U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

LCOM Checkers

Degree in Informatics and Computing Engineering

Computer Laboratory

Class 2 – Group 6

28th May, 2023

Bruno Miguel Lopes da Rocha Fernandes up202108871@up.pt
Marcos António Sousa Costa up202108869@up.pt
Vasco Moutinho de Oliveira up202108881@up.pt

Index

User Instructions

Our project is a *Player vs Player* implementation of the classic *Checkers* board game with a different set of rules that we think make the game more enjoyable for the players.

Rules

The famous checkers game is a 2 player board game that is played on a 8x8 board with 24 pieces that consists of each player moving one piece at a time to try and capture the opposite player's pieces. The pieces can only move diagonally, forward and one square at a time, but there is an exception. When a piece reaches the opposite side of the board, it turns into a "King". The King works the same way as the other pieces but it can move backwards and multiple squares at a time.

To capture a piece, you have to jump over it, technically moving two squares. It is also possible that a player captures multiple pieces in one move, given it is possible after capturing the first one. If there is the possibility of a move that captures pieces, the player is forced to make the one that captures the most.

The winner is the first one to capture all the opponent's pieces.

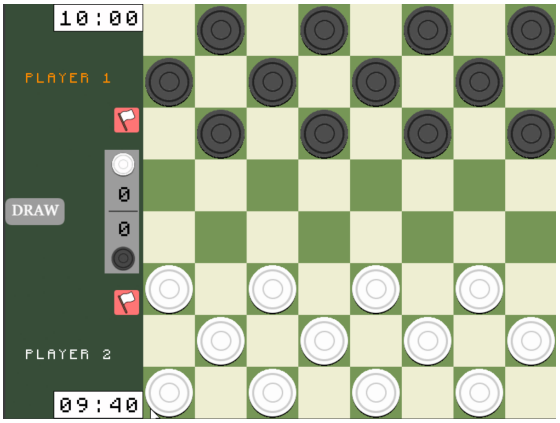
Start Instructions

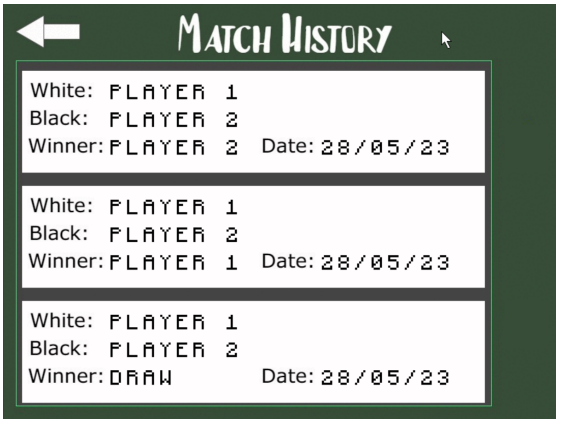
To start the game enter your Minix virtual machine. You should be in the directory proj/src. Inside that directory, run the following command: *make && lcom_run proj*.

Now the game should be loaded. Hit "Play" to start a new game, "Records" to see the game history or "Exit" to leave the game.

When you hit the "Play" button, you have the possibility to choose the name of the players before starting the match.

When in game, grab a piece using the left button of your mouse and move it to where you want to move the piece to. When you select a piece some squares will be highlighted to show you the possible moves. There is the possibility to forfeit clicking the white flag button on your side of the board, or make it a draw by clicking the "Draw" button.





Project Status

Device table

Device	Usage	Interrupts
Timer	Updates display and game logic at a constant framerate	Yes
Graphics Card	Image/Sprite/Font drawing	Yes
Keyboard	Name Choosing and Page Scrolling	Yes
Mouse	Piece movement and Menu selections	Yes
RTC	Saving record history	Yes

Timer

Used to update the screen and game logic at a consistent pace and as a timer to limit the amount of time players have to make their moves. Function timer_event_handler() is called every 2 interrupts from the timer.

Major functions:

- timer_event_handler() – updates display and the remaining time for the player whose turn is to play. If said player has no time left, then the match ends and the other player wins.
- update_sprites() - updates the positions of sprites in a game, taking into account their velocities and destination coordinates.

Minor functions:

- timer.c file - all other functions. A few of them were harnessed from lab2.

Graphics Card

Used to draw pixels that can be part of sprites, fonts, images, etc. We decided to use the video mode 0x115 with a resolution of 800x600 and a 24 bit direct color model. The configuration was set up to operate using a linear framebuffer.

Double buffering via copy was used.

Structs are utilized to manage the current animation status of an object, enabling the implementation of straightforward animations. The animation itself is achieved either by utilizing distinct frames or by utilizing sprite sheets.

