Trappy

A Minix game for the “Laboratório de Computadores” curricular unit.

Mestrado Integrado em Engenharia Informática e Computação

Janeiro de 2019/2020

PROJECT REPORT

Turma 4 Grupo 3

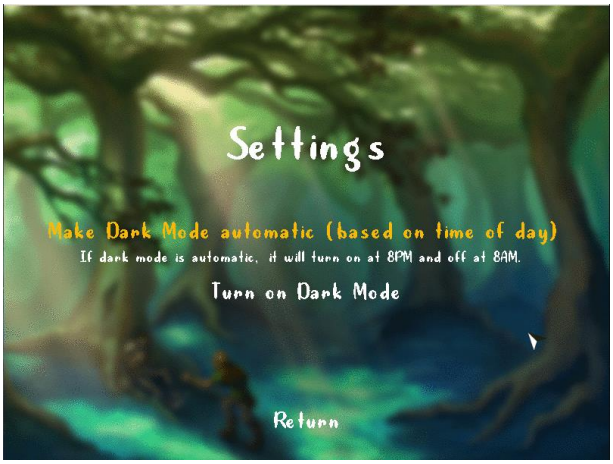
Rafael Cristino (up201806680)
Xavier Pisco (up201806134)

Index

User instructions

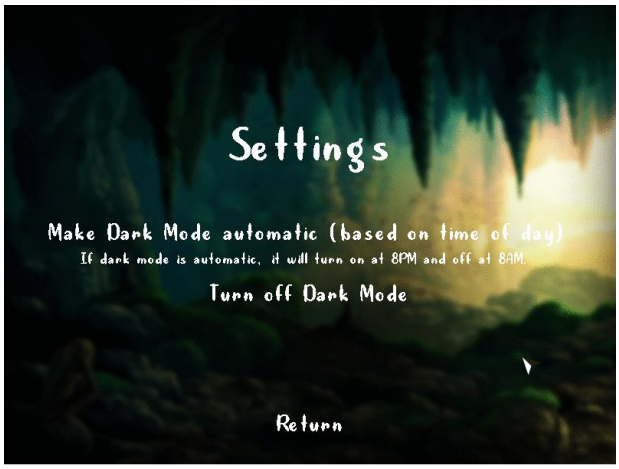
Welcome screen

The user can navigate by clicking the options in the menu



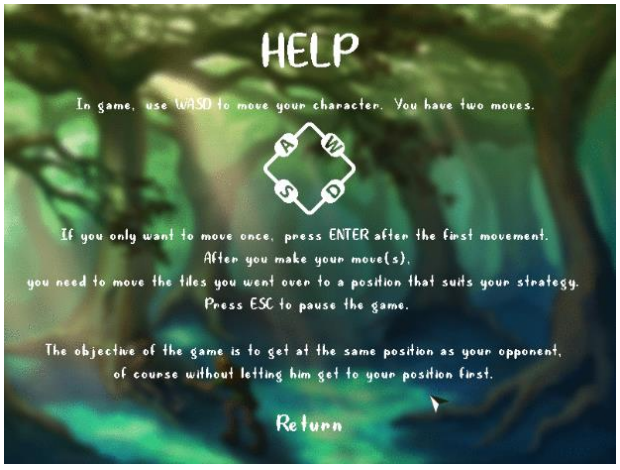
Settings screen

The user can set the dark mode to automatic (like it is now) or manually turn it on or off



