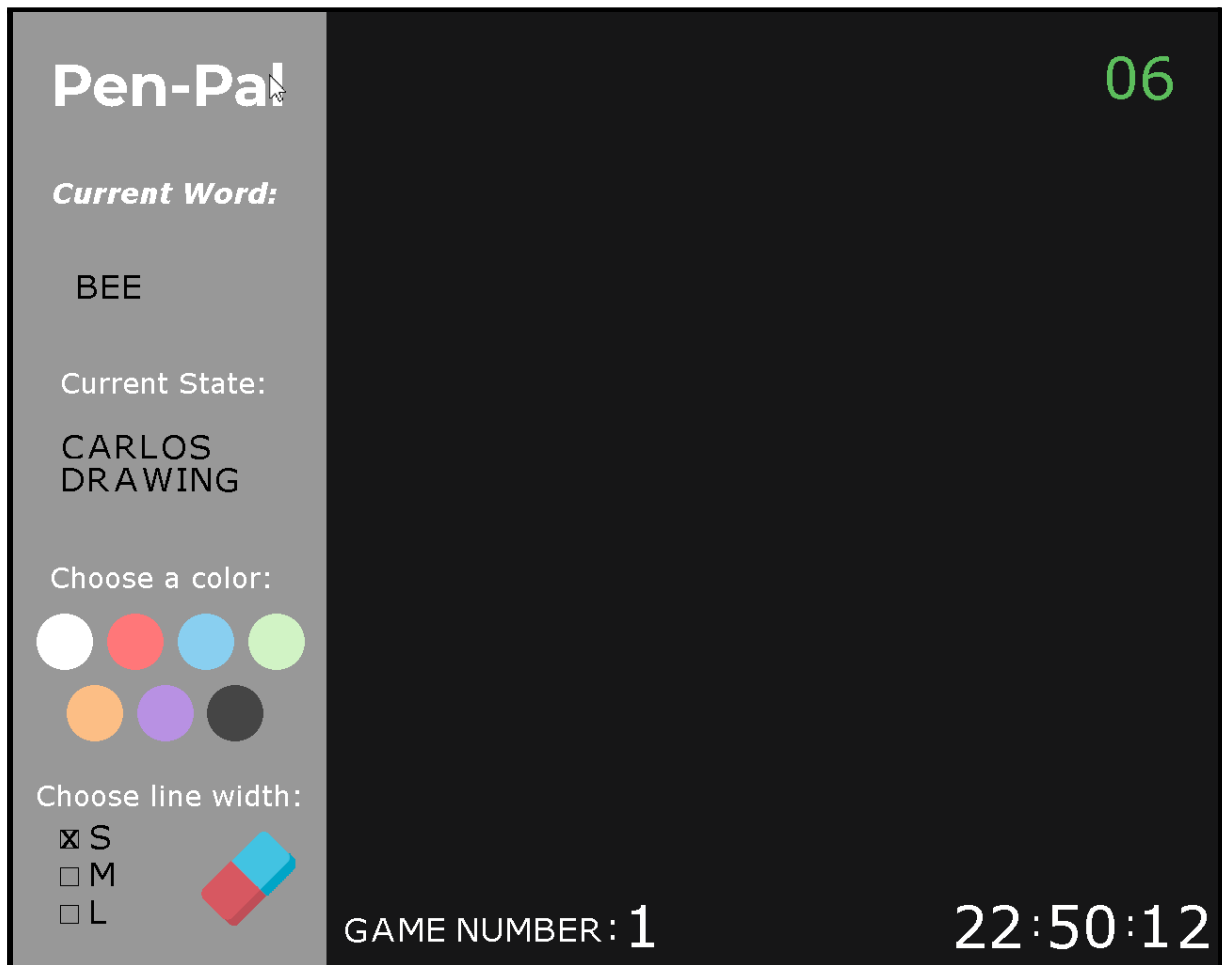


Figure 5a: Game in dark mode

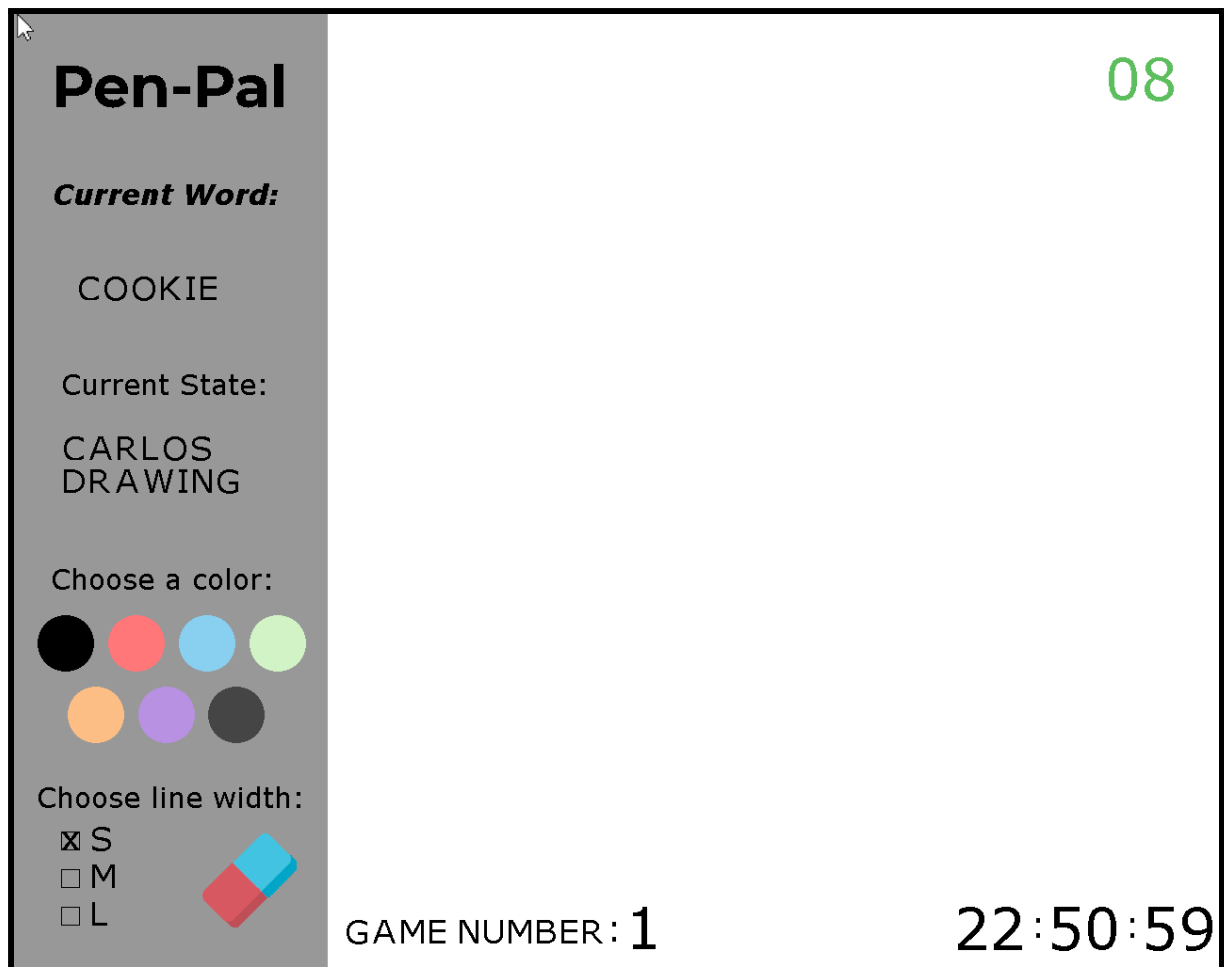


Figure 5b: Game in light mode

After 60 seconds, player 2 should approach the computer and will have about half of the drawing time to analyse what player 1 drew.

After that, they should take a guess at what player 1 has just drawn. There are 