Sketch Master

Paint and Gartic inspired game

Bachelor in Informatics and Computing Engineering

Computer Laboratory 2022/2023

T13G1

Students:

Daniel José de Aguiar Gago (202108791)

João Tomás Matos Fernandes Garcia Padrão (202108766)

Pedro Rafael Angélico Madureira (202108866)

Tomás Carvalho Gaspar (202108828)

Index

1. User’s instructions

1.1. Main Menu

The Main Menu offers two distinct modes: "Single Player" and "Multi Player." In "Single Player" mode, users can draw without time limits. Selecting the "Multi Player" option takes users to the Multi Player Menu. To exit the game, players can either press ESC or use the “Exit” button.

Figure 1 – Main Menu on Light Mode and on Dark Mode

1.2. Multi Player Menu

The Multi Player Menu provides options for selecting the roles of the players. Players can choose to be the drawer or the guesser. Pressing ESC or the “Exit” button allows users to return to the main menu. If a player chooses to be the "Start” they will attempt to draw a given word within a specified time limit. If a player selects the "Guess" role, they will try to guess what the other player is drawing. While waiting for the other player, a waiting screen will be displayed.

Figure 2 – Multi player Menu on Light Mode

1.3. Drawing page – one user

When a single user is playing, he will have access to a drawing canvas. On this canvas, he can freely draw without any time restrictions or assigned word prompts. Additionally, he can utilize the following options:

(1) Buttons to change the colour

(2) Buttons to change the size of the pencil

(3) Button to clear the canvas

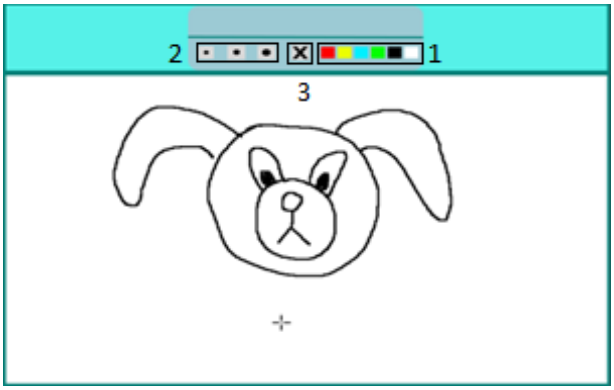

Figure 3 - Drawing canvas on single player

1.4. Drawing page – two users

Player 1 (Drawing): This player has a drawing canvas. He will receive a word prompt that he must attempt to draw within a time limit of 99 seconds. Player 1 can make use of the same buttons as the player on single player.

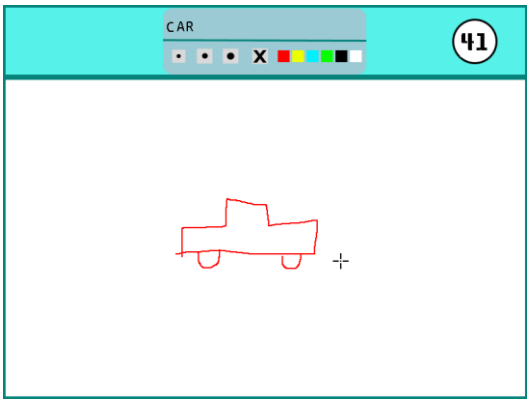

Figure 4 - Drawing canvas on multi player (player 1)

Player 2 (Guessing): This player's objective is to guess what the other player is drawing. They can enter their guesses by typing and pressing ENTER. If the timer runs out before the player guesses the word, a game over screen will be displayed to both players. However, if the player successfully guesses the word, a winning screen will be displayed to both players.

