U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Asteroids

Laboratório de Computadores

2018/2019

Turma 8 – Grupo 6

7 de janeiro de 2019

Nuno Miguel Fernandes Marques, up201708997@fe.up.pt

Eduardo Ferreira Campos, up201604920@fe.up.pt

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Index

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

User's Instructions

Main Menu

Upon entering the game, the user will be presented with the game's Main Menu. From here, the player can perform several actions by pressing one of the following buttons:

- **Play** will begin a new game;
- **Connect to game** will begin a Player versus Player game, hosted in the other player's machine;
- **Host game** will create a local game to which another player may join;
- **Exit** will leave the game;
- **Options** will enter a new menu where several options can be changed.

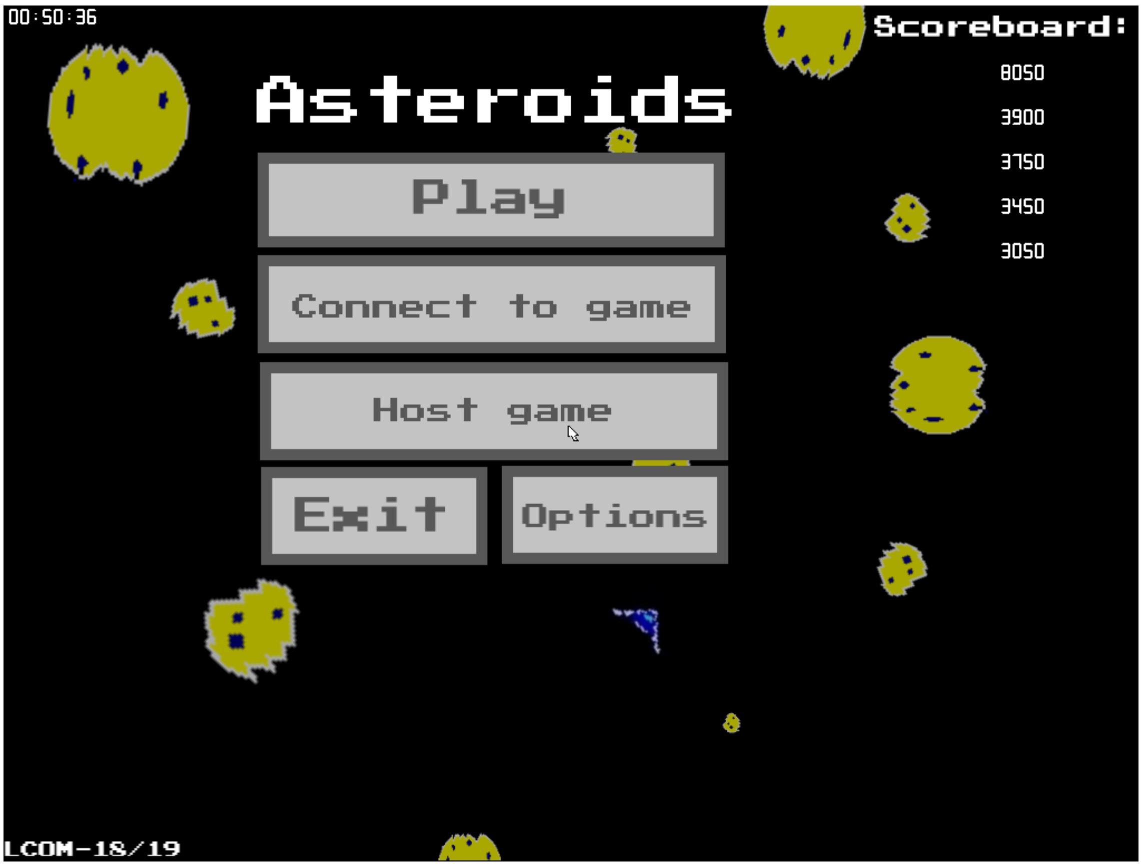


Figure 1 – The main menu

Additionally, at the top right corner there is present a scoreboard, where the player can check his/her previous games' scores. On the top left corner, the current time of day is displayed as well.

Play

Not unlike the original game, our version of Asteroids has, as the main goal, surviving as long as possible while destroying asteroids and enemy ships.

The player begins with a score of zero and full Health Points (HP), i.e. 100. Destroying asteroids, as well as enemy ships, yields a certain amount of points. However, being hit by an asteroid, or by an enemy laser, decrements the player's HP. If the player manages to destroy all entities (asteroids and/or enemy ships), then a new round commences, and the player receives 20 HP, to a max of 100.

