Lowlife Champions of Mosscrest

Masters in Informatics and Computing Engineering

Computer Laboratory (EIC0020)

Class 6 - Group 1

November 2020 - January 2021

Joel Alexandre Vieira Fernandes - up201904977

João Ferreira Baltazar - up201905616

Index

User Instructions

After compiling, the game is run using the following command: `lcom_run proj [path_to_xpm_dir]`, where path_to_xpm_dir (an optional argument) will tell the program where to look for the images to load in game, since no relative paths are allowed.
You will be greeted with the main menu:

Fig 1 - Main menu

Here, the user will be able to choose one of two game modes - singleplayer if he is playing alone or coop mode if the user is playing with another player. Both of these menus will lead the user to the difficulty menu.

Fig 2 - Difficulty menu

This menu will allow the user to choose the difficulty the user desires to play in. We advise all our players to start by playing the easy difficulty mode so as to gain experience to play harder modes. Pressing any difficulty will take the user to one of the two following states depending on the gamemode chosen:

- Will enter in game if the gamemode chosen was singleplayer;
- Will enter in the syncing menu if the gamemode chosen was coop.

Fig 3 - Syncing menu

WARNING: If no other player connects to the game within 10 seconds in coop mode with the same difficulty the user will be led back to the main menu.

When the user gets in game he will be presented with a board similar to the one in Fig 4.

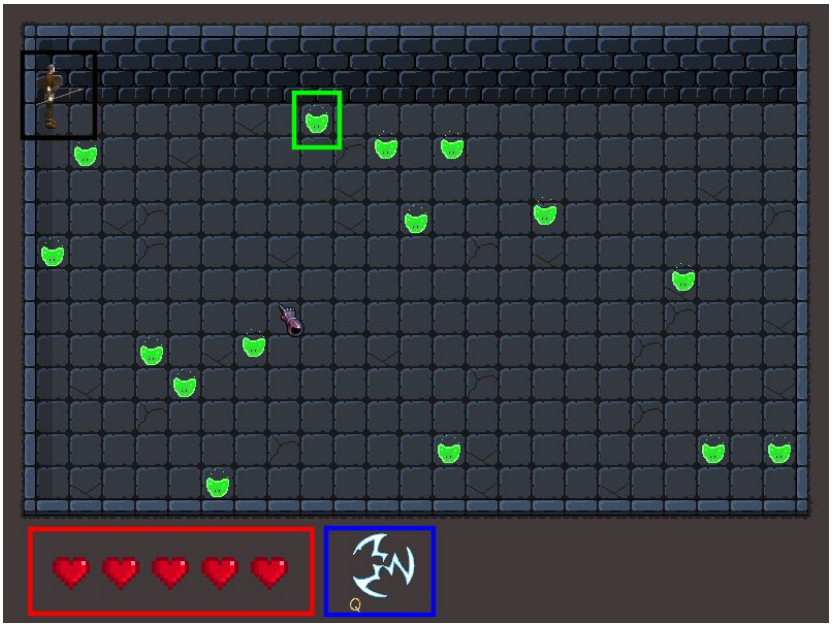

Fig 4 - In game board example (easy difficulty)

In this figure there are some topics that are important to highlight:

- The area in **blue** represents the character's special ability (if there is nothing in there that means that the ability is on cooldown);
- The area in **red** represents the character's health;
- The area in **green** represents an enemy;
- The area in **black** represents a character. If coop mode is activated 2 of these will appear.

In game, enemy creatures will walk around and attack the player if they can not walk in the desired direction. Eventually, if the player's health drops below zero he will be presented with the following screen:

Fig 5 - Death screen

On the other hand, if you win you will be presented with the following screen:

