Faculdade de Engenharia da Universidade do Porto

Laboratório de Computadores 2022/23

Space Enemies

By T07G05

Miguel Marinho up202108822

Rubén Fonseca up202108830

António Matos up202006866

Eva Carvalho up202006379

Contents

1. About

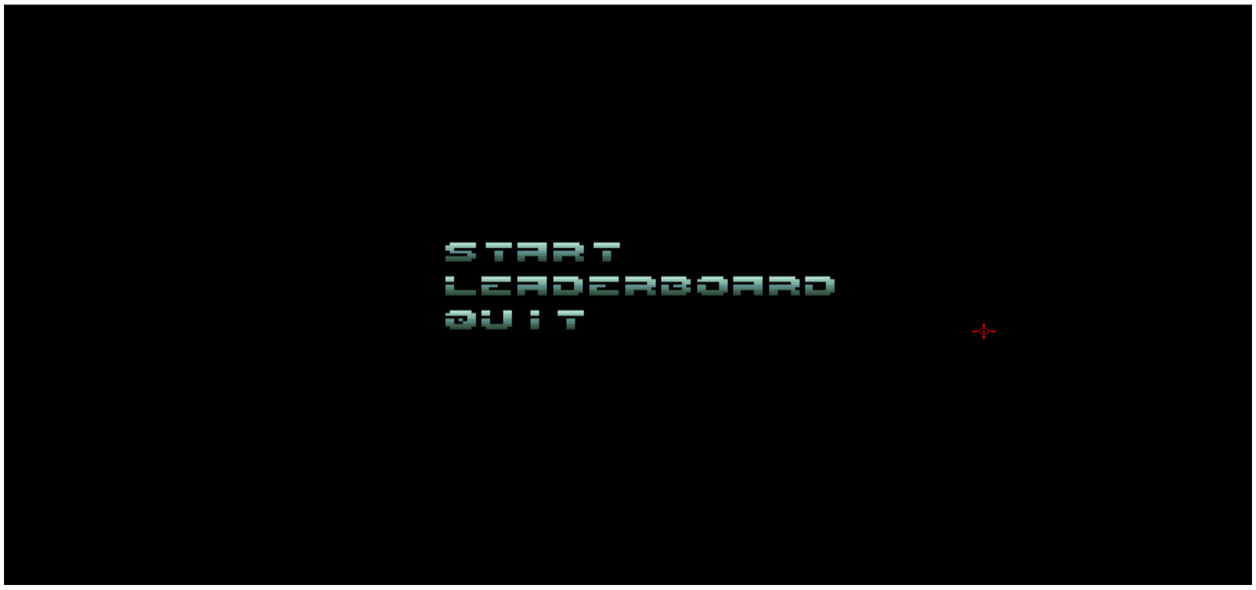
- Game Concept

Our objective was to recreate the hit game *Space Invaders* using the C programming language and using MINIX's system calls. We also wanted to implement mouse functionality so we could make the game even more frenetic than before. The goal of the game is to kill every single enemy before the time runs out or before your ship is destroyed.

