

Ω CLASS

Final Report

LCOM 2021/2022

T09-G03

João Alves - up202007614
Adam Nogueira - up202007519
Ana Costa - up202007602
Rúben Monteiro - up202006478

Index

Index	2
User Instructions	4
1.1 Description	4
1.2 Main Menu	4
1.3 Gameplay	5
1.4 Controls	6
1.5 Scoring	7
1.6 Game Over	8
1.7 Leaderboard	9
Project Status	11
2.1 Devices Used	11
2.1.1 Timer	11
2.1.2 Keyboard	11
2.1.3 Mouse	12
2.1.4 Video Card	12
2.1.5 RTC - Real Time Clock	12
Code Structure	13
3.1 Modules	13
3.1.1 utils.c	13
3.1.2 timer.c	13
3.1.3 keyboard.c	13
3.1.4 mouse.c	13
3.1.5 video_card.c	14
3.1.6 rtc.c	14
3.1.6 sprites	14
3.1.7 leaderboard.c	14
3.1.8 game.c	14
3.1.9 proj.c	15
3.2 Function call Graph	16
Implementation Details	17
4.1 State	17
4.2 Animations	17
4.3 Collision Detection	18

4.4 Event Driven	18
4.5 Object Orientation	18
Conclusions	19
5.1 Opinion	19
5.2 Problem	19
5.3 Main achievements	19

1. User Instructions

1.1 Description

In this project we developed a game, called Omega Class. The main objective of the game is to kill as many aliens as possible before the inevitable death of the player, since in an alien invasion, Humanity is doomed to die. By shooting the aliens, the player can survive for a little more time, but if an alien survives for too long, the player will die.

1.2 Main Menu

When you run the application, the first thing you see is the Menu.



The Menu gives 2 options, which are very simple:

- The first button, PLAY, just takes the user to the game and starts playing right away, after all, there is no time to spare in an alien invasion.
- The second button, EXIT, just exits the game, and stops the execution of the program.

1.3 Gameplay


When the game starts, after a few brief moments, the aliens start appearing one after the other. The spawning rate of the aliens may start slow, but their spawning rate increases as time goes by. Fortunately for the player, there is a limit to how fast the aliens can appear.

The player should use the mouse to point at the aliens, and shoot them.



As you can see in the image above, the aliens can spawn in random positions on the screen, and the player has no way to predict where the next alien will appear. The player should try to kill the aliens in the same order as they come into sight, because the death of the player is not determined by how many aliens are on the screen, but by the maximum time that any alien is on the screen. Although, it's possible for players to come up with different strategies for the game.

Although unrealistic in an alien invasion scenario, we have a pause feature that allows the player to rest for as long as necessary. The game completely stops during this time and the only thing displayed is the message PAUSED.