3 guesses available. You submit a guess by pressing the ENTER key.

If you are lucky to have correctly guessed the word, the following screen will be shown:



Figure 6a: Correct guess

However, if you submit a wrong guess, one of the following prompts will pop up:

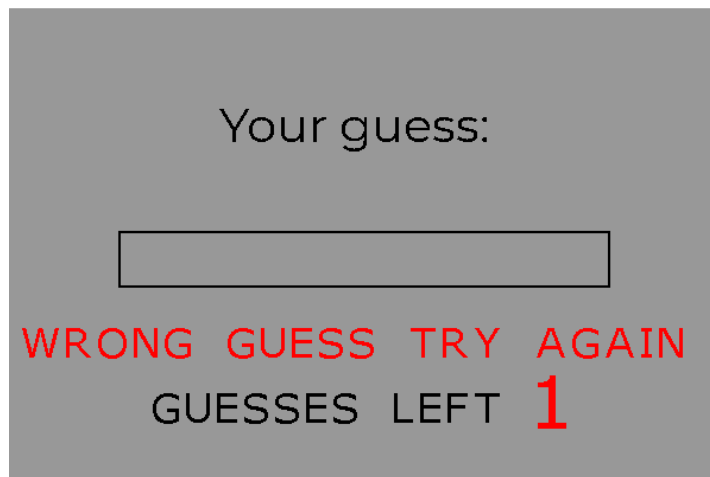


Figure 6b: Wrong guess (still have guesses left)