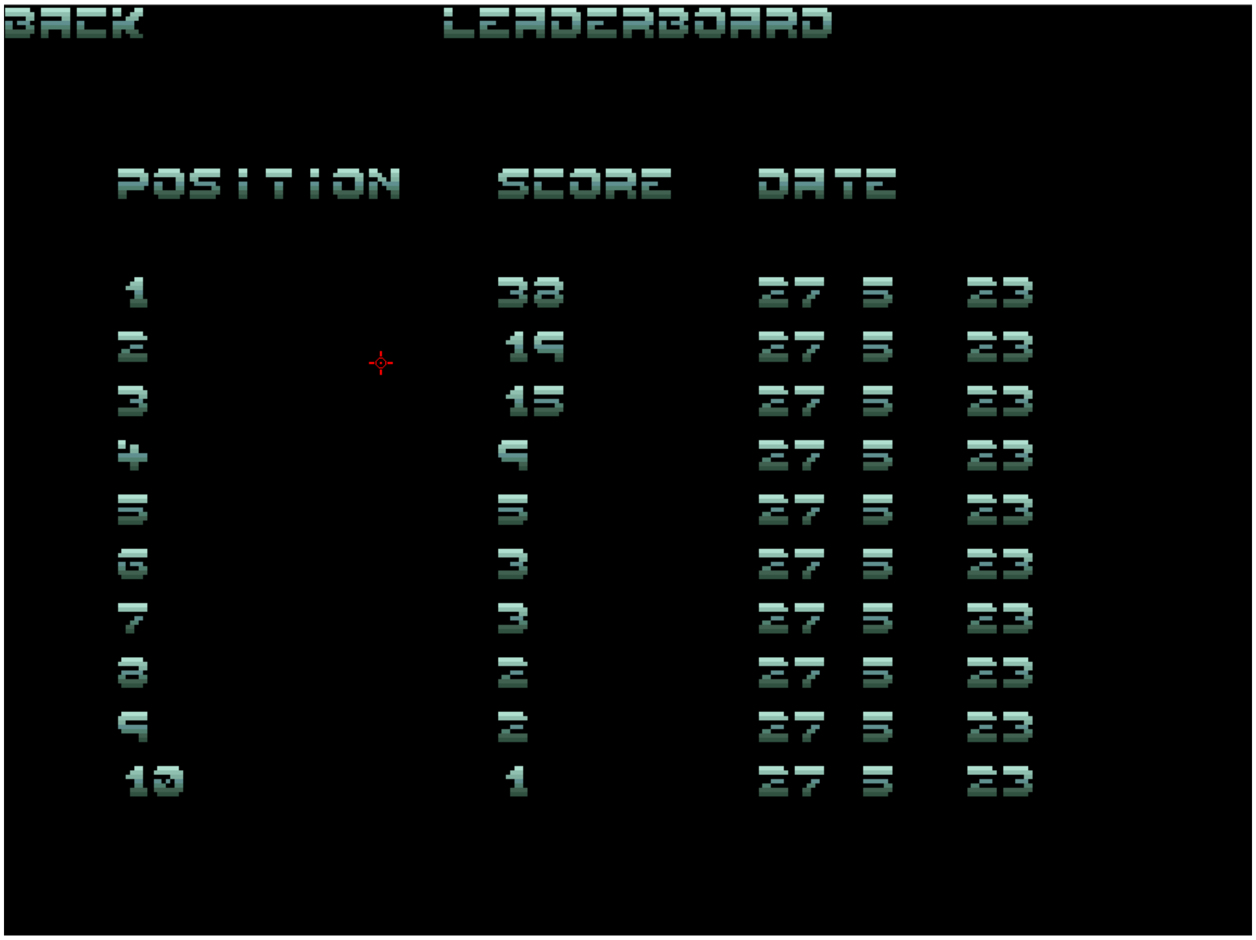
- User Instructions

The way the game works does not deviate much from the original, other than the controls of course.

In the menu, the user will be able to quit the game by pressing the ESC key or using the mouse to click on the quit button. The user will also be able to see the high scores previously registered by clicking the leaderboard button and play the game by pressing the start button.

