

* **Laboratório de Computadores 2018/2019**
* **Turma 4 – Grupo 7**
* **27 de dezembro de 2017**

Martim Silva  **-------------------------------------------------------- up201705205**

João Nunes **--------------------------------------------------------** **up201705457**

Index

**Menus** 2

Main Menu 2

Play 3

Options Menu 4

Game Over Menu 5

Highscores menu 5

**IO devices** 6

Timer 7

Keyboard 7

Mouse 7

Graphics Video 7-8

Real Time Clock 9

**Code Structure** 7

Bitmap 7

Game 8

Extras 9

RTC 10

Objects 11

Proj, Vetor 12

Highscores 13

State 14-16

Timer, Mouse, Keyboard, Vbe, Video\_GR, Video 16

ibm\_8042, i8254 16

**Conclusion** 17-18

Call Graph 18

# **Menus**

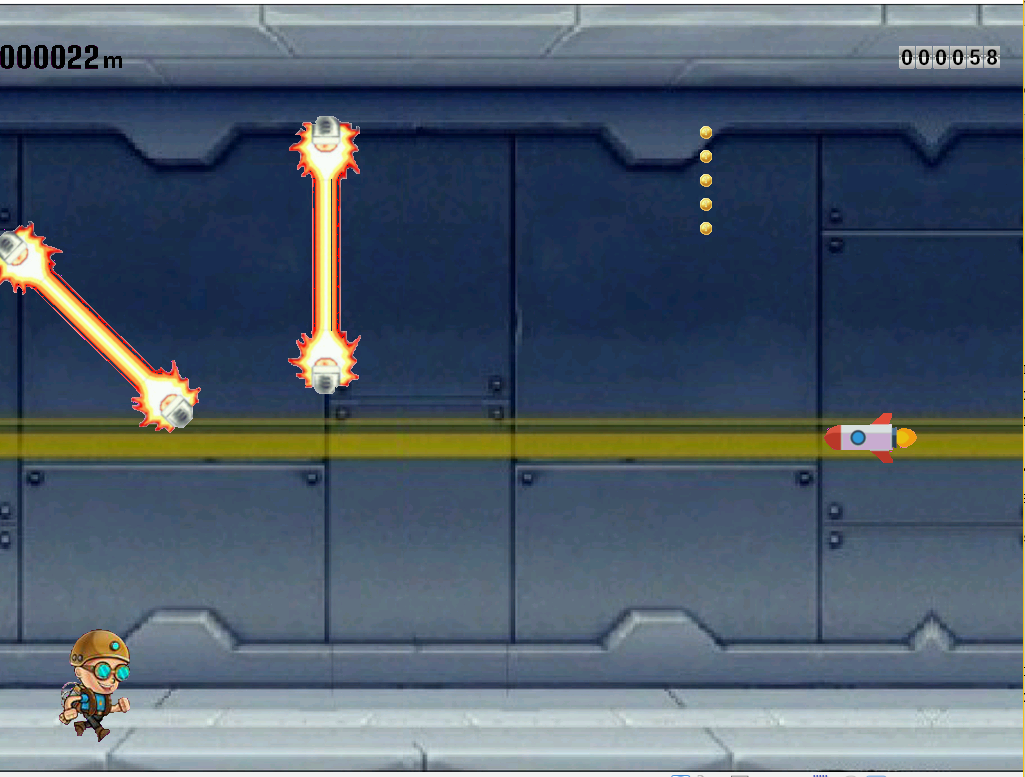
## **Main Menu**

When you start the game you will be sent to this menu, where you have the option to play single-player, multiplayer, to change the options, consult the highscores and finally to exit the game

To select the menu you want you should click on the menu with the mouse and you will be sent to it.

In the options menu after clicking an options, it will be saved and then you can close (or press ESC) to go back

## **Single-Player**

The objective of the game is to survive the longest as possible, while avoiding objects like lasers, rockets and collecting coins to get the maximum score possible.

You can control the jetpack characte with the arrow keys, space bar and/or mouse left button.

In the left upper corner you can the the distance convered ando n the other side you can see your score.

