BLOCK DESTROYER

PROJECT REPORT

PROJECT BY:
Daniel Garcia Silva - up201806524
Nuno Oliveira - up201806525

for Laboratório de Computadores, MIEIC 2nd year

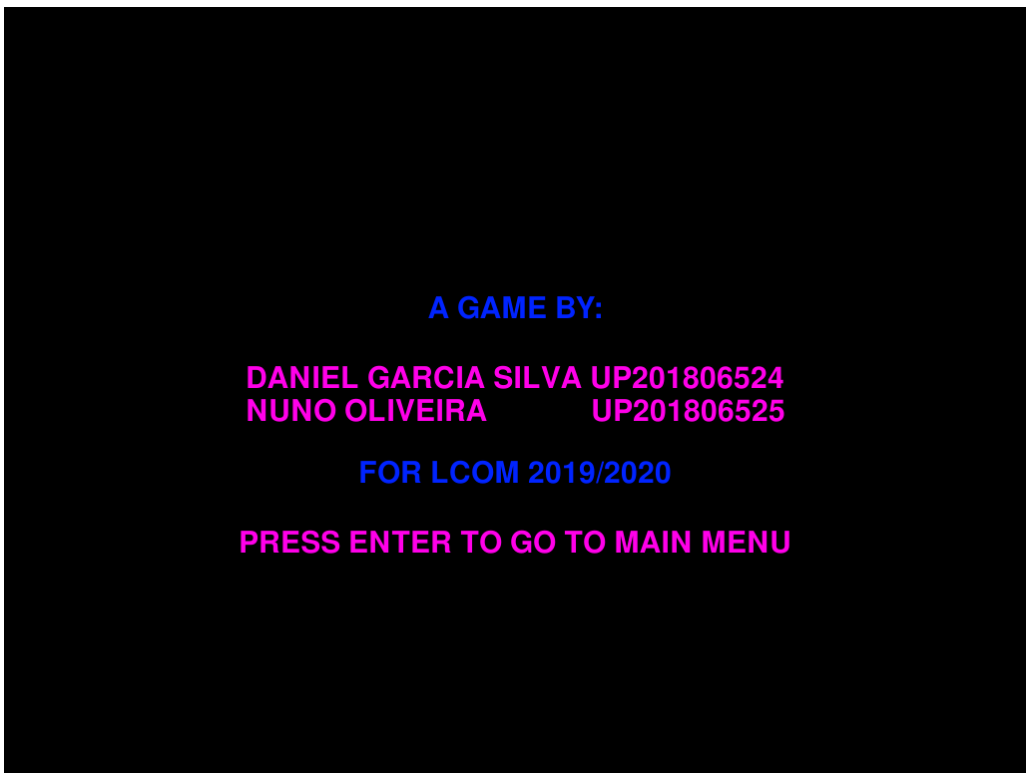
Trabalho do grupo 09 da turma 04

6 de Janeiro de 2020

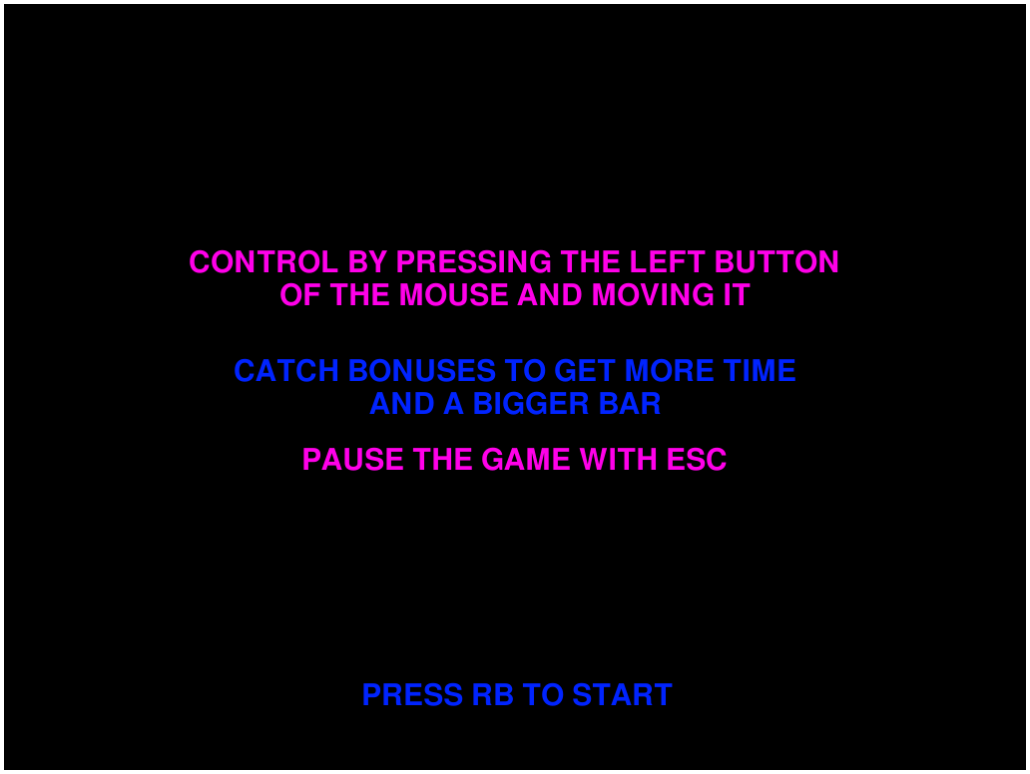
1. USER'S INSTRUCTIONS

1.1. MAIN MENU

Upon running, the program begins with a main menu (see image above). The user can navigate the menu using the UP and DOWN arrow keys and select an option with the ENTER key of the keyboard. The first option, START, will take