## Faculdade de Engenharia da Universidade do Porto


# LCOM Project-Tetirs

T06G05


| António O. Santos | Pedro A. F. Silva | Pedro M. M. Nunes | José L. N. Osório |
|---|---|---|---|
| up202008004 | up202004985 | up202004714 | up202004653 |

# Contents

# 1. About

- **Game Concept**

Our project target was to recreate the famous videogame Tetris, more precisely the 1989 version developed by Nintendo for the NES, in the C programming language, using Minix's system calls. Despite its' simplicity, some functionalities required careful analysis of the code and thought about edge cases to be done properly, avoiding bugs, and most importantly to deliver a finished product.

- **User Instructions**

The project doesn't deviate from much the usual Tetris control layout, making it so players will have no trouble adapting to this version.

In the main menu there are 4 buttons: single player, 2-player versus, leaderboard and exit, all of them being self-explanatory. The "Single Player" button allows the user to play a traditional score based version of Tetris, the "Leaderboard" displays recorded top scores and "Exit" gracefully terminates the program. The "Multiplayer" mode was not implemented, which will be discussed later on the report.


*Figure 1 – Main Menu*

While playing the game, the goal is to drop tetrominos (the name of the blocks) and filling up lines in order to clear those lines and freeing space for more tetrominos to fall and keep the game going. However, as the tetrominos start piling up and eventually reach the top, the game ends. The score is measured by the number of lines able to cleared.

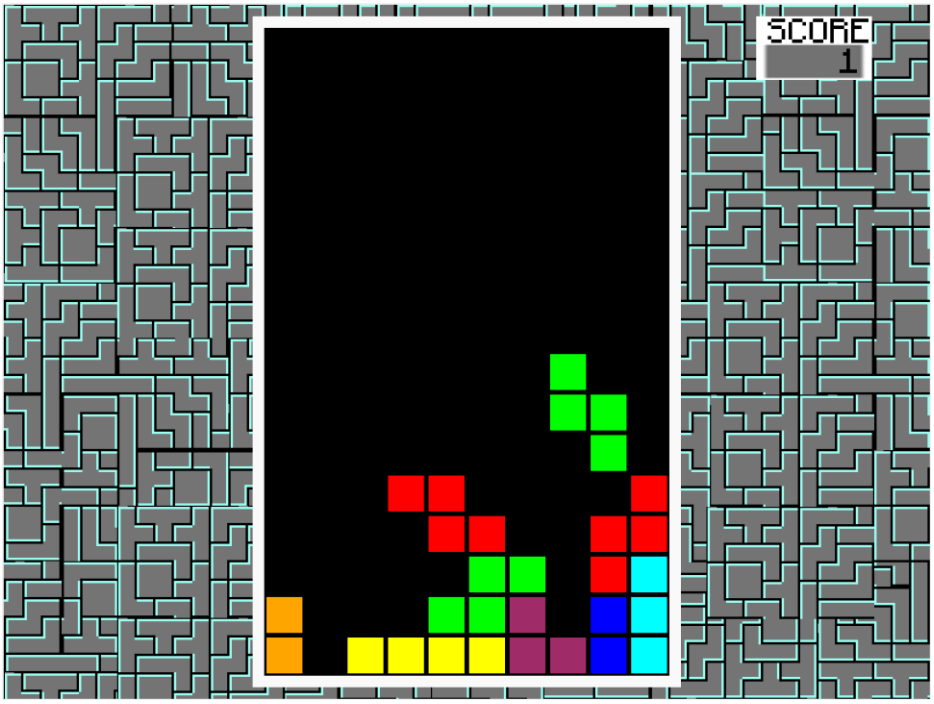


*Figure 2 – Single Player gameplay*

When the game ends, the player is prompted to write their name in order to save the score and add it to the leaderboard.


*Figure 3 – Game Over Screen*


*Figure 4 – Leaderboard*

Unfortunately, the 2-player versus mode was not implemented, however it would function just like the single player. 2 players, in separate computers, clear lines and send lines with a missing space to their opponent, in order to make increase the difficulty of clearing the board and forcing a loss on the opponent.

To control the game, the following tables illustrates the available options:

| Keyboard | |
|---|---|
| **Left Arrow** | Move piece left |
| **Right Arrow** | Move piece right |
| **Down Arrow** | Move piece down |
| **Z** | Rotate left |
| **X** | Rotate right |

| Mouse | |
|---|---|
| **Move Mouse Left** | Move piece left |
| **Move Mouse Right** | Move piece right |
| **Right Click** | Move piece down |
| **Left Click** | Rotate left |

The main menu is controlled with the mouse and left mouse button.

