TYPE RACER by T04G06

in LCOM - LEIC

Guilherme Pereira (up202007375@up.pt)

Gustavo Costa (up202004187@up.pt)

João Gigante (up202008133@up.pt)

Ricardo Cavalheiro (up202005103@up.pt)

Index

User instructions

The game

Our game is a type racer type game in which the objective is to type two given sentences as fast as possible.

The game can be executed with the command “lcom_run proj” while having the proj executable in the working directory. This can be achieved using either the MINIX terminal, or via ssh.

After doing so, the following screen will be presented to the user (Fig1):

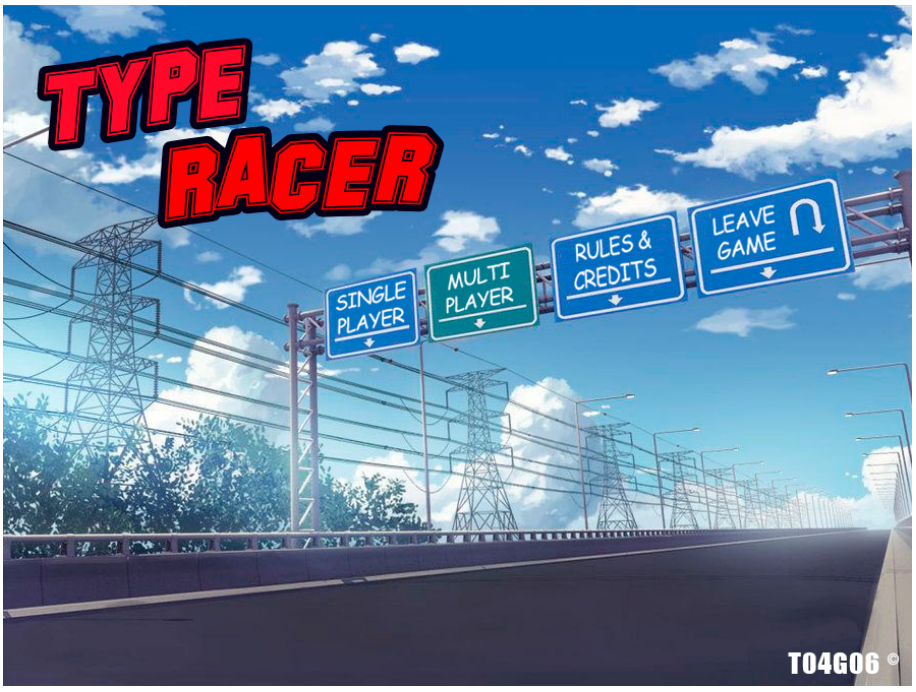

Fig1:Game’s start screen

As you can tell, multiple options are available. They can be selected either using the mouse and clicking on it, or using the keyboard arrows and pressing Enter.

Run-through of the available options:

- Single player: Single player version of the game, where the user competes against himself to try and type phrases as fast as possible;
- Multiplayer: Multiplayer version of the game, where (in theory) the user would compete against another user running the same version of the game on another machine. The goal here is to see who is the fastest. Sadly, this was not implemented due to some constraints, more on that later;
- Rules and credits: This one speaks for itself;
- Leave game: Exits the game.