Dark mode set manually on

Dark mode set manually on



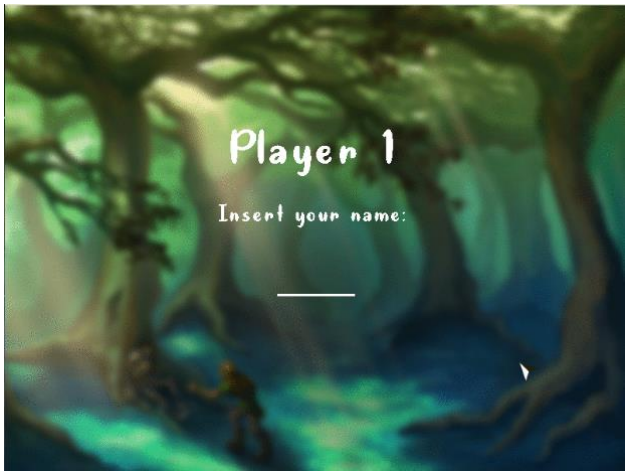
Help screen

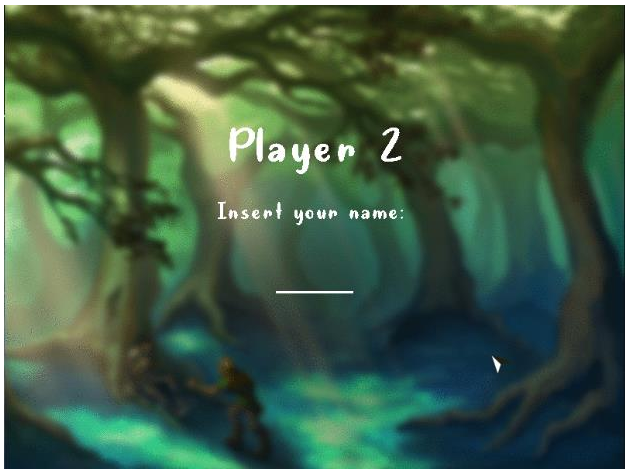
Brief explanation of the game rules

Start Game screen

The user chooses whether the game is going to be on one computer or two

Multiplayer on single computer mode

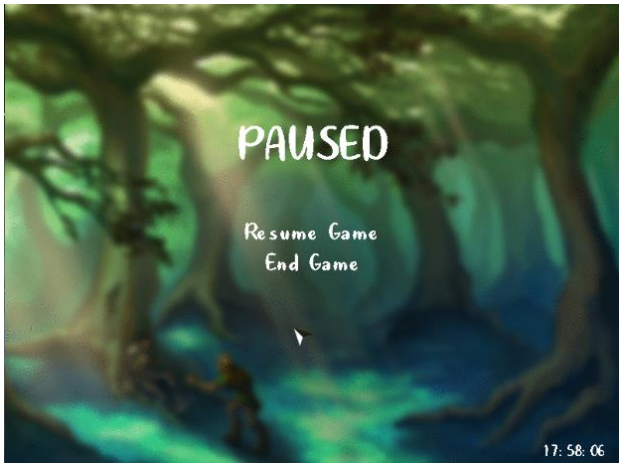




In these two screens, both players insert their names by typing on keyboard. Only letters are accepted. When they've entered their names, they press ENTER to submit.

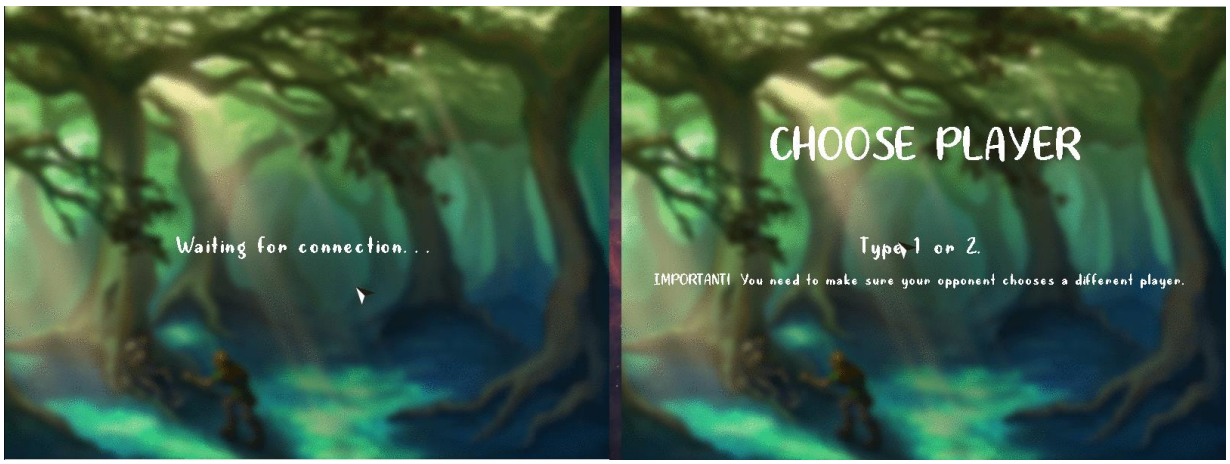
Game has started. Players move their characters using WASD keys. If they only want to move once, they move, and press ENTER. After moving once or twice, they must drag the tiles they moved over to a position that suits their interests, using the mouse. After moving the tiles it's the other player's turn to play.

During each play the player has 10 seconds for each ball movement and 30 seconds to move the tiles. The moved tiles must be placed near another tile.

