Project Report

LCOM 21/22

Group T07_G03

- Nuno Gonçalves (up201706864)
- Henrique Silva (up202007242)
- João Araújo (up202004293)
- Pedro Pereira (up201905508)

U. PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

1. User instructions

We developed a single player top-down shooter video-game where a player has to fight against the enemies invading his safe-house.

In our game, one can freely move around the map and use the cursor to shoot down their enemies. The player aims to survive as long as possible, killing as many enemies as possible without losing their health points.

1.1. Usage and Make Commands

Make commands:

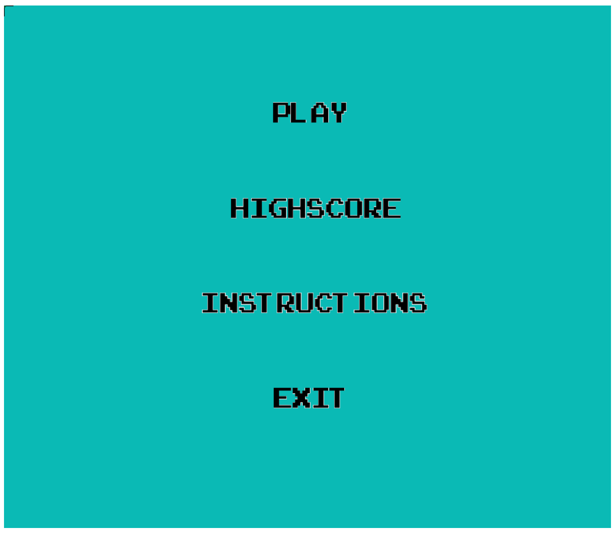
```
cd proj/src
make depend && make                # Install
```

Usage:

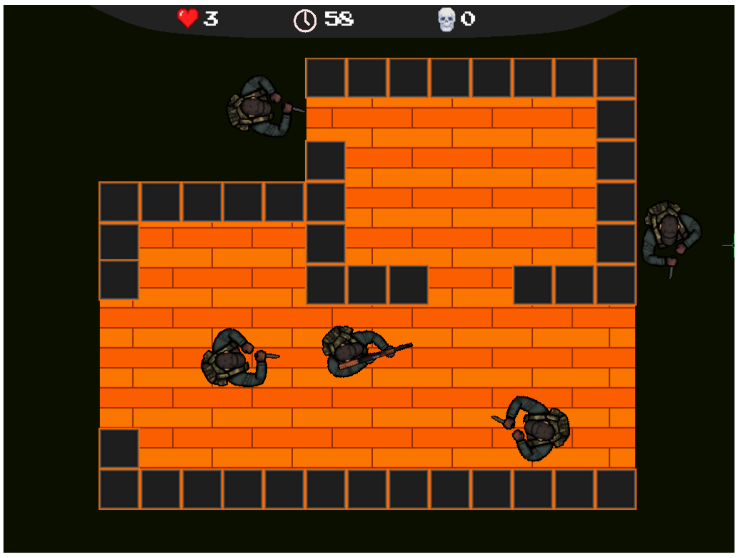
```
cd proj/src
lcom_run proj
```

1.2. How to Play

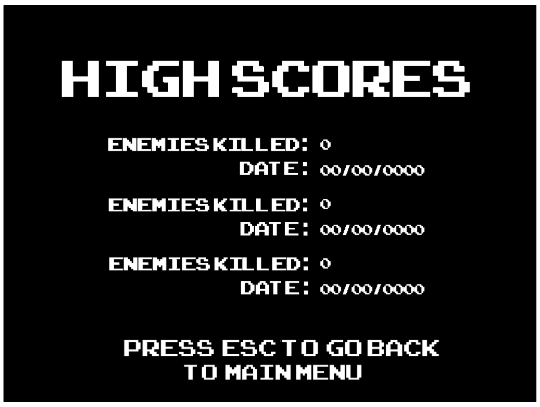
Opening our project application, it greets us with a menu where we can select 1 of 4 options using our mouse pointer:



- Play button -> Starts the game



- Highscore -> Displays the highscore table of the game



HIGH SCORES

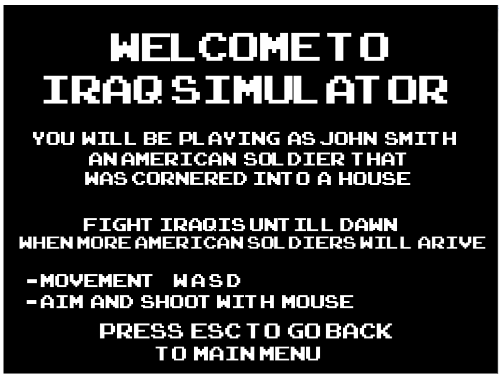
ENEMIES KILLED: 0
DATE: 00/00/0000

ENEMIES KILLED: 0
DATE: 00/00/0000

ENEMIES KILLED: 0
DATE: 00/00/0000

PRESS ESC TO GO BACK
TO MAIN MENU

- Instructions -> Gives instructions on how to play the game



WELCOME TO
IRAQ SIMULATOR

YOU WILL BE PLAYING AS JOHN SMITH
AN AMERICAN SOLDIER THAT
WAS CORNERED INTO A HOUSE

FIGHT IRAQIS UNTILL DAWN
WHEN MORE AMERICAN SOLDIERS WILL ARIVE

-MOVEMENT WASD
-AIM AND SHOOT WITH MOUSE

PRESS ESC TO GO BACK
TO MAIN MENU

- Exit button -> Closes the application

Hovering the mouse over any of the options will highlight the selected option.

In game we feature a map, where the player spawns inside a house, featuring collidable walls and enemies.

We can aim and shoot at our enemies using our mouse as a crosshair. Firing a shot with a mouse left click will trigger an animation and a bullet will be fired. If that bullet hits an enemy it will die and instantly despawn from the map, spawning a new enemy at the spawn position.

Our enemies can attack by stabbing us, which also features an animation.

The player can move around freely in the map by using the keyboard:

- W to move up
- S to move down
- A to move left
- D to move right

We also feature a HUD which gives the player important in-game information:

- The player's remaining lives
- The time left to survive
- The number of killed enemies

2. Project Status

Device	**What for**	**Interrupt/polling**
Timer	Controlling frame rate, counting time left	I
KBD	Moving player in-game	I
Mouse	Menu selection, in-game aiming and shooting	P
Video Card	Menu and game display	NA
RTC	Displaying highscore date	NA

TIMER

We used the timer to keep control of our application frame rate and to count the time left for the player to survive.

KEYBOARD

We used the KBD for the player to be able to move in-game inside the map, to leave game and also the end game, instructions and highscores tab.

MOUSE

We used the mouse as a pointer for the selection of menu options and to aim in-game. We also used the mouse's left click to select an option and to shoot the player's bullets.

VIDEO CARD

Our video card implementation features double buffering, collision detection such as players colliding with walls and bullets with enemies. It also features animated sprites for idling and attacking (stabbing for enemies and shooting for player)..

RTC

The Real Time Clock (RTC) is used to display the date when the player got each highscore in the highscore table.

3. Code organization/structure

Our code is split into two main parts: the code specific to the project in the "project" folder and the libraries with the code related to the I/O devices like the timer, keyboard, mouse and video card, present in the "libraries" folder and mostly reused from the classe's labs, with some changes. However, the RTC folder had to be created from scratch since we haven't implemented it in any of the labs. We also have another folder called media with the sprites for the player, enemies, bullets, background, UI, etc. and functions to get them into the game. In the main project folder we have the 5 following modules:

- **aux_functions.c:** In this module, there are functions for various purposes, related to the processing of interrupts received from the mouse and keyboard and to the drawing of the menus and backgrounds. For this, it also has the data structure keys_t to store which keys the player has pressed in the moment for every relevant press, having a value of 0 if not pressed and 1 if pressed.