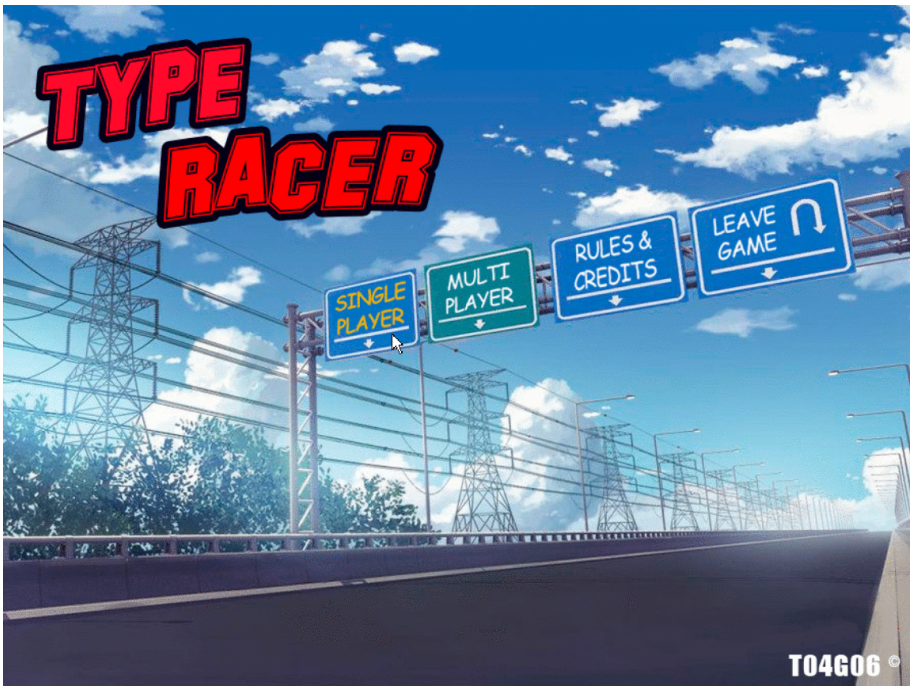

Fig2: Selecting the first option with the mouse

We recommend you select the “Single player” option, as that would be the most interesting feature of the game as it is.

One would think about playing multiplayer, but that feature isn’t functional. We found ourselves stuck between 4 projects that we had to simultaneously develop, and a mountain of errors, especially related to the mouse.

Singleplayer

If you did decide to choose the “Single player” option, you’d be greeted with the following screen (Fig3):

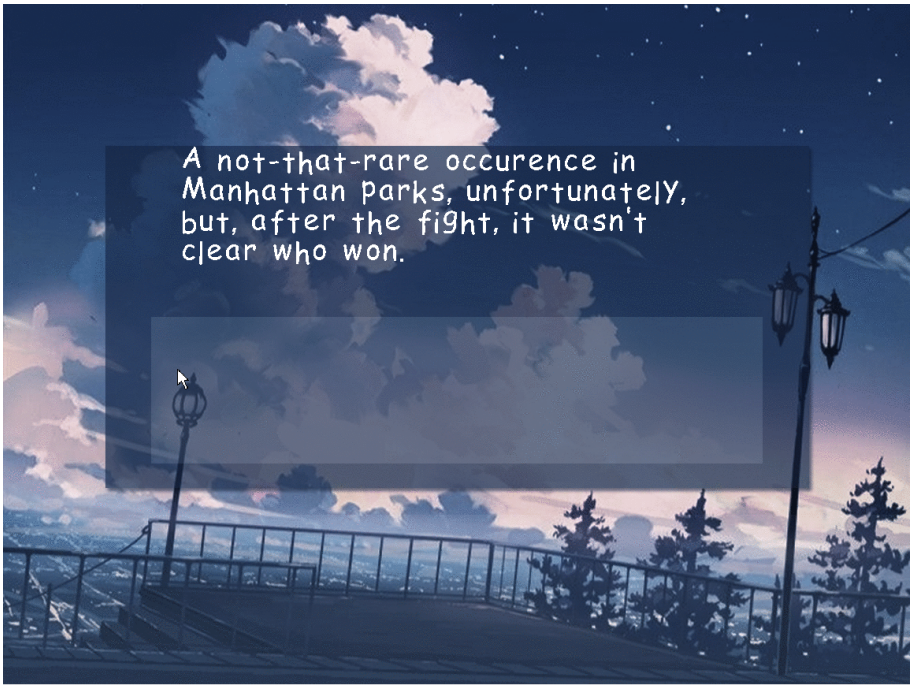

Fig3: Single player mode window

You’re prompted to type a sentence, and you can start doing so right away.

You can’t use the CapsLock key to represent uppercase letters, but you can use the left shift key.

You’ll know you’re typing in caps because a small indicator will popup on your screen, like so (Fig4):

