Faculdade de Engenharia da Universidade do Porto

Asteroids – Lcom project

Computer Labs: Project Report

Projeto LCOM 22/23
(L.EC018: Laboratório de Computadores)
Licenciatura em Engenharia Informática e Computação

Theoretical professor: Pedro Alexandre Guimarães Lobo Ferreira Souto

Laboratory practices professor: Nuno Filipe Gomes Cardoso

Students & Authors:

David Fang up202004179@fe.up.pt

Afonso Pinto up202008014@fe.up.pt

Pedro Balazeiro up202005097@fe.up.pt

Table of contents

Documents to Submit

- **Final report:**
    - Same as the project´s deadline.
- **Evaluation form:**
    - Your evaluation of yourself and the other group member’s work.
- **Video:**
    - Demonstration of your project.
    - Max 5 mins.
- **Project/Demo form:**
    - Must include URL of the video demo.
    - will help us to better assess the ceiling of your project.
- **Documentation:**
    - Doxygen documentation on Git and code.

Project report goals

- **Show you've met the goals of the course:**
  - use the hardware interface of the most common PC peripherals.
  - develop low level software and embedded software.
  - program in the C language (in a structured way).
  - use various SW development tools.
- **Help us grading fairly your project:**
  - There is a lot "under the hood" that may go unnoticed ➔ point us to.
  - Very hard to evaluate the work of each member ➔indicate it.
- **The project report is worth much more than the nominal 20%:**
  - We use it to estimate the complexity of your project and therefore the ceiling of your grade.