Figure 6c: Wrong guess (no guesses left)

Nevertheless, after that, a prompt appears with three options: you can either exit the game, returning to the Minix command-line interface, choose to play again (and the game will restart), or even choose to switch sides (that is, assuming that player 1 was the drawer and player 2 the guesser, if this option is chosen, player 1 will now be the guesser and player 2 the drawer). The latter option is achieved by swapping the nicknames of the players.

Project Status

Device	What for	Interrupt/Polling
Timer	Restrict the time the drawer has to draw and print the real-time	I
Keyboard	Insert the players' names, the guesses and possibly choose the word to be drawn.	I
Mouse	Navigate and select options in the menu and draw	I
Video card	Used to draw everything displayed on the screen.	N/A
RTC	Switch between dark and light mode and retrieve the real-time.	N/A

Timer

The timer is used to restrict the time that the players have to perform their tasks. The timer is set to a 60Hz frequency, i.e., in one second, 60 timer interrupts occur. So, at each one of those 60 interrupts, a function that handles these is called (`timer_event_handler()`).

At every 60 interrupts, a second passes and a function that reads the real current time from the RTC registers is called (`read_curr_time_from_rtc(Time *time)`) and another one that prints this real-time is also called (`print_time(Time time)`).

In the case of counting down the drawing time left, a variable (`drawing_time_left`) that tracks it is decremented (it starts at 60) at each 60 interruptions by one second, and at each second, a function is called to display the time left (`display_time_left(int time_left)`). When it reaches 0 (60 seconds passed) the time for the player to draw is over.

The situation is similar for the case when, after the drawing time is over, the time that the guesser has to analyse the draw before the input prompt that takes the guess appears is counted down. However, instead of 60 seconds, the guesser has only 30 seconds to analyse the draw before the input prompt appears. This time, a variable (`analysis_time_left`) that tracks the analysis time left is decremented by one second from 30 until it reaches 0, and, at each second, a function that displays that time left is called (`display_time_left (int time_left)`).

Keyboard

The keyboard is used here for text input on two different occasions: first, at the beginning of the game, when both players are choosing their names. Player 1 starts by typing their name, and, for each character that they type, a make code of the character is sent by the KBD to the output buffer. The interrupt handler (`kbc_ih()`) reads that code from the output buffer and a function that translates the code into an ASCII character is called (`get_character_from_code(unsigned char code)`). Finally, this character is added to the end of the string. This process repeats itself until the user presses the 'enter' key, which marks the end of the input and the confirmation of the submission. The exact same happens when the guesser is typing the word to infer what the drawer has crafted.

The keyboard can also be used by the drawer to choose the word to be drawn (although this can and should be done with the mouse). As we have three words randomly chosen between a group of words in a predefined dictionary, pressing the key '1' on the keyboard selects the first one, pressing '2' selects the second one, and pressing '3' selects the third one. The process is very similar to the aforementioned. An interrupt is generated when the user presses one of those keys and a make code is sent to the output buffer, the interrupt handler reads that code from the output buffer, the function that translates the code to ASCII is called and finally is verified if the character pressed is one of the previously mentioned three. If yes, one of the words will be chosen, depending on what key was pressed. Otherwise, nothing will happen.

Mouse

The mouse is used to navigate and select options from the menu and is also responsible for the main focus of the game: the drawing.

In terms of buttons of the mouse, the only one that has some kind of functionality is the left button, which is used to select some option in the menu, to perform the drawing, as you need to have this button pressed down while the drawing is being executed and to select different colours/sizes for the drawing.

Depending on the position of the mouse on the plane, pressing this button can cause an option in one of the several menus (initial, end of the game,...) to be selected. In terms of the drawing action, after the left button is pressed down, each change in the position of the mouse will cause the generation of interrupts (interrupts that are handled by the `mouse_ih()` and by the `mouse_event_handler()`). Each time the mouse changes position in this situation, a pixel is drawn to the screen, being the draw completed this way.

Video Card

The video card is responsible for all the screen displays. The graphics card is initialised in mode 0x11b, with a resolution of 1280x1024 pixels, 16.8 million colours and 24 bits per pixel (8:8:8, i.e., 8 for the red component, 8 bits for the green component and 8 bits for the blue component, respectively), being the colour model the direct mode (using the function `set_graphics_mode(uint16_t mode)`).