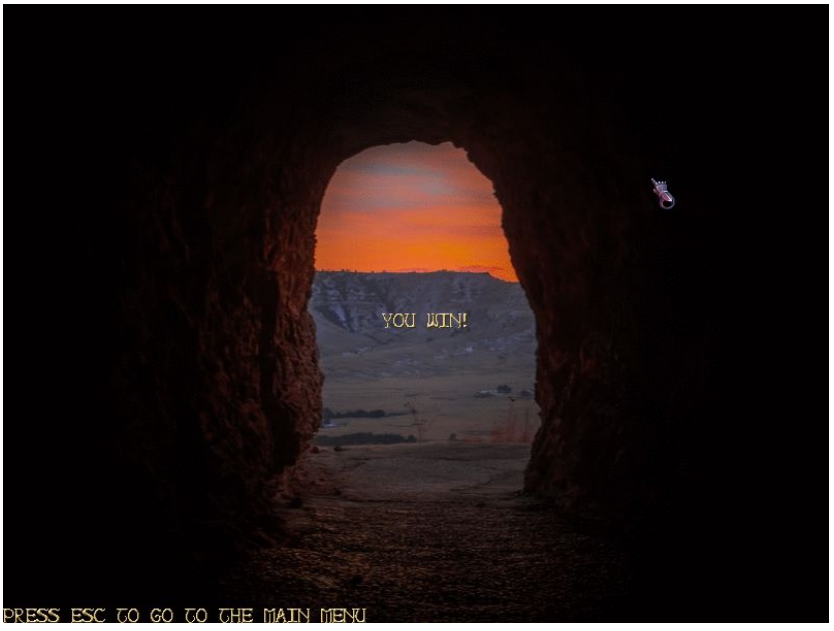

Fig 6 - Victory screen

Note: In all these menus (except the syncing menu - Fig 3) you can go back or exit if you press the ESC key.

Keybinds and controls

Q - Channel the special ability. This ability can be triggered by pressing the **left mouse button** in a valid direction and canceled by pressing the **right mouse button**.
W - To move forward.
S - To move backwards.
A - To move to the left.
D - To move to the right.
ESC - To exit or navigate the menus.

Project Status

I/O devices used

Device	In game functionality	Interrupts
Timer	Set frame rate for display and program logic	Yes
Keyboard	Player movement and interaction; escaping game and menus	Yes
Graphics card	Sprites/menus/animations/font display	N/A
Mouse	Cursor, menu navigation and special ability aim	Yes
RTC	Read date for the game's random seed; alarm for the special ability's cooldown	Yes
Serial port	Communicate between the 2 symmetric players	Yes (receive) / No (transmit)

Timer

Used to update the screen and game logic at 60Hz. The player and enemy actions are parsed every 60 frames (1 second). These actions are 30 frames out of phase between each other.

Functions: **on_timer_int()**, which is called every time a timer interrupt is received and handles the internal game clock and graphical display.

Keyboard

Used to control the player and their actions in addition to navigating the menus. Check the section before for more information about the keybindings of the game.

Functions: **process_kbd_input(),** which handles keyboard inputs from the user in game and in all the menus, helped by **kbd_parse_data()**, an improved version of the work done in lab3, which converts KBC bytes to keyboard usable data (scancodes).

Graphics card

Used to display the game and animate the various characters and creatures on the screen. We made use of double buffering (with an auxiliary buffer to store the static backgrounds). We also used fonts to draw text. We tried using VBE's 07h function to implement page flipping but it just introduced a flickering problem with no noticeable performance improvement.

Functions: **vg_init_ours()**, **vg_exit_ours()**, and **paint_pixel()** and all derived functions (paint_xpm(), paint_area(), etc.).

Mouse

Used as a cursor to navigate through the menus, clicking buttons, and to aim the player's special ability. Check the section before for more information about in game use.

Functions: **process_mouse_input(),** which handles mouse inputs from the user in game and in all the menus, helped by **mouse_parse_data()**, an improved version of the work done in lab4, which converts KBC bytes to mouse usable data (mouse packets).

RTC

Used to generate new and random game boards as well as manage the player's special ability cooldown with alarms.

Functions: **rtc_set_alarm_delta_s()** which sets an alarm that will ring *x* seconds after it is defined. **rtc_read_time()** that reads the current time of the day.

Serial port

Used for Coop (cooperation) mode where two players play along with each other. The serial port guarantees that one player action taken on one side is taken on the other side with the help of polling to send, interrupts to receive, and a single FIFO. Our implementation is not flawless as we will discuss later in more detail.