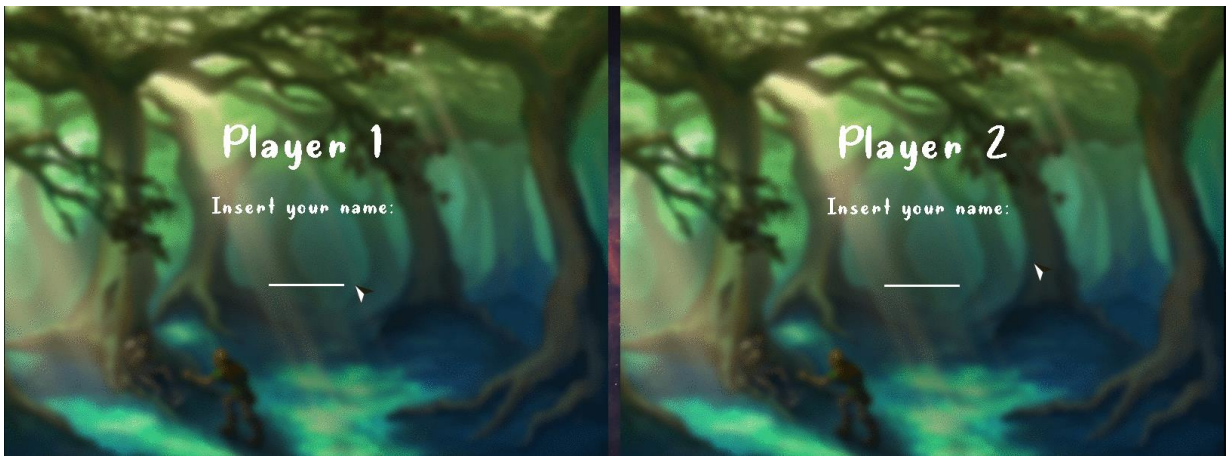


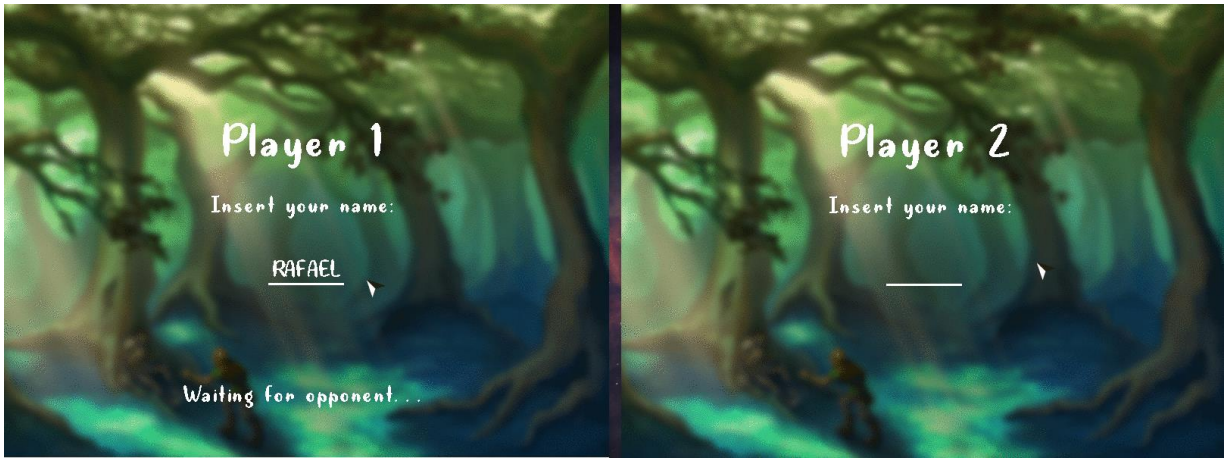
The players can pause the game by pressing the ESC key on the keyboard. The game ends when both players' characters are on the same spot in the game board, and the player who wins is the one who made the move, or when the time expires, the player that was playing loses.

Multiplayer on two computers mode (serial port)