the user to the actual game (see section 1.2). The second, CREDITS, takes the user to the credits screen (see image below). The last option, EXIT, will exit the program.

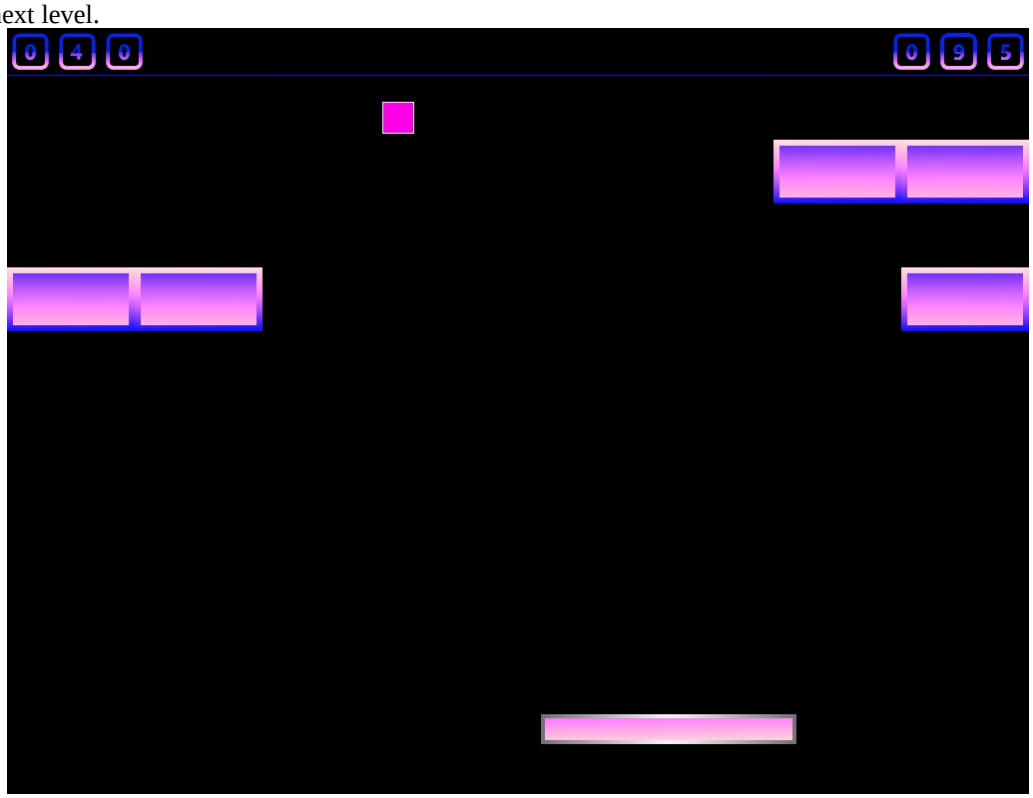


1.2. GAME MODE



The game begins with an explanation of how to play (see image above): you have to use your mouse to drag the player bar on the bottom of the screen to avoid letting the ball fall down into the abyss, while also pointing it to the remaining blocks. You have to press the mouse's right button to begin play (see image below).