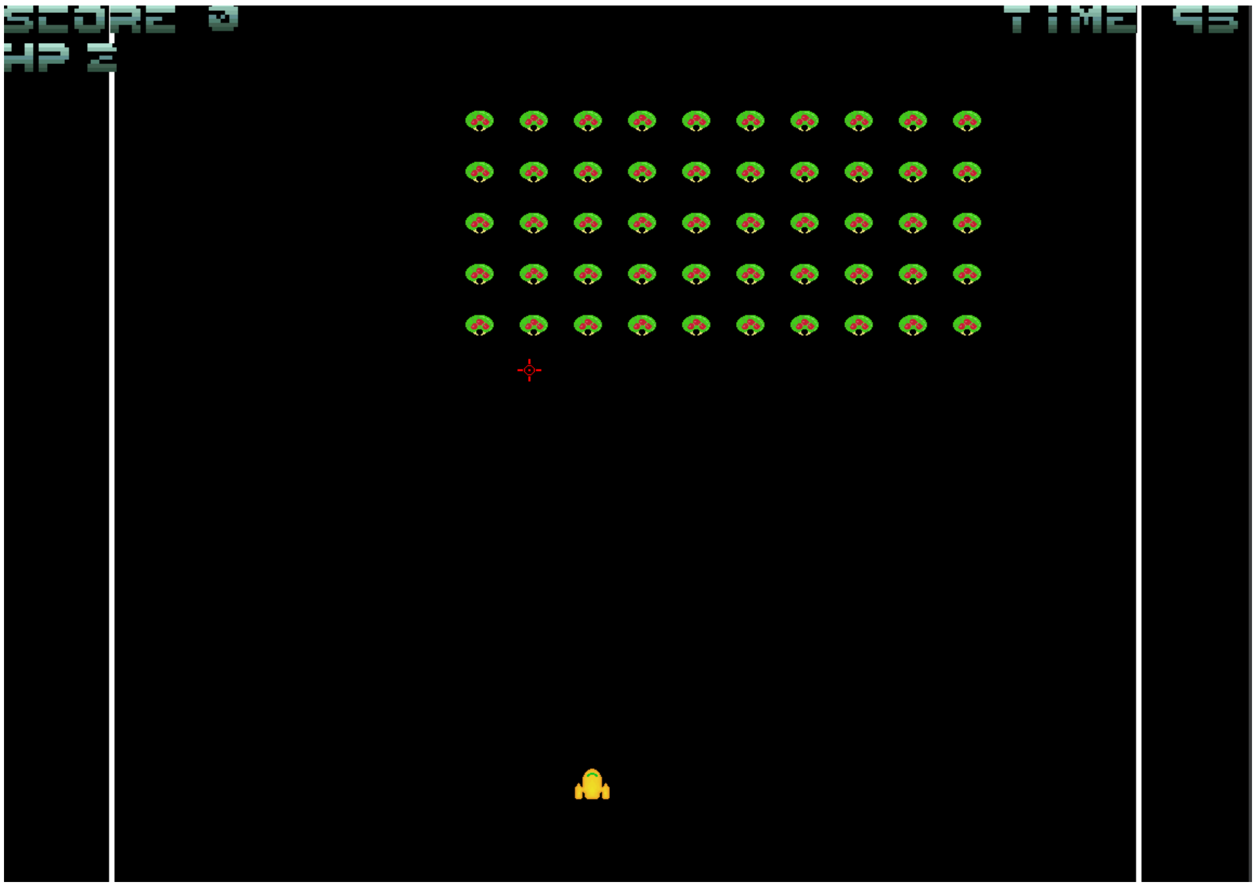


Once on the leaderboard screen, the user will see the 10 best scores displayed in a readable way as well as the date they were achieved. You can quit the game by pressing ESC or go back to the menu by pressing the BACK button with mouse.



POSITION	SCORE	DATE		
1	38	27	5	23
2	19	27	5	23
3	15	27	5	23
4	9	27	5	23
5	5	27	5	23
6	3	27	5	23
7	3	27	5	23
8	2	27	5	23
9	2	27	5	23
10	1	27	5	23

While playing the game, the screen will print relevant information like how many hits you can take before a gameover (HP), the time you have left and the score. You can shoot by aiming with the mouse and pressing the left mouse button. You can also move your ship by pressing either the A or D keys. You can also "dodge enemy bullets" by shooting them with your own bullets. You can quit the game anytime by pressing the ESC key but be aware that it won't save your score.



- Controls

In the menu and in the leaderboard page you can only use the mouse. Press the left mouse button to go to a page and move the mouse to move the cursor. While in the game, you can move your ship to the right by pressing the D key and move to the left by pressing the A key. You shoot with the left mouse button and aim using by moving the mouse.

2. Project Status

Device	What for	INT.
Timer	Controlling frame rate, animation duration and time to finish the game.	Y
KBD	Controlling the player ship and quitting the program.	Y
Mouse	Menu navigation, aim and shooting.	Y
GPU	Display the game. Double buffering also works.	N
RTC	Fetch the date a high score was achieved.	N

- GPU

Mode 14c was used with a resolution of "1152x864" with Direct color mode and transparency. We used the function vg_init() to achieve this.