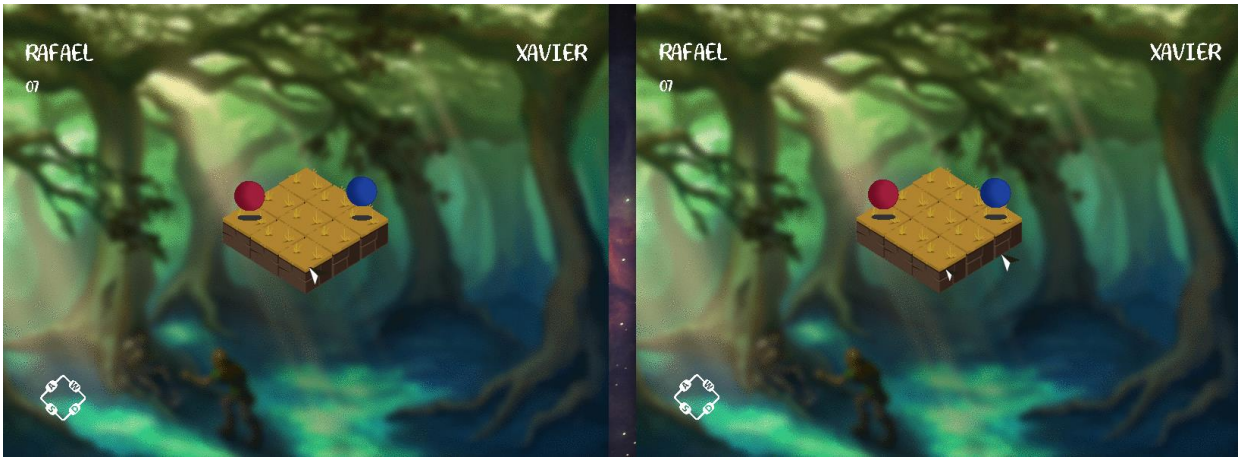


This is the connection screen. Both players need to choose player 1 or 2, while making sure they don't choose the same player.