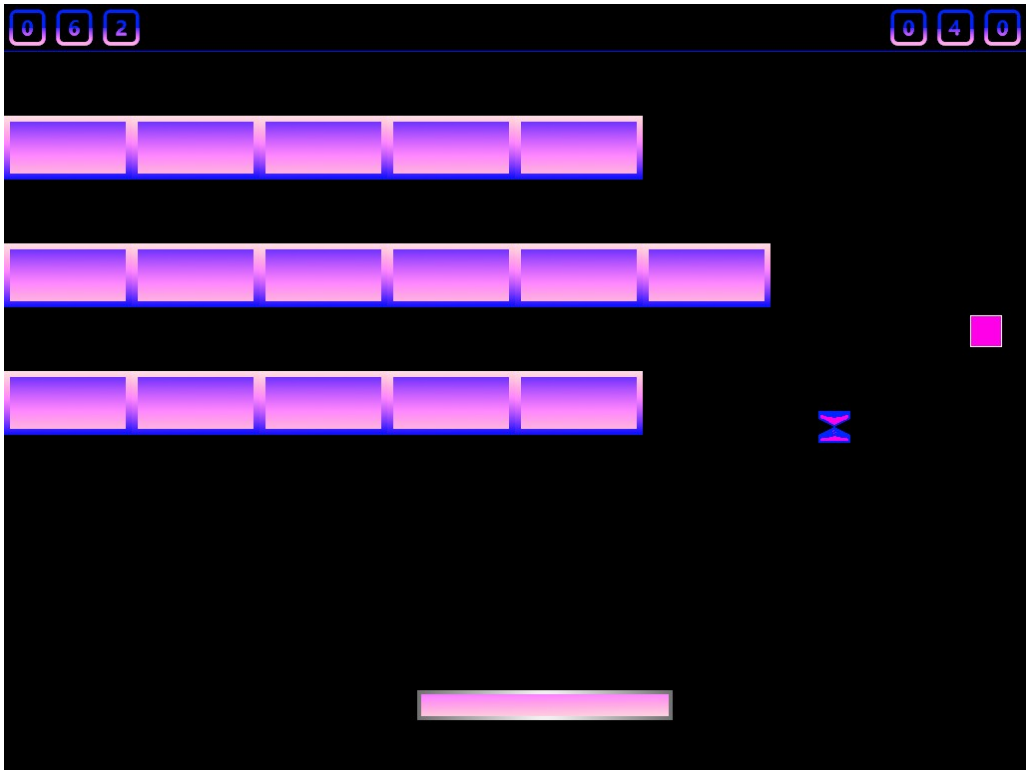
Note: for debugging and demonstration purposes, if you press the middle button of the mouse, it automatically takes you to the next level.

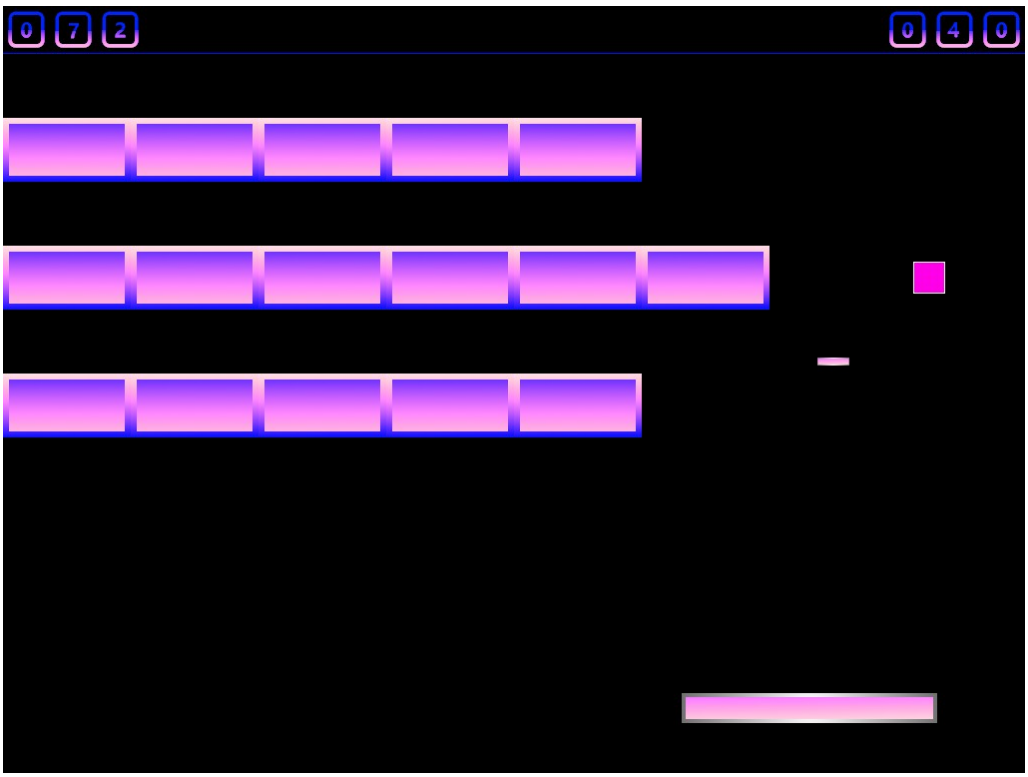


While you’re playing, the point at which the ball hits the player bar changes the angle at which it goes back up. If it hits at the exact middle point of the player, it will go straight up. There’s a region surrouding the center of the player where the ball simply keeps its angle, and beyond that, the further from the center you hit it, the smaller the angle you give to the ball. When the ball hits a block, wall or the ceiling, its angle is maintained.