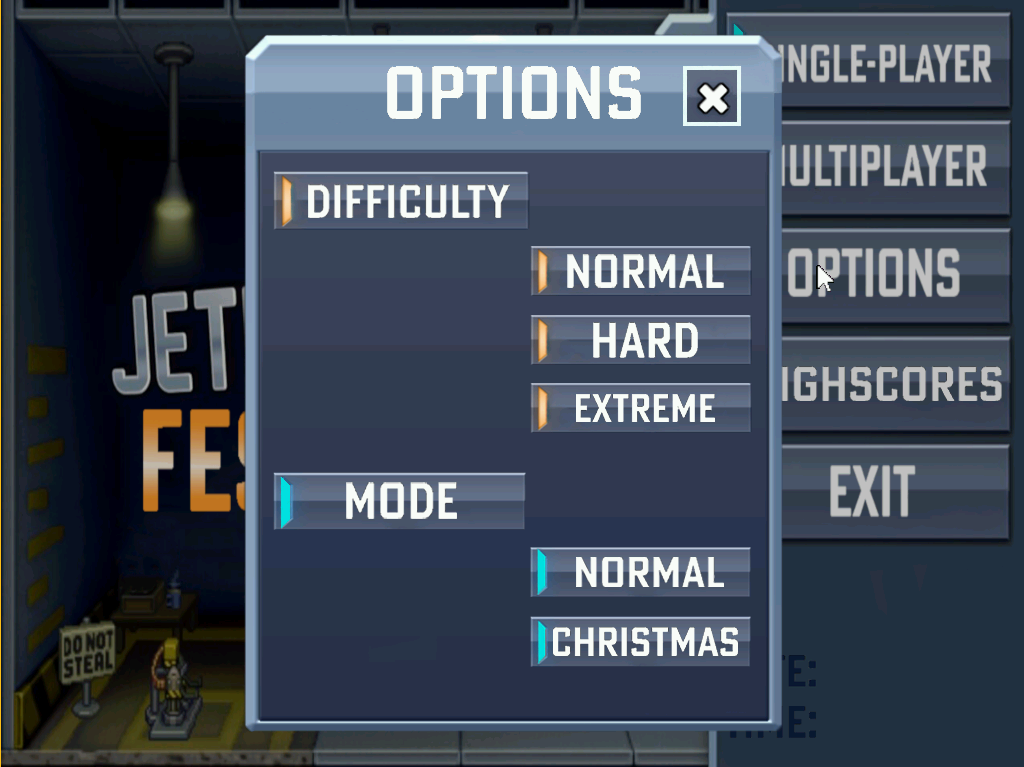
The game is lost when the player collides with a rocket or a laser, and then the player is directed to the gameover option, or if it wants to quiti t can press ESC and also be directed to the gameover menu.

## **Options Menu**

In this menu the player has the option to choose the difficulty (which will impact the speed at which the leve lis played, including the rockets, lasers etc but this will Grant more point per “meter” in the global score counter).

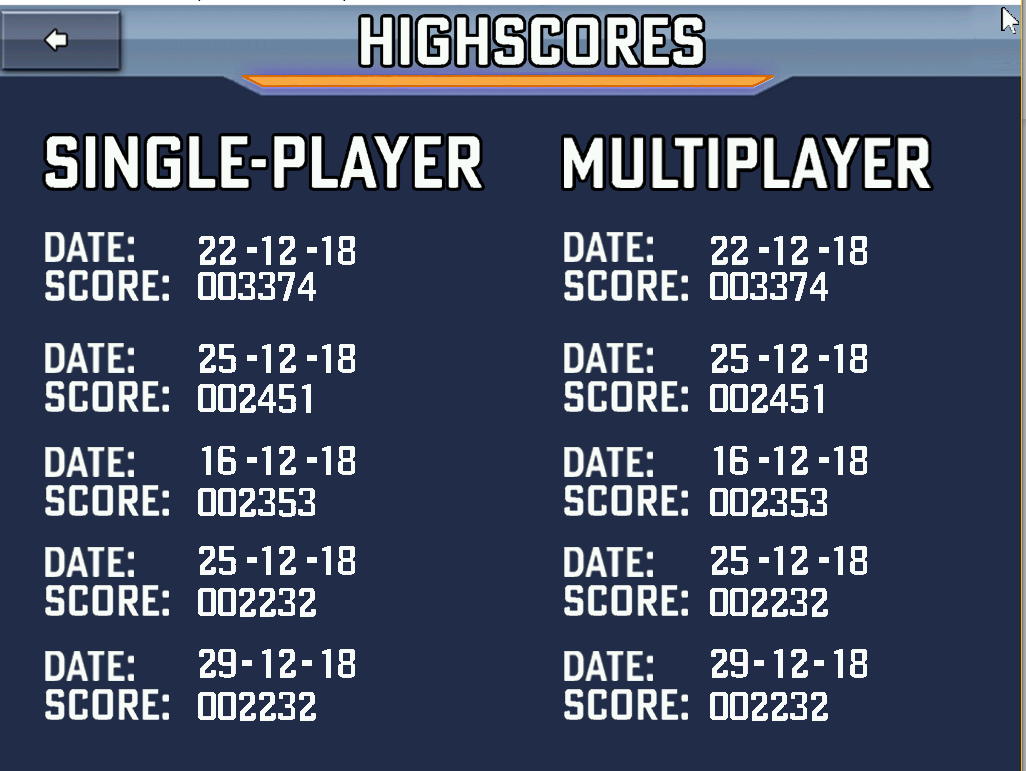
Also, because of the time of the year we are in, we decided to add some snow (with a limito of 20 snow balls, to not became to destracting Also, because of the time of the year we are in, we decided to add some snow (with a limito of 20 snow balls, to not became to distracting).