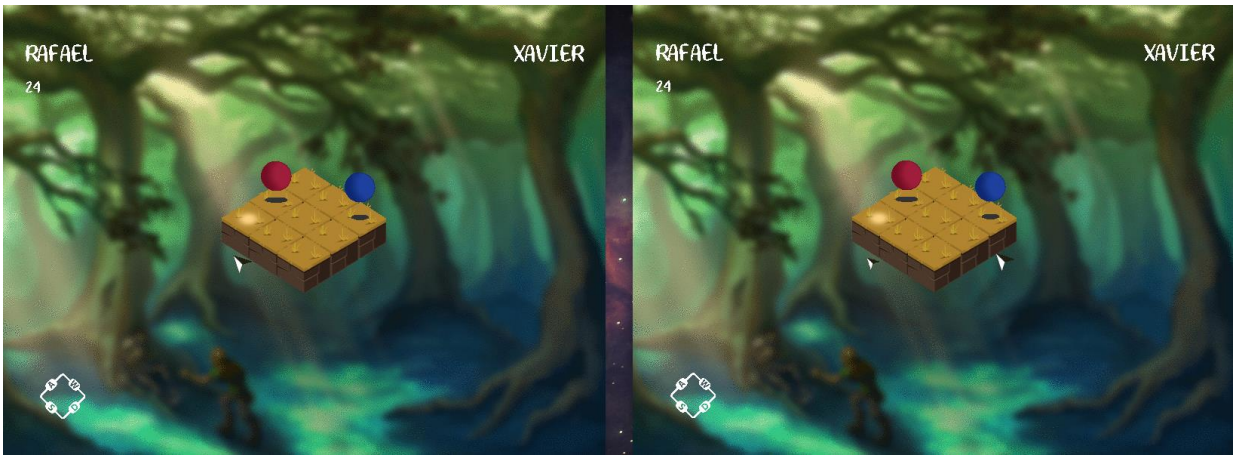


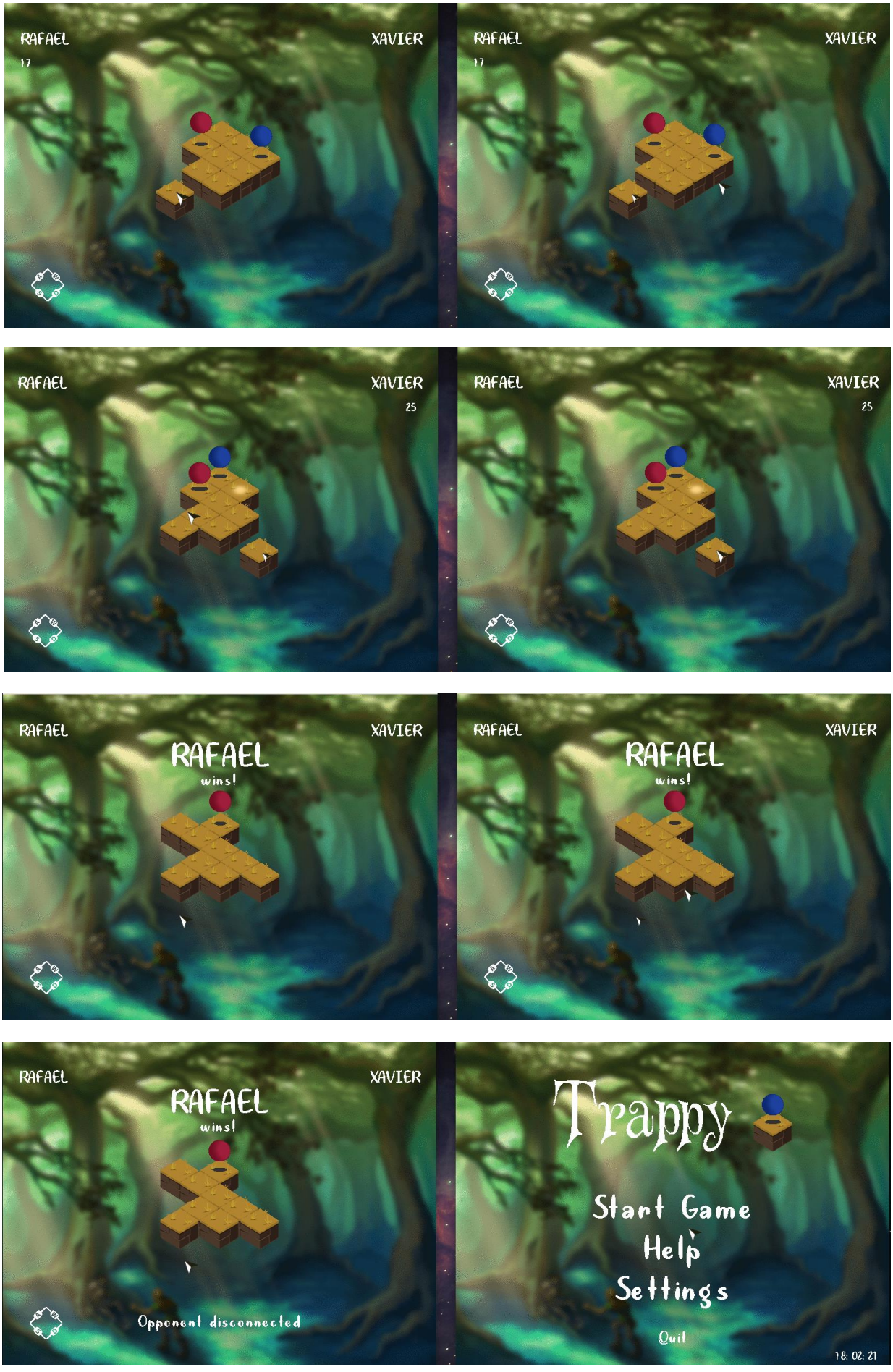


After connecting, players are taken to the name entering prompt, that works the same way as in single computer mode (they enter their names through the keyboard and then press ENTER).



The game starts and works exactly in the same way as single computer mode, only it's in two computers. A user can only pause the game when it's their turn (and the game pauses for both). If a user is disconnected, the other wins automatically.




RAFAEL
XAVIER
17
RAFAEL
XAVIER
17
RAFAEL
XAVIER
25
RAFAEL
XAVIER
25
RAFAEL
XAVIER
RAFAEL
wins!
RAFAEL
XAVIER
RAFAEL
wins!
RAFAEL
XAVIER
RAFAEL
wins!
Opponent disconnected
Trappy
Start Game
Help
Settings
Quit
18: 02: 21

Project Status

Peripherals used

Device	What for	Interrupts
Timer	Counting time between plays, making sprite animations and keeping a constant frame rate.	Y
Keyboard	Receiving user input (text, game actions and menu navigation).	Y
Mouse	Selecting options in menus and dragging tiles.	Y
Graphics Card	Rendering the graphics viewed on the screen.	N
RTC	Knowing the time, for setting dark mode. Writing to its free registers, to save settings page options.	Y
Serial Port	Transmitting data between computers (mouse information, keyboard codes, strings, among others)	Y

Timer

Role: Keep the game at exactly 60 frames per second and count the time each player has left in its play.

Functionality: Interrupt handling to count the time and the frames of the game.

Code: timer_subscribe_int(), timer_unsubscribe_int(), timer_int_handler(), interrupt loop in function game().

Keyboard

Role: Move the player's character on the board, pause the game and write the player's names.

Functionality: Interrupt handling to see which key was pressed and/or released.