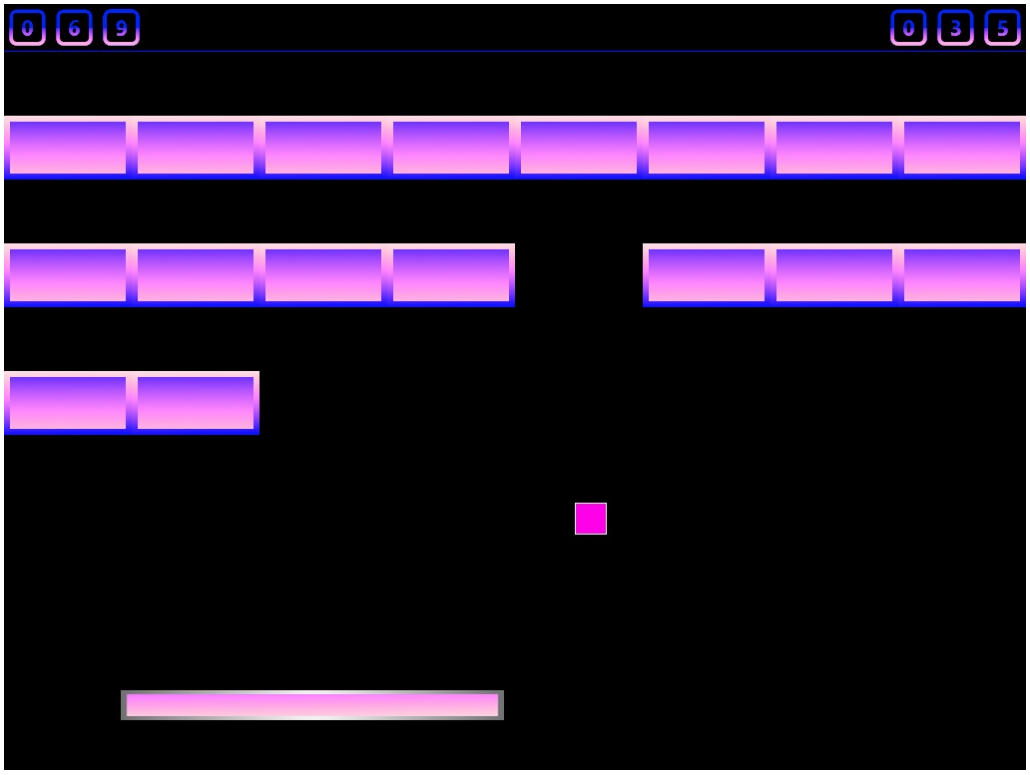
Each time you hit a block, it disappears and 5 points are added to your score, which is displayed at the upper right corner of the screen. You have to destroy the all the blocks before the timer at the upper left corner reaches 0.

There is a small chance (15%) that you get a bonus from destroying a block. There are two kinds of bonuses: time bonus and bar bonus (see images below).