Double buffering implemented via copy. We had to implement this since the flickering looked terrible and affected the enjoyment of the game. The function is called "flip_buffer()" and can be found in gpu module.

Sprites, collision detection and animations used. We animated the enemies' ships so they looked better. Collision detection works for both the enemies and the player. You can also shoot the enemies' bullets. The animations were handled in the main loop with the help of the timer. Most sprite functions can be found in the sprite or draw_elements modules.

We also used a font so we could write whatever we wanted without having a xpm file for every single character. Functions "draw_string" in "draw_elements" and "vg_draw_char" in "gpu.c".

- KBC

The keyboard was used to control the movement of the player's ship and to provide a shortcut to quitting the game. Most of this was handled inside the game loop.

- Mouse

We used both the buttons and position of the mouse. We use the buttons to shoot or click on an option and the movement to aim. This was handled during the main loop, and in the functions (mouse_*_state()).

- Timer

Used to control the framerate the animations and the counter of the game.

This was handled during the main loop, and in the functions (timer_*_state()).

- RTC

The RTC was used to fetch the date a certain high score was achieved. We simply read the register using the function (fecthCurrentDate()) the game was finished and saved it in a separate file.

3. Code Organization/Structure

- proj.c

Call the main loop and initialization of the bit_nos.

1%.

- utils.c

Some utils functions used in the timer implementation. This is not really used since we didn’t end up changing the framerate.

0%.

- timer.c

Increments the global counter as an interrupt handler, subscribes interrupts, changes the frequency of the timer and displays its configuration.

10%.

- sprite.c

Contains the bullet(contains a sprite field and other fields relevant to the speed and destination of the bullet) and sprite data structures. It also keeps all the sprites as global variables so they can be accessed anywhere in the code.

10%.

- rtc.c

Contains the functions that fetch the date to save a high score. Also contains the Date data structure that has a month, day and year fields.

4%.

- keyboard.c

Subscribes and unsubscribes interrupts for the KBC. Also has functions that write and read from the KBC using pooling. The interrupt handler gets the information given by the KBC, and that information is then parsed by a function.

10%.

- mouse.c

Very similar to the keyboard module but adapted to the mouse. Also has functions to enable/disable data reporting.

10%.

- gpu.c

Module that handles the low-level gpu functions. It initializes the gpu to a certain mode, maps the video memory, handles the double buffering and draws the font's characters as well as the xpms and regular horizontal and vertical lines.

15%.

- game_loop.c

Contains the main loop of the project where we change states and where interruptions are "caught".

15%.

- draw_elements.c

Module that calls functions in the gpu module to clear and draw on the screen. This module also handles bullet creation and leaderboard management.