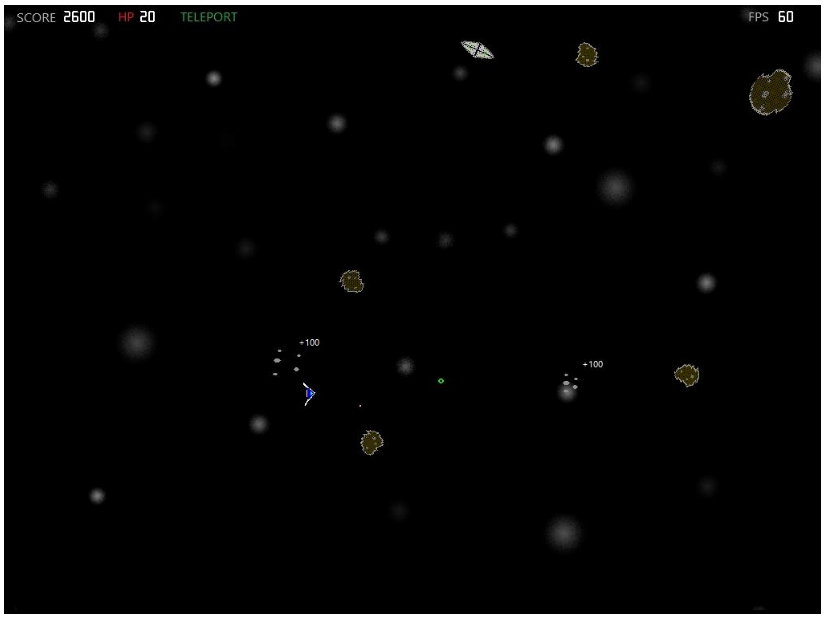


Figure 2 – The game. Notice the score, HP and Teleport indicators on the upper right, as well as the FPS counter on the upper left corner.

The player can control the movement of the ship with WASD keys, the ships direction, i.e. where it's aiming at, with the mouse cursor, fire lasers with the left mouse button and teleporting to another on-screen location with the right mouse button.

By teleporting, there is the possibility that the player might land on an asteroid, hence damaging the player. This, like in the original game, is actually a game mechanic, to force the player to measure the benefits of moving to a perhaps better position against the risk of hitting an asteroid and possibly losing the game.

When the player's HP reaches zero, then the game ends and a Death Screen is shown to the player. Here the player can return to the Main Menu or play another game.

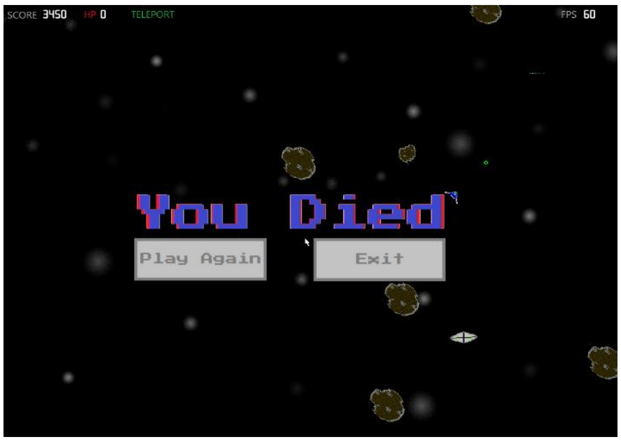


Figure 3 – The Death Screen.

Should the player reach a new highscore, a message will be shown (see below). These highscores are also shown in the Main Menu and are kept in a text file.

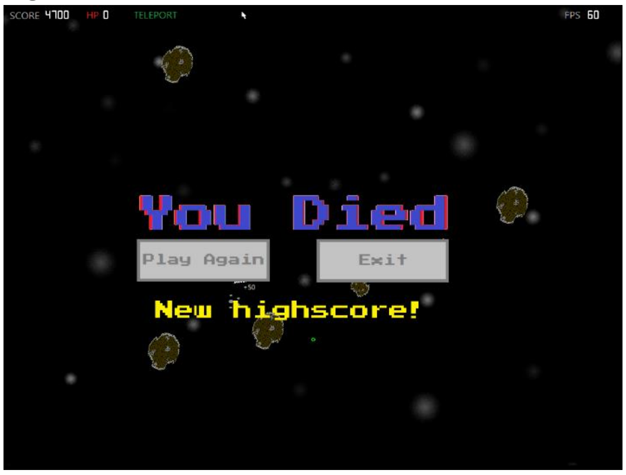


Figure 4 – The Death Screen with the "New Highscore" message.

At any moment during the playthrough the player is free to pause the game. If he chooses to continue he needs only press ESC or if he wishes to quit the game, press the space bar. This, however, results in all progress being lost.