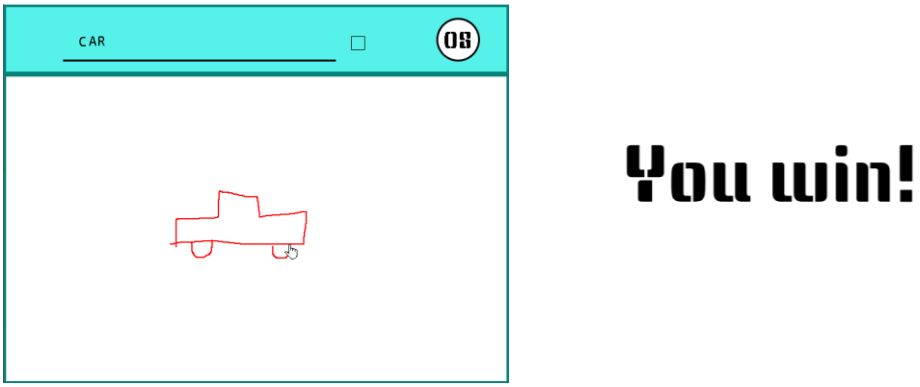

Figure 5 - Drawing canvas on multi player (player 2) + Winning screen

2. Project Status

2.1. I/O devices used

Device	What for	Int.
Timer	Controlling frame rate and the timer of the game	Y
Mouse	Click on buttons and drawing	Y
Keyboard	Writing and going back to previous screens	Y
Video Card	Application menus, screens, and font display	N/A
RTC	Opting between dark mode and light mode	Y
Serial Port	Transferring mouse info and words guessed between machines	Y

2.2. Timer

The timer is used to generate interrupts, allowing to measure the time that the players have to draw or guess the given word (99 seconds) and to display the finish screen for only 5 seconds. Furthermore, it is used to update the screen on every interrupt, by copying the content of the second buffer into the main buffer and drawing the cursor on the main buffer. To implement this, the following functions were used:

- timer_subscribe_int: used to subscribe the interrupts of the Timer 0
- timer_unsubscribe_int: used to unsubscribe the interrupts of the Timer 0
- timer_int_handler: used to increment the variable timer_counter which is useful to measure time

2.3. Mouse

The mouse is used to update the position of the cursor and, when the left button is pressed, depending on where the cursor is, to click on a button or to draw on the screen. To implement this, the following functions were used:

- mouse_subscribe_int: used to subscribe the interrupts of the mouse
- mouse_unsubscribe_int: used to unsubscribe the interrupts of the mouse
- mouse_read_data: used to read a byte from the output buffer
- mouse_ih: calls mouse_read_data
- mouse_data_reporting: used to enable/disable the data reporting from the mouse
- mouse_parse_packet: parses an array of bytes from the mouse into a packet

2.4. Keyboard

The keyboard plays a crucial role in the "Multi Player" mode, allowing the guessing player to input the correct word. Additionally, the ESC key is often used to interact with menus and options, basically it redirects to the previous page and in the main menu it closes the game. To implement this, the following functions were used:

- kb_subscribe_int: used to subscribe the interrupts of the keyboard
- kb_unsubscribe_int: used to unsubscribe the interrupts of the keyboard
- kbc_read: used to read from the keyboard controller output buffer
- kbc_ih: calls kbc_read

2.5. Video card

The video card is used throughout the whole project to display all the necessary information to the user, including the ability to draw, in the form of pixels on the screen. The video cards mode utilized was the 0x115, which is characterized by its 800x600 resolution and 24-bit direct colour model. Double buffering was utilized to keep track of what is behind the user's cursor after it moves. To implement all of this, the following functions were used:

- vg_draw_pixel: used to draw a pixel in one of the buffers
- vg_draw_point: calls the above function, used to draw a filled circle to the secondary buffer
- vg_draw_line: calls the above function, used to draw a line to the secondary buffer
- vg_draw_xpm: used to draw an XPM image in one of the buffers
- init_buffers: initializes the graphics mode and creates the second buffer
- swap_buffers: copies the content of the second buffer to the main buffer
- free_second_buffer: frees the memory allocated to the second buffer

You can see a more in-depth description of our double buffering implementation at ***Double Buffering***.