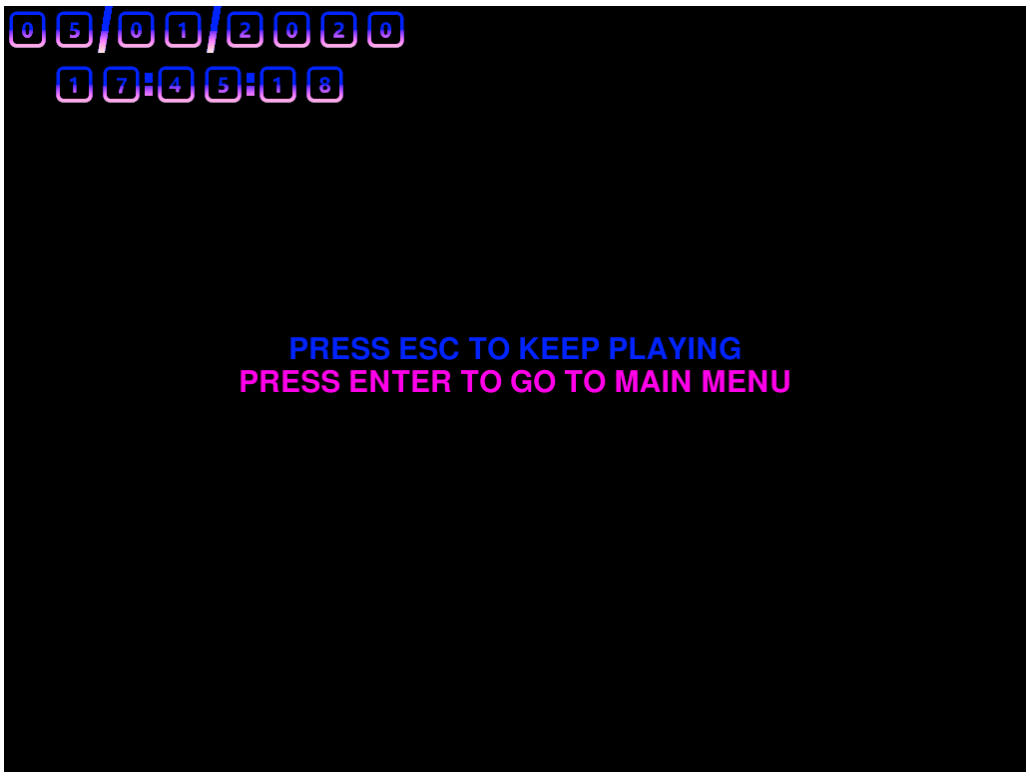




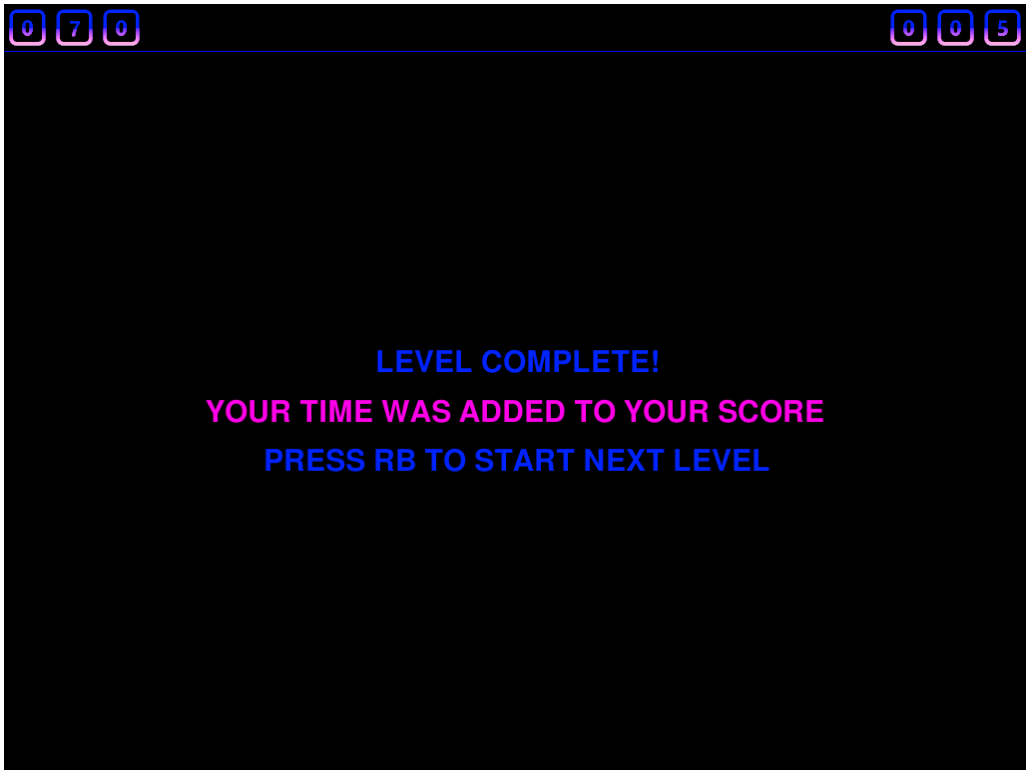
If you manage to catch the time bonus, you get 10 seconds added to your remaining time. However, if you are able to catch the bar bonus, your player will double it's size for 10 seconds (see image below).



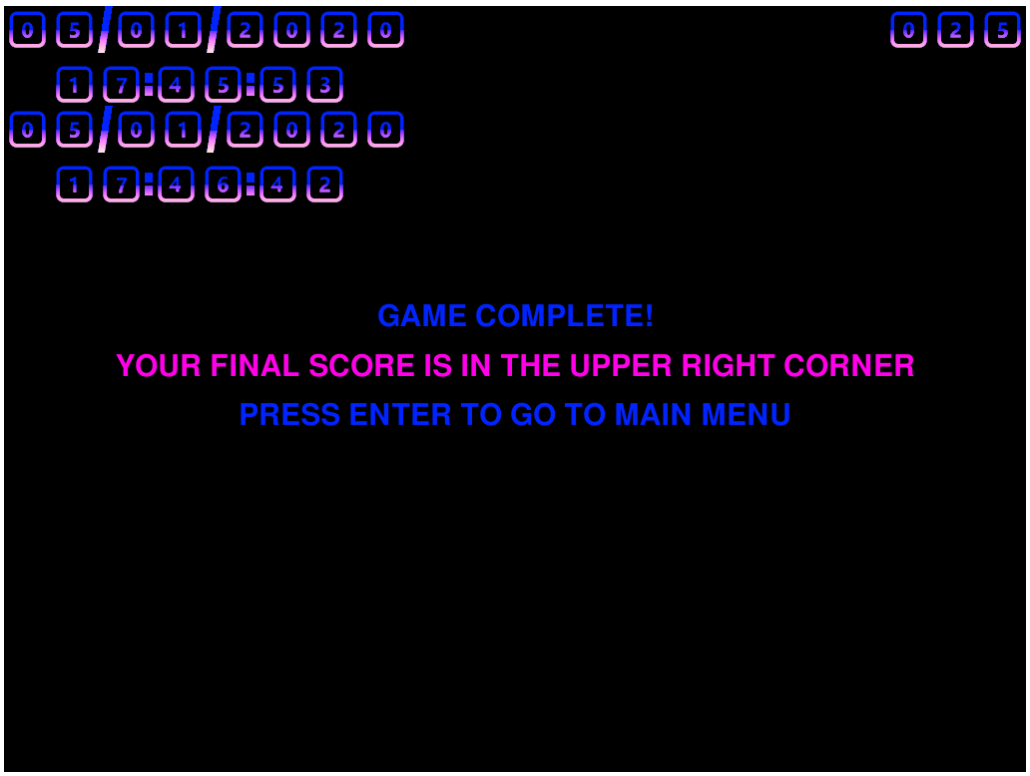
You can pause your game at any time simply by pressing the ESC key on your keyboard. This will take you to the pause screen (see image below). Here, you can see the real time on the upper left corner, and you're given a choice: keep playing or get back to the main menu.