Project Report: General Structure

1. **User instructions.**

2. **Project status.**

3. **Code organization/structure.**

4. **Implementation details.**

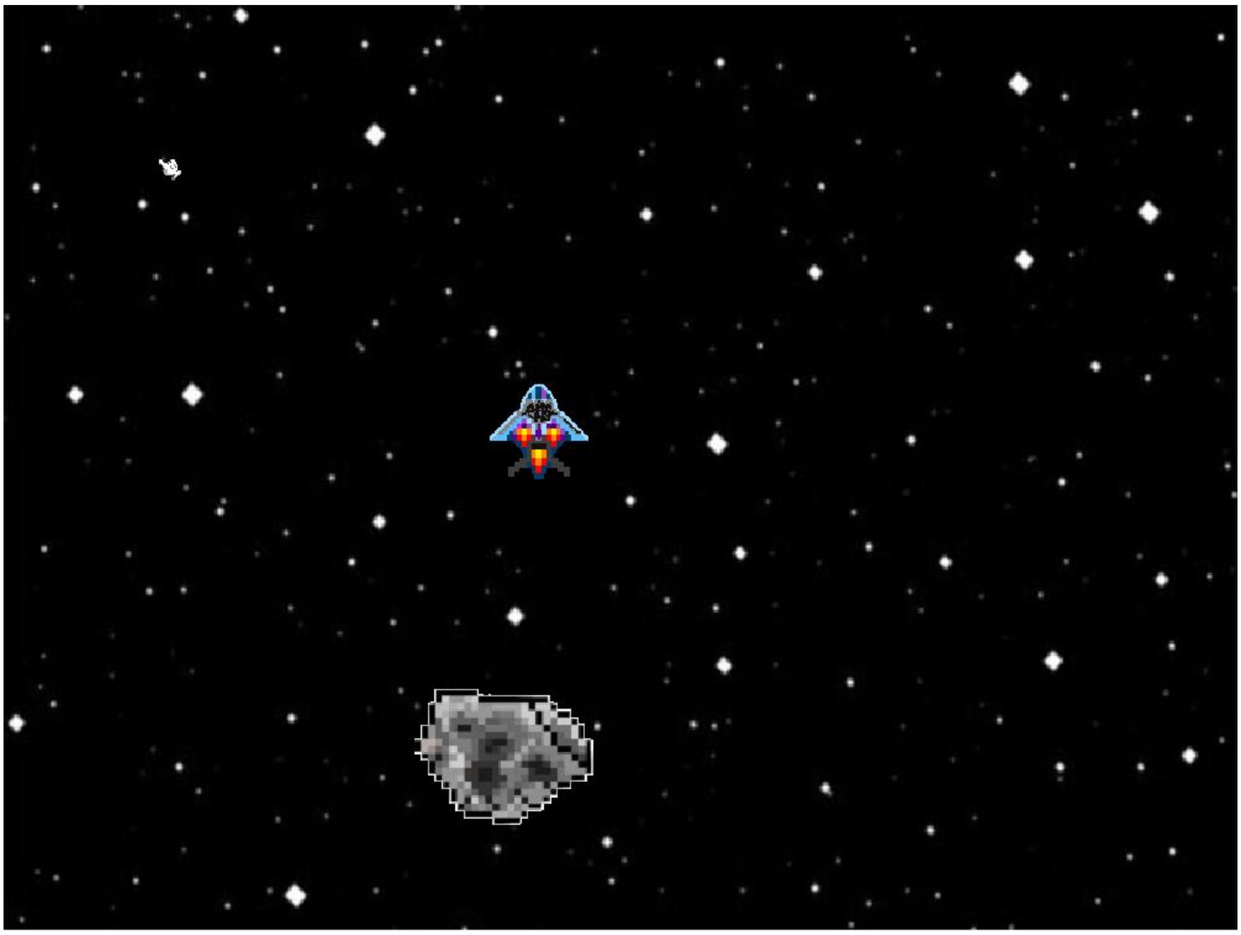
5. **Conclusions.**

- **Appendix: Installation instructions (optional).**

Section 1: user instructions

- **What is your project about?**

Our project is inspired by Asteroids. It is a classic arcade game released by Atari in 1979. It consists in a space-themed shooter where the player controls a small spaceship and must navigate through an asteroid field while avoiding collisions with asteroids. The gameplay is set in a two-dimensional, black-and-white environment. Our spaceship can rotate in 8 directions, also moving forward and backward allowing for movements and evasion. The main objective is to shoot and destroy asteroids. The game features several challenging elements like asteroids moving in random patterns. This game was highly popular upon its release, renowned for its addictive gameplay, fast-paced action, and pioneering vector graphics. It has since become an iconic and influential title in the history of video games, spawning numerous sequels, adaptations, and references in popular culture.



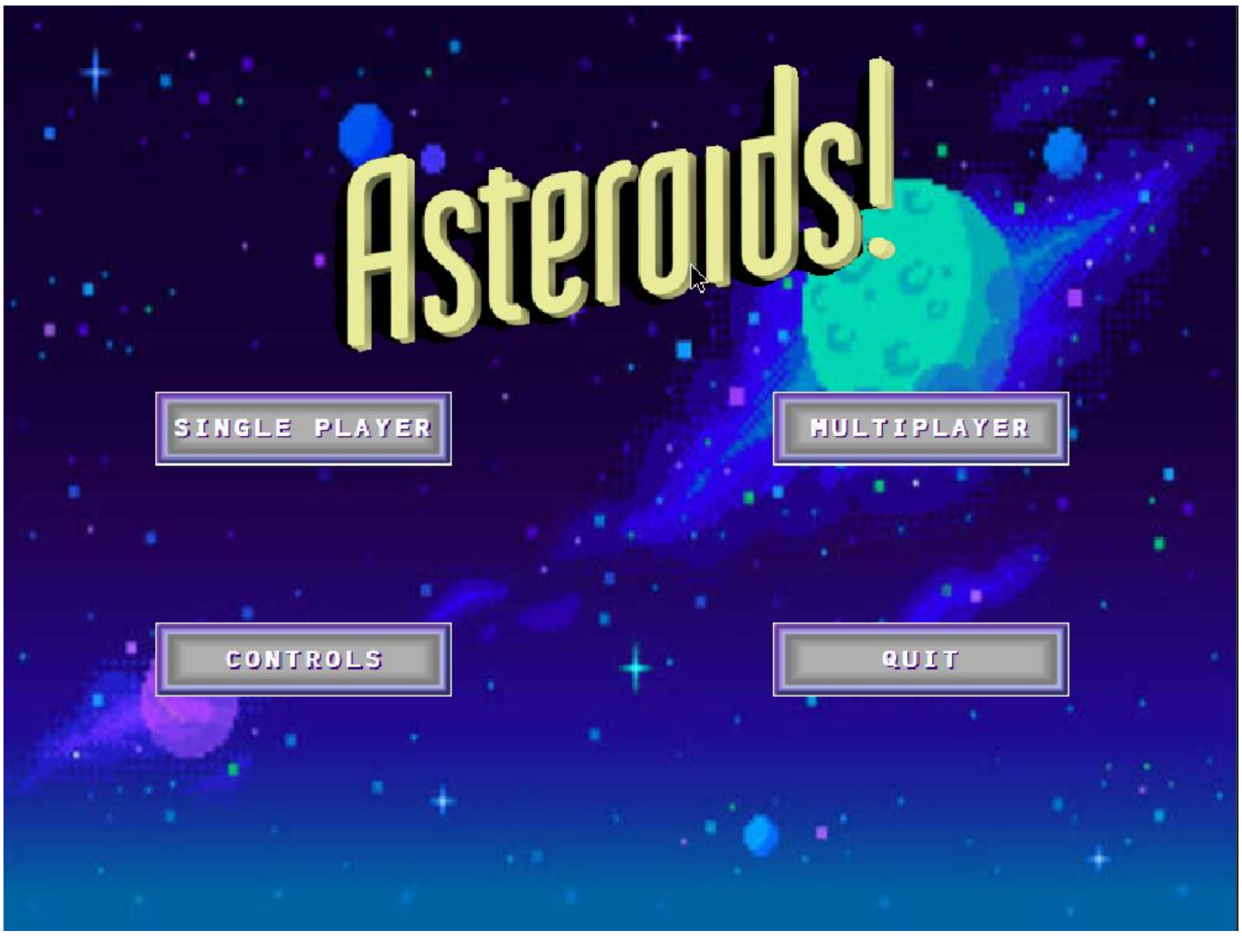
• How to use your project:

• use it to provide an overview of your project’s.

• may include aims not accomplished if they’re clearly stated (see next section).

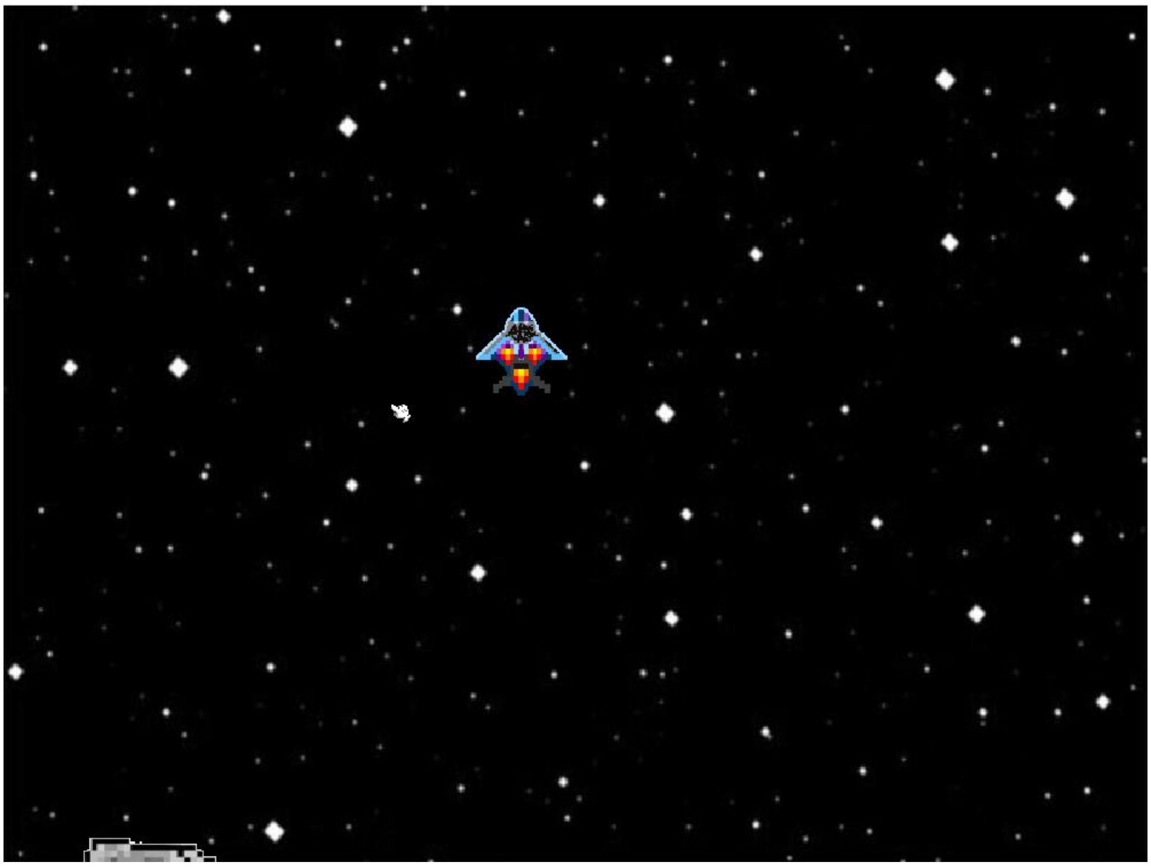
• Use and abuse of images ("1 picture is worth 1000 words").