2.6. RTC

The RTC was used for generating alarm interrupts and reading date/time. When the program starts, the following function calls related to the RTC are made:

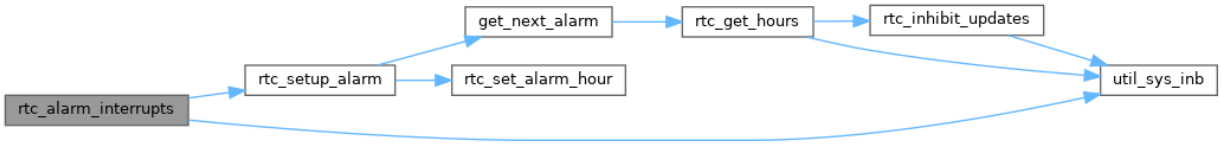

Figure 6 - Function Call Diagram for the rtc_alarm_interrupts function

- rtc_alarm_interrupts: enables alarm interrupts and sets up alarm
- rtc_setup_alarm: sets up the RTC alarm
- get_next_alarm: based on the current time, finds out when the next alarm should be
- rtc_set_alarm_hour: sets the RTC alarm hour
- rtc_get_hours: gets the current hours from the RTC
- rtc_inhibit_updates: inhibits or enables updates to the RTC registers, this is useful for changing the value of one of the time registers, to prevent inconsistencies caused by asynchronous updates.

On top of the above-mentioned functions, we also used some others:

- rtc_subscribe_int: used to subscribe the interrupts of the real time clock
- rtc_unsubscribe_int: used to unsubscribe the interrupts of the real time clock
- rtc_enable_int: enables interrupts for the real time clock
- rtc_disable_int: disables interrupts for the real time clock
- rtc_ih: when an interrupt is received, it automatically configures the next alarm
- wait_valid_rtc: used in the interrupt handler to make sure there isn't an update in progress while trying to read time/date registers

You can see a more in-depth description of our RTC implementation at ***RTC***.

2.7. Serial Port

The serial port was used for full duplex communication in our project. In the sp_init function we configured the device with a word length of 8 bits, one stop bit, no parity, a trigger level of 1, FIFO enabled, and interrupts for received data and transmitter empty. The default baud rate of 115200 was used. In sp_cleanup, the interrupts and the FIFO were disabled.

Other relevant functions for our implementation include:

- sp_subscribe_int: used to subscribe the interrupts of the serial port
- sp_unsubscribe_int: used to unsubscribe the interrupts of the serial port
- sp_get_status: which returns the contents of the Line Status Register
- sp_read: reads data from the serial port, and, if it doesn't have any errors, pushes it into the receive queue. This is done until there is no more data to be read
- sp_write: writes the item at the front of the transmit queue to the serial port
- sp_ih: checks what interrupt was received and invokes either sp_read or sp_write accordingly.

You can see a more in-depth description of our Serial Port / UART implementation at ***UART***.

3. Code organization/structure

3.1. Modules

3.1.1. keyboard

The keyboard module has the necessary functionality to interact with the user's keyboard input. It includes functions to subscribe and unsubscribe to the keyboard interrupts, as well as a function to read the make and break codes generated by the keyboard with those interrupts.

3.1.2. mouse

The mouse module has the necessary functionality to interact with the user's mouse input. It includes functions to subscribe and unsubscribe to the mouse interrupts, as well as a function to process the packets sent by the mouse with those interrupts and store the relevant information in a more readable data structure.

3.1.3. rtc

The RTC module provides the necessary functionality to interact with the real-time clock. It includes functions for initializing the RTC, getting the current hours and working with alarm interrupts.

3.1.4. queue

To use the serial port, we needed a software FIFO to facilitate things. Because we only intended to use the queue for serial port communications, our implementation is restricted to a transmit queue and a receive queue, which have the same implementation but are used for different purposes. For each of them we can push an item to the back, pop an item from the front and check if it is empty.

3.1.5. serial_port

The serial port module has the necessary functionality for the correct initialization and clean-up of the device driver. It also has a read function that reads all the data available on the receive buffer and places it on the receive queue, and a write function which writes the first item available on the transmit queue, to the transmit buffer. Depending on the interrupt type, the interrupt handler, which is also available on this module, calls one of the two functions.

3.1.6. timer

The timer module provides the necessary functionality to measure the time allotted for players to draw or guess a given word. It includes functions for subscribe and unsubscribe the

interrupts of the timer 0 as well as increment a variable that is useful for measuring the elapsed time.