Basically, the physical address of the VRAM is first discovered (by calling a function that retrieves the VBE mode information, `vbe_get_mode_info(uint16_t mode, vbe_mode_info_t *vbe_info)`) and then it is mapped to the process's virtual address space (by calling the function `map_vram(uint16_t mode)`). A change on the screen (that happens for each pixel that is drawn on the screen or for each action that causes the screen to change, like when a menu option is clicked with the left mouse button and makes the screen state change, by showing another screen) happens because of changes applied to the frame buffer, a buffer that contains a full-screen image.

Although we cannot point out the use of VBE functions besides the function that was used to initialise the graphics card with the aforementioned mode, nor double buffering, nor fonts, we can make reference to sprite collisions, for example, between the mouse cursor and the menu options (function `cursor_collisions_in_word_selection()`) and between the mouse cursor and the colours in the palette (collisions handled by the function `handle_mouse_collision_with_colors(uint32_t *new_color, Collision collision)`).

Real-Time-Clock (RTC)

The Real-Time-Clock(RTC) is used to allow the real-time to be displayed on the game screen, as well as the light and dark modes. Essentially, from 7 AM to 19 PM, the game works in light mode, and between 19 PM and 7 AM, the dark mode is introduced, which, in its essence, makes the board where the draws are done turn black.

This is done by reading the seconds, minutes and hours from the RTC registers (using the function `read_curr_time_from_rtc(Time *time)`) and translating these BCD (Binary-Coded Decimal) values into decimal values (using the function `bcd_to_decimal(uint32_t x)`). The value of the hours is then compared with the values 19 and 7. If it is between these values, then dark mode must be displayed. Otherwise, the game will be in light mode.

Code Organization/Structure

Modules

CodeToCharacter.c

- Description: Module to ease the translation between keyboard codes and ASCII characters. Contains functions that, given a character, return its corresponding break code and vice versa.
- Data Structure:
 - **CodeToCharacter**
 - uint8_t break_code;
 - char character;
- Weight: 2%
- Members involved in this module:
 - Carlos Veríssimo

Color.c

- Description: Module used to associate an RGB value to the visual representation of a colour, using a sprite. Contains a function that loads all the colours in the game into an array of Colours, a function that, given a colour index, returns its RGB value and a function to draw all the colours on the screen.
- Data Structure:
 - **Colour**
 - uint32_t colour;
 - Sprite *sp;
- Weight: 2%
- Members involved in this module:
 - Carlos Veríssimo

graphics.c

- Description: contains functions and logic to deal with the graphics card.
 - Set the graphics card mode
 - Draw a pixel in a given position with a given colour
 - Draw a line in a given position with a given colour and length
 - Includes a variation to draw a line and add it to the current frame
 - Draw a rectangle in a given position with a given colour, length and width
 - Includes a variation to draw a rectangle and add it to the current frame
 - Map the virtual memory
 - Draw a pixmap
 - Includes a variation to draw it with a fixed colour
 - Get information about the dimensions of the screen
 - Draw an unfilled square
- Weight: 15%
- Members involved in this module:
 - All

keyboard.c

- Description: contains functions and logic to deal with the keyboard.
 - Subscribe keyboard interruptions
 - Unsubscribe keyboard interruptions
 - Interruption handler
 - Check for errors
 - Reads a byte from the output buffer and adds into to the correct scancode array position
 - Read the status of the keyboard controller
 - Set the command byte
 - Get the command byte
 - Read from the output buffer
 - Find the correct position in the scancode array to add the byte read
 - Write a command to the keyboard
 - Find whether a certain keyboard code is a make or a break code
- Weight: 10%
- Members involved in this module:
 - All

mouse.c

- Description: contains functions and logic to deal with the mouse.
 - Interrupt handler:
 - Checks if the output buffer is full
 - Check for errors
 - Reads a byte from the output buffer.
 - Disable data reporting
 - Subscribe mouse interruptions
 - Unsubscribe mouse interruptions
 - Read from the output buffer
 - Parse a mouse packet
 - Create a new cursor
 - Draw the cursor
 - Delete cursor's old position
 - Draw a pixel
 - This function will call a graphics function that will draw a rectangle on the screen, at the cursor current coordinates.
 - The size of the rectangle will be the selected option for the draw size multiplied by an expanding factor of 3;
 - This function is also used for the eraser and will draw a rectangle with the colour of the background.
 - Before drawing, will check if the cursor is within the drawing area.

```
void(mouse_draw_pixel)(uint32_t selected_color, char width,
bool addToBackgroundFrame) {
    if (cursor->x < 336)
        return;

    int expanding_factor = 3;
```

```

    if (width == 'M')
        expanding_factor = 6;
    else if (width == 'L')
        expanding_factor = 9;

    if (eraser_on)
        expanding_factor *= 3;

    my_vg_draw_rectangle(cursor->x, cursor->y,
        expanding_factor, expanding_factor, selected_color,
        addToBackgroundFrame);
}