After selecting all wished options the user must either clicked in the cross to go bak or pressed ESC to go back to the main menu.



## **Game Over Menu**

Every time the player loses (by a collision with a rocket/laser or by quiting, by pressing ESC) the player is reencaminha do/direcionado to this menu, where he was the option to either go to the main menu or to play again with the previous settings i.e same difficulty,mode, ..

***Highscores Over Menu*** 

This menu is accesible through the main menu to see the highscores stored since the creation of the game. This is done via test files (writing and reading)

**Note**: We are trying to implement the multiplayer with the serial port. Since this will be done until the last day, if we can’t finish it we won’t be able to change all the menus including the main menu

## **Timer**

The main purpose of the timer is that at each interruption, it draws all bitmap, updates all objects (mouse, player, lasers, …) positions, and scores.

Basically it updates the frame at 60 fps. The most importante information is dealed in here (after each timer interruption) and is of the most importante devices.

The timer interrupt handler used is in assembly.

## **Keyboard**

This IO device is used for some shortcuts in the menus (exit options and higscores menus and also while playing to quit the game, which leads to the gameover menu).

In the game, with the space bar or the arrow key (up) is able to move the player up and down . The keyboard interrupt handler used is in assembly.

## **Mouse**

This is the device that allow navigation between all menus with left button and with the position of the mouse (obtain with delta x and delta y of mouse packet).

It can also move the player up and down while in the game with left button.

## **Graphics Video**

The graphics mode selected was 117h (1024\*764)

It was implemented the double buffering tecnique (drawing all images in another buffer and after all images are drawn, it draws in the vídeo memory).

We also developed animated sprites for running, flying and even while falling, after a collisions (this was implemented with global counters that kept track of the current bmp images).

In regard of collision detections, to check the collision between rockets, coins and lasers we first check if there is an intersection between 2 bitmap (to avoid unncessary calculations) and then draw a third buffer all rose with the first obstacle (laser, rocket, coin) and while drawing the player bitmap in that buffer we check, before drawing, if that pixel is not rose (in both bitmap) if that is the case then there is a collision.

## **RTC**

Used to read current time and date in the main menu and also to submit highscores.

# Code Structure

## **Bitmap**

This module is responsible to load,delete and draw all bitmap.

Some parts of the code were copied from Henrique Ferrolho bitmap library but with major alterations (professor Pedro Souto allowed it).

Firstly we implemented the images drawn with transparency, something that was not available to us, in the library. Also for collisions detections we created another draw bitmap function that draws it in another buffer to check collisions. We were only told to use xpm library while in the middle of the project, and the unnecessary work to change eerything was not worth it.

All images were done by João Nunes (except rocket.bmp), that used GIMP as the main tool to edit and colored the background of all images in rose.

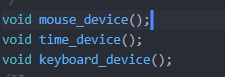
Development: **Martim Silva**

## **Game**

This is the main library that contain the most importante information.

It has a singleton in game\_info, that contain all game information, such as all images, scores, interrupt counter, scan codes,…

The *game\_init()* inicializes the game struct and loads all bitmap, as well as subscribing all IO interrupts.

The *universal\_ih()* is the main loop where everything, in regard of interrupts notifications is handled and dealed in other 3 major functions: *mouse\_device()*, *time\_device()* and *time\_device()* each will deal with its device.

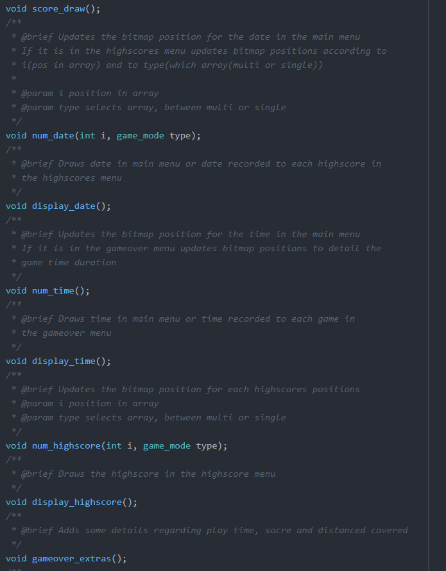
The most complex one is time\_device where the “update frame” is done. Also is here that all objects are randomized, where is checked if there are any collisions and also (obviously) where the page flippinfg function is located (at the end, after generating all bitmap in the double buffer).