3.1.7. utils

The utils module provides the necessary functionality for various utility operations. It includes functions to get the LSB and MSB of a 16-bit number, a function to facilitate calls to the sys_inb system call, and a function to convert an 8-bit binary-coded decimal to a decimal number.

3.1.8. video

This module provides the necessary functionality to interact with the VRAM and, consequently, the user's screen. It contains a function to draw a pixel to the screen and then multiple other functions based on the previous one that draw various shapes (lines and filled rectangles/circles) and a function to draw XPM images. We also have functions used to be able to work with a second frame buffer.

3.1.9. paint_window

This module is responsible for painting on the screen. It provides functions to paint different elements on screen, such as images and menus. Regardless on the buffer we want to draw, the called function may be different: If we want to draw the cursor, then we will call the paint_image_cursor function that will draw directly on the VRAM. For all other elements we call the paint_image. We explain the reason we use this approach in the ***Double Buffering*** section.

3.1.10. drawing_state_graphics

This module contains a detailed state machine for the cursor behaviour on the drawing page, as shown on ***Figure 7***. This state machine allows for different actions to be taken based on the current state and the type of cursor movement.

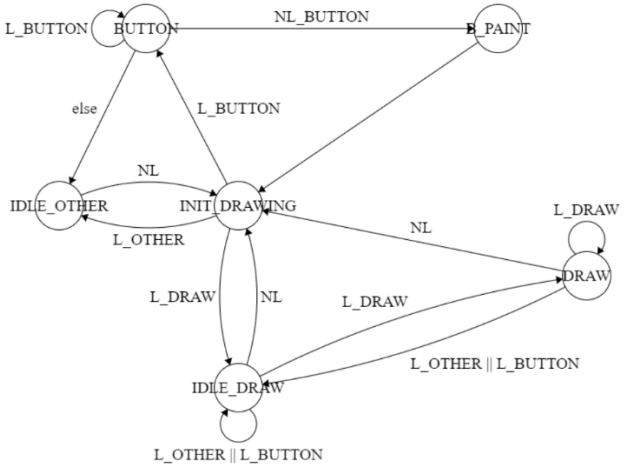

Figure 7 - State machine of the mouse for the canvas

The type of movement is defined based on the left button status (clicked – L or not clicked – NL) and on the cursor position (button – BUTTON, drawing area – DRAW or other area – OTHER). With the permutation of this information, we can build all the relevant movements, for instance, L_DRAW means that the left button of the mouse is pressed and the cursor in on the drawing area.

The main idea is to prevent unintended actions when the left button is clicked and the cursor moves across different areas, for instance if we click on the left button and start moving around, we could accidently change the colour of the pencil or even clear the canvas.

To achieve this, the state machine differentiates the various states based on the cursor's position, left button status, and movement. By establishing distinct states, the machine can manage the behaviour of the drawing application and enforce the desired actions while maintaining a clear separation of functionalities.

3.1.11. game

This module contains the game_state_handler function which is responsible for the game state machine. It processes all interrupts as well as keeps track of both player's current state and bases the interrupt processing on them. It is also responsible for sending and receiving information through the serial port which enables the possibility of it being a multiplayer game.

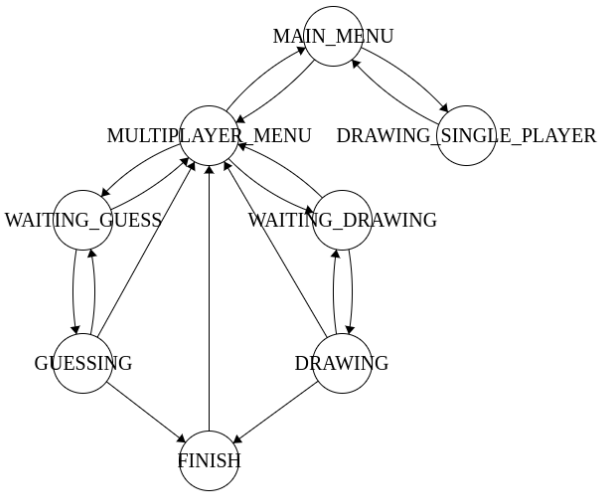

Figure 8 - State machine for the game

3.1.12. proj

This module contains the proj_main_loop function which is responsible for the main game loop. The function starts by subscribing to all relevant interrupts, initializes the serial port and graphics mode and then creates the cursor before entering the loop. The main loop is solely responsible for receiving interrupts and delegating its processing to the game_state_handler, described in the above module. After the loop is over, the function unsubscribes from all the interrupts that it previously subscribed to and exits from the graphics mode.