```

- Update the cursor position
- Update the cursor
 - Deletes the previous cursor (avoids cursor dragging)
 - Updates the cursor position
 - Draws the cursor at the new position
- Check collision with words
- Check collision with colours and eraser
- Check collision with end game options
- Given a collision with a colour, update the selected colour
- Weight: 25%
- Data structure:
 - **Cursor**
 - uint16_t x, y; // x and y coordinates of the cursor
 - xpm_image_t img; // xpm for the normal cursor
 - xpm_image_t eraser_img; // xpm for the eraser cursor
 - xpm_image_t select_img; // xpm for the hover cursor
 - bool isOverOption; // indicate whether cursor is over an option
 - **Collision**
 - Enumerable that contains all the possible collisions in the game.

Members involved in this module:

- Carlos Veríssimo

number.c

- Description: Handles logic to draw numbers on the screen. Functions:
 - Load numbers' images
 - Display the time left
 - Draw a number on the screen
 - Print the time in the hh:mm:ss format
- Weight: 5%

- Members involved in this module:
 - Carlos Veríssimo

pen_pal.c

- Description: The main file, contains the game loop and all the event handlers
- Weight: 10%
- Members involved in this module:
 - Carlos Verissimo

proj.c

- Description: Entry file to the project, sets the video mode, fills the list of Code-Character pairs and calls the game loop.
- Weight: 5%
- Members involved in this module:
 - Carlos Verissimo

rtc.c

- Description: Used to get information about the current time. Reads from the Real-Time-Clock registers. Functions:
 - Find whether or not the dark mode should be toggled
 - Reads the hours register
 - If the current hour is between 7 pm and 7 am, toggle dark mode.
 - Read from an RTC register
 - First writes to the address register, which register it wants to read from
 - Reads from the register
 - Read the current time
 - Read hours, minutes and seconds registers
 - Converts the values read into the decimal format and add them to the *Time* data structure
 - Convert from BCD to decimal format
 - [Credits](#)
- Weight: 5%
- Data structure:
 - **Time:**
 - int seconds;
 - int minutes;
 - int hours;
- Members involved in this module:
 - All

sprite.c

- Description: Used instead of plain xpm_image_t when elements are fixed.
functions:
 - Create a sprite
 - Delete a sprite
 - Draw a sprite
- Weight: 5%
- Data structure:
 - **Sprite:**
 - int x,y; // current sprite position
 - int xspeed, yspeed; // current speeds in the x and y direction */
 - uint8_t *map; // the sprite pixmap
 - xpm_image_t img; // the sprite image
 -
- Members involved in this module:
 - Carlos Verissimo

timer.c

- Description: Description: contains functions and logic to deal with the timer.
 - Set the timer frequency
 - Subscribe timer interruptions
 - Unsubscribe timer interruptions
 - Handle the interruption
 - Increments a global counter
 - Read the timer configuration
- Weight: 5%
- Members involved in this module:
 - All