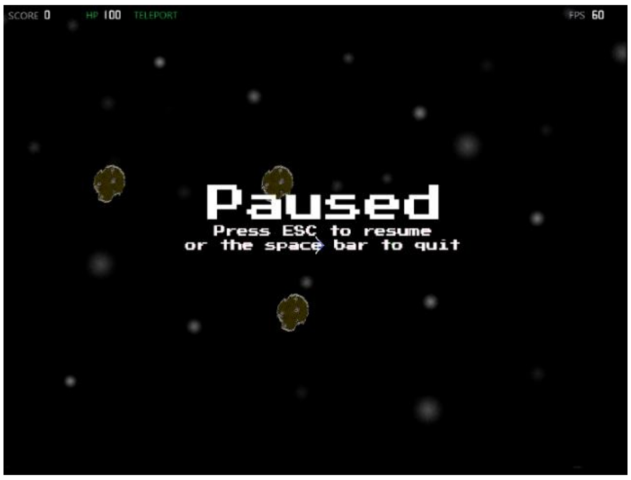
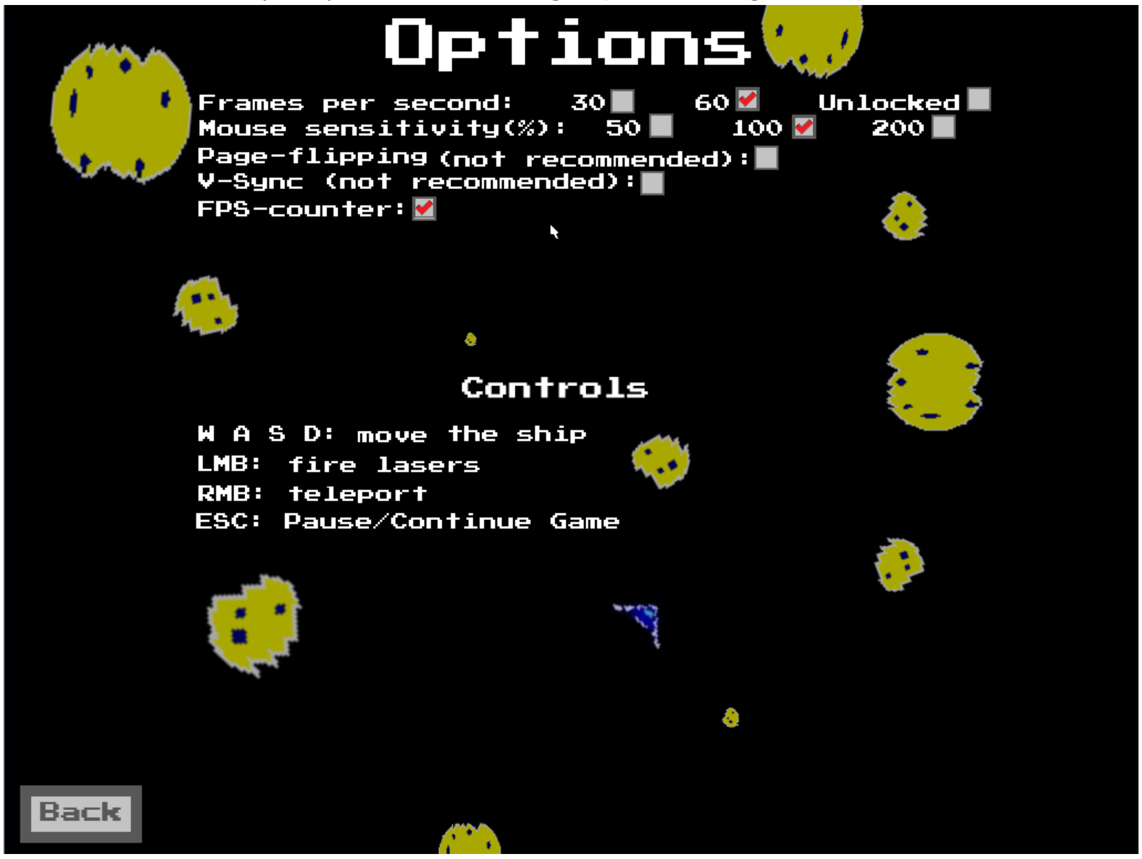


Figure 5 – The Pause Screen.

Options Menu

By pressing the options button of the Main Menu, the player is taken to the Options Menu, where a variety of options can be changed (see the image below).



Frames per second: 30 ☐ 60 ✓ Unlocked ☐
Mouse sensitivity(%): 50 ☐ 100 ✓ 200 ☐
Page-flipping (not recommended): ☐
V-Sync (not recommended): ☐
FPS-counter: ✓

Controls

W A S D: move the ship
LMB: fire lasers
RMB: teleport
ESC: Pause/Continue Game

Figure 6 – The Options menu.

Most options are self-explanatory. The Frames per second option allows the player to have the game's FPS locked on either 30 FPS, 60 FPS or unlocked, allowing for higher FPS during the game. The Mouse sensitivity option allows the player to choose with which mouse sensitivity he wishes to play the game, half the normal (50%), the normal (100%), or double the normal (200%). The Page-flipping and V-Sync options enable the page-flipping and V-sync features, however, as stated in the menu, we do not recommend using them. The FPS-counter option enables or disables the FPS counter during the game. The options checked in figure 6 are the default options.

In the bottom half of the menu, there are shown the game's controls. The Back button returns the player to the Main Menu.

Multiplayer

To play the multiplayer one versus one mode a player must first host a game using the *Host game* button and then the other player connects using the *Connect to Game button.* Upon connecting the players will face each other, the locally controlled ship is always blue, with blue lasers, and the enemy ship controlled remotely is always red, with red lasers. When one of the players health points goes down to zero both players will have a victory or loss screen and return to the Main Menu, where they are free to start a new game. The mouse sensitivity setting in figure 6 does not apply to the multiplayer mode.