We detect the collision of objects by checking their positions.

Fonts are used by employing an .xpm file that contains the font design.

Major functions:

- vg_init() - uses VBE Function 01h to set up video mode.
- video_map_phys_mem() - maps the physical memory to the virtual memory.

Minor functions:

- video.c file - all remaining functions that draw the components of the game.

Keyboard

Used to read players' names at the beginning of each match or to scroll through the match history.

Major functions:

- kbd_event_handler() - handles keyboard events, including the handling of specific keys and the state of the program. It performs actions such as changing the state, selecting names, and handling history-related events.

Minor functions:

- keyboard.c – all remaining functions, most of them harnessed from lab3

Mouse

Used by players to move the cursor displayed on the screen, the left button is used to select options on the menus, grab and move pieces and, also, to select fields to write the name of the players.

Major functions:

- mouse_event_handler() - processes mouse events, tracks the state of mouse buttons, updates the cursor position, and handles the movement of game pieces or objects based on the mouse input.
- attempt_move() - verifies the validity of a move, updates the board and game state accordingly, captures pieces, handles promotion of pieces, switches turns, calculates new moves, updates the draw board, checks for game over conditions, and returns a boolean value indicating the success of the move.

Minor functions:

- mouse.c and mouse_event_handler.c files – all remaining functions, some harnessed from lab4, other used to handle left button being pressed on the different screens.

Real Time Clock (RTC)

Used to capture and save the date of a match to save the match on the history later on. Once a user completes a match, the system stores a record of the match, including information such as the winning player, the names of the participants, and the date of the match. The date is obtained from the real time clock (RTC) module.

Major functions:

- rtc_read_time() - reads the current time from the RTC and returns a struct of type Time. This struct contains the day, year, month, hours, minutes, and seconds of the current time.

Minor functions:

- rtc_write_data() and rtc_read_data() - other functions.

Code Structure

Modules

Keyboard

Mainly identical to lab3, reads scancodes from the KBC, writes commands to KBC and utility functions that ensure the output buffer is cleared to prevent the operating system (MINIX) from becoming unresponsive or hanging.

Mouse

Mainly identical to lab4. Handles mouse events, updates the cursor position, and performs specific actions based on the mouse button presses and cursor position. It also includes lower-level functions for interacting with the mouse device at a hardware level.

RTC

Provides functions to read and write data from/to the RTC and retrieves the current time from the RTC registers.

Structs:

Time – stores the day, month, year, hour, minute and second read from the RTC and is returned in rtc_read_time()

Timer

Mainly identical to lab2.

VBE

Mainly identical to lab5.

Graphics Card

Mainly identical to lab5, but modified to suit project structures, implemented dedicated drawing functions for structs that hold game world information.