# 2. Project Status

| Device | What for | Interrupt/polling |
|---|---|---|
| **Timer** | - Frame rate and tetromino falling speed; | I |
| **KBD** | - Controlling tetromino placement: This includes moving horizontally, rotating and making them fall faster.<br>- Writing name to the leaderboard | I |
| **Mouse** | - Selecting an options in the menu;<br>- Controlling tetromino placement: This includes moving horizontally, rotating and making them fall fast; | I |
| **Video Card** | - Displaying the menus and game;<br>- Double-buffering implemented; | N/A |
| **RTC** | - Getting date to register in the leaderboard; | I |

As it was previously stated, the 2-player mode was not implemented in time for the final iteration of the project, and we wouldn't want to rush the development and implementation of the serial port since it would feel unfinished and cause a lot of bugs.

- **Graphics card**

Video mode 0x115 was used, which features an 800x600 resolution and a direct color mode with 24 different colors. (**vg_init()**).

Double buffering was also implemented as we were seeing flickering issues when rendering the main menu alongside the text options and the cursor. This implementation also saw a big performance boost when drawing falling pieces which could previously be flickery.

We use a monospaced font rendered from an *.xpm* file, with two versions, one in white and one in black in other to make text written to any sort of background readable.

- **Keyboard**

The keyboard was used to control the tetrominos during the game (**handle_kbd_playing_event()**), as an alternative to the mouse, through the use of interrupts. It is also used to write the player name to the leaderboard upon completing a game. In-game, the left/right arrow keys move the tetromino left/right, the down arrow key makes it fall faster, Z rotates it left and X rotates it right, all of these inputs also make sure that the movement is possible.

In both game over and leaderboard screens the keyboard is used to go to the main menu screen, with the ESC key (**handle_kbd_finished_event()** and **handle_kbd_leaderboard_event()**).

- **Mouse**

The mouse was used through interrupts in order to detect its positioning, used to control the tetrominos left and right (**handle_mouse_playing_event()**) and selecting the options in the menu (**handle_mouse_menu_event()**), while also using the left mouse button to make a tetromino fall faster/selecting an option in the menu and the right mouse button to rotate a piece to the right.

- **RTC**

RTC was a simple implementation as it was simply used to read and register the time when a game was finished in the leaderboard (**rtc_read_time()**).

- **Game Logic & Functionalities**

We were able to explore ways to refine and make sure collision detection was being correctly done and as bug proof as possible (**check_collision()**). The issues that emerged from this endeavor in conjunction with the rotation of the pieces (**piece_rotate()**) made for a complex challenge.

# 3. Code organization/structure

- **proj.c**

Calls to the initialization, main loop, cleanup functions and some lcf functions to make the project run.

**Weight:** 1%
**Contributors:** António Santos, José Osório, Pedro Silva, Pedro Nunes

- **utils.c**

Some useful function not directly related to the game in specific. Mostly calculations done with matrixes such as transposition, column/row reversion, array shuffling, getting strings for keys in the keyboard, checking if a character is capitalized and a sleep function in milliseconds.

**Weight:** 4%
**Contributors:** Pedro Nunes

- **device_utils.c**

Harbors the very important **util_sys_inb()** function.

**Weight:** 0.5%
**Contributors:** José Osório

- **video_card.c**

Contains most of the functions that manipulates the graphics card “directly”. Initiating the video card in the proper mode, mapping the video memory, changing the video mode, swapping buffers (double-buffering), drawing a rectangle (which calls a function that draws a line that in turn calls the draw_pixel function), drawing an xpm (used in the font, for example), drawing a font character and drawing a block (part of a tetromino).

**Weight:** 10%
**Contributors:** António Santos

- **keyboard.c**

Home of the keyboard interrupt handler and other keyboard functions, such as subscribing/unsubscribing interrupts and reenabling them, getting its status, reading the output buffer, writing the command byte and reading its return value.

**Weight:** 10%
**Contributors:** José Osório

- **mouse.c**

Similar to the keyboard module but adapted to the mouse. Also needs to enable/disable data reporting.

**Weight:** 10%
**Contributors:** José Osório

- **timer.c**

The timer module gets, sets and displays the timer configuration, sets its frequency, subscribes/unsubscribes to interrupts and increments a global counter as an interrupt handler.

**Weight:** 5%
**Contributors:** António Santos, José Osório, Pedro Silva, Pedro Nunes

- **rtc.c**

In the rtc module we read/write to the data register, write to the address register and read the time, from the year to the second.

**Weight:** 1%
**Contributors:** José Osório

- **kbd_event_handler.c/mouse_event_handler.c/timer_event_handler.c**

These 3 modules serve as middlemen between the device interrupts and the application actions, that are dependent on the game state. For example, every timer interrupt we draw the cursor if we're not in the "playing" state, draw the respective menus or board, etc.; translate every keyboard and mouse interrupt into movement/piece rotation/menu selection depending on which key is pressed or the positioning of the mouse, and so on.