utils.c

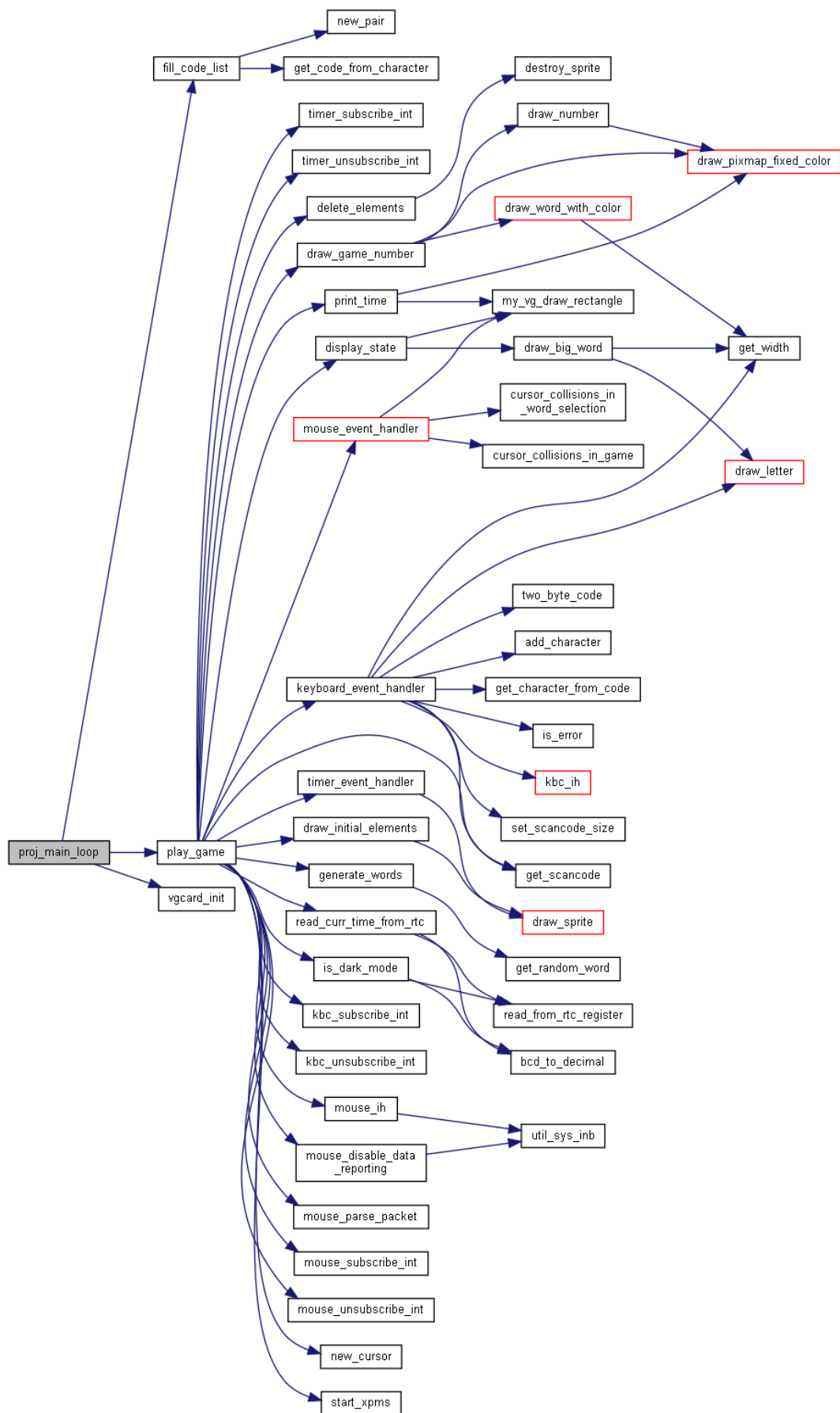
- Description: Utility module, contains wrapper functions for system calls.
- Weight: 1%
- Members involved in this module:
 - All

words.c

- Description: Handles logic to draw letters (words) on the screen and contains a dictionary with some words. Functions:
 - Load a letter image
 - Load the whole alphabet
 - For loop from 'A' to 'Z' that calls the function that loads a letter
 - Draw a letter on the screen

- Includes a variation to draw a letter on the screen with a given colour
- Draw a word on the screen
 - Includes a variation to draw a big word on the screen.
 - Also has a variation that draws it with a given colour
 - Includes a variation to draw a word on the screen with a given colour
- Get the width of a letter image
 - Useful when spacing the letters in a word
- Get a random word from the dictionary
- Add a character to the end of a string
- Weight: 10%
- Members involved in this module:
 - Carlos Verissimo

Function Call Graph



Implementation Details

Keeping Track of the Current Frame

After implementing the cursor image and drawing it on the screen, we faced a problem that we had not planned to encounter.

When moving the mouse, a drag would appear, something like this:



Furthermore, moving the mouse over other game elements would draw over them, essentially deleting them and, obviously, that is not what we wanted.

We quickly found that in order to have the mouse work as it should, we had to keep track of what was on the old frame, so that, when updating the cursor position, we would be able to reset the frame and avoid the dragging.

To implement this, the following was done:

- Every time something is drawn on the screen, we also add it to something that we call the frame
 - It's basically a copy of what's on the screen.
- This is not the case for the mouse, i.e, when drawing the mouse, we don't add it to the frame, we only draw it on the screen.
 - This is done so that, when updating it to a new position, we can restore what was drawn on the cursor's old position.
- Only needed to change the function that draws a pixel on the screen to add a parameter that specifies whether or not that pixel should be added to the frame.
- Note: Moving the mouse, i.e, updating the cursor position, does not imply adding it to the frame, however, when drawing, we obviously, need to add the drawing to the frame.