Also in this library there are 4 functions responsible to see what menu was clicked in gameover, main menu, highscores, options menu.

Finally, here are located the functions responsible if there is a “retangle colisions” and if so it then checks, pixel by pixel, if there is a collision.

Development: **Martim Silva**

## **Extras**

This library deals with all graphics extras, such as displaying the score, the distance covered, and updating its values.

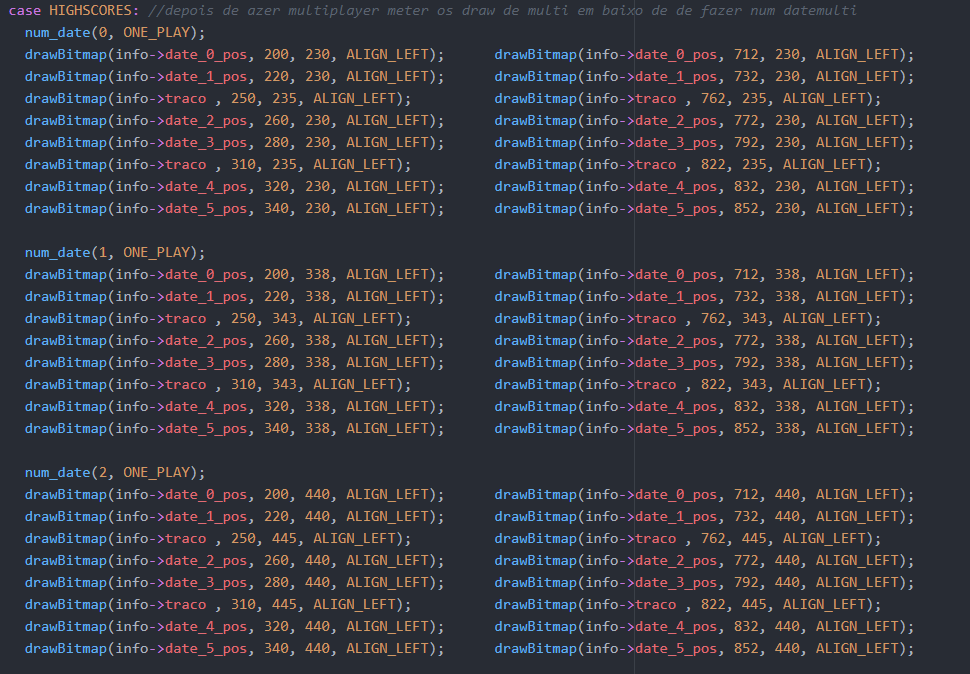
Also, is responsible to display current time and date in the main menu, updating its values and to display the high scores in the highscores menu alongside to the date of the recorded highscore.

Furthermore, displays the game time in the gameover menu with the score of that game and the distance covered .

Finally, have 3 different function (for 3 different number types) that return a bitmap depending on the number received.

For example, in *distance\_update\_metros()* we, each time the function is called separate the counter by digit and then specify which bitmap correspond to pos [0-5] of info->metros, and then draw each variable in a certain position

Development: **Martim Silva**



## **RTC**

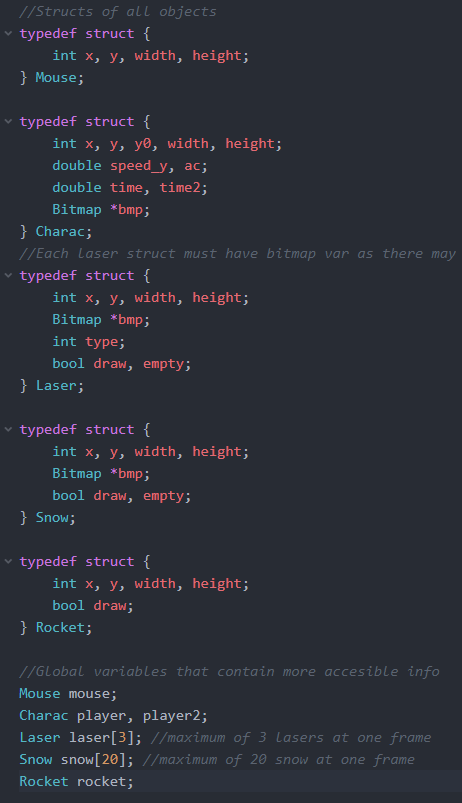
This library reads the time and date in *read\_time()* and *read\_date().*

First, we confirm whether the information is ready to be read (only when updating in progress (UIP) flag is not set).