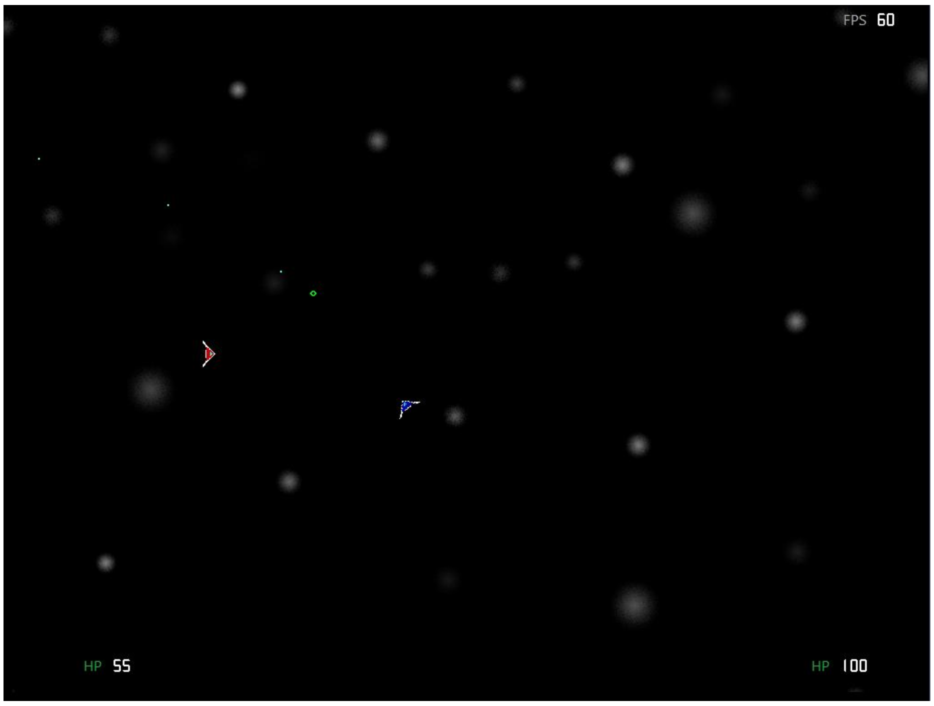


Figure 7 – The multiplayer game, with the red and blue ships being both human players.

Project Status

Game Logic

Most of the game logic was developed using cartesian coordinates and mathematical vectors to control the movement, collision, direction and the rest of the game's physics, which are separate from the graphical representation. All vector-related functions are in ***mvector.c***.

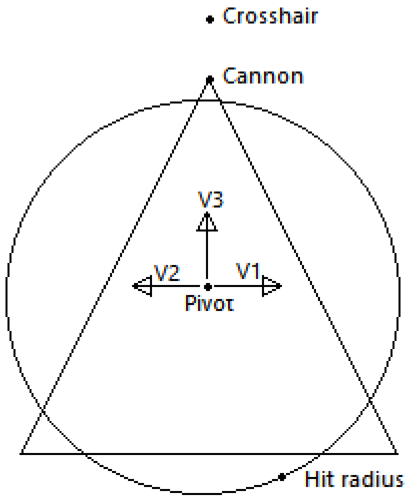


Figure 8 – Ship Overview

- **Ship's movement:** The ship's movement is achieved using a force vector. When using the engines, a force is applied in the direction of the respective engine versor, *V1*, *V2* or *V3*, represented in figure 7. On every physics update the force is applied to the ship's x and y coordinates.
- **Ship's direction:** The ship's cannon always points to where the crosshair is. This is done by creating a vector using the pivot and the crosshair, calculating the angle between *V3* and this vector, and finally using the rotation matrix to rotate the cannon point and the versors.
- **Ship's lasers:** Each laser starts in the cannon point and has a force vector with the same direction as *V3*. On each physics update, a force vector is applied to the laser's *x* and *y* coordinates until it collides with something or exits the screen.
- **Collision:** Laser collision is achieved by creating a vector between the laser and each entity and getting the vector's magnitude. If the magnitude is smaller than the hit radius of the entity a hit is registered. Collision between entities is similar, but instead a vector between pivots is created and if its magnitude is smaller than the sum of both hit radiuses, a collision is registered.

All functions described above can be found in ***ship.c*** and ***asteroidfield.c***.

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Devices Used

Device:	Description:	Method:
Timer:	Controls game physics and timers	Interrupt
Keyboard:	Ship Controls	Interrupt
Mouse:	Menu navigation and ship controls	Interrupt
Video Card:	Displays rendered frames	Polling
Real Time Clock:	Clock in Main Menu and FPS counter	Interrupt
Serial Port:	Multiplayer data transmission	Interrupt and Polling

TIMER

Timer 0 interrupts are used to update the game's various timers, control the frequency of the physics updates and if in locked fps mode it controls the frequency of frames displayed. Timer functions are in ***devices.c***.

KEYBOARD

Keyboard interrupts are used in game to activate the ship's engines and apply a force to the ship, it does not directly control the ship's position. The effect on the ship depends on the orientation. On figure 7:

- ***W*** activates ***V3***;
- ***A*** activates ***V2***;
- ***D*** activates ***V1***;
- ***S*** activates ***V3*** in reverse.

ESC is also used to pause the game, and ***SPACE BAR*** exits the game when in paused mode. Keyboard functions are in ***devices.c***.

MOUSE

The mouse is used to navigate the game's menus by controlling the cursor and using the left mouse button to choose options. In game, the mouse is used to control the ship's crosshair, the left mouse button fires lasers and the right mouse button teleports the ship to a random location. Mouse functions are in ***devices.c***.