Event-Driven Code and State Machines

Game State Machine

Depicts the current state of the game. Devices should handle events accordingly to the game state. The possible game states are:

- GETTING_NICKNAME_1,
- GETTING_NICKNAME_2,
- CHOOSING_WORD,
- PLAYER1_DRAWING,
- ANALYSE_DRAWING,
- PLAYER2_GUESSING,

- CORRECT_GUESS,
- WRONG_GUESS,
- OUT_OF_GUESSES,
- PLAY_AGAIN,
- EXIT_GAME

Most of the event-driven code for the game state is related to the timer and the function that will handle its events is the **timer_event_handler**. This function will always increment a global counter and will perform different instructions, accordingly to the state of the game.

- If the game state is **GETTING_NICKNAME_1**, **GETTING_NICKNAME_2** or **CHOOSING_WORD**, the event handler will only check if the corresponding prompt has been drawn. If not, draws it and toggles the state variable of that prompt.
- If the game state is **PLAYER1_DRAWING**, the handler will
 - Check if the prompt has been drawn. If not, draws it and toggles its state variable.
 - Check if the global counter is divisible by 60 (i.e, a second has passed). If so, decrements it and calls a function that will display the remaining time to draw
 - Check if the time left to draw has expired (≤ 0). If so, resets the counter and prepare the next game state, by changing the state of all the devices.
- If the game state is **ANALYSING_DRAWING**, the event handler will
 - Check if the prompt has been drawn. If not, draws it and toggles its state variable.
 - Check if the global counter is divisible by 60 (i.e, a second has passed). If so, decrements it and calls a function that will display the remaining time to analyse.
 - Check if the time left to draw has expired (≤ 0). If so, resets the counter and prepare the next game state, by changing the state of all the devices.
- If the game state is **PLAYER2_GUESSING**, the event handler will
 - Check if the prompt has been drawn. If not, draws it and toggles its state variable.
 - Draw the number of guesses left on the screen
 - Check if a guess has been submitted. If so
 - Check if it's correct. If so
 - New game state is **CORRECT_GUESS**, otherwise, increment the number of guesses and check if there are any guesses left. If so,
 - new state is **WRONG_GUESS**, otherwise, the new state is **OUT_OF_GUESSES**
- If the game state is **WRONG_GUESS**, the event handler will
 - Check if the prompt has been drawn. If not, draws it and toggles its state variable.
 - Display a message saying that the guess was wrong.

- Draw the number of guesses left on the screen
- Check if a guess has been submitted. If so
 - Check if it's correct. If so
 - New game state is **CORRECT_GUESS**, otherwise, increment the number of guesses and check if there are any guesses left. If so,
 - new state is **WRONG_GUESS**, otherwise, the new state is **OUT_OF_GUESSES**
- If the game state is **OUT_OF_GUESSES**, the event handler will
 - Check if the prompt has been drawn. If not, draws it and toggles its state variable.
 - Display a message saying that there are no more guesses left
- If the game state is **PLAY_AGAIN**
 - New game state is **CHOOSING_WORD**

Mouse State Machine

Depicts the current state of the mouse. Devices should handle events accordingly to the game state. The possible game states are:

- MOUSE_GETTING_NICKNAMES,
- MOUSE_CHOOSING_WORD,
- MOUSE_DRAWING,
- MOUSE_ANALYSING,
- MOUSE_GUESSING,
- MOUSE_PLAY_AGAIN

Most of the event-driven code for the mouse state is related to the mouse and the function that will handle its events is the **mouse_event_handler**. This function will always update the cursor and perform different instructions, accordingly to the state of the mouse.

- If the mouse state is **MOUSE_GETTING_NICKNAMES**, **MOUSE_ANALYSIS** or **MOUSE_GUESSING**, nothing is performed.
- If the mouse state is **MOUSE_CHOOSING_WORD**:
 - Call function that checks collision between mouse cursor and different words.
 - E.g.: If the function returns collision #1, it means the user clicked on the 1st word and so on for the other words.
 - Set the current word.
 - If the left button is not pressed, break
 - If the current word is not null, it means there was a collision with one of the words
 - Prepare the next game state, by changing the state of all the devices.
- If the mouse state is **MOUSE_DRAWING**:
 - Call function that checks collision between mouse cursor and the different elements in the game (colours, sizes and eraser)
 - If the left button is not pressed, break