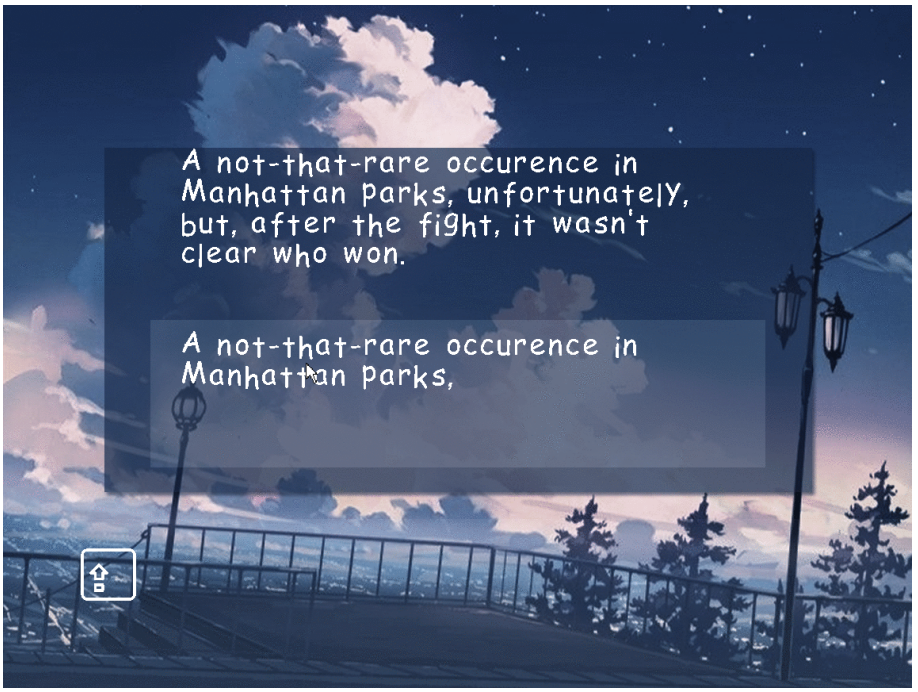

Fig4: Player written text and uppercase indicator

Obviously, since you can only move on if you type the sentence correctly, and you can make mistakes on your jouney to doing so, you're free to delete your progress anytime,using the Backspace key (Fig5 and Fig6).

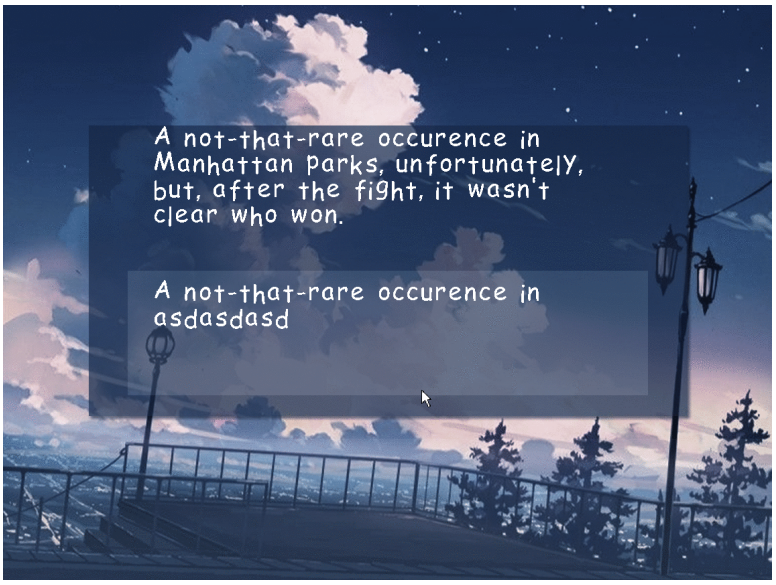

Fig5: Wrong text input

Fig6: Deleting the wrong text

Once you've finished typing the entire sentence correctly, an orange button will appear in the bottom right corner of the screen indicating that you can proceed onto the next sentence. (Fig7)