When you destroy all the blocks, the next level screen appears (see image below). Your remaining time is added to your score, each second giving you 5 points. It tells you to press the right button of the mouse to begin the next level.



If you lose the game, either by running out of time or by letting the ball go past the player, you are automatically taken to the main menu. However, if you manage to beat the game (all 9 levels), then you reach the end game screen (see image below). It provides your beginning time and your finishing time, on the upper left corner, and your score, on the upper right corner. You can press ENTER to go to the main menu.



2. PROJECT STATUS

2.1. OVERALL USE

Device	Uses	Interrupts
Timer	Frame rate, level timer and bonus timer.	Yes
Keyboard	Navigating the menus and pausing the game.	Yes
Mouse	Controlling the player and advancing the levels.	Yes
Video Card	Display and collision detection.	Yes
RTC	Telling the real time while game paused and the begin and end times when game is finished.	No

2.2 TIMER

The timer is used to controll the frame rate (60fps) on the main menu (main_menu) and while the game is being played (play). It is also used to countdown the level timer (play) and the bar bonus timer (play), and to poll and update the RTC on the pause screen (pause_game).

2.3 KEYBOARD

The keyboard is used to navigate the main menu (main_menu) with the UP and DOWN arrow keys, while ENTER selects and option. This key also has to be pressed in order to go back to the main menu, when you are in the credits screen (credits). This device is also used to pause the game (play), with the ESC key. While in the pause screen, it can be used to get back to the game with ESC or go to the main menu with ENTER (pause).

2.4 MOUSE

The mouse is used to control the player, by pressing and holding the left button, while moving it sideways (play). It is also used to advance through the levels (next_level_card), by pressing the right button while on the instructions screen and the next level screen.

2.5 VIDEO CARD

The vídeo card is used to display all the menus (main_menu, next_level_card, credits, pause_game), and game elements (play, draw_player, draw_ball, draw_grid, draw_number).

This is accomplished using mode 118. Its resolution is 1024x768 and it uses a direct color model (8:8:8 = 512 colors). Despite the wide array of available colors, we only use two of them (black and blue), doing everything eles with xpms. We use double buffering by drawing everything on a back buffer, and then copying it to the front buffer (copybuffer). The only moving elements of the game are the ball and the player. All collisions are handled by checking their positions and the block’s positions as well (handle_ball_collision).

2.6 RTC

The RTC is used to tell the user what time it is when they pause the game (pause_game), and what time it was when they began it and ended it (play).

3. CODE STRUCTURE

3.1. PROJ.C – 3%

This file is where the program begins. It calls the function proj_main_loop, which subscribes the timer, keyboard and mouse devices. It also initializes graphic mode and allocates memory for the back buffer. Then, it calls main_menu until the user chooses to exit the program, when it reverts all subscriptions, changes the minix to text mode and frees the memory allocated. This is also where some global variables are declared.

3.2. GAME.H AND GAME.C - 82%

This is all the functions related to the actual game are declared and implemented. Menu, level, collision detection and element drawing functions are all here. It also contains all the global variables from proj.c, along with some global variables that are only needed on the functions here.

3.3 TIMER.H AND TIMER.C – 3%

All timer and RTC device related functions are in the timer.h and timer.c files. These include:

- Subscribing timer interrupts (timer_subscribe_int);
- Unsubscribing timer interrupts (timer_unsubscribe_int);
- Incrementing global variable time_elapsed (timer_int_handler).
- Reading the real time clock (rtc_time).

An important data structure in the timer.h file is struct Date: it stores information retrieved from RTC in unsigned variables.

3.4 KBD.H AND KBD.C – 3%

All keyboard device related functions are in the kbd.h and kbd.c files. These include:

- Subscribing keyboard interrupts (kbd_subscribe);
- Unsubscribing timer interrupts (unsubscribe_kbd);
- Reading the status register to check for errors and to read the scancode byte from the output buffer (kbc_ih);
- Reading the command byte of the Keyboard Controler (kbd_read_cmd);
- Writing a command to a KBC port (kbd_write_cmd).

3.5 MOUSE.H AND MOUSE.C – 3%

All mouse device related functions are in the mouse.h and mouse.c files. These include:

- Subscribing mouse interrupts (mouse_subscribe_int);
- Unsubscribing timer interrupts (mouse_unsubscribe_int);
- Reading the status register to check for errors and to read the scancode byte from the output buffer (mouse_ih);
- Parsing the mouse bytes to get information about the mouse (buttons, position and overflow) (parse_mouse_packet);
- Writing a command to the KBC (mouse_write_cmd);

- Using a mouse packet to detect which event occured (get_new_event).