3.1.13. menu_state

This module is used to check if the user is hovering over the menu buttons and if he clicks them. Each button on the menu has a function associated with it that checks if the user's mouse is between the button's coordinates.

3.1.14. word_picker

This module is responsible for opening the file word_list.txt and choosing a random word that is present in it. This word is then returned to be used in the multiplayer game mode.

3.1.15. cursor

This module is used to handle all the necessary information related to a cursor. The cursor is an entity that is created at the beginning of the program and that we update whenever mouse interrupts are received.

The cursor has some attributes that define its behaviour: It stores information about the pencil's colour and size used when the user is drawing and it maintains the current position on the screen, represented by its X and Y coordinates.

In addition, it holds a cursor map, represented as an XPM array. This cursor map defines the appearance of the cursor. It allows for different cursor styles such as a precise cursor, a normal cursor, or even a hand cursor, depending on its position and depending on the mode (Light Mode or Dark Mode).

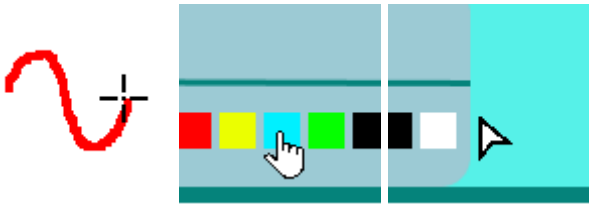

Figure 9 - Different cursors based on the position on Light Mode

Regarding the movement, the module provides two essential functions. The first function, move_cursor, updates the cursor's position based on specified deltas. It ensures that the cursor stays within the defined boundaries by considering the screen's limits.

The second function, draw_and_move_cursor, not only updates the cursor's position but also simultaneously draws the on the screen using the specified colour and thickness.

Lastly, it includes a utility function, is_on_limits, which allows checking if the cursor is within the limits it receives as parameters. This function is mainly used to check if the cursor is above a button or above the drawing canvas.

3.2. Weight of modules

Module	Weight (%)
keyboard	6
mouse	7
rtc	6
queue	4
serial_port	17
timer	4
utils	2
video	9
paint_window	5
drawing_state_graphics	9
proj	3
game	10
menu_state	3
word_picker	3
cursor	12
	100