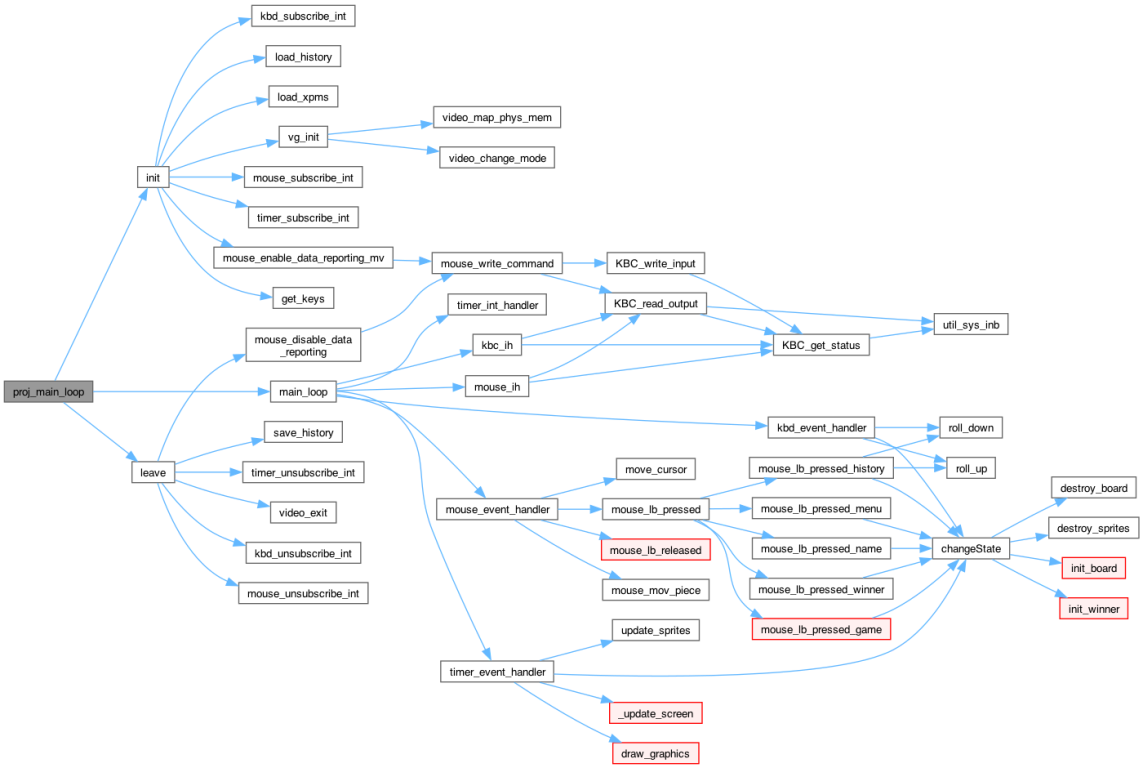
Game Logic

The game logic is structured in the files in the game directory. Consists of various variables and functions that control the flow and behavior of the game. Handles the logic of the checkers game like the possible moves, who's the next player to make a move, if there is a possibility to capture a piece, if the game has ended, if a piece is a King or not, etc.

Structs:

- Animation: manages an animation, including the frames, their durations, and its position.
- Piece: represents a game piece on the board.
- Move: represents a move made on the game board.
- Trajectory: represents the trajectory of a piece on the game board.
- PieceSprite: represents a sprite of a game piece for animation purposes.
- MatchRecord: saves information about a match.

Function Call Graph



Implementation Details

In the development of our checkers game, several implementation details required careful consideration and ingenuity to ensure smooth device usage and an engaging user experience. This section delves into the key topics that were addressed during the implementation phase, highlighting the decisions made regarding layering, event-driven code, object orientation, and frame generation.

Layering

To create a visually appealing and interactive checkers game, a layered approach was employed. The game board, checker pieces, and additional graphical elements were organized into separate layers. By employing layers, we achieved a modular and flexible design that allows for easy manipulation and updating of specific elements without affecting others. This approach facilitates efficient rendering and optimizes resource usage on the graphics card, enabling smooth and responsive gameplay.

Event-Driven Code

Implementing an event-driven architecture was crucial for capturing user input and enabling interactive gameplay. By leveraging event-driven code, we established a system that responds to user actions in a timely manner, enhancing the overall user experience. Events such as mouse clicks, keyboard inputs, and timer-triggered updates were seamlessly integrated into the game's logic. This design choice allowed for real-time interaction, enabling players to make moves, select pieces, and navigate the user interface with ease and responsiveness.

Object Orientation

Object-oriented programming played a pivotal role in the implementation of our checkers game. By modeling the game entities, such as the game board and checker pieces as objects, we achieved encapsulation, modularity, and code reusability. This approach facilitated the organization and management of game data, logic, and behavior.

Frame Generation

Efficient frame generation was a key consideration to ensure smooth animation and visual updates. By utilizing double buffering, we employed a technique that reduces visual artifacts like tearing and flickering. This method involves generating the game frames on a back buffer while the front buffer is displayed. Upon completion of the frame generation, the buffers are swapped, ensuring that complete frames are presented to the user.

Timer, Keyboard, Mouse, and RTC Integration

The integration of various input devices was crucial to enhance user interaction. Timers were utilized to regulate game logic and provide timely updates, ensuring that game actions occurred at the appropriate moments. Keyboard support enabled players to navigate menus and provide input. Mouse integration facilitated piece selection, movement, and interaction with the user interface elements. Additionally, Real-Time Clock (RTC) integration allowed for the implementation of time-based features such as match history timestamps.

In conclusion, the implementation of our checkers game involved thoughtful consideration and ingenuity in various aspects. Layering, event-driven code, object orientation, and frame generation were vital elements that contributed to a visually appealing, interactive, and responsive gaming experience. The integration of timers, keyboards, mice, and RTC further enriched user interaction, making the game engaging and enjoyable. These implementation details have resulted in a high-quality checkers game that leverages device capabilities to their fullest extent.

Conclusion

During the development of this game, we had some difficulties implementing the game logic because of the amount of possibilities on every play. We also had some problems with the fact that the path to the history.csv file could be different in different machines.

On a future update, we would like to implement some connection features to play online and create accounts. It would be also good to implement the possibility to play single-player, playing against an AI. Both these ideas would give the user a way to play the game when they are by themselves.

We think our main achievement is the fact that we were able to implement all the projected features and modules except for the serial port.

During the making of this project we learned a lot about what is “behind” the input/output devices we use on a day-to-day basis.