L.EIC
LCOM
2021/2022
Group T02G05

BATTLEGROUND

Project Development Report

Afonso Poças (202008323)
Diogo Silva (202004288)
João Araújo (202007855)
Tomás Martins (201704976)

Index

1. User Instructions

Our project consists of a game called “Battleground” which is reminiscent of battle arena style games that everyone is so fond of.

Initially in the main menu, the user has the option of clicking on the ‘0’ key, returning to the minix text mode or the ‘1’ key, passing to gameplay

Then, after the game starts, you can move the avatar with the WASD keys and shoot the enemies down using the mouse (point and shoot style). Also the fences can electrocute the hero but can also serve as protection against robots. The goal is to stand alive for last, knocking all the enemies down.

The user can exit the game by clicking the ESC key at any time. Furthermore the game ends when all robots are dead or the hero dies.

Fig. 1: Menu interface

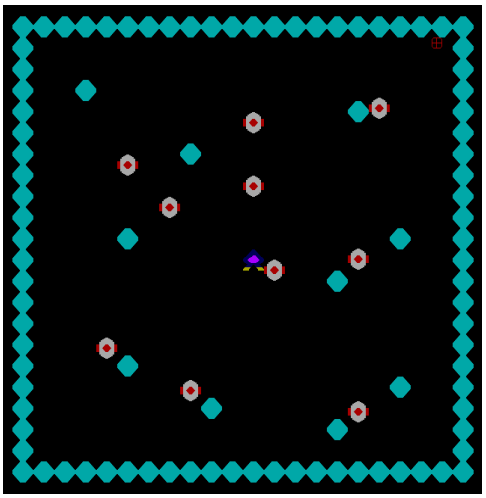

Fig. 2: Arena Interface

2. Project Status

The functionalities implemented are:

A. The game has a menu where the user can choose an option (play, exit).

B. There is a map which is made of fences and contains a player and some robots.

C. The robots will chase the player and try to kill him.

D. The user can move the player around and eliminate robots by shooting them

E. Whenever the player or a robot hits a fence, the player gets electrocuted and is killed immediately but robots won't get electrocuted.

F. The player wins if all robots are eliminated.

G. The player loses if he is caught by one of the robots or if he gets electrocuted

Device	What for	Interrupt/Polling
Timer	Used to control the frames per second, and to	I
KBD	Controlling actions within the game (player movements and the menu option)	I
Mouse	Controls actions within the game (attacks to robots)	I
Video Card	Displays the game and everything related (shows the maze, player and robots on the on the screen in addition to the main menu)	N/A

Timer

- Used to deal with frames per second and control the movement's rate of robots and the bullet.
- Used in the function **gameplay()** loop (**module <game.c>**) (**timer_int_handler()**).

KBD

- Used to select fields, player movements and exit the gameplay.
- The keys used are W (to move the hero up), A (to move the hero left), S (to move the hero down), D (to move the hero right), ESC (to exit the gameplay), 0 (to exit the program) and 1 (to start the gameplay).
- The key is pressed and the function **uint8_t get_key()** assigns a given action to the key stroke, depending on the key pressed.

Mouse

- Used to kill the robots, shooting a bullet.
- Moving the mouse changes the crosshair relative position on the screen, clicking the mouse left button shoots a bullet using the crosshair's actual position as trajectory.

Video Card (VBE)

- Indexed mode (0x105) with 1024x768 for the resolution, with 256 colors. Used double buffering, used **vg_draw_content()** to pass buffer's content to the video physical memory.

3. Code Organization/Structure

Module <controller.c> (weight: 10%)

- Responsible for initializing and terminating the devices used.
- Diogo made this module's code.
- No data structures used.
- Function **init()** for initializing actions for all devices and **leave()** for terminating them.

Module <game.c> (weight: 30%)

- Where the main loop happens and the handling of events related to the game logic occurs.
- Made by Diogo alongside João, with opinions from the remaining elements.
- Used structs to represent the game elements and respective attributes (hero, robots, arena fences, bullets and the crosshair).
- The **gameplay()** function where the handling of interrupts (**driver_receive()**) and events takes place coordinating the entire game logic, **displayArena()** for passing all the arena's content to the console screen and **checkCollisions()** to check collisions between all types of game entities.

Module <kbd.c> (weight: 10%)

- Code responsible for dealing with information from the keyboard.
- Made by Diogo.
- An array containing the scancode byte(s).
- The **kbd_scancode()** function interprets the byte read from the output buffer and fills the contents of the scancode array and **get_key()** to know what key was pressed/released.

Module <menu.c> (weight: 5%)

- Responsible for the game's main menu execution and logic.
- Made by Diogo alongside João, with opinions from the remaining elements.
- No data structures used.

- The **main_menu()** function where a field can be selected by the user, dealing with keyboard interrupts (**driver_receive()**).

Module <proj.c> (weight: 2%)

- Initializes the program process.
- Made by the group in general.
- No data structures used.
- **proj_main_loop()** function that initializes the program main process.

Module <timer.c> (weight: 15%)

- Code responsible for programming the timer and/or dealing with information from it.
- Made by João.
- No data structures used.
- **timer_int_handler()** function increments the number of timer interrupts (ticks), **timer_set_frequency()** to change the rate at which the timer sends interrupts.

Module <utils.c> (weight: 3%)

- Code to facilitate read operations and interpretation of bytes.
- Made by Diogo.
- No data structures used.
- **util_sys_inb()** to properly read data from a port.