The keyboard is used for text input and application and game control. In any screen, ESC is usually the key that will take you to where you were before, or that will exit out of the game. In game, ESC is used to pause the game, and WASD and ENTER keys are used to control your character.

Code: get_letter_input(), handle_keyboard_events(), kbc_ih(), kbd_subscribe_int(), kbd_unsubscribe_int(), among others from the lab.

Mouse

Role: Change menus, move the tiles in the game and keep track of the mouse to draw.

Functionality: Interrupt handling to count the movement in each direction and see which buttons were pressed and/or released.

Our game uses the mouse for the position and the buttons. Both the mouse and the buttons are used to select options presented on the screen and in-game to drag the board tiles when you must (by pressing the left button to start dragging and letting the left button go to stop).

Code: The *mouse_trigger* module is a way to make it easier to detect collisions between the mouse and the objects on the screen, draw_text_button(), handle_mouse_events(), mouse_subscribe_int(), mouse_unsubscribe_int(), among others from the lab.

Graphics Card

Role: Draw all the menus, the mouse, the background and all the pieces in the game on the screen.

Functionality: Change VBE mode to video, write all the xpms to the buffer. Used triple buffer to improve the image.

The graphics card is used in video mode 0x115 (direct color), with 255*255*255 colors and 800x600 pixels in resolution. We used the triple buffering technique: one buffer for the background and another for the rest of the image. Then, both were written to the video memory, avoiding 'watching' the frame being drawn on the screen.

There is moving objects (animated sprites) and collision detection. We used fonts and devised a way to scale text and pixmaps, although very rudimentary (for every 2 pixels drawing only 1, or for every 3 pixels drawing only 1). We used the VBE functions that were necessary in the lab5.

Code: draw_game() (one of the most important), all the functions in the video module (examples: draw_pixel(), draw_pixmap(), draw_string(), draw_string_centered()), draw_player(), draw_tile(), draw_text_button(), draw_main_menu(), draw_tutorial(), draw_pause_menu(), draw_settings(), draw_choosing_mode_menu(), draw_waiting_for_connection(), draw_choosing_host_menu(), draw_string_input(), draw_player1_prompt().

Real Time Clock (RTC)

Role: Changes game to light or dark mode. Shows current time on the welcome screen.

Functionality: Interrupts based on update interrupts to update time every second.

The Real Time Clock was used to read the date and time (to enable automatically switching from dark to light mode and vice-versa) and to store data in its free registers.

Code: the interrupt loop in the game() function, read_rtc(), write_rtc(), write_dark_mode(), rtc_subscribe_int(), rtc_unsubscribe_int() and all the other functions in the RTC module.

Serial Port

Role: Allow the game to be played in two computers, one for each player.

Functionality: Used interrupts, FIFOS.

The serial port was used to enable playing the game in two computers. We used COM1 and COM2, both with the following parameters:

- Word length: 8bits
- Stop bits: 2
- Parity: even
- Bitrate: 115200

We also developed a charqueue C “class” that enables us to store the transmit data and the received data in a more friendly way, making it easier to know what we’re looking at.

The data exchanged is: player data (strings), keyboard codes, mouse variations and whether left button is pressed and critical events (mainly composited of chars, that tell the game what is happening on the other side).

Code: sp_init(), sp_enable_interrupts(), sp_disable_interrupts(), sp_enable_fifo(), sp_set_conf(), sp_set_bitrate(), transmit_string(), transmit_player_data(), transmit_kbd_code(), transmit_mouse_bytes(), transmit_critical_event(), retrieve_info_from_queue(), sp_terminate(), sp_ih() and all the other functions in the SP module.

Code organization/structure

Modules

Proj

Description: Project file given by the teacher to start the game. Only calls the function game(), which is the main function of the game.

Weight: 0%

Game

Description: Most important file in the project. Subscribes, receives, and unsubscribes all interrupts. Starts most of the variables and handles the info received from each the interrupt handlers.

Weight: 30%

Board Tile

Description: Class definition of board's tiles. Has the functions associated with the current position of the tiles.

Weight: 5%

Keyboard

Description: Handles keyboards interrupts and transforms bytes received to readable and usable data. Has the functions to subscribe and to unsubscribe those interrupts.

Weight: 5%

Mouse

Description: Handles mouse interrupts and transforms bytes received to readable and usable data. Has the functions to subscribe and to unsubscribe those interrupts.

Weight: 5%

Timer

Description: Handles timer interrupts and transforms bytes received to readable and usable data. Has the functions to subscribe and to unsubscribe those interrupts.

Weight: 5%

Utils

Description: Utility functions used on various parts of the program.