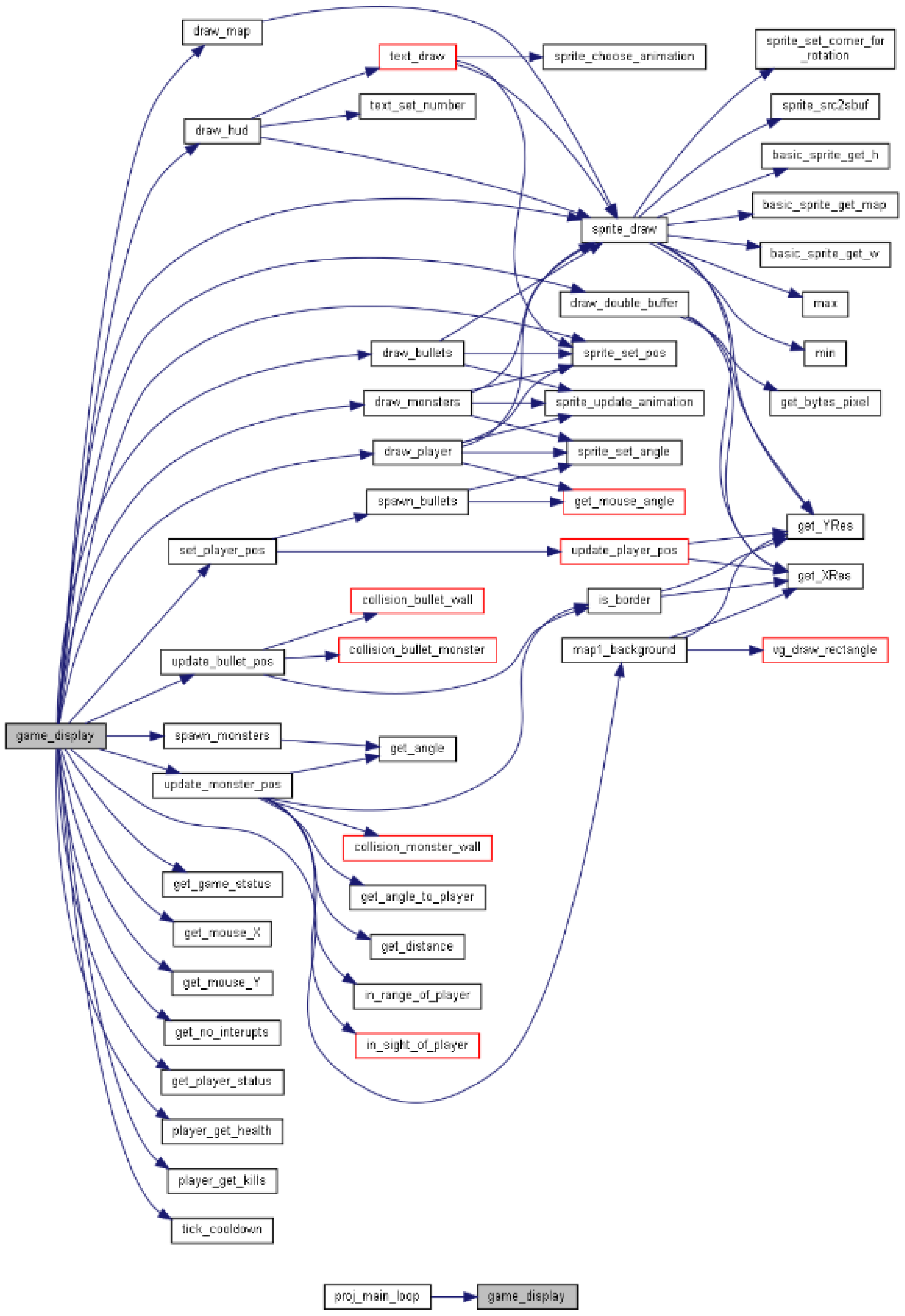
- **elements.c:** This module contains the information and functions related to objects in the game, having a different structure for the player, the enemies and the bullets. These objects have stored all values important to their behavior, like their position, speed, sprites, whether they're alive or spawned in and other additional values that were needed for their implementation. All the functions directly related to these objects, like their creation, spawning, drawing, movement and collisions are all defined on this file, as well as various auxiliary functions to help us manipulate these elements. There are also functions to build the game map and to process collisions of the objects with walls.

- **game.c:** This module takes care of functions involving the game cycle of drawing all of the elements from the previous module as well as despawning them at the end of the game. This module was going to have it's own game loop after the menu loop, however, errors on implementing it made it so we scrapped it, implementing this loop in the main loop instead.

- **interrupts.c:** This module has functions to help us subscribe and unsubscribe all interrupts when the execution of the program starts or finishes, as well as a function to help us retrieve the irq bit that each device is assigned to.

- **proj.c.:** This is the main module of the project, containing the game loop and the main function. As such, it takes care of calling all the functions needed to set up the game, and then it starts the driver-receive loop. For this, there are some variables to keep track of which state the game is in. ***finished*** is used for knowing when the program should stop executing the loop. At first, the loop starts executing the main menu, and then if the 'Play' button is pressed, the ***game_enter*** variable is used to signal that the gameplay loop should be ran. Then, there are other variables representing other states of the program. ***instructions*** signals if the instructions menu should be displayed, ***gameover*** for the game over screen and ***highscore*** for the high scores table. At the end of execution, when ***finished*** is set to true, all interrupts are unsubscribed and memory used in the execution of the program is freed, closing it.

To see how the functions are called, here you can see the graph of the main game function:



4. Implementation details

4.1. Object-oriented programming in non object-oriented language

Despite C not being an object-oriented language, with the use of typedef struct, we tried to make each of our game elements work as a class. All of them were defined in elements.h and it's methods implemented in elements.c.

4.2. Animations with XPMs

Through the use of various slightly different XPMs stored in the folder media, we created sprites, each composed from multiple basic sprites that constitute animations. To complement it, each sprite has an associated angle (ranging from 0 to 2π) which allows the animation to rotate over itself.

5. Conclusion

While developing our project we realized that some features like local multiplayer and a map creator/generator could have been a good addition to our project.

Although we felt we were heavily time limited, which definitely contributed to us not being able to implement all planned features, we believe the game's playability, animations and graphics, which are all working well, are our main achievements of the project.