Module <vbe.c> (weight: 15%)

- Code responsible for managing the data associated with the video graphics.
- Made by Diogo.
- Struct that contains information about the graphics mode being used.
- **vg_init()** to set a specific video graphics mode, **vg_generate_pixel()** to generate a pixel with a color content inside the video buffer and **vg_draw_content()** to pass video buffer's content to the video physical memory.

Module <mouse.c> (weight: 10%)

- Code responsible for dealing with information from the mouse.

- Made by João.
- An array containing the information related to the packet bytes.
- **mouse_state()** to get the information about a mouse interaction.

Besides the code the group developed, we also used code that we've been using for the whole semester and was made available to all students by the teachers (i8042.h, i8254.h, etc.) and was either used as it was given or adapted to our necessities. There's no code being used besides this and the one developed by us.

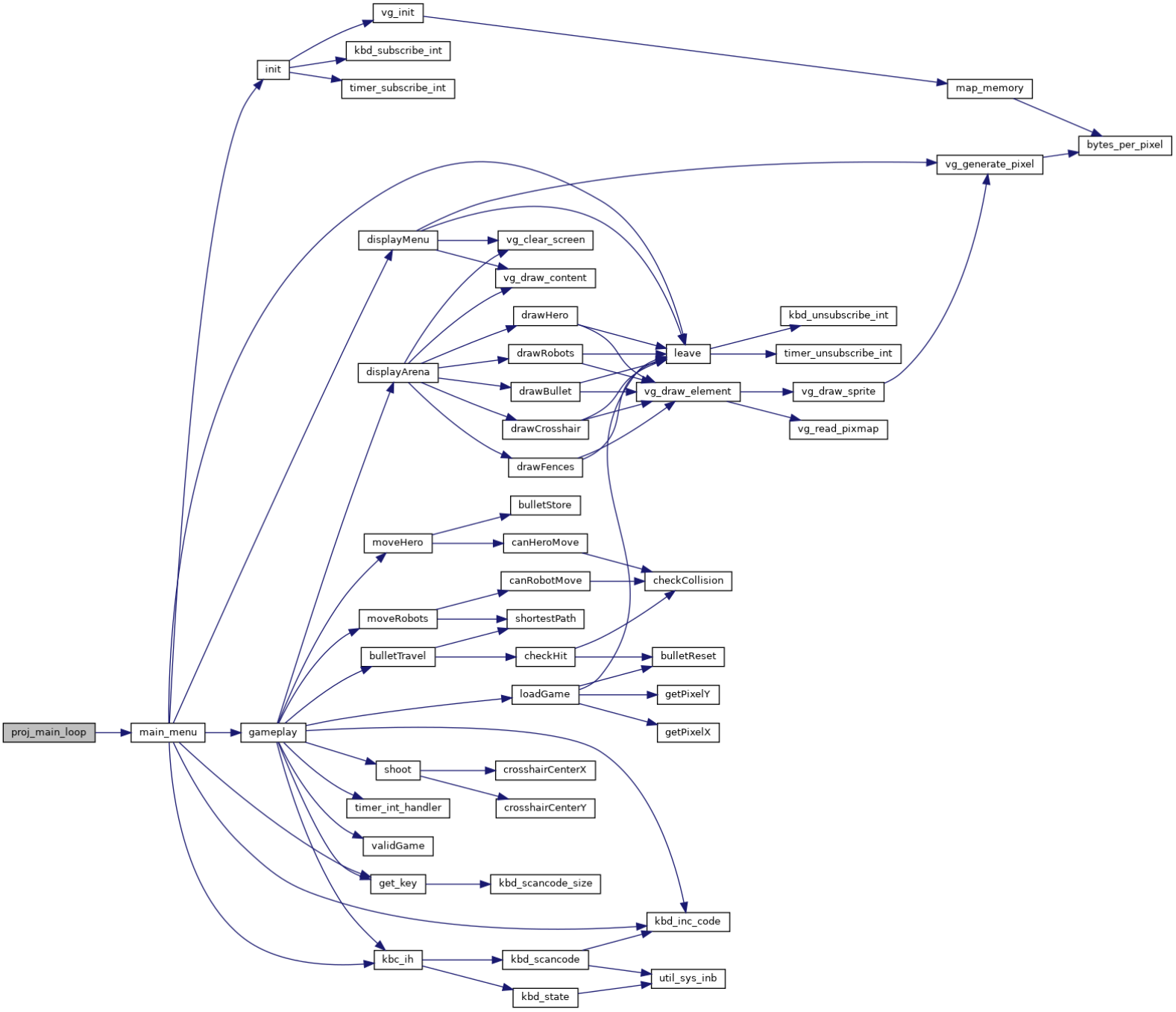

Fig. 3: Dependencies graph

4. Implementation details

For this project, we used the double buffering concept in order to improve the efficiency of our graphics part of the code. In addition to that we could consider our code as event driven since after a certain interaction with the mouse/keyboard, a specific action takes place in our game logic

Regarding the topics not covered in the lectures/labs, there's the example of a collision detection algorithm that we had to develop (in the **module <game.c>**, **checkCollision()** function) in this case, collision between two sprites (entities). Also, the way we dealt with sprites is different from the way it was presented in the lectures/slides, instead of defining a sprite as a struct and using functions to create and destroy the respective elements, we simply read the content stored in our pixmaps and have different structs each one characterizing a different entity of our game with unique variables.

5. Conclusions

The problems we encountered relate to the movement of the crosshair as well as the shooting itself.

Due to time constraints we were unable to design an instructional menu display. As well as gameplay difficulty levels. These two features were initially planned . However, it would be interesting to have the possibility of multiplying.

The main achievement of this work was undoubtedly to develop a game at a low level, and to understand better how the hardware works behind, what at first seems simple.