- If the collision corresponds to a collision with the size selection boxes
 - Update the selected box by drawing a rectangle with the background colour inside the old box and drawing the tick (X) in the new box.
 - Otherwise, call the function that handles the mouse collision with colours
 - If, after that, the selected colour was changed to the background colour, it means we have a collision with the eraser. We act as if we were dealing with any other colour.
 - Call the function that draws in the current cursor position, passing it the selected colour and the selected width(size).
 - Set the current word.
- If the mouse state is **MOUSE_PLAY_AGAIN:**
 - Call function that checks collision between mouse cursor and the different options in the play again menu.
 - If the left button is not pressed, break
 - Both collisions with the play again and play again & switch roles buttons call the function that prepares the game restart, however only the latter calls the functions to swap the nicknames, The restart game function:
 - Resets the drawing and guessing times.
 - Zeroes the guess
 - Resets state variables of prompts
 - Resets selected width
 - Prepares the next game state, by changing the state of all the devices.
 - Collision with the exit game button just sets the game state to **EXIT_GAME**, so it exits the main loop.

Keyboard State Machine

Depicts the current state of the keyboard. Devices should handle events accordingly to the game state. The possible game states are:

- KBD_IN_GAME,
- KBD_GETTING_NICKNAME_1,
- KBD_GETTING_NICKNAME_2,
- KBD_CHOOSING_WORD,
- KBD_GETTING_GUESSES

Most of the event-driven code for the keyboard state is related to the keyboard and the function that will handle its events is the **keyboard_event_handler**. This function will perform different instructions, accordingly to the state of the keyboard.

- If the keyboard state is **KBD_IN_GAME:**
 - Call the keyboard interrupt handler to clear the buffer

- If the keyboard state is **KBD_GETTING_NICKNAME_1** or **KBD_GETTING_NICKNAME_2**
 - Call the interrupt handler
 - If scancode complete:
 - Call function that, given a code, retrieves its ASCII representation.
 - Check if the length of the nickname is equal to the maximum allowed. If so
 - Prepare the next game state, by changing the state of all the devices.
 - If the character retrieved is valid, add it to the end of the nickname.
 - Draw the letter on the screen in the correct position.
 - Spacing between letters is achieved by retrieving the width of each letter image and adding it to a variable
 - If the key pressed is the ENTER, and the length of the nickname is greater than 0
 - Prepare the next game state, by changing the state of all the devices.
- If the keyboard state is **KBD_CHOOSING_WORD:**
 - Call the interrupt handler
 - If scancode complete:
 - Check if code is either corresponding to the 1, 2 or 3 keys. If so, set the word.
 - If the current word is not null, it means that one of the above keys was pressed
 - Prepare the next game state, by changing the state of all the devices.
- If the keyboard state is **KBD_GETTING_GUESSES:**
 - Call the interrupt handler
 - If scancode complete:
 - Call function that, given a code, retrieves its ASCII representation.
 - If the key pressed is the ENTER, and the length of the guess is greater than 0
 - Set submitted guess variable to true.
 - Check if the length of the nickname is equal to the maximum allowed. If so
 - Set submitted guess variable to true.
 - If the character retrieved is valid, add it to the end of the nickname.
 - Draw the letter on the screen in the correct position.
 - Spacing between letters is achieved by retrieving the width of each letter image and adding it to a variable

Conclusions

Unfortunately, a feature that we thought we could implement from the beginning, but, due to lack of time, we could not: the serial port. However, its use would basically be to establish communication between the two players' computers and allow each one of them to play on their personal computers. This would be possible by exchanging data between the two computers by means of the serial port at a certain frequency rate. The initial idea was to, after a certain number of moves done by the drawer, send the state of the screen to the guesser's computer, so he could see on his screen what the other player was crafting and try to infer what it was from his computer. Then, a similar approach would be taken to send the information about the correct or incorrect guess by the guesser to the drawer computer, so he could see whether the other could guess correctly or not.

Imagining this project could be continued in the future, one simple feature that could be implemented to make the game more interesting and dynamic would be a point system, which would work something like this (note that it would only make sense to include this if player 1 and player 2 were always switching roles): the less time the guesser would take to guess what was drawn, the more points he would earn (of course, if he did not infer what was drawn in three guesses or less, he would gain 0 points). For the drawer, it would work the same way, because this would encourage him to do the draft as well as he could. The first player to reach a certain number of points or higher would win.

So, summing all up, we can say that we achieved almost everything we planned for (excluding what it is mentioned in the first paragraph) and some other small additional features that were not planned from the beginning but that, although simple, we found out were interesting and give a finer touch to the game itself (like the switch between light and dark mode, the different sizes of the rubber or even the switch roles feature).