VIDEO CARD

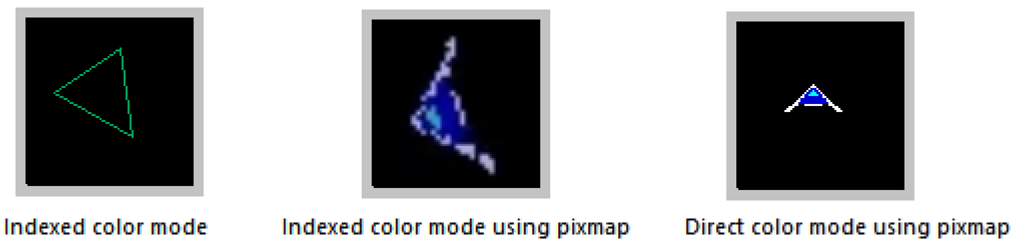

Indexed color mode
Indexed color mode using pixmap
Direct color mode using pixmap

Figure 9 – The effects of using different modes and assets on the game's graphical quality.

The Video Card is used in 24-bit Direct Colour Mode RGB (8:8:8), with a resolution of 1024 by 768. By default, the game is rendered using double buffering.

There are options to use Page-flipping and Vertical Synchronization in the menu using VBE's function 07h. However, these options are mostly for proof of concept as every time function 07h is called, to switch the display start pointer, *sys_int86* needs to temporarily switch to real address mode and back to protected mode. This is a very taxing operation and defeats the purpose of swapping pointers. In our tests when in page flipping mode the game can only draw about 25 frames per second, on average. In some environments function 07h might not work at all. Functions ***vbe_mode_info()***, ***vmode_init()***, ***vmode_exit()***, ***vbe_switch_display_start()*** and ***display_frame()*** can be found in ***vcard.c***.

The frames are rendered using both 24-bit colour Pixmaps and Bitmaps. Both Pixmaps and Bitmaps are loaded at the start of the game and freed at the end. The Pixmaps loading function, ***read_pixmap24()***, was based on function ***read_xpm()*** by Joao Cardoso, but it was later heavily modified to support 24-bit colour pixmaps. To draw Pixmaps, a new function, ***draw_pixmap()***, was created that allows transparency by ignoring a chosen colour and Pixmap rotation in a 360 degree range. The functions mentioned are in ***graphics.h***. Animated sprites are used for asteroid and alien destruction animations in the ***render_frame()*** function in ***vcard.c***.

Bitmaps were mainly used for the User-Interface. Henrique Ferrolho's code was used to load and draw Bitmaps. The ***drawBitmap()***function was altered to allow for transparency by ignoring a colour.

Collision was done separately from the graphical representation as part of the game's physics updates using the method explained in the *Game Logic* section. Collision function ***ast_collision()*** can be found in ***asteroidfield.c***, ***alien_collision()*** can be found in ***ship.c*** and ***mp_ship_collision()*** can be found in ***multiplayer.c***.

RTC

The Real Time Clock interrupts are used to read the time (hours: minutes: seconds), which is displayed on the Main Menu. The interrupts are also used to measure the number of frames drawn per second which is displayed in-game, if the Frame-counter option is enabled in the Options Menu. RTC functions can be found in ***devices.c***.

Serial Port

The serial port is used to pass keyboard events, mouse packets and the player's health points between computers for the multiplayer mode. COM1 is used to send data to the host and COM2 to the client. Interrupts are used when receiving data and polling is used to transmit. The maximum rate of 115200 bits/s is used. A function to enable FIFO exists but was deemed not necessary. All serial port related functions are in ***serialport.c*** and ***multiplayer.c***.

Code structure

Asteroidfield

This module is responsible for the creation and management of the asteroids. This includes creating the variants of the asteroid (***ast_spawn()***), processing their movement through the screen and, should they cross the screen border, warp them to the other edge of the screen (***ast_update()***), deal with their collision with the player ship (***ast_collision()***), fragment the asteroid into smaller asteroids (***ast_fragment()***) or destroy it, should the asteroid be hit by one of the player's lasers. This module has an overall weight of 5% to the project and was created by Nuno Marques.

Devices

This module contains all the several functions of the peripherals used (except the Serial-port and Video card). This includes all the peripherals addressed in the various Labs, and where directly imported from them, as well as the RTC. This module has an overall weight of 10% and was created by both Nuno Marques and Eduardo Campos.

Gamestate

This is the most important module as it contains the interrupt cycle and most of the game logic, such as handling the menus, the game state machine (*figure below*) and the single player event handler. This module has an overall weight of 25% and was created by Nuno Marques.

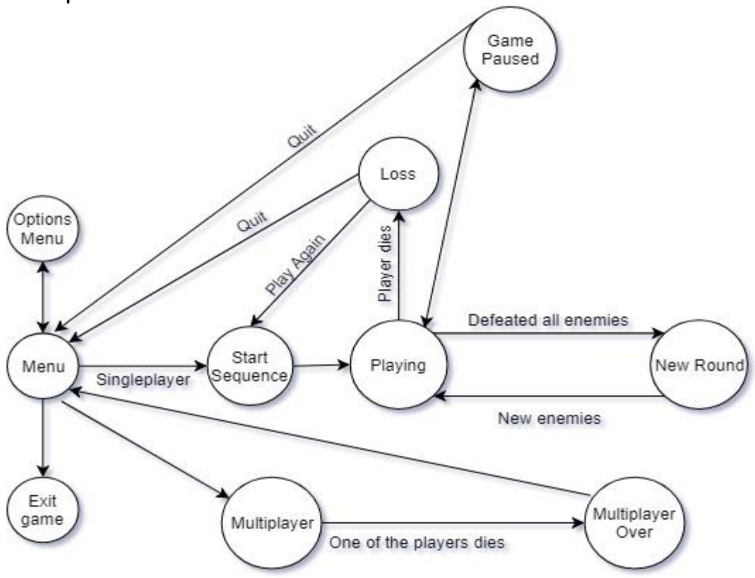


Figure 10 – The game's state machine.