3.3. Function call graph

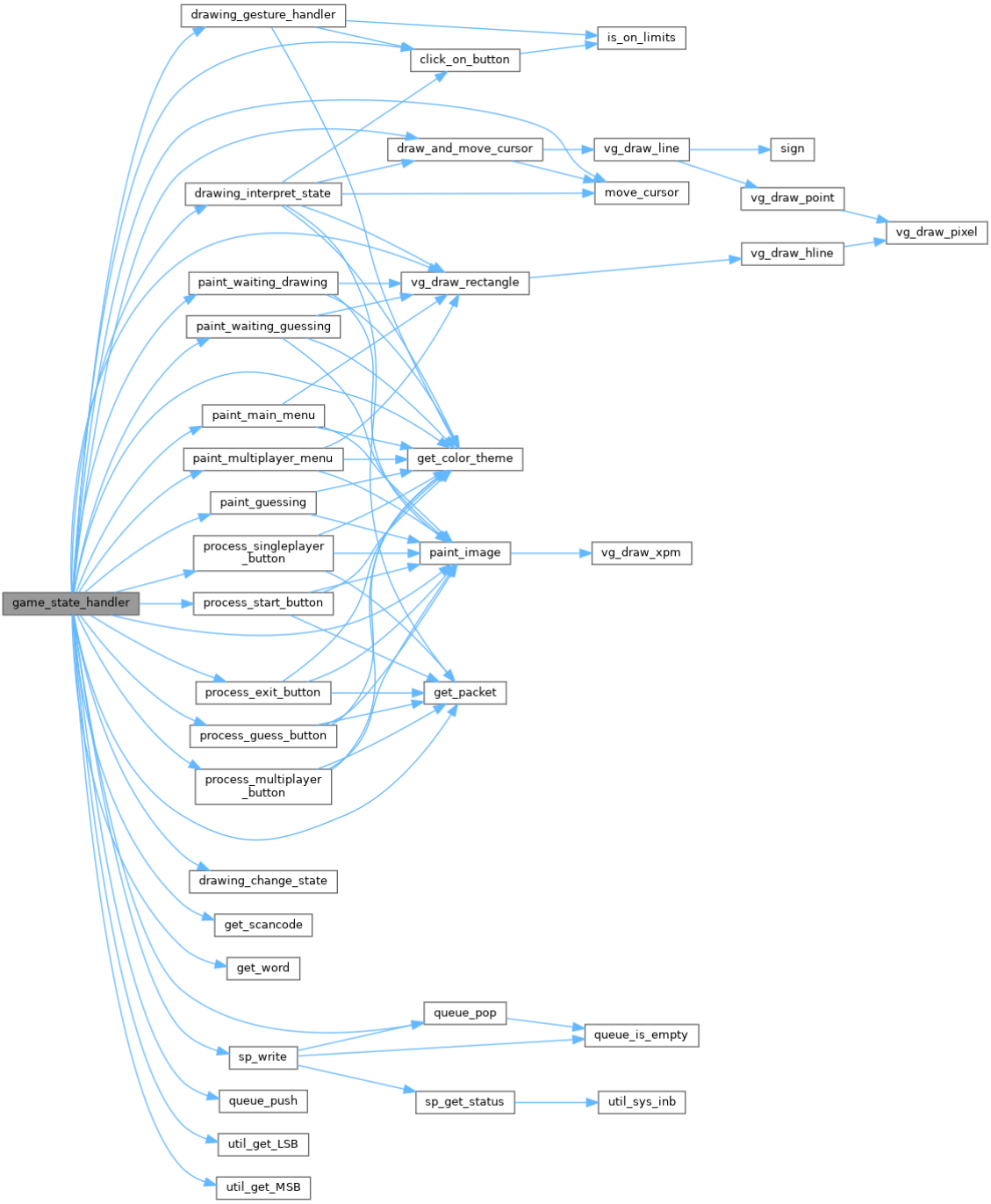

Figure 10 - Function call graph for game_state_handler

The above picture represents the function call graph for the game_state_handler, which is the main state machine of the program. We couldn't include the call graph for proj_main_loop due to Doxygen's limitations. The functions that the main loop would call, apart from the game handler are functions for subscribing and unsubscribing interrupts, interrupt handlers, functions for initializing the cursor and graphics buffers, painting the first menu, and setting up the alarm on the RTC.

4. Implementation details

4.1. RTC

The RTC required the implementation of the usual device driver functions for subscribing and unsubscribing interrupts, for initializing the device with the correct configuration and for resetting the configuration back to the original. On top of that, we have also implemented functions that help prevent data inconsistencies. These functions include checking if there is an update in progress (and waiting if that is the case) before reading the time, and inhibiting updates before changing the value of one of the time registers to prevent inconsistencies caused by asynchronous updates.

In the project we made use of alarm interrupts. The first alarm is set up by the function responsible for initializing the RTC. This function checks two macros, LIGHTMODE_HOURS and DARKMODE_HOURS, to determine which of these events happens first. Suppose LIGHTMODE_HOURS is set to 7 and DARKMODE_HOURS to 20, and that at the time of initialization of the real time clock the hour is 16. In that case, the alarm would be set up for 20. Every time an alarm interrupt is triggered, the next alarm is automatically configured using the same process.

