Whack'a Diglett - A Minix-based Game

LCOM Project Report

Group T02G04:
Francisca Portugal (up202303640@up.pt)
Gabriela de Mattos (up202304064@up.pt)
Maria Luiza Vieira (up202304306@up.pt)
Sara García (up202306877@up.pt)

Laboratório de Computadores (L.EIC018) 2024/2025

Index

1. What was our goal? What is our application?

Our goal with this project was to design and implement a simple yet engaging graphical application using low-level programming concepts, specifically targeting the Minix operating system and leveraging direct hardware interaction. We aimed to deepen our understanding of computer architecture, graphics programming, and device control by building an interactive game from scratch.

The application we developed is a 2D game (inspired by Whack'A Mole) that showcases real-time graphics, user input handling (keyboard and mouse), and smooth animations through techniques like triple buffering. It features multiple game states, such as menus, instructions, gameplay, pause, and game over screens. The game logic, rendering routines, and input controllers were all implemented at a low level, giving us hands-on experience with topics like video memory management, interrupt handling, and drawing algorithms.

Overall, our application serves as both a fun game and a demonstration of our ability to work close to the hardware, applying theoretical knowledge from our coursework to a practical, interactive project.

2. How did we structure the project?

2.1. Core Architecture

The project follows a layered architecture with clear separation of concerns:

- **Main Program (main.c)**: Entry point that initializes the video mode and launches the game loop.
- **Game Core (game.c/h)**:
    - Controls the main loop and state machine
    - Manages transitions between different game modes
    - Handles interrupt processing from hardware devices
- **Hardware Controllers (/controllers)**:
    - **Video**: Manages display, buffers, and drawing operations
    - **KBD/Mouse**: Processes user input
    - **Timer:** Controls game timing and animation
- **Game Modes (/game/modes):** Each mode (Menu, Playing, Paused, Instructions, GameOver) encapsulates its own:
    - State management
    - Input handling

    - Rendering logic
- **Visual Components:**
    - Sprites (static and animated)
    - Background rendering
    - UI elements (cursor, text)

2.2. Key Dependencies

- **Main Program** initializes video and launches the game core
- **Game Core** manages modes, coordinates hardware access, and controls rendering
- **Game Modes** depend on the core, controllers, and visual assets
- **Visual Components** utilize the video controller for rendering
- **Hardware Controllers** provide essential services to all other components

2.3. Data Flow

- **Input:** Hardware interrupts → Controllers → Game Core → Mode handlers
- **Rendering:** Mode rendering → Video controller → Buffer management → Display
- **State:** Game Core manages global state and mode transitions

2.4. Buffer Management