The Initial/Main menu appears when we start the program, and we use the mouse to display cursor movement and select with mouse right button between four options (buttons). We can also use some key shortcuts:

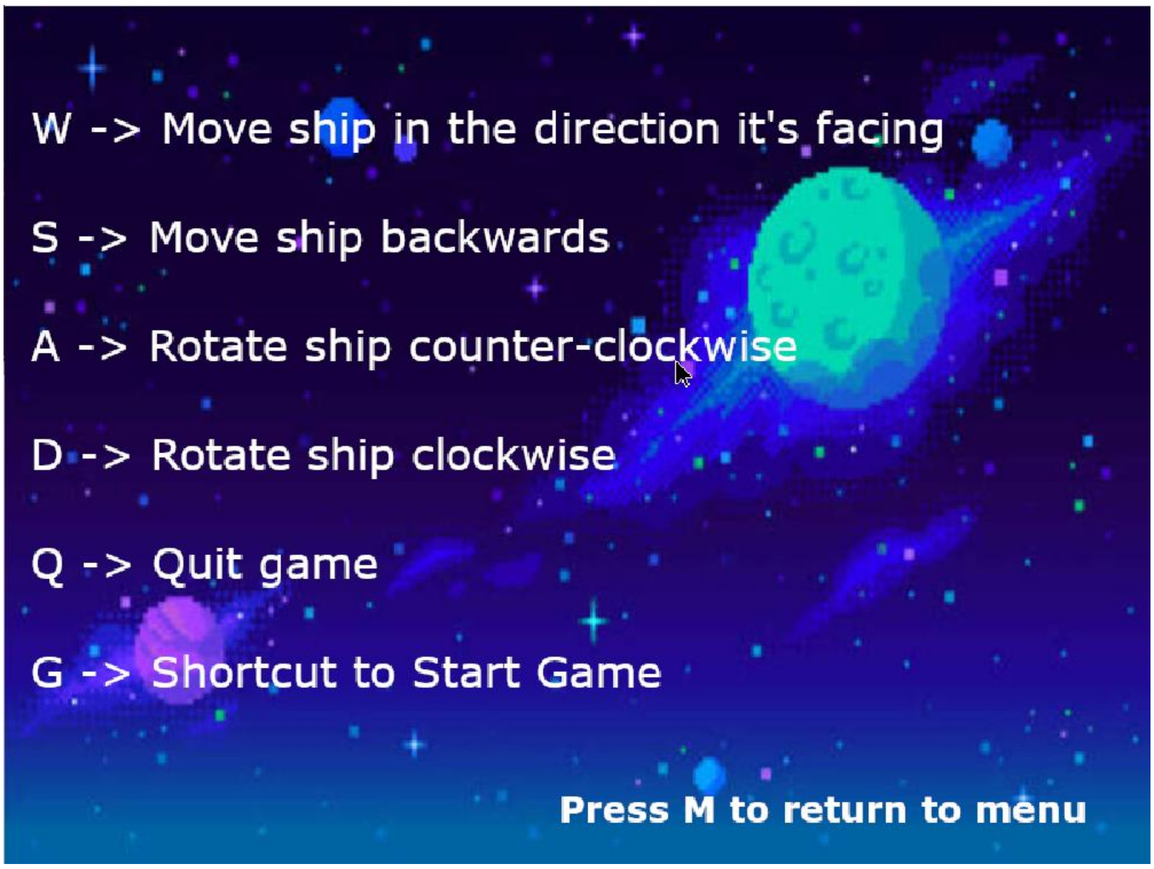


- M (goes to Main Menu).
- Esc (same as above).
- C (goes to Controls Menu).
- G (goes to Game).
- E (goes to End).
- Q (Quit).