An important data structure in the mouse.h file is the enum event: it is used to know what event ocurred upon a mouse interrupt chain (3 interrupts, to be exact).

3.6 VIDEO.H AND VIDEO.C – 3%

All vídeo card device related functions are in the video.h and video.c files. These include:

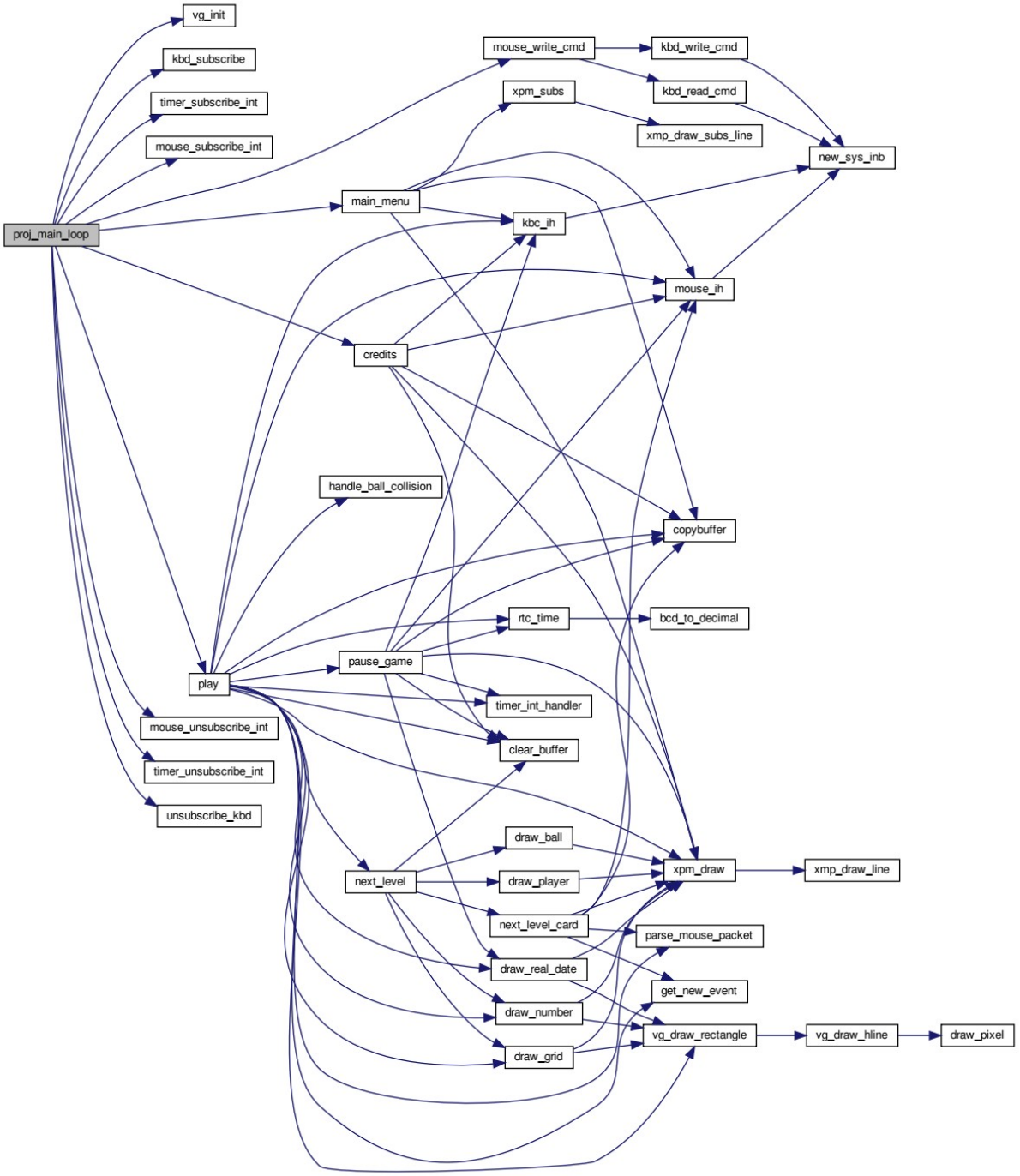
- Initializing vídeo mode (vg_init);
- Drawing with specific colors (draw_pixel, vg_draw_hline, vg_draw_rectangle)
- Drawing xpms (xpm_draw_line, xpm_draw, xpm_draw_subs_line, xpm_subs).

3.7 UTILS – 3%

This is where all the macros are stored and xpm files are included. The utils.h and utils.c also include libraries and a few functions used throughout the program:

- Getting LSB and MSB (util_get_LSB and util_get_MSB);
- Reading a byte from a port (new_sys_inb);
- Clearing and copying the back buffer (clear_buffer and copybuffer);
- Converting BCD into decimal (bcd_to_decimal);

3.8 FUNCTION CALL GRAPH



4. IMPLEMENTATION

4.1. LAYERING

In this project, a good example of layering is in the main_menu function. We have an arrow xpm drawn over a background xpm. This is achieved by having the xpm file of the arrow with a transparent background, and checking for that transparency when we draw it. If the pixel is transparent, we don’t draw it, so it maintains its previous color. Every time the user switches options, we redraw the arrow in the new place, while we also redraw the area where the arrow was with the corresponding pixels from the background xpm (xpm_subs function).