**Weight:** 10%
**Contributors:** José Osório

- **int_manager.c**

This module is the glue that ties everything together, all of the functions (excluding the LCF ones) that are called in proj.c are implemented here. It subscribes every interrupt, enables data reporting on the mouse, loads xpm files and initializes the array of keys on startup; starts the interrupt loop and on exit returns Minix to its default state (reenables text mode, unsubscribes interrupts and data reporting on the mouse).

**Weight:** 1%
**Contributors:** António Santos, José Osório, Pedro Silva, Pedro Nunes

- **tetromino.c**

These modules takes care of actions aimed specifically at the tetrominos, such as creating/deleting them, managing rotations and keeping them inside of the border.

**Weight:** 5%
**Contributors:** António Santos, Pedro Nunes

- **game_state.c**

This big module controls and calls functions that manipulate everything related (mostly) to the playing state and some on the leaderboard/main menu state, such as moving the tetrominos on the board through keyboard and mouse, checking for collisions, drawing/clearing tetrominos, drawing ui (board, scores, background, etc.), rotating tetrominos (the function that handles this specifically is in tetromino.c), drawing/clearing the mouse, checking for mouse clicks in certain places, etc.

**Weight:** 30%
**Contributors:** António Santos, Pedro Silva, Pedro Nunes

- **draw_graphics.c**

Like the event handlers, this modules servers as the middleman between the functions that get called to draw specific stuff on screen and the graphics card functions that write to it, and also loads the tetromino images/xpm files. Everything from the board to the leaderboard, the characters, the background, (…) gets dispatched to a proper graphics card function in video_card.c.

**Weight:** 10%
**Contributors:** António Santos

- **leaderboard.c**

This module deals mostly with storing and updating the leaderboard correctly. Leaderboard info is read from a text file to memory, updated at game over, and stored back in the text file when the program closes.

**Weight:** 2.5%
**Contributors:** José Osório

# 4. Implementation Details

The program is based around user events/interrupts like key presses, mouse presses and movement or timer interrupts, each handled by the respective handler and triggering a different response, therefore making it event driven.

The notion of states (**enum game_state**) is also key to the program, as most functions handling events have a different behavior depending on the current state of the program. This way the program functions as a state machine.

Collision checking has been mentioned throughout the report, as it was essencial to the game, encompassing a significant part of the game logic behind Tetris. Possible collisions between the tetromino been moved and the tetrominos already placed (which are part of the **board**) as well as the other board delimiters are avoided by checking any overlap between the two entities after moving the tetromino, as well as ensuring the coordinates of every part of the tetromino stay inside the designated board area.
Additionally, an extra check is done when rotating a tetromino motivated by the fact that while the overall coordinates don't change, the new placement could leave part of the piece out of the borders. To solve this, we check if this would be the case and if so, immediately change the x value of this tetromino to correct it (to either left or right, as needed). This is done by a specific check function (**check_rotate_inside_border**()) inside the rotate functions (**rotatePieceLeft** and **rotatePieceRight**).

In terms of rotating tetrominos we also have an interesting implementation to mention. The piece itself is stored as a single 3x3 or 4x4 matrix (implemented as **uint8_t****) instead of storing each of the 4 possible placements for every piece. For a rotation to be done we perform operations over the matrix, namely transposing and reversing its columns/rows.

# 5. Conclusion

Throughout the project we had issues with the collisions, as sometimes they were inconsistent or simply didn't work as expected. A lot of work was done to fix them and make sure they behaved as planned. Moreover, we initially didn't expect we would have to implement double buffering, but the mouse being used in the menu introduced a high amount of flickering that we had to correct.

If time was not a restriction (and after implementing everything we planned initially), maybe we would have considered adding another mode besides single and two player mode. Multiplayer Tetris exists, even up to 99 players, and perhaps we could have replicated that experience. Another option would be to modify the classic Tetris flow: add gravity to individual blocks, power-ups like clearing tetrominos in an area (like a bomb), controlling the tetromino queue, modifying the board size... The possibilities are (or would've been) endless.

However, the project still feels like a success. The freedom of choice we were given to pick anything we wanted to recreate (even something other than a game) was crucial to motivate us, and we're sure the majority of the students feel this way.

The fidelity of our project to the real, classic game (other than the slight difference in the name) is something we are proud of: Even the menu and the surrounding background around the game board is straight out of the NES game. The behavior of the game is practically identical as well.

Another thing we are proud of is how entertaining it is to play our own game: usually the case with projects is that we develop them, get the grade and then we never touch them again. However, since our project is a game whose foundation was well written and was inspired in a famous game, we can see ourselves and our friends playing it purely for fun, which is a very exciting prospect and it makes us happy that we could create something like this.

In conclusion, this project was not without its problems and challenges, but overall we feel like it was an amazing way of consolidating what we learned. We enjoyed working on it and are satisfied with what was accomplished.