Functions: **process_uart_packet()** which processes the incoming data from the UART. **send_data_uart()** which sends (using polling) information to the UART.

Code Organization/Structure

Modules

Timer

Functionality from lab2.
Developed by: both, previously in the lab.
Data Structures: None
Weight - 2%

Keyboard

Functionality from lab3 - added an enum to express when the data is ready to be processed.
Developed by: both, previously in the lab.
Data Structures: None
Weight - 5%

Mouse

Functionality from lab4, and a new cursor object.
Developed by: both, part of it developed previously.
Data Structures: packet (mouse)
Weight - 5%

Graphics

Some functionality from lab5, but mostly new content - new VBE mode, more robust xpm handling, sprites, animations, fonts, backgrounds, UI, grid.
Developed by: Joel Fernandes (xpms, fonts, backgrounds, UI) and João Baltazar (sprites, animations, grid).
Data Structures: xpm_img_t, GridImage
Weight - 20%

RTC

Read/write registers, time polling, interrupt handling for time update and alarms, seeding enemies from current time, alarm setting for cooldown.
Developed by: João Baltazar.
Data Structures: None
Weight - 5%

UART

Send/receive synchronization information and in-game action requests.

Developed by: Joel Fernandes.

Data Structures: None

Weight - 5%

Dispatcher

Initializes some game values and holds driver_receive() loop.

Developed by: Joel Fernandes and João Baltazar.

Data Structures: None

Weight - 3%

Game Logic

Mainly held in game_helper(), this is the big one. Handles the events, including interrupts, according to the current game state. Also updates the grid and handles entity interactions. Calls virtually all other modules - for example, to synchronize it resorts to the UART, to update the visual representation of the grid and menu it calls Graphics, etc.

Developed by: Joel Fernandes and João Baltazar (50-50).

Data Structures: GreenBlob (enemy), SpearMan (player), Game (game state), grid (holds entities)