- Single Player option (displays and starts the game).



- Control Menu (shows the commands available)



W -> Move ship in the direction it's facing

S -> Move ship backwards

A -> Rotate ship counter-clockwise

D -> Rotate ship clockwise

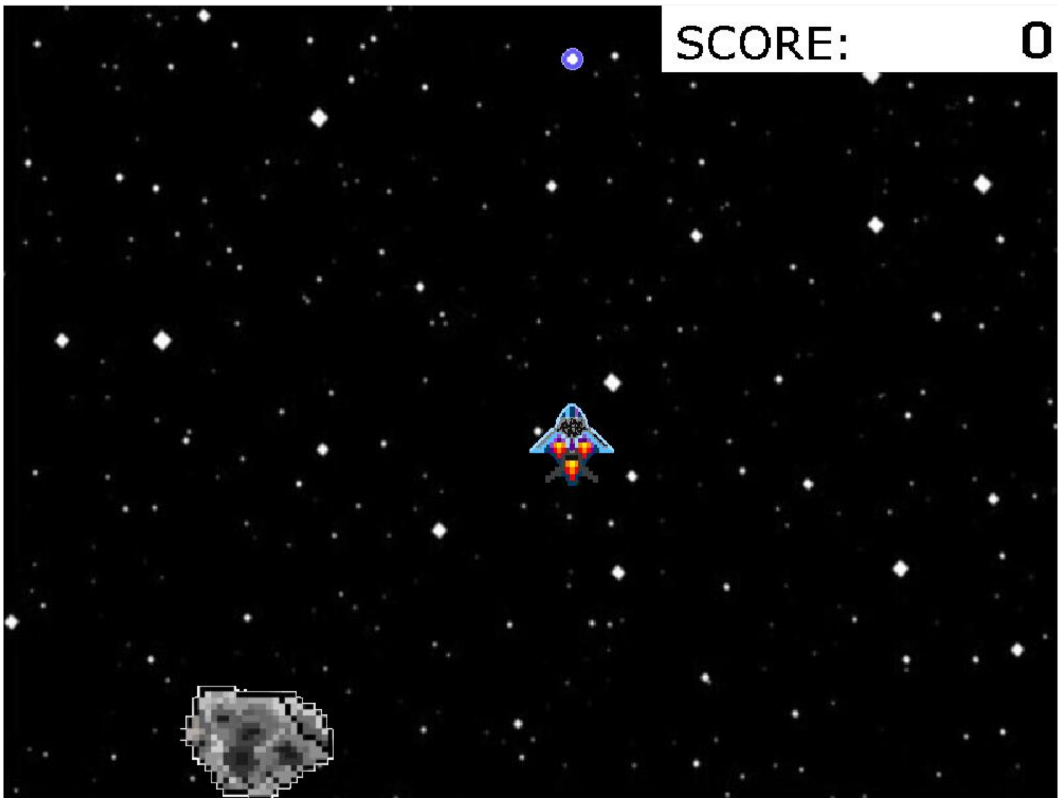
Q -> Quit game

G -> Shortcut to Start Game

Press M to return to menu

- Quit (that allows you to exit the program, that is, the same functionality as ESC key).

```
minix$ _
```



The objective of the game was already described above. The movement is done using the W, A, S and D keys (up, left, down and right respectively). Also the bullets are released by pressing the SPACE key (mouse left button pressed shots).

Section 2: Project Status (1/3)

- **What functionality did you really implement?**

We implemented the following functionalities:

- Moving between states using buttons (ex: from main menu to game, main menu to controls menu, main menu to exit program and game to main menu, etc.).
- Spaceship rotation in 8 directions, and movement forward and backward.
- Shooting the asteroids with bullets in game state.
- Collisions between asteroids and spaceships and between bullets and asteroids.
- Different types of asteroids trajectories.
- Quit game with quit button option.
- Score goes up as asteroids get destroyed.

- **May be easier to list which functionality mentioned in the previous section you did not implement:**

  • E.g., your GUI allows you to choose some functionality not implemented.

We wanted or thought of implementing the following functionalities:

- Leaderboard.
- Multiplayer possibility.
- Show time survived while in game state.