Weight: 5%

Mouse Trigger

Description: Class definition for squares that execute events when clicked or when mouse is over them. Used mostly in menus and game tiles to check if user is over or clicking somewhere.

Weight: 10%

Player

Description: Class for the characters and their movements on the tiles.

Weight: 10%

RTC

Description: Handles mouse RTC interrupts upon update and transforms bytes received to readable and usable data. Saves mode info for the next usage.

Weight: 5%

Video

Description: Draw functions for all the used strings and xpms, including characters, tiles, background and mouse.

Weight: 5%

XPM Includes

Description: File used to load and store all the programs xpms that can be used later in the program.

Weight: 5%

Charqueue (queue)

Description: Class for queue's of chars used mainly in the serial port communication.

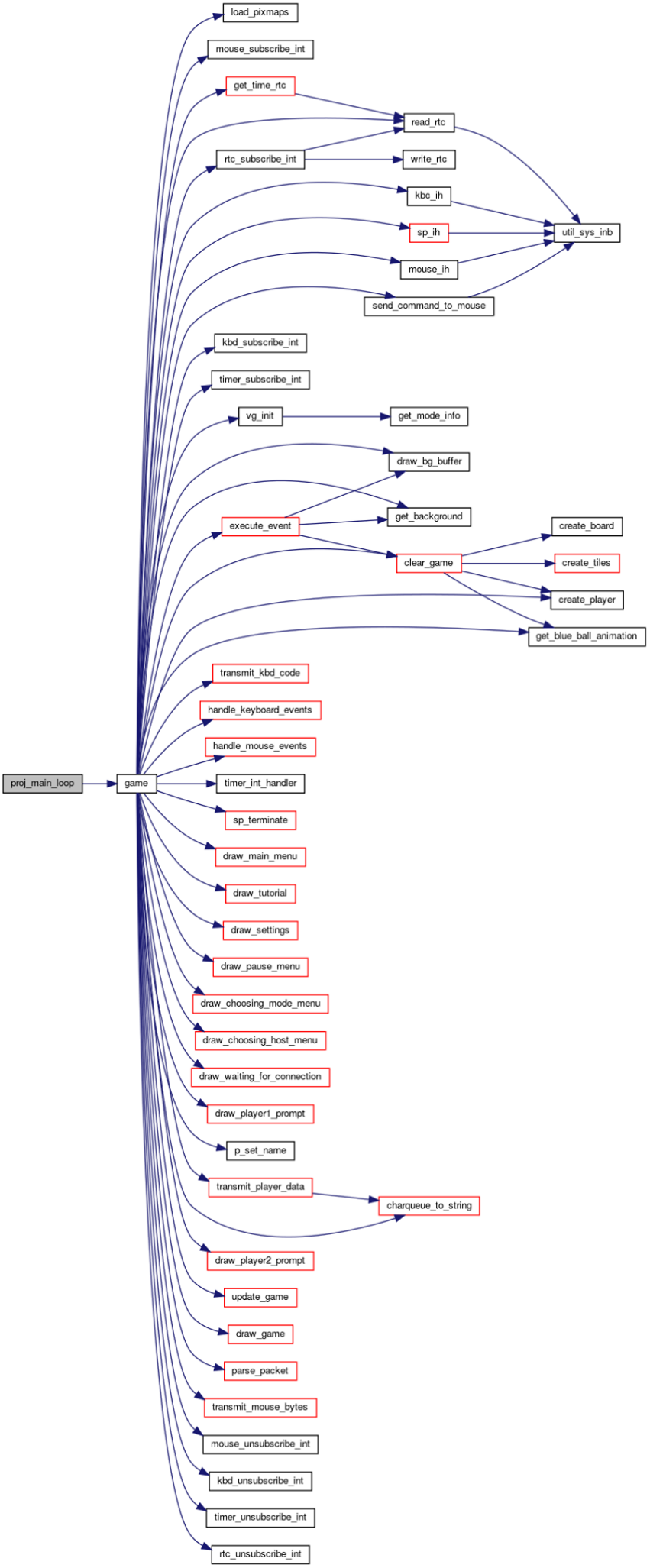
Weight: 5%

Serial Port

Description: Handles all the communication between both computers while playing on two computers. Handles the interrupts generated by both of the computers and subscribes and unsubscribes them.

Weight: 10%

Function Call Graph



Main functions

- Proj_main_loop()
  It's the function that evokes the game (the function itself hasn't much to be said about it).