After that, if the information is in bcd, is converted to binary, if not, then the values that were read are “corrected”.

Development: **Martim Silva**

## **Objects**



This library deals with the global structs Mouse, Charac, Laser, Snow, Rocket. It was decided to implement this technique, to use them as singletons *(“A singleton is a class that allows only a single instance of itself to be created and gives access to that created instance.”*).

The mouse functions only initializes and continuously update the mouse.x mouse.y coordinates.

Because of a rational limit impose by us, it was decided to make the laser (limit of 3) and snow (limit os 20, to not become to distracting) non-dynamic arrays. In contrary, the coins (because of their high quantity) is a dynamic array.

All other functions have init, randomise, update\_pos and draw. The init functions are declared in proj main loop, the randomise is not implemented always the exact same away, but with rand()% randomises the y and/or x coordinate of the object and updates some bool statements . In update\_pos it updates the y and/or x coordinate and if the object “gets out of the screen” its stops drawing (the bool draw and empty have always opposite values, but for understanding purposes we decided to use both). In draw functions it is drawn the bitmap with its x and y coordinate.

Development: **Martim Silva**

## **Proj**

This file inicializes the vídeo graphics in mode 117(1024\*768). Also inicializes the game\_info global struct and the state machine. Finally inicializes same objects positions and values before calling iniversal\_ih() which will deal with all interrupts before delegating to other functions in game.c

The main loop after entering in *universal\_ih()*, continues indefinitely until the user selects the options exit in the main menu. After that the *game\_exit()* is called and returns to minix default text mode in *vg\_exit()*.

Development: **Martim Silva**

***Report***

This report was done by Martim Silva during the festive period (23/12 - 03/01)

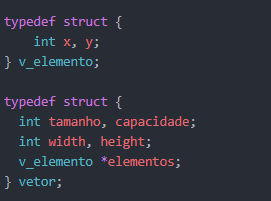
Development: **Martim Silva**

***Video Demo***

The video demo with audio explaining briefly the project was done by Martim Silva during the festive period (23/12 - 03/01)

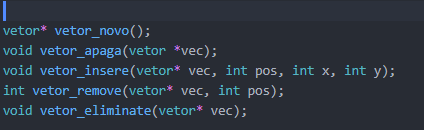
Development: **Martim Silva**

## **Vetor**



## This struct has the x and y coordinate in v\_elemento and then between elements, because it is about the same element (same bitmap) the width, height is equal between all elements of the vetor.

## This library creates dynamically a vetor, in the game to create the coins vetor, and also has functions that insert a new element, that eliminate a specific element or to completely eliminate the vetor.

This is done because, unlike the other object (laser, snow and rocket) the coins have much more quantity in the game and it didn’t make any sense to limit it’s amount to a specific number, therefore we decided to implement this.

Development: **Martim Silva**

## **Highscores**

## This library is responsible to read highscores from a txt file and storing them in a struct of highscore.

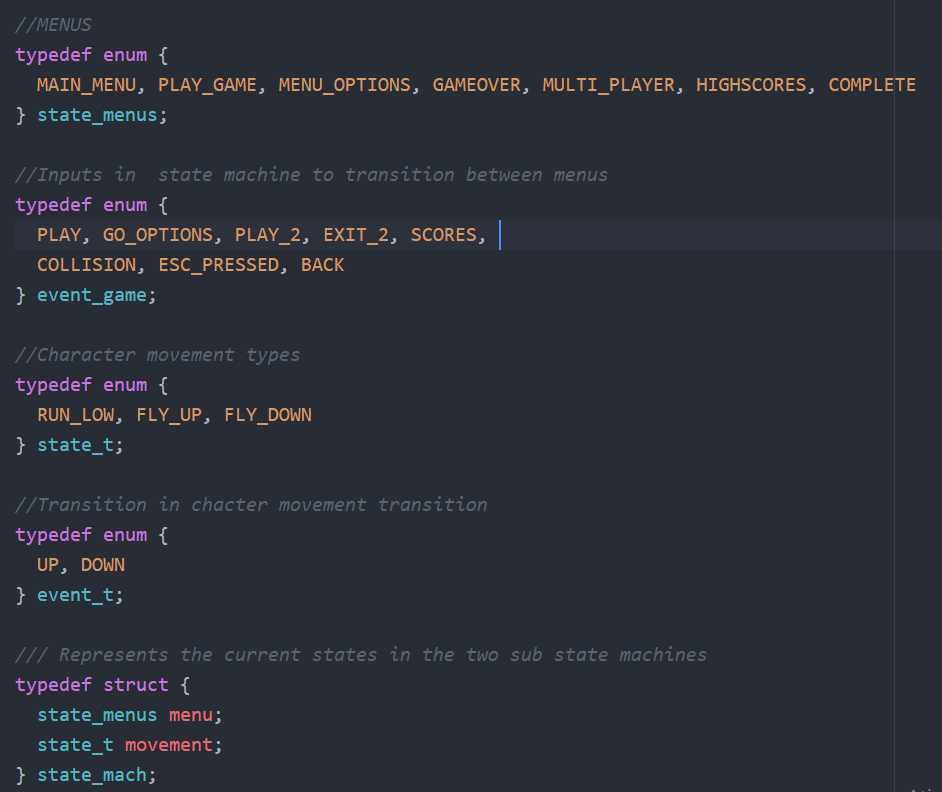
## While the game is running after each game an assement is done. If the score is good enough to enter in the top 5 is putted in the last position(because it is already ordered) and later on is ordered again.