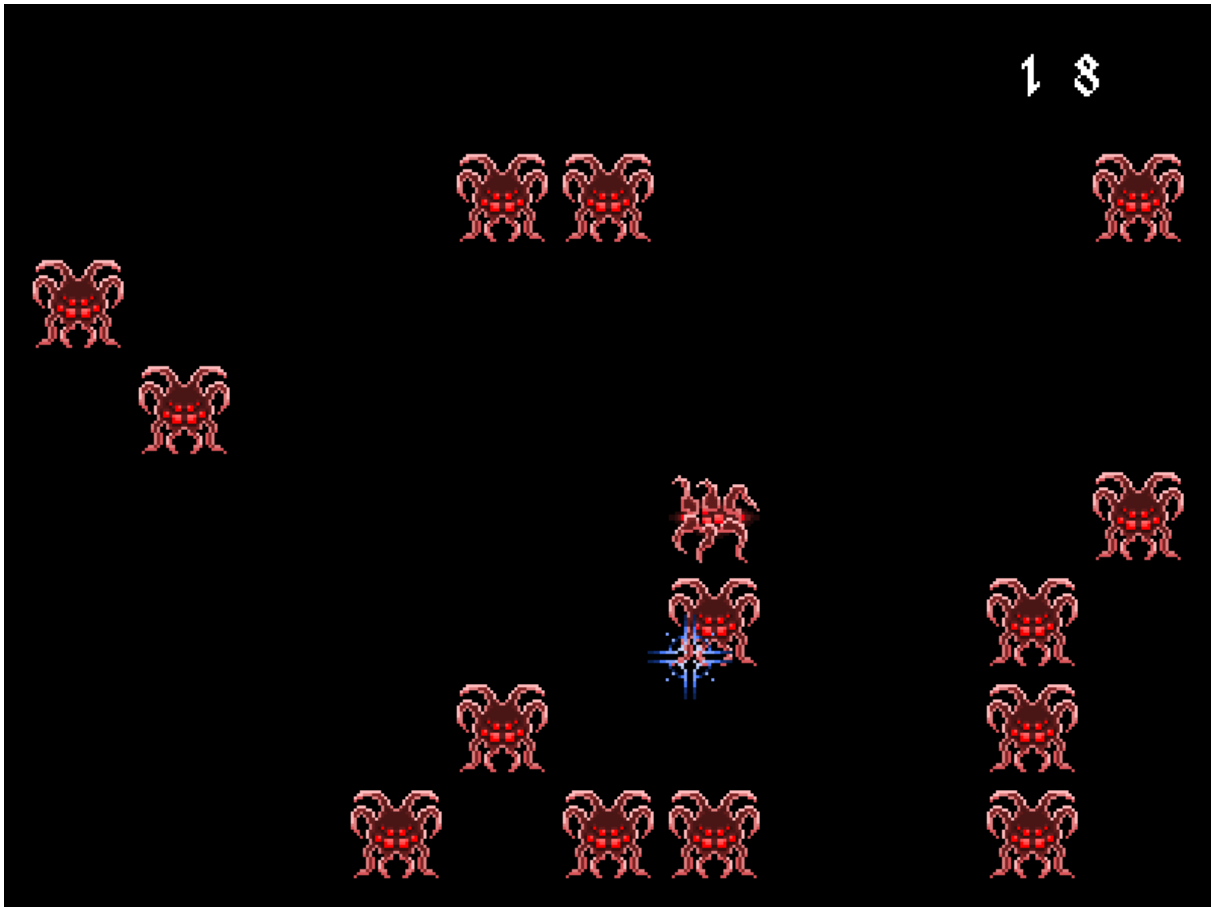


PAUSED

1.4 Controls

- **Mouse left-click** : shoot.
- **Esc** : quit the game.
- **P** : pause the game

1.5 Scoring



On the top right corner of the image, you can also see a number. That number represents the current score of the player, which is the number of aliens killed. Evidently, the number increments each time the player kills an alien. This score is later used for the leaderboard.

1.6 Game Over

As mentioned before, if an alien is left alive for too long, the player dies. After the player's death, a GAME OVER screen is displayed.



In this state, all that is left is the game over message, the crosshair, and the surviving alien that stayed alive for too long and killed the player. I believe this feature is quite helpful, since it allows the player to know where they made the mistake during the game, and maybe improve in the next game.