Graphics

This module is responsible for managing the graphic assets of the game, i.e. both in Pixmap form and Bitmap form (***pixmap_data*** and ***bitmap_data*** respectively), as well as drawing them (***draw_pixmap()*** and ***drawBitmap()***), loading them from the .bmp files (***loadBitmap()***) and deleting them when they are no longer needed. The Bitmap part of the module was originally created by *Henrique Ferrolho* but was slightly edited by us to better fit our needs. This module has an overall weight of 10% and was created by Nuno Marques and Eduardo Campos.

Macros

This module is a support module which holds macros for all the device functions. It also holds all the game physics macros such as accelerations, velocities, health points and the alien ship's parameters.

Multiplayer

This module contains various auxiliary functions for the multiplayer mode, such as functions to transmit information over serial port, the multiplayer event handler, and altered functions from other modules for multiplayer. This module has an overall weight of 10%.

MVector

This module is responsible for all vector-related operations such as adding, multiplying, rotating and limiting vectors, calculating a vector's magnitude and others. It's used for game physics and rotating pixmaps. This module has an overall weight of 10% and was designed by Nuno Marques.

Pixmaps

This module contains all the pixmaps used in the game. This module has and overall weight of 5% and was created by both Nuno Marques and Eduardo Campos.

Proj

This is the starting module of the project, where the game data, the game itself and the several peripherals are initialized.

Serial Port

This module contains all functions pertaining to the workings of the serial port, such as subscribing and unsubscribing interrupts, setting up the configuration between machines and transmitting game data. This module has an overall weight of 5% and was created by Nuno Marques.

Ship

This module is responsible for dealing with the functionalities of both the player's and the alien ships. Within it, the ships' spawn, movement, collision detection and management, teleportation capabilities and firing capabilities are managed. Additionally, this module manages the highscores, i.e., storing, loading and updating them every game session. This module has an overall weight of 10% and was created by Nuno Marques and Eduardo Campos.

Vcard

This module is responsible for managing the Video Card and its memory, drawing the various Pixmaps and loading the Bitmaps, and freeing the memory when the assets are no longer needed. It is also responsible for rendering and displaying the frames. This module has an overall weight of 10% and was created by Nuno Marques and Eduardo Campos.