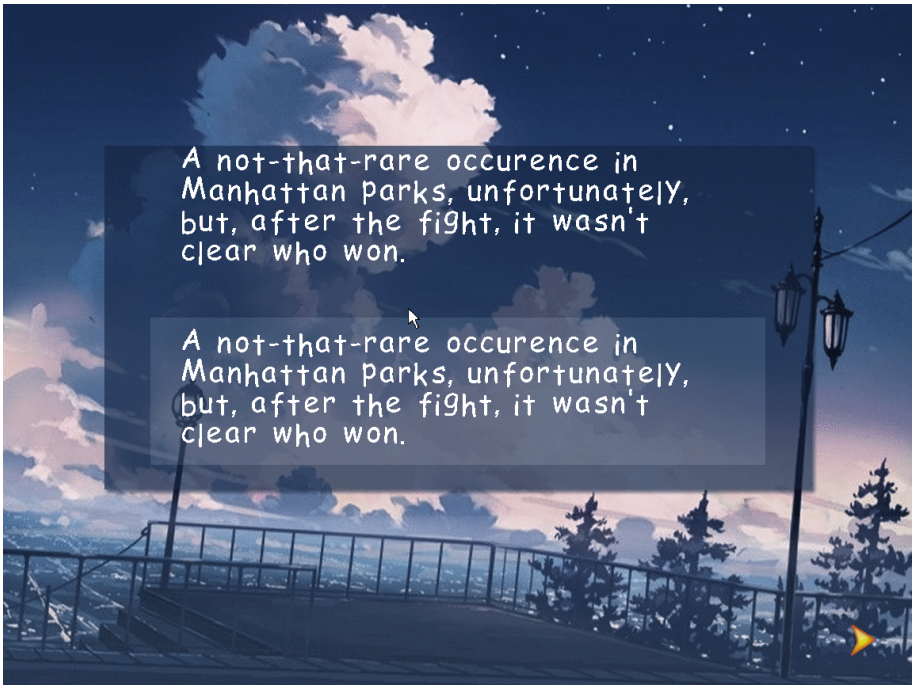

Fig7: Next phase indicator

Pressing the Enter key will display another sentence and the player has to repeat the process for a second time. Successfully doing so, will prompt a menu telling the player that they have won (Fig8).

Fig8: End screen

A player can also pause the game, if they're tired of typing so fast, by pressing on the ESC key. This pause menu allows them to either resume their game, or exit the game. (Fig9)

Fig9: Pause screen

Contrary to the first menu, the pause and finish menu can only be navigated using the up and down arrow keys.

Rules & Credits

Another selectable menu is the Rules & Credits, where the player can see a brief text explaining the game's objective (Fig10).

Fig10: Rules and credits menu

There's a credits section displaying all the group members.
To leave this menu, just press the ESC key.

Project Status

Initially we planned on having the player pick the order between three sentences. He had to type them all in order to win.

Since then, we've reduced the number down to two sentences, and he cannot pick between them, as they're picked randomly for him.

Alongside the sentence picking feature, we planned on having the typical word per minute (WPM) counter, featured in a leaderboard of the best players, but that required the RTC to be implemented, and we didn't have the time to do so.

The RTC would also allow us to have a small car, representing a player's progress in typing a certain sentence (Fig11)

Fig11: Car that would represent the player's progress at the bottom of the screen

As previously talked about, the game was intended to have a multiplayer gamemode, which required the Serial Port device to be used.

We found this to be very ambitious, considering what we managed to do in the first week, so we abandoned that idea.

The list of used devices and their uses goes as follows:

Device	Use	Communication method
Timer	Controlling frame rate	Interrupts
Keyboard	Menu navigation and typing	Interrupts
Video card	Displaying the game	N/A
Mouse	Main menu selection	Interrupts

Timer

This device's main function is to update the graphics and run the logic frequently.

All the menu's functions (changeMenuState(), singlePlayer_mode(), rules(), pause(), and win_menu_f()) are hosted here, as well as the function responsible for drawing the mouse cursor.

Keyboard

The main objective of this devide is to get input from the user so that they can type the given sentences. We have an array of letter struct objects that maps relevant information (Fig12) and is used by the kbd_handler() function to determine which character to draw.

```
typedef struct letter{
    char letter;
    uint8_t makeCode;
    xpm_image_t img;
    uint8_t * xpm;
    int shift;
}letter;
```

Fig12: letter struct definition with relevant information

While smaller, another function the keyboard has is menu navigation. Since we're using game states, it's really easy to make the keyboard change between menu buttons.

Video card

Without the video card, there would be no game, as it's all visual and needs to be displayed on screen. Drawing pixel by pixel, we managed to display anything, from whole images to small cursor pointers.

We chose a mode(0x14C) that allows us to have a 1152x864 screen with direct color mode and 32bits per pixel ((8:)8:8:8), which is more than enough for our humble typing game.

Double buffering was our main drawing mechanism, achieved through the double_buffering() function call..

As good as they may seem, don't be fooled, because our game doesn't have animations. It's all drawn on screen and just changing between two different xpms

Stumbling into performance issues, since drawing over one million pixels per timer interrupt, we had to resort to refreshing the screen only on the small parts that needed to be refreshed. This was done by the clear_xpm_with_cover() function.

All the text that's manually written by us was done so using individual xpms, that we also made, using Comic Sans as a font.

Mouse

The heart of our problems, the mouse started to be a headache, because of the immense lag it produced.

Turns out, it wasn't really the poor mouse's fault, and we later realized that drawing multiple xpm was the source.

We didn't have that much time to make it very useful or, at least, as useful as we'd like, but we managed to allow the user to hover over the different starting menu options and even left click on them.