## Before exiting the game the scores are written in the corresponding text file.

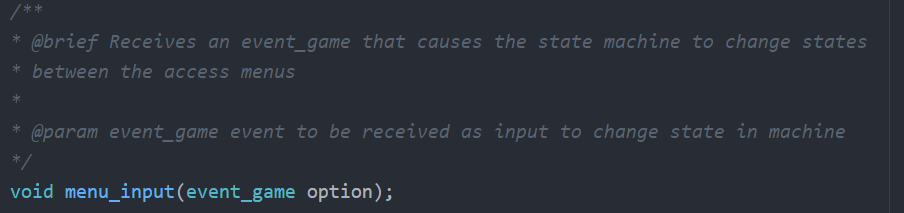
The *highscores\_cleanup()* is able to delete all highscores (is not called) and highscores\_init() just inicializes the struct with values before reading from the txt file.

Development: **Martim** **Silva**

## **State**

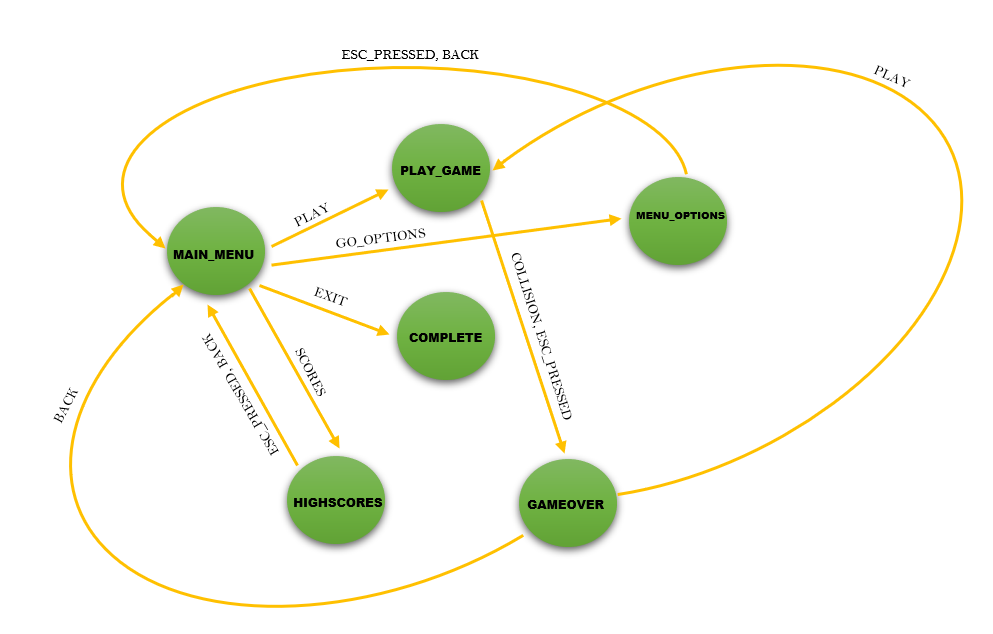


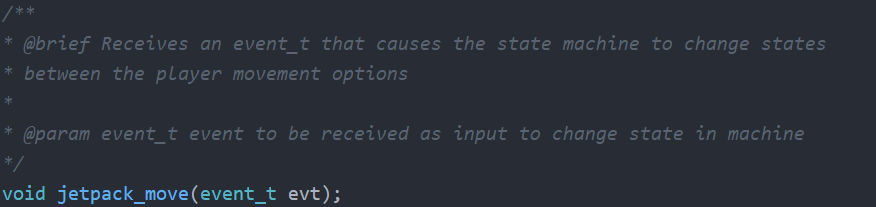
This library includes two state sub-machines within one, where the states of the player movement and menu navigation are encompassed.