4.2. EVENT DRIVEN CODE

The event driven code in this project is present in the menus and next level screens, however, it's also this type of code that is used to control the player. While on main_menu, we use an integer to keep tabs on which option the user is choosing, which is done through the UP, DOWN and ENTER keys, we also use the ESC and ENTER keys to pause the game and get back to it, or to the main menu. The mouse's right button is also used to advance the player to the next level.

But perhaps the most important piece of event driven code in this project is get_new_event, which takes the parsed mouse packet and returns an event that tells the program what the user did with the mouse: pressed/released any relevant buttons, moved right or left, or nothing relevant happened.

4.3. BLOCKS

The way we designed our levels is quite simplistic: there is a global variable grid, a two-dimensional array of bools, 5 rows by 8 columns. Each of these bools corresponds to a block of 128x64 pixels, an if a given bool is true, then the block is "active", meaning it is to be drawn on screen. The grid occupies the entirety of the screen's width, and it begins on y = 112. Function draw_grid is called to draw the active blocks everytime it is necessary. We also keep tabs on the number of active blocks through the global variable blocks.

The first 4 levels are design by us, while the rest are random, with a 70% chance a block gets drawn at the beginning of the level (function next_level). The level is beat when blocks reaches 0.

4.4. BALL MOVEMENT AND COLLISION DETECTION

The "ball" is a 32x32 square. We keep the information about the ball with 4 global variables: ball_x, ball_y, ball_vx and ball_vy. The x and y correspond to the upper left corner of the square, and vx and vy correspond to how much these increment per frame. Every level, the ball starts on top of the player with x = 512 and y = 624. The way we initialize their speeds depends on the level (above level 5, the speed is doubled), but they always start at a 45º angle, going up and right (draw_ball).

Every frame, after we increment the ball position by its speed, we call handle_ball_collision, which checks, in order:

1. if the ball is hitting a wall
2. if the ball is hitting the ceiling
3. if the ball is hitting the player
4. if the ball is past the player
5. for every active block, if the ball is hitting it:

1. vertically
2. horizontally
3. any corner of the ball with a corner of the block

For wall and ceiling collisions, the correspondin speed's sign is switched. If the ball hits the player, it checks where exactly did it hit:

- exact middle: the ball goes straight up;
- neutral region: if it was between the 40px behind the middle and the 40px beyond, the middle, it merely switches the sign on ball_vy;
- moderate left region: 68px behind the neutral region, where the ball gains a 45º angle to the left;
- moderate right region: 68px beyond the neutral region, where the ball gains a 45º angel to the right;
- far left region: between the beginning of the player and the moderate left region, the ball gains a speed that sends it faster to the left than upwards;
- far right region: between the moderate right region and the end of the player, the ball gains a speed that sends it faster to the right than upwards;

If the ball is past the player, the function returns 2 immediately, telling the play function that the game is lost. However, if it hits a block, it switches the adequate speed's sign, turns that block inactive, assigns values to the bonus position and decrements the blocks global variable. The function then returns 1, telling the play function that at least one block was destroyed, so it has to redraw the grid and check for bonuses. It also has to check if the level was beat.

All other collisions or no collisions return 0.

4.5. PLAYER MOVEMENT

As described above (see section 4.2), the player movement is mouse event driven. However, it only moves when the user is both pressing the mouse's left button and moving it left or right. It won't move if the player is against a wall.

4.6 BONUSES

Everytime the function handle_ball_collision returns 1, the play function checks for bonuses. It generates a random number between 1 and 20, and turns the bonus_drop variable true if it is lower than 4 (15% chance). The bonus will alternate between time and bar, although it won't appear if another bonus is on screen, if the bar bonus is active or if there are no more blocks.

If bonus_drop is true, it switches the value of bonus_type (to make it change between bonuses) and draws the corresponding bonus on the screen, dropping it 4px every frame. If its position is above the end of the block grid, it draws the grid before redrawing the bonus xpm, so as to not have black bars above blocks when the bonus goes through them. If it reaches the player's y, it checks if there is collision and activates the bonus accordingly.

However, if the bonus chance is not met, it resets its position, so as to permit its assignment on handle_ball_collision. If this was not done, everytime the ball hits a block, the bonus position changes to where the block was destroyed.

4.7 RTC

When on the pause menu, the user can observe the time and date at which they are playing. This is done through the rtc_time and draw_real_date functions. The first reads all date and time components and stores it in a struct Date, while the second takes that struct, and draws each number at the given position.

Another use of the RTC is upon starting the game, the date and time is stored, so as to later be displayed when the player beats the game, along with its end date and time.