- Game()
  This function is the center of the program. It sports the interrupt loop, which is the basis of peripheral use and is deeply correlated with it. In the interrupt loop calls are made to other really important functions, like update functions, draw functions, event execution function and i/o interrupt handlers.

- Draw_game() and other draw functions
  These functions are the functions that draw everything to the screen, from the game to all the strings on the menus, by writing to the video memory (we use triple buffering throughout the program).

- Update_game()
  This function is the brains of the game, as it interprets all that is happening in the screen and draws an outcome from it.

- Execute_event()
  This function is the heart of our state machine that is the game. It takes an event, and the current state of the machine and produces an outcome, changing (or not) the state of the game and some of its variables.

- Handle_mouse_events() and handle_keyboard_events()
  These two functions are at the heart of dealing with user input, as they get the information given by the keyboard and mouse interrupt handlers and turn it into a useful meaning to the programme. These functions turn mouse and keyboard input into events.

Implementation details

Peripherals that weren't taught in practical lessons

In our opinion one of the hardest things was the serial port. Even though we think it doesn't need to be a must to pass the subject, in our opinion it would be great if we had already had contact with it in practice. Since that part of the subject is only taught on the theorical lectures it was harder to implement it on the project so we think it should be part of a lab so that people who want to use it to get a better grade and learn more won't face so much of a challenge.

For the real time clock, even if our first practical contact with it was on the project it was an easier part so probably the lectures are enough for most people to understand.

Event Driven Code

The nature of our project is itself driven from event driven code. The game is basically an enum type State (defined in game.h), that jumps from state to state until it ends using events that themselves are also part of an enum type Event (defined in mouse_trigger.h).

In the code, the main event handler we have is the function execute_event(). It takes the state of the game, interprets the event that just happened and produces an outcome, changing (or not) the state of the game, and altering the variables it needed to alter.

Event driven code is a different way of thinking about the flow of a program than what we were used to. It will come handy in the future.

State Machines

The enum type State mentioned above is, essentially a state machine. Depending on the state it currently is, and the information it receives (event), it will jump state, if that jump is so defined. This comes in great relation to this semester's theory of computation curricular unit.

Object orientation in C

When our teacher spoke in the theorical lecture about object orientation in c, we were a bit sceptical, of course. But it works. Some examples of that are our queue module, the player module, the mouse trigger module and the board tile module. It is something that helped greatly to the outcome of this project.

We implemented these “classes” using structs and functions. These functions included a constructor (which returned a pointer to the object created) and get and set methods, among other useful utilities. All of these functions, but the constructor of course, required being passed a pointer to the object as parameter, guaranteeing that that was the object whose data we were getting

Frame generation

Frame generation helped us understand better how graphical computing works, by having to draw every single pixel to the video memory. Also, learning about buffers and what their purpose is was very interesting. We implemented a triple buffer, that increased efficiency because we didn’t have to be drawing the background repeatedly and the image appeared at once on the screen, while with no buffering you could almost see it being drawn.

Developing an abstract data type in C

Developing an abstract data type in C is another cool relation to another of this semester’s curricular units, algorithms and data structures. With the knowledge obtained from this unit, we were able to develop a queue in C, that was very useful to the serial port implementation, as it made everything a lot easier. This abstract data type is another example of “object orientation” in C.

Conclusions

Starting by the bad things, one of the most difficult challenges we had the whole semester was finding all the info we needed. We started by searching on the lab handouts, then we had to go to 1 or 2 lecture slides and search. Sometimes if we missed it, we had to do it all again. So, one of the most important things that could be improved in this subject is the making of new slides, new lab handouts and organize everything in a better way so that future students won't lose so much time going around in circles searching for a little bit of info.

Another big problem we faced was on the MINIX itself, sometimes when calling a function, we lost info on other variables because their reference would change, after some time we found out that if we reduced the number of global variables that would happen less often. Sometimes, the functions would lose the return address and cause havoc at return, which forced us to have to hardcode interrupt handlers in the interrupt loop, instead of having them as a separate function. We know that most likely has nothing to do with LCOM but maybe there is a way to improve that or at least warn students against that.

At last, one of the biggest and simplest improvements this class could have is teaching the basic commands of the linux terminal, not really a big problem on our group but for a lot of students this was their first interaction with UNIX-like systems and that set some people back, one thing that could have been done was on the initial classes teach some commands like cd, ls, pwd, mkdir and some others that could have been useful.

Onto the good things, one of the best things from the subject is the freedom we have when doing the project, we almost didn't have boundaries and could create whatever we wanted. That has motivated us a lot for the subject because we would finally be able to do what we liked.