- **Must include a table with the I/O devices you have used, what you have used them for, how did you use them (interrupt vs. polling).**

Used Devices:

Devices	Utility	Interrupt
Timer	Handles loops	Yes
Mouse	Cursor movement and item selection	Yes
Keyboard	Spaceship movement and shoot bullets and states shortcuts.	Yes
Video Graphic	Game display	Yes
Real Time Clock	-	No
Serial Port	-	No

Section 2: Project Status (2/3)

- **Must include a subsection for each I/O device:**

1. describing the device's functionality used.
2. referring to the code (function name) where you use it.

- **Graphics card should mention/indicate use of:**

• video mode (with resolution, color mode and number of colors).

• Double/triple buffering.

• Moving objects (collision detection, animated sprites).

• Fonts.

• VBE functions, e.g., to change the palette or for page flipping.

Our graphic card is utilized in video mode with resolution 800 pixels of width per 600 pixels per height (code mode 0x115), in direct color mode (Bits per pixel (R: G: B), 24 (8:8:8)).

We also use double buffering to allow the game to become more fluid and dynamic.

The images displayed result from the interactions in the game, like collision between asteroid spaceship and animated sprites like the asteroids moving with different patterns. We have all the xpms in a xpm directory.

The functions where we use it are:

- Functions in video.c and functions that call the ones inside it.

- **Keyboard used for:**

  • Application control.

  • Text input.

We use the keyboard for shortcuts between states and to shoot bullets as already described above.

The functions where we use it are:

- Functions in keyboard.c and functions that call the ones inside it.

Section 2: Project Status (3/3)

- **Mouse use of:**

  • Position.

  • Buttons.

  • and for each of them what do you use it for.

We use the mouse as a form of communication between the user and the program. Like when you move the cursor to the option/button you choose on the main menu.

The functions where we use it are:

- Functions in mouse.c and functions that call the ones inside it.

- **RTC used for:**

  • Reading date/time.

  • Generating an alarm.

  • Periodic interrupts.

As RTC makes it possible to read the current date and time.

The functions where we use it are:

- Not implemented.

UART Should describe:

- Features used, e.g., interrupts or FIFOs.
- Communication parameters used.
- Data exchanged and exchange frequency.

Not implemented.

Timer describe:

As the essential device of this project, the timer is responsible for handling the interrupts according to the game state at a constant rate, displaying the respective content (spaceship, bullets, asteroids, buttons, background, menu) and updating the objects' coordinates whenever there is a movement of objects (spaceship, bullets, asteroids).

The functions where we use it are:

- Functions in timer.c and functions that call the ones inside it.

Section 3: Code Organization/Structure (1/2)

- **For code you developed, one subsection per module (C file), with:**

  1. "A one paragraph description" of the code contained in the module:
     - Also include the relative weight (in %) of module in project.
  2. Information on who (group member) did what (in the module).

main.c

This module is responsible for initializing the devices, handling all interrupts of main cycle, and closing the devices whenever main cycle ends.

The device initialization is made from functions:

- main() – lcf.
- setup() – setting all up.

Once the devices are ready, the main cycle handles the interrupts through driver_receive() until the EXIT state of the game is met with proj_main_loop().

Before quitting the program, the following functions are called:

- teardown() – shutting it all down.

Module developed by: Afonso Pinto, David Fang and Pedro Balazeiro.

game.c

This module contains a struct game, which has all the information needed to run the game. It has the setup_bitmaps() that initializes all bitmaps and destroy_bitmaps that does the opposite. The update_timer_state() that occurs at every timer interruption allows to update the game state constantly (asteroids, spaceship, collisions, etc.). The update_keyboard_state() that occurs at every keyboard interruption is responsible for changing the menu states upon selection of any shortcut. The update_mouse_state() that occurs at every mouse interruption allows to do the same thing as the keyboard basically. The update_button_state controls if the buttons are selected with the mouse.

Module developed by: Afonso Pinto, David Fang and Pedro Balazeiro.

bullet.c

This module contains:

- get_ship_direction() responsible for obtaining in which direction the spaceship will shoot the bullet.
- create_bullet() allocating the bullet.
- shoot () responsible for checking if the condition to shoot the bullet is met to create it.
- update_bullet() that updates bullet state.
- destroy_bullet() that does what the name says.
- check_bullet_collision() that checks collisions with asteroids.