The function responsible for all the mouse logic is its handler, mouse_handler(), that updates the mouse's position on the screen, as well as check for collisions with invisible boxes around the menu items (Fig13).

```
if(game->mouse.mouse_x + p->delta_x >= 0 && game->mouse.mouse_x + p->delta_x + 15 <= (int) get_hres()){
    game->mouse.mouse_x += p->delta_x;
}
if(game->mouse.mouse_y - p->delta_y >= 0 && game->mouse.mouse_y - p->delta_y + 26 <= (int) get_vres()){
    game->mouse.mouse_y -= p->delta_y;
}

Point mouse_pt = {game->mouse.mouse_x,game->mouse.mouse_y};
if(isInsideOpt(singlePlayerOpt, mouse_pt)){

    if(selectorMenu!=1){
        selectorMenu = 1;
        startMenuEntry = SINGLE;
        game->state.draw = true;

    }
    if(game->mouse.lmb){
        game->state.mode = SINGLEPLAYER;
        game->state.start = true;
        current_menu = game_background;
        current_menu_img = game_background_img;
    }

}
```

Fig13: Mouse handler code snippet for updating posittion and checking for "Single Player" option selection.

Code Organization

The modules that were necessary to run the program are the following:
We did not include the device modules that didn't have addeed/altered code from the labs.

Game.c/Game.h

This is the module where basically all the functions of the games were made. We have a game_init() that initializes the game and handles all the Game States. It also has a while loop that handles all the interruptions until the game ends. This module corresponds to a 70% weight of the entire project.

Letters.c/Letters.h

This module is responsible for storing all available sentences and characters xpms, allocating, loading and getting rid of them. The functions load() and loadSentences() are both called at the start of the game, to make sure all needed resources are available when needed. This module corresponds to a 9% weight of the entire project.

Vg.c/Vg.h

This is the module responsible for all the visual heavy lifting, providing the user with the correct images so he can visually understand what they're supposed to do. It's versatille. We added a function to load all the xpms that were needed for the execution of the game. This module corresponds to a 15% weight of the entire project.

Xpm.h

Xpm.h is simply a file that includes all the needed "xpm_name".h files, for convenience. This module corresponds to a 1% weight of the entire project.

Mouse_collision.c/Mouse_collision.h

This module is used for, well, checking if the mouse collides with the invisible rectangles over each main menu option. It has defined a struct Point, which represents coordinates, and function, isInsideOpt(), that takes a Point vector representing the rectangle, and the coordinates of the mouse. This module corresponds to a 5% weight of the entire project.