Call Graphs

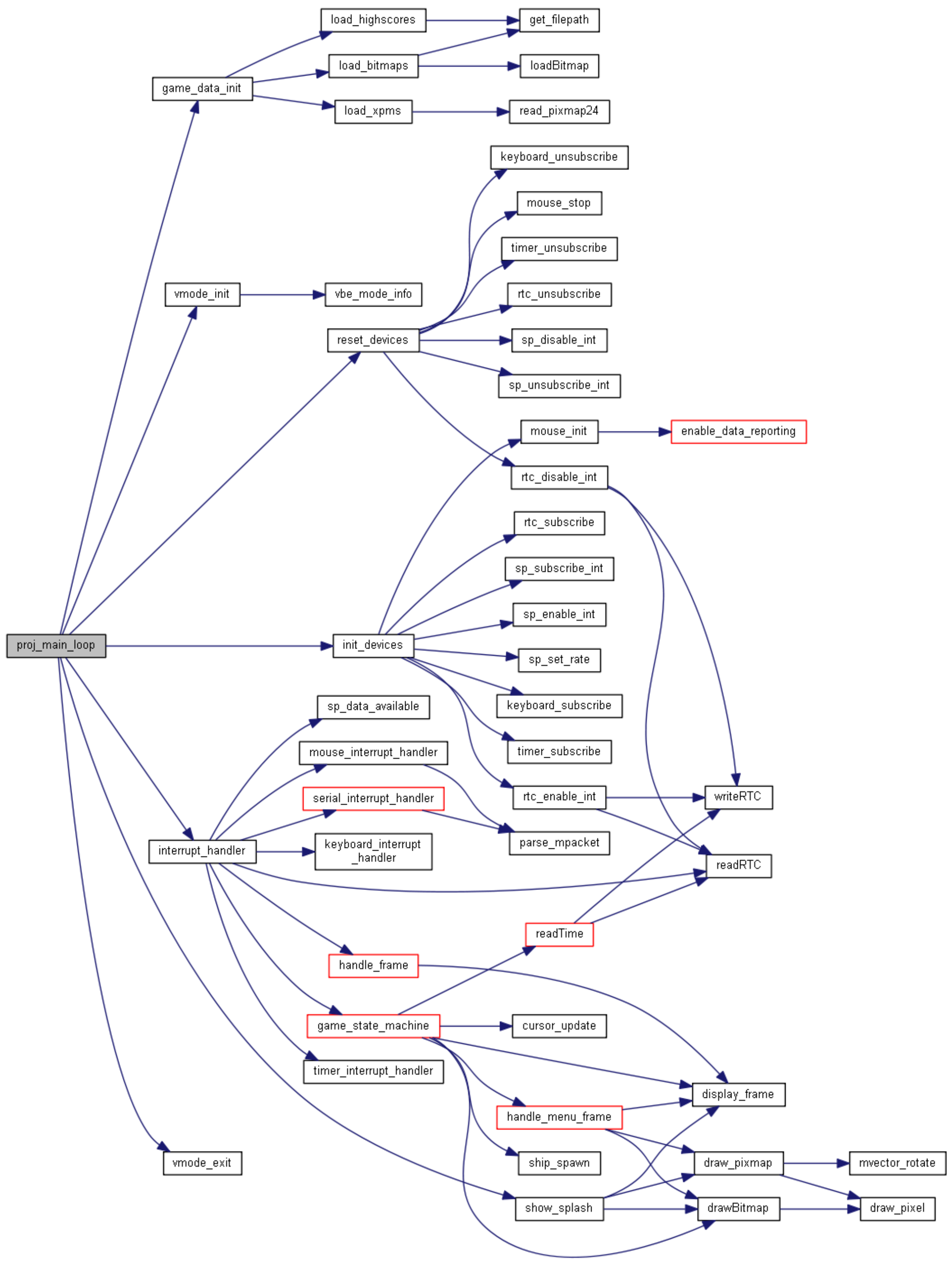


Figure 11 – The function call graph from proj.c's proj_main_loop().

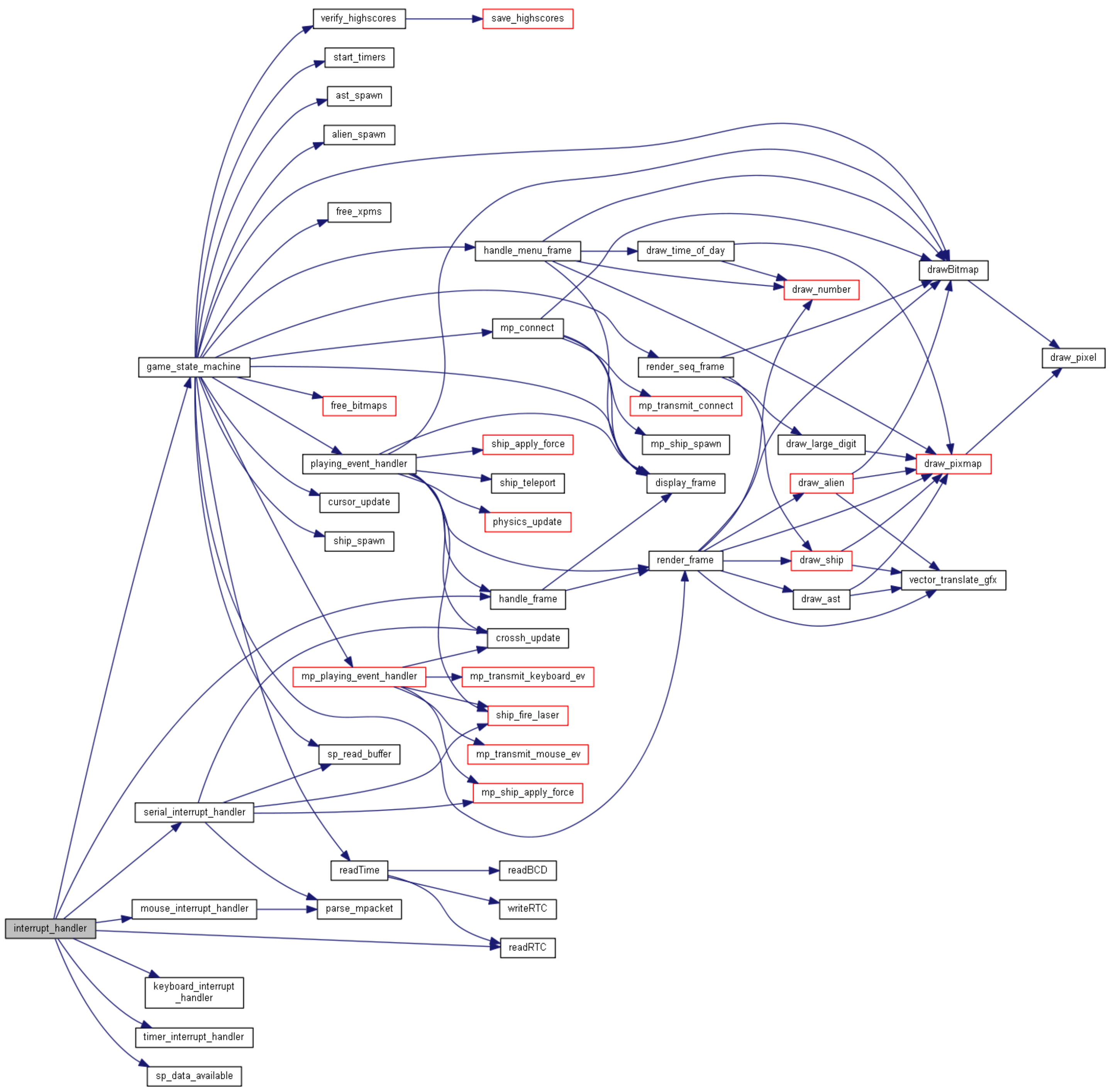


Figure 12 – The function call graph from the interrupt_handler().