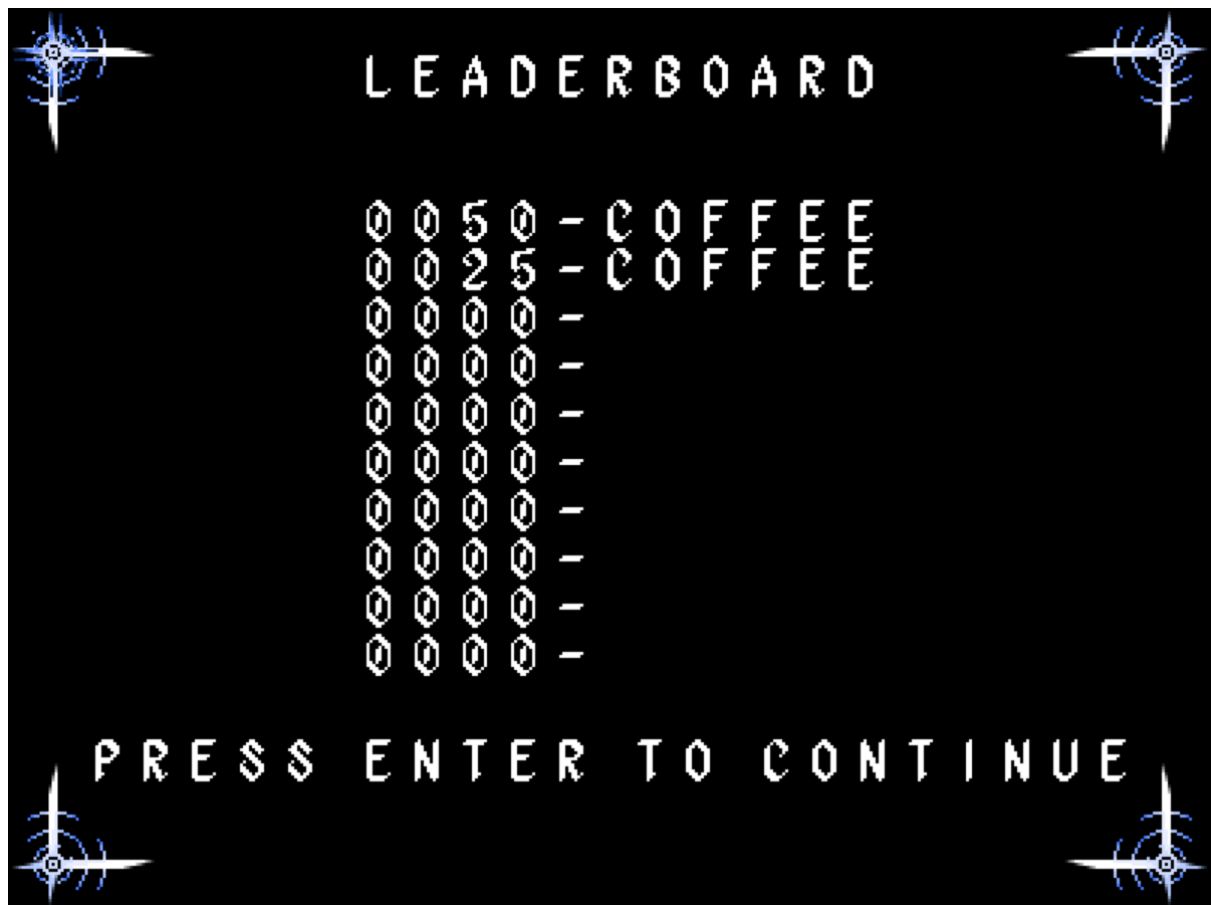
After a short time, the game over screen disappears, and changes to the next state, the leaderboard.

1.7 Leaderboard

The game also stores the best scores of all the players. At the end of the game, if the player was good enough to be added to the leaderboard, the game asks for the player's name.



If the user didn't obtain a good enough score or after the user writes their name and presses enter, the game will show the current leaderboard.



As you can see in the image, the leaderboard records both the name of the player, as well as their own score. The results appear ordered by the highest score. The game also displays an appealing border around the scores.

2. Project Status

In the current project all the above features are already implemented, and we believe they make the game quite fun to play. However, we realize it's still possible to make some improvements to our work. One of the possible upgrades to the game could be to supply the player with special powers or more weapons. Another idea is to add more types of aliens that may have unusual behaviors.

Unfortunately, due to the lack of time, we were unable to add these features. Nevertheless, we are pleased with the work we currently have.

2.1 Devices Used

In this project we used 5 devices, which are described below:

Device	Functionality used for	Interrupt / Polling
Timer	Frame Rate Control	Interrupt
Keyboard	Player name input and pause during the game	Interrupt
Mouse	Selection of the buttons in the menu and killing the aliens	Interrupt
Video Card	All displays: Menu, Pause, Game, Game Over...	NA
RTC	Display current time (hours:minutes)	Interrupt

2.1.1 Timer

The timer device was, exactly as described above, used for the frame rate control. In the game, we subscribe to the timer interrupts, and every time the timer is called, it considers the game frame rate, and decides whether or not to draw the game model.

This behavior is seen in the `game_loop()` function in the `game.c` file, which is the main loop of the game.

2.1.2 Keyboard

The keyboard device is used in 2 states of the game:

- The first one is during the game, the user can press the p button to stop the game execution. This was mentioned before in section 1.3.
- The second one, is at the end of the game, when the game might ask for the player's name.

The keyboard events are handled by the `keyboard_event_handler()` function, in the `keyboard.c` file, which decides what to do, based on the state of the game.

2.1.3 Mouse

The mouse is a very crucial device in our game. The user can use the mouse in the menu, to select the button they wish, but the most important role of the mouse in our project is obviously to kill the aliens during the game.

In the beginning of the game, the mouse is attributed a center position, and from there, it computes the displacement that occurred for each event. The left button is used, as mentioned above, to shoot.