Module developed by: David Fang.

asteroid.c

Identic module as bullet that contains:

- create_asteroid() allocating the asteroid.
- update_asteroid() that updates asteroid state.
- destroy_asteroid() that does what the name says.
- check_collision() that checks collisions with spaceship.

Module developed by: Afonso Pinto.

spaceShip.c

This module contains:

- update_spaceship_position() that updates spaceship state, draws it and checks for collisions.

Module developed by: Pedro Balazeiro.

timer.c

Group of functions important for the use and configuration of the timer.
Imported from labs 2.

Module developed by: Afonso Pinto.

keyboard.c and KBC.c

Group of functions important for the use and functionality of the keyboard.
Imported from labs 3.

Module developed by: Pedro Balazeiro.

video.c

Group of functions important for the operation and configuration of the graphic card.
Imported from labs 5.

Module developed by: David Fang.

mouse.c and KBC.c

Group of functions important for the use and functionality of the mouse.
Imported from labs 4.

Module developed by: Afonso Pinto.

utils.c

Group of important auxiliary functions used in all labs.

Module developed by: Pedro Balazeiro.

bitmap.c

Group of important auxiliary functions that create and destroy bitmaps.

Module developed by: David Fang.

Different weights table:

	Weight
timer.c and utils.c	10%
keyboard.c and KBC.c	10%
mouse.c and KBC.c	10%
video.c and bitmap.c	10%
game.c	20%
bullet.c	5%
asteroid.c	5%
spaceShip.c	5%
-	-
main.c	25%

- **May also include a description of the:**
  - main data structures per module.

- **For code you downloaded, one subsection per module/function, with:**

  1. "A one paragraph description" of the code.
  2. A description of the changes you had to make, if any.
  3. The URL you got the code from.

Section 3: Code Organization/Structure (2/2)

- **Function call graph:**
  - can be generated automatically by Doxygen.
  - You may limit the depth for a more report-wise size figure.
  - full Doxygen documentation should be include in the Git repo.

- **Also include a short description of the main functions:**
  - These must include functions that call driver_receive().

Section 4: Implementation Details

- **Topics covered in the lectures, but that required some ingenuity in their application to your project (e.g., layering, event driven code, state machines, object orientation, frame generation, ...):**

• Details regarding the use of the RTC and the UART.

All referenced above. State machine to control the states, frame generation every timer interruption with the help of double buffering to turn the game fluid, etc.

- **Topics not covered in the lectures/labs and that you had to learn by yourself (and maybe you wished we had talked about it) (e.g., collision detection, ...).**

I think we should have learnt more but the reality is that for the time we have and counting on the consumption of time required for this and other subjects (huge time consumption), it's impossible to do and accomplish more.

Section 5: Conclusions

- **Problems, delays that occurred and led to some functionality not implemented/faulty.**

- **“Like to haves” that were not defined at first but would add to the program if done in the future.**

- **Main achievements of the project.**

Due to the huge time consumption with this and other projects of other subjects we couldn´t accomplish all the functionalities we would want to, but we are satisfied with the work done. We think that we understood much more about how the devices work and now we can visualize the interactions easily. Sometimes we had difficulties to find errors that weren´t visible but as knowledge grew, we got better, and the mistakes disappeared. So, in conclusion we made a basic and playable game with menu transitions that uses the devices and that is our main achievement.

Appendix: Installation Instructions (Optional)

- **required only if we need to do something else other than invoke make and lcom_run proj in your project's top directory:**

• If you use files, avoid "hard-coding" them: specify the directory with them via command line arguments.

  - This is a major source of "headaches" and delays in the final demos.

Project Report: Final Recommendations

- **Remember, the project report is worth 20% of your project's grade:**
  - And it also affects other aspects of your project's grade.
  - In the self-evaluation form, there is a field to specify your contribution to the project report.
- **Do not leave it for the last minute:**
  - Start writing the report now.
  - Write it incrementally.
  - Most of the information we ask for is available rather early in the project.
  - You can always review it, if later you change something already mentioned in the report.
- **Leave "refinements" to closer to the deadline:**
  - o E.g., do not worry much with the writing style in early versions.
- **You can add sections, if the information you want to transmit does not fit in any of the sections enumerated:**
  - But include all the sections mentioned.