The menus are changed in this function, that accept as argument an event\_game option (see above) and the transition is performed in case of a valid option.

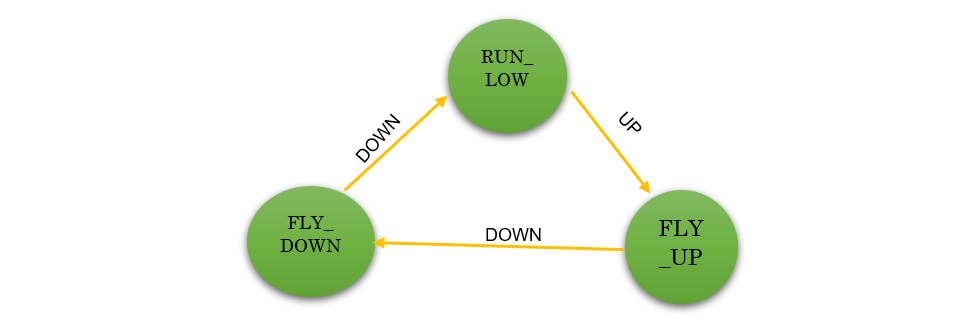
* state\_menus are the states that contain: MAIN\_MENU, PLAY\_GAME, MENU\_OPTIONS, GAMEOVER, MULTI\_PLAYER, HIGHSCORES, COMPLETE
* event\_game are the input options that contain: PLAY, GO\_OPTIONS, PLAY\_2, EXIT, SCORES, COLLISION, ESC\_PRESSED, BACK





The transitions between player movement are changed in this function, that accept as argument an event\_t option (see above) and the transition is performed in case of a valid option.

* state\_t are the states that contain: RUN\_LOW, FLY\_UP, FLY\_DOWN
* event\_t are the input options that contain: UP, DOWN
* To know when the transition between FLY\_DOWN and RUN\_LOW occurs, it has been decided a limit y coordinate that when the player reaches it, it transitions to RUN\_LOW (this is done while updating the player position).



DOWN

UP

The state machine is initialized in proj.c in *state\_mach \*machine\_start()* which allocates memory to the structure and is destroy in *void delete\_machine()* to free all allocated memory in *game\_exit()*.

Development: **Martim Silva**

## **Timer, Mouse, Keyboard, VBE, Video\_GR, Video, ibm\_8042, i8254**

* Almost all code was imported from the labs.
* There were some changes in video\_gr to add some get methods regarding double buffer and a third buffer (for collisions detection purposes) and a page flipping function. The page flipping function copies all memory that is stored in the double buffer (were all bitmap are drawn) and displays it in video\_mem (what the user sees).
* There were also some changes in ibm\_8042 for the mapping of some keys, including the space bar, and the arrow keys
* Finally, there were some changes done by **João Paulo** to put the timer interrupt handler in assembly.

Development: **Martim Silva** (all changes, except the assembly)