Similarly to the keyboard, the function which handles the mouse events is the `mouse_event_handler()` in the `mouse.c` file.

2.1.4 Video Card

The video card is responsible for drawing the model of our game. There are multiple functions with the purpose of drawing depending on the game state, but their actions are easily comprehensible if you take a look at the `vg_ih()` in the `video_card.c` file.

The video card implementation we made uses the double buffering strategy in order to avoid the *snow effect*. This is important, since we wouldn't want our crosshair appearing deformed.

The mode we use in the game is (0x)115. We made this decision because we wanted a relatively small resolution, and at the same time a wider variety of colors that might be required for the different sprites. We believe that choosing a smaller resolution screen was the right call to improve the user experience. That is because in a wider screen, the player would be moving the mouse around too much, from one side of the screen to the other, leading to in-game frustration.

2.1.5 RTC - Real Time Clock

The RTC is used in the Menu. It displays the current time, in the Hour:Minute format.

The main function of this device is the `read_time()` function implemented in the `rtc.c` file. In order to read time consistently, we choose to use the method of pooling the bit UIP of `rtc`'s register A. The reading will be done in BCD mode, and then translated to decimal values.

Although we didn't subscribe to the `rtc` interrupts, a function was created (`rtc_ih()`) in `game.c` which is going to read the current time every video card interrupt.

3. Code Structure

Our code is structured into different modules. They are mostly modules corresponding to the individual devices used, and the rest are related to the game configuration. Most of the files are organized into folders that contain a specific functionality.

3.1 Modules

3.1.1 `utils.c`

Small auxiliary functions used overall. Developed during lab2.

Contributions:

- All group members contributed equally in this module.

Percentage: 1%

3.1.2 `timer.c`

Functions related to the timer, and were developed in lab2. Functions used in the project are just the `timer_subscribe_int()` and `timer_unsubscribe_int()`. Very straightforward implementation.

Contributions:

- All group members contributed equally in this module.

Percentage: 4%

3.1.3 `keyboard.c`

Functions related to the keyboard. Include code developed in lab3. Most functions were used in the project, but they were already completed in classes.

Contributions:

- João Alves and Adam Nogueira developed `keyboard_event_handler()`
- The rest was equally done by all members of the group

Percentage: 9%

3.1.4 `mouse.c`

Functions related to the mouse. Include code developed in lab4.

Contributions:

- Ana Sofia: subscriptions of interrupts, both enable and disable data reporting, `read_ACK_byte()`, and interrupt handler.
- João Alves: changes made to interrupt handler, event handler, and collision checks with buttons and aliens

Percentage: 8%

3.1.5 video_card.c

Functions related to the video card. Include code developed in lab5. This file handles all that is related to drawing to the VRAM.

Contributions:

- João Alves: prepare and set graphics, draw sprite, crosshair, points, aliens, buttons, menu, game, game_over and pause.
- Adam Nogueira: draw killer alien, string, input screen and leaderboard
- Ana Sofia: draw time
- Joint effort: interrupt handler and load sprites

Percentage: 31%

3.1.6 rtc.c

Functions related to the real time clock. This module provides us functionality to access the current time and date that we use to show in the menu.

Contributions:

- All functions were developed by Ana Sofia

Percentage: 7%

3.1.6 sprites

All files that contain the sprites are in this module.

Contributions:

- All sprites were developed by Rúben Monteiro

Percentage: 9%

3.1.7 leaderboard.c

Module responsible for all regarding the leaderboard. It handles the logic behind it, as well as the reading and writing files operations.

Contributions:

- All functions were developed by Adam Nogueira

Percentage: 5%

3.1.8 game.c

Module that handles the main logic of the home program. Responsible for coordinating all previous modules mentioned. This is where the main loop of the program is (game_loop() function), and where driver_receive() is called. In this function, we start by subscribing to the devices that work with interrupts in our program, and inside the loop containing

driver_receive() we call the interrupt handlers for each of the devices as well as the game handler. After the player decides to exit the game, we unsubscribe to the devices.