Function call graph

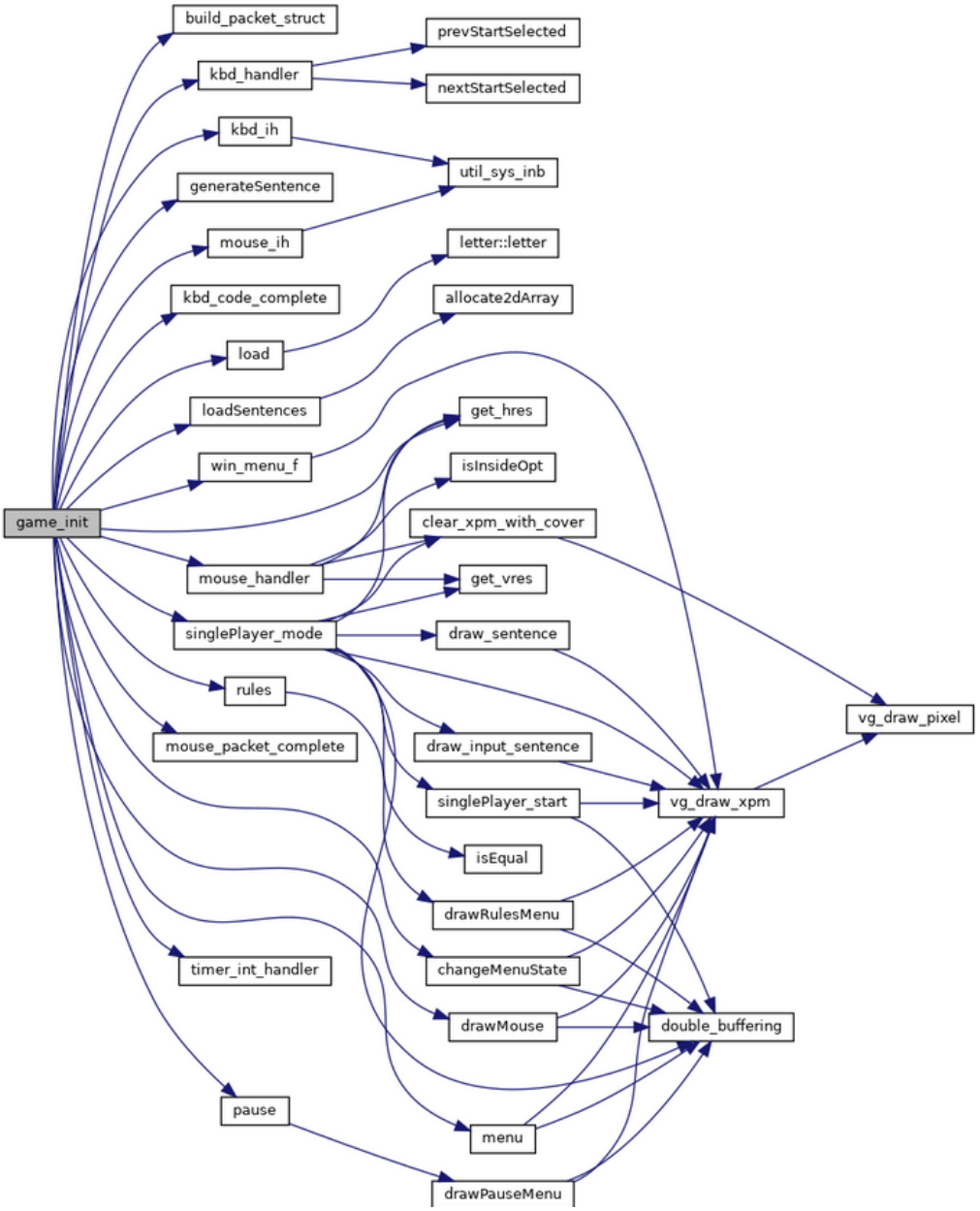

Fig14: Function call graph

Work division

The following table represents the work division amongst all the group members, relative to the entire project, by module, so 30% would mean 30% the entire game, and 100% of the module, if the module represented 30% of the game.

Part(total %) / Member	Guilherme	Gustavo	João	Ricardo
Game.c/h (70%)	0%	23.3%	23.3%	23.3%
Letters.c/h (9%)	0%	0%	0%	9%
Vg.c/h (15%)	0%	5%	10%	0%
Xpm.h (1%)	0%	0%	0%	1%
Mouse_collision .c/h (5%)	0%	5%	0%	0%

It’s important to note that the labs used as foundation are not taken into account in this work division table, as it’s “mandatory” work to all group members, by participating in the lab classes.

Implementation Details

Overview

The implementation of the game's features was possible due to the knowledge obtained in the Theoretical classes and also in all the Labs.

Interrupts and game states

One aspect of our game was that we have only one while loop which handles all the interruptions of the devices and inside each interruption we have a switch case for every Game State.

Timer

Concerning the timer, during the labs we used the timer's interrupt only to increment a global counter. This is more than sufficient as we "only" use it to update the game's logic and draw everything needed in each frame.

Since we couldn't manage to have the RTC working, we tried to have the timer count passing time, altough, not in seconds.

Drawing and optimization

To prevent our game from being inefficient and slow we had to figure out a way to not draw every visual at each timer interrupt. We solved that by cleaning only the specific part of the screen where we will draw the corresponding xpm (This increased considerably the performance of some devices, especially the mouse). Also we only draw the screen if the state of the game changes making the game fluid.

During labs we learned to display xpms of images, however in this project we had to deal with images of letters which had to be displayed according to each other's width.

Keyboard

In regards to the keyboard, it had a big role as this was a typing game. We had to learn how to deal with each specific makecode. The keyboard was used in many features, such as the user needing to be able to write a phrase to match the sentence provided and also to change between buttons in each Menu.

Double buffering

Another feature implemented was double buffering. Double buffering helps speed up the program because it overlaps the process of drawing.

Memory allocation/deallocation

Since we need to allocate a lot of space, in order for the code to be clean, and for MINIX to be healthy when the game ends, we had to deallocate all the memory we allocated for all the letters and sentences.

Conclusions

Even though we're proud of our work, we feel like it's lacking many characteristics of a type racer. The car is missing, the WPM counter is missing, there's no leaderboard so you can't even compete with others.

Another thing we feel is lacking is support materials for students. The slides are hard to follow and the lab handouts are hideous and crowded with information that's not necessarily needed. The tests are written in a way better format, as they're more straightforward, and the handouts should resemble them.

All in all, LCOM is a fun subject that we enjoyed partaking in, but it's not one easy to master.