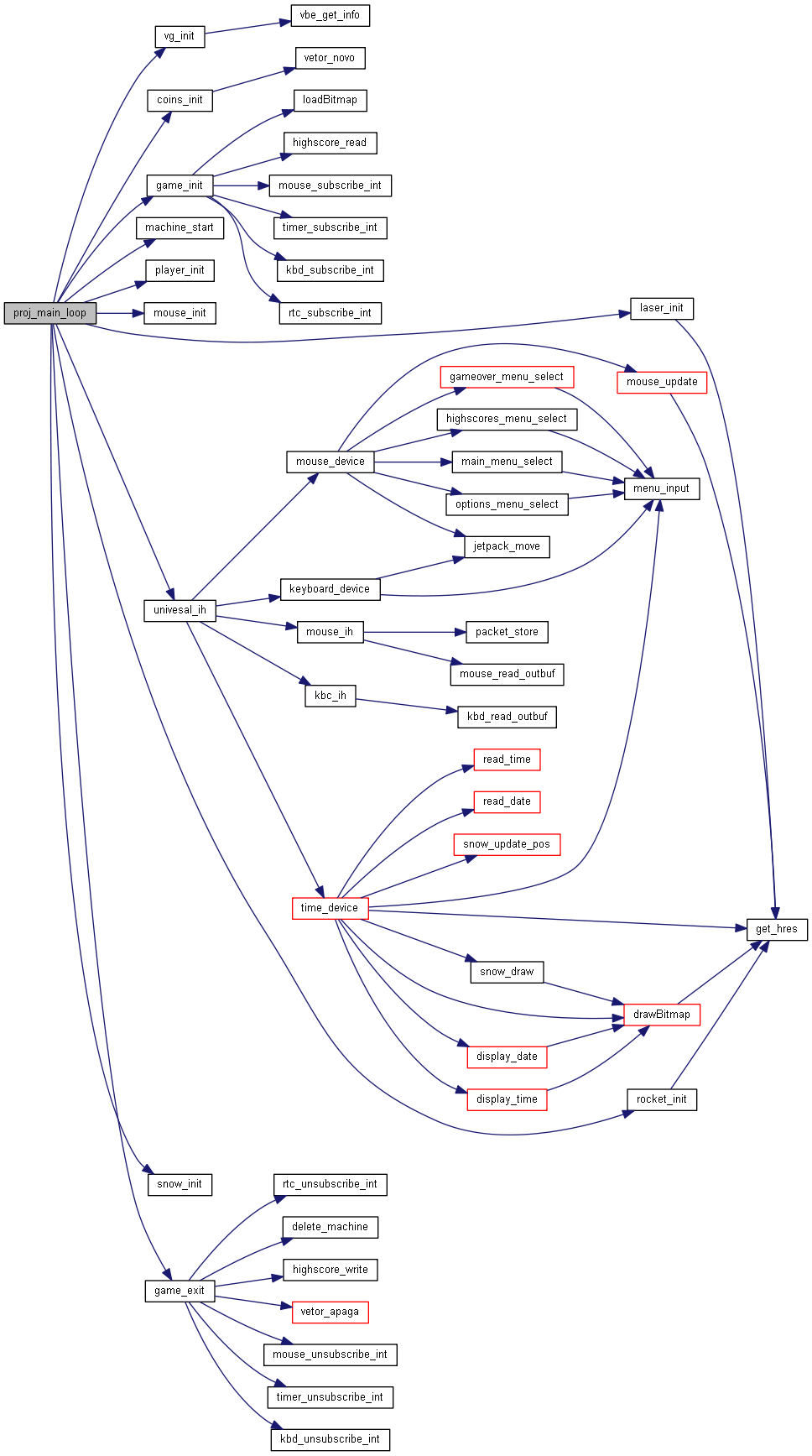
**Doxygen**

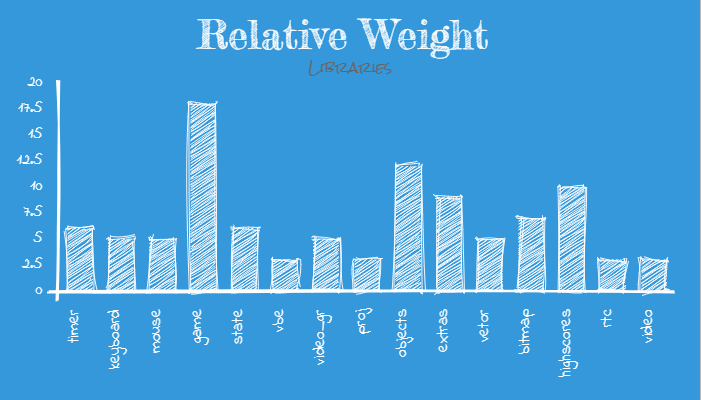
* All doxygen comments in the code was done by Martim Silva (brief, param, returns)
* João Paulo used “Doxygen Wizard” to do the Doxygen with those comments

Development: **Martim Silva** and **João Paulo** (80% / 20%)

## **Conclusion**

**Call graph**





The most difficult and challenging modules were game, objects, extras, bitmap and highscores but all for different reasons. Some demanded knowledge in physics, others in how to allocate properly memory and free it, also how to manage files (reading/writing), also to detect collisions(pixel by pixel) , among other things.

The first thing that was done, was the state machines that help us planned the game and is the base of it. The creation of “classes” like Singletons helped us to share all information between files and that way it simplifies the code by avoiding to, for example always write the same arguments in different functions to share the different structures between files.

As given in the lectures, we decided to start with the more simple functions and work our way up, when it reached a point that we were just calling other functions inside the more complexed ones, for organization and code structure purposes.

The teachers were very helpful and were always ready and available. I would like to give a special thanks to my two TP professors, to professor Pedro Souto who always answered after the lectures and also in the Moodle platform, to professor Pedro Silva and finally to my class mate Miguel Moás who help me understanding some details in the development process.

All in all it was a very fun and challenging subject.

Note: Due to time constrains and the difficult taks we were not able to complete the multiplayer mode and thus the multiplayer highscores as well as the multiplayer options in the main menu does not hava any affect or meaning.