Weight - 55%

Function Call Graph

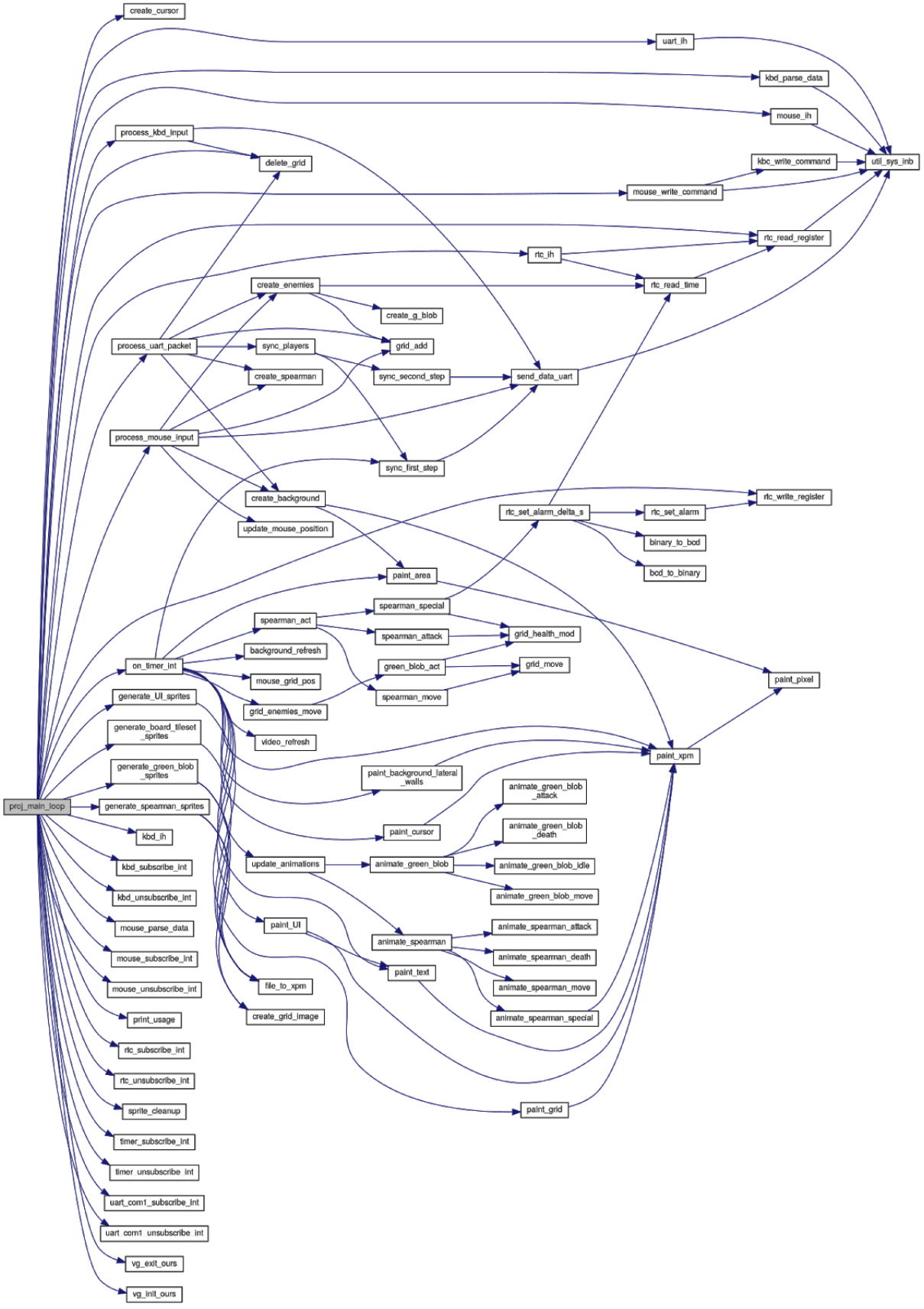

Fig 7 - proj_main_loop function call graph

Implementation Details

Sprites and Animations

We make use of file reading in order to read xpm files in run time in order to lower the size of the executable.

In regards to our implementation of sprites and animations, since our game is grid based, we paint the entities in the screen in a position related to their grid position. Not only that, but we developed a data structure that holds offsets to where to start the painting related to this grid position - **GridImage** - which allows us to be more modular in terms of sprite size and animations.

As mentioned before, animations are done with the help of this data structure which totally fixes the problem of having discrete locations for the entities. This way, in order to create seamless animations we need only to adjust these offsets to satisfy our needs.

RTC

We didn't encounter much difficulty at all implementing the RTC, though we rapidly realised that the RTC uses BCD to represent the time. We found there were some challenges that we needed to address, for example, how could we add seconds to the current time to set an alarm? We had to build functions that could convert from BCD to actual binary and vice versa so we can do direct binary addition.

UART and serial communication protocol

As we mentioned in the project status our implementation of the serial port is not flawless. We knew from the beginning that, because our game and actions are very time-sensitive the serial port would be hard to implement (not to mention the additional hurdle of not knowing anything about the UART before the project started, which hindered our decision-making process when designing the general solution). Until now, we developed various iterations of our communication protocol as described below:

- Send one packet some frames before an actions is taken
    - This presents us with some advantages as we do not have to communicate with the UART so frequently but any delay or problem with the connection could lead to instant desync.
- Send packets every time an action is requested (current implementation)
    - Although this implementation is quite naive (which is to be expected), it is far more robust than the previous one since there is much more redundancy in the communication.
    - A variation of this protocol consists in sending an action request only if it is different from the previous one. While this does reduce the number of packets sent by a fair amount, it does so at some risk as well, since if the first request fails, the others will be silent which comes at a disadvantage when compared with the previous approach.
- One possible addition which we did not attempt because of its burden on the communications would be to, every once in a while, send the board state to potentially fix any errors.

None of them are perfect, and none can assure a safe and stable connection at all times. However, we're sure there's a far better solution than ours and that is why we weighed many pros and cons.

Special thanks

To Francisco Teixeira Lopes, a.k.a. EZSPECIAL, for his constant support throughout the frequent adversities LCOM presents us with. His contribution to this year's class has been massive. Thank you.