Implementation Details

SERIAL PORT

Serial Port's communication protocol

Message type	Byte 1 (*Header*)	Byte 2	Byte 3	Byte 4
Connection:	*"O"*	*"L"*	---	---
Keyboard event:	*"K"*	Keyboard event (0-6)	---	---
Mouse packet:	*"M"*	Mouse byte 1	Mouse *delta-x*	Mouse *delta-y*
HP update:	*"H"*	Host HP (0-100)	Client HP (0-100)	---

In multiplayer mode, the host computer alone does the collision checks and updates for both computers. This is to avoid both players winning simultaneously due to latency, the rest is transmitted equally between both computers.

The serial port was the most challenging part of this project due to the nature of the game, being simultaneous and in real-time instead of a turn-based game. The force applied to the ship depends on the ship's angle and because of the latency in receiving user inputs, a small error is generated on every input since, when the other computer receives the input, the ship is already at a slightly different angle. This error is small but adds up quickly over time. A possible solution would be to have "correction packets" a few times a second, this proved difficult due to UART's maximum transfer speed of 14,400 Kbytes per second, as these correction packets would have to transmit the ship's *x* and *y*-coordinates, the force vector and the crosshair's coordinates on top of the other messages. Instead, we minimized the error by simplifying the game, both side engines were removed as these are auxiliary and caused the most error, the ship's acceleration was changed to keep the ship at maximum velocity at all times, and the game was kept as a simple one versus one between players with no other elements, such as asteroids, involved. While the multiplayer experience seems smooth when using two virtual machines on the same computer, the latency is somewhat noticeable when using different computers by TCP.

GRAPHICS

Initially, the game was rendered by drawing points and lines. When swapping to pixmaps, the first challenge was to find a way to rotate the pixmaps. This ended up being accomplished by, when rendering the pixmap, creating vectors with *P* always being the centre of the pixmap and *Q* being each of the other points of the pixmap and applying the rotation matrix. This method worked well but it can generate errors depending on angle since there is a limited number of pixels, and a non-integer coordinate is not directly representable on screen, so an approximation is made. This can lead to small deformations of the pixmap or even some pixels inside the pixmap not being filled. However, the errors are small enough that they are not very noticeable.

RTC

While the implementation of RTC was straightforward we ended up using the RTC's interrupts as an unexpected solution to count the frames per second drawn. Since *timer 0* is used to regulate the amounts of FPS, using it to also measure FPS would be unreliable, and since the RTC generates interrupts every second it became a great solution to this problem.

VIDEO CARD

The Video Card had various interesting topics not covered in the labs, such as *page-flipping* and *vertical synchronization*. With page-flipping, we found that VBE function 07h worked on both our environments but not as we hoped, as explained above. We attempted to used function 0Ah to operate VBE in protected-mode, but it did not work in our environments. We also tried using the VGA registers as VBE did report our computers to be VGA-compatible, but we found that polling *Vsync* still does not work. We attempted page-flipping using the VGA registers, to bypass VBE's problem, but were unable to make it functional. With this in mind, we found triple-buffering to be pointless, and instead use simple double-buffering as default.

DEVICES

Inline assembly was used for all register input and output operations, as we found it to work just as well as linked assembly and didn't require separate files and/or functions for the small blocks of code we wanted to optimize.

Conclusion

Nuno Marques

I found this class to be a great learning experience, dealing with the devices directly, applying the notions of interrupts and polling, rendering techniques, etc. In my opinion, this class is one of the most relevant and best classes I've taken so far but could be improved in three ways:

First, as I had the labs on Monday, I found that at times either the handout or the Minix environment was not ready even at the time of the class, and the tests were available too close to the deadline.

Second, I found the serial-port to be far more interesting and worthier of a lab than the keyboard or mouse.

Finally, the Minix environment proved to be very restrictive on what we could accomplish, for example: no sound, VGA registers don't work properly, VBE protected-mode doesn't work, etc…

Eduardo Campos

In general, I found this project to be quite interesting and challenging, but in a positive manner. Overall, this project was a lot of work and I personally give a lot of credit to my colleague, Nuno Marques, who had some brilliant ideas throughout the development of this game, which made our jobs a lot easier.

Regarding the UC itself, I find that it could be better organized, specially some divergencies we found throughout the semester between the classes' slides and the labs' handouts which made some of the labs unnecessarily confusing. Personally, I found some of the subjects addressed during classes to be a bit outdated in relation to current the technologies used in the "real-world", but I won't opine much about that since I might just be failing to see the true point of the class. Finally, although I think this is because of the implementation of the LCF, the tardiness of the launching of the labs and test grades is somewhat annoying, from a student's point of view, of course, simply because we have no idea whether or not we are succeeding at the UC or not.

Project References

- Henrique Ferrolho's Bitmap works: http://difusal.blogspot.com
- This project was heavily inspired by 1979's game *Asteroids*, produced and owned by *Atari*: https://en.wikipedia.org/wiki/Asteroids_(video_game)
- Rotation matrix: https://en.wikipedia.org/wiki/Rotation_matrix

Installation Instructions

- Enable Serial-port COM1 and COM2;
- Add config to /etc/system.conf.d/ with command lcom_conf add conf/proj;
- To run use:
  - *lcom_run proj* "default", for path "/home/lcom/labs/proj/src/";
  - *lcom_run proj* "path to files", for a different file location.