One challenge we encountered during the implementation was related to the output format of the RTC. While we originally intended to use a binary format for the time representation, we faced issues with this approach as it did not always work even after setting the correspondent bit in register B. To overcome this obstacle, we opted to use BCD instead.

4.2. UART

Much like the previous devices, the UART also demanded the implementation of functions for subscribing and unsubscribing interrupts, initialization and clean up.

For the UART configuration, we chose a word length of 8 bits with the objective of simplifying data transmission and receival; enabled both received data interrupts and transmitter empty interrupts; set trigger level to 1, mainly with the transmission of individual letters in mind (for the guessing mode), and finally enabled the FIFO.

In addition to the use of the hardware FIFO, we also took advantage of two software FIFOs, a transmit queue and a receive queue. These have a quite simple implementation but play a crucial role in data transmission via serial port. The interrupt handler is responsible for invoking the functions that write and read data to and from the serial port. The read function reads every item that is in the hardware buffer and pushes it into the receive queue. On the other hand, the write function writes the item that is at the front of the transmit queue (if existent) to the

transmitter hardware buffer.

The toughest part of using UART was undoubtedly the protocol. The protocol was especially important for communication between the player who is drawing and the one who is guessing because it involved on top of the state change data many other types of data: mouse, pencil size, screen clear, pencil colour, correct guess, and wrong guess.

Every type of data except for mouse data has a unique 2-byte code that is sent before it. These codes had to be made of 2 bytes, so that they would not be mistaken for mouse data.

4.3. Double Buffering

We have made a slight change to the concept of double buffering due to memory optimization. Instead of drawing screens, menus, and the timer directly on the primary buffer, we now utilize a secondary buffer for these elements. During the buffer swap, the contents of the secondary buffer are copied to the VRAM, and we also draw the pixmap of the cursor directly onto the VRAM.

This approach allows us to avoid the need for an extra buffer dedicated to merging the content of the secondary buffer and the pixmap of the cursor. From a visual standpoint, there is no noticeable difference. However, by eliminating this extra buffer, we optimize memory usage.

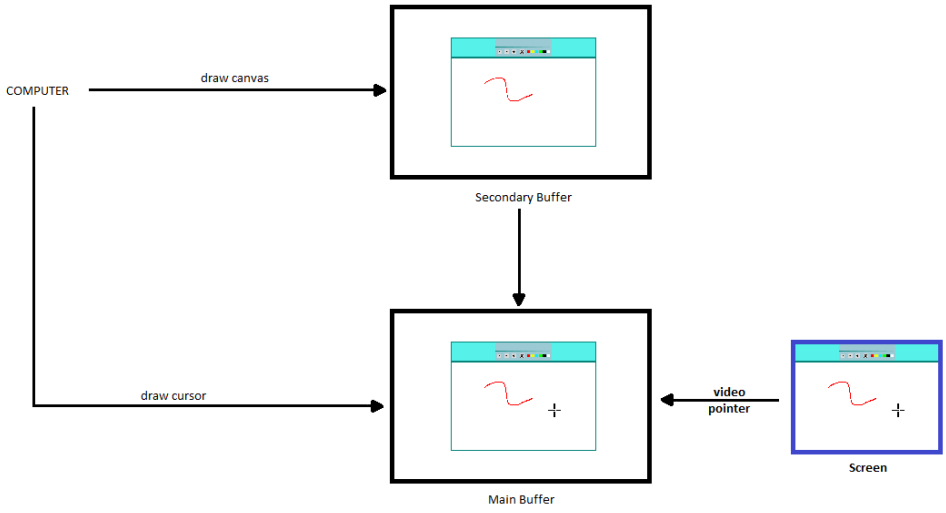

Figure 11 - Double Buffering representation

5. Conclusions

We were able to successfully complete all the assigned tasks and even went beyond the initial requirements by implementing the serial not only to transfer the guessed words, but also to transfer the drawing. Furthermore, there are some "like to haves" that were not initially defined but could greatly enhance the program if implemented in the future, such as adding additional options to the canvas, such as a paint bucket tool and various shapes.

Overall, this project allowed us to greatly improve our knowledge in the C programming language and in the use of the hardware interface offered by I/O devices.