Contributions:

- Adam Nogueira: small changes in game loop, game leaderboard, save leaderboard
- João Alves: game initialization, most of game loop, game step (update aliens time and generate new alien), game reset
- Joint effort: interrupt handler

Percentage: 25%

3.1.9 proj.c

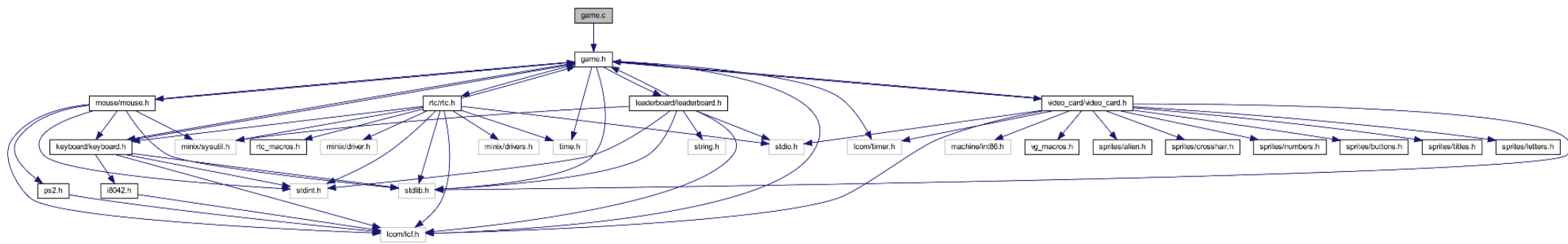
Module containing the main function of the game. It starts the graphics mode, calls the game loop, and then switches back to the text mode.

Contributions:

- All functions were developed by João Alves

Percentage: 1%

3.2 Function call Graph (generated by Doxygen)



4. Implementation Details

4.1 State

Our project doesn't use a state machine for the mouse, like the one talked about in class, since it wasn't required for our game. However, we still use a state strategy in the overall game and in the alien lifetime. The existing game states declaration is present in the game.h file. This programming design pattern allows us to have an organized and easy to understand code.

4.2 Animations

One of the most fun parts to implement in this project were the animations, because their results were really enjoyable.

Both menu buttons have animations. When they are hovered by the mouse (crosshair), they appear to be pressed down, causing a subtle, yet satisfying effect on the user experience.



The aliens, just like the buttons in the initial menu, have animations. But the animations for the aliens are more complex. The aliens have a sequence of animation states that change according to the progress of the game. The aliens have 1 sprite used for their appearance, 1 for when they are alive, and 5 for when they die. It's important to mention that it's also possible to kill an alien in an appearing state.



Besides this alien animation, the killer alien (the one that killed the player) has its own special animation while the game over message is displayed. It creates an enjoyable moving effect on the alien.



4.3 Collision Detection

This was a rather straightforward feature to implement in our game compared to others, since the only collisions we had to take into account were the mouse “collisions” with the buttons, and the aliens. Nevertheless, it’s still an important feature that ensures the reliability of the playing experience.

4.4 Event Driven

Both the keyboard and the mouse have event handler functions that handle the events generated by the interrupt handlers, and choose what action to take. By adopting this design we can make our code flexible and easily add new features to the project!

4.5 Object Orientation

In order to have a similar code structure that is possible to have with object oriented programming languages, we made use of static variables inside the modules. The biggest examples are the `game.c` and the `video_card.c`.

5. Conclusions

5.1 Group Opinion

In this project, just like many others, there are things we consider more interesting than others, but as a group, we consider that this was an enjoyable project. However, we have one main complaint, which is the lack of time to further develop our game.

We only finished the code for devices required rather late in this semester, and that by itself would leave us with little time to increase the complexity of our work. When we combine this, with the fact that we have group projects in all other subjects lectured in this semester, it becomes an extremely difficult and exhausting task to expand our game features as we desire.

We hope you can take this into consideration in the following years.

5.2 Problems

During development we had some problems with the sprites. For some reason, the sprites that had names of colors or *None* on their definition, couldn't be read properly. Unfortunately, we weren't able to solve this problem, but we found a way to fix this, by changing the colors to their values in RGB, and considering the black color as *None* when we had to take overlapping of sprites into consideration.

Another problem we had is the handling of files in minix. As mentioned before, we have a leaderboard feature in our project, and our objective is to be able to store the information inside the files, but due to path problems in minix, we had some problems regarding this functionality. The way we solved this problem was by asking the program to look for a file to read from, but if the file is not found, the game still works properly, by creating a new clean leaderboard.

5.3 Main achievements

Overall, we are happy with the work we have done. We believe that the game can provide a fun experience to anyone who decides to play it.

While developing the game we also learned a lot through experience, and we feel like we are better prepared to develop any similar project in the future, if such is required.