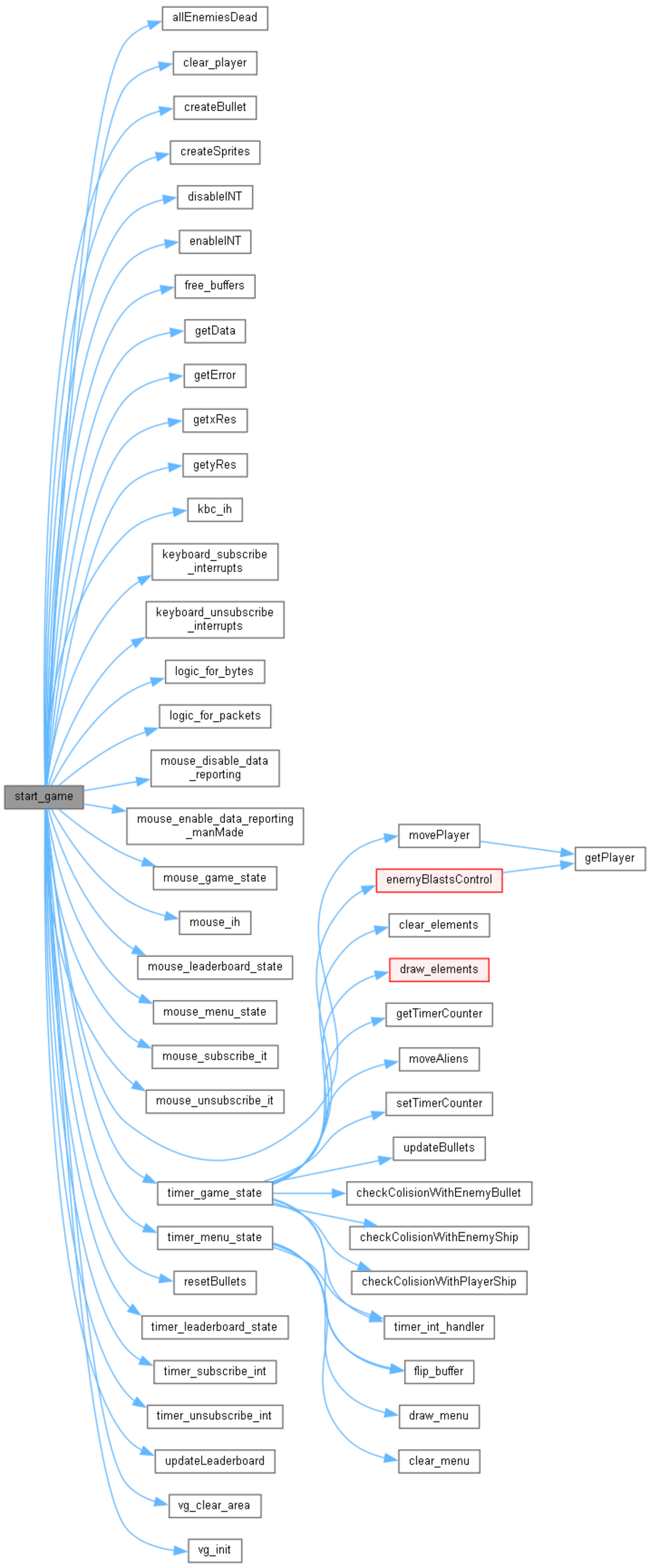
15%.

- controller.c

Module that handles the controls, collision system and the “AI” of the enemies.

10%.

In the next page we present you the function call graph.



start_game
allEnemiesDead
clear_player
createBullet
createSprites
disableINT
enableINT
free_buffers
getData
getError
getxRes
getyRes
kbc_ih
keyboard_subscribe_interrupts
keyboard_unsubscribe_interrupts
logic_for_bytes
logic_for_packets
mouse_disable_data_reporting
mouse_enable_data_reporting_manMade
mouse_game_state
mouse_ih
mouse_leaderboard_state
mouse_menu_state
mouse_subscribe_it
mouse_unsubscribe_it
timer_game_state
timer_menu_state
resetBullets
timer_leaderboard_state
timer_subscribe_int
timer_unsubscribe_int
updateLeaderboard
vg_clear_area
vg_init
movePlayer
enemyBlastsControl
clear_elements
draw_elements
getTimerCounter
moveAliens
setTimerCounter
updateBullets
checkColisionWithEnemyBullet
checkColisionWithEnemyShip
checkColisionWithPlayerShip
timer_int_handler
flip_buffer
draw_menu
clear_menu
getPlayer

4. Implementation details

- ## OOP

Object oriented programming was something we had present in our project development, by having static variables across different modules and by making use of struct objects with their own data.

- ## Frame generation

We implemented a double buffer strategy. By flipping between buffers, we can avoid problems like the overlapping of different frames and delays in displaying elements, since the whole frame is already generated and stored in the buffer when it is going to be displayed.

- ## RTC

We implemented RTC to fetch the date of the high score achievements to our leader board, which was something we had to learn on our own but we can admit it would have been easier to implement if we had dedicated some lab time for it.

5. Conclusions

- **Problems**:

Displaying letters: Since we were using a font xpm instead of having an individual file for each letter of the alphabet, we ran into some issues when traversing through it to display specific elements.

Minix running make: we had some issues configuring our project to run in minix, which made our progress more difficult and we struggled to solve it.

- **Lessons learned:**

We, as a group, are satisfied with our project's final state. However, we think the development could have gone more smoothly had we had more time to dedicate to it, since this happened concurrently with every other classes' projects, which includes different group members and schedules, this can difficult time and task management within the group, which ended up happening in ours. So we would say this was an issue we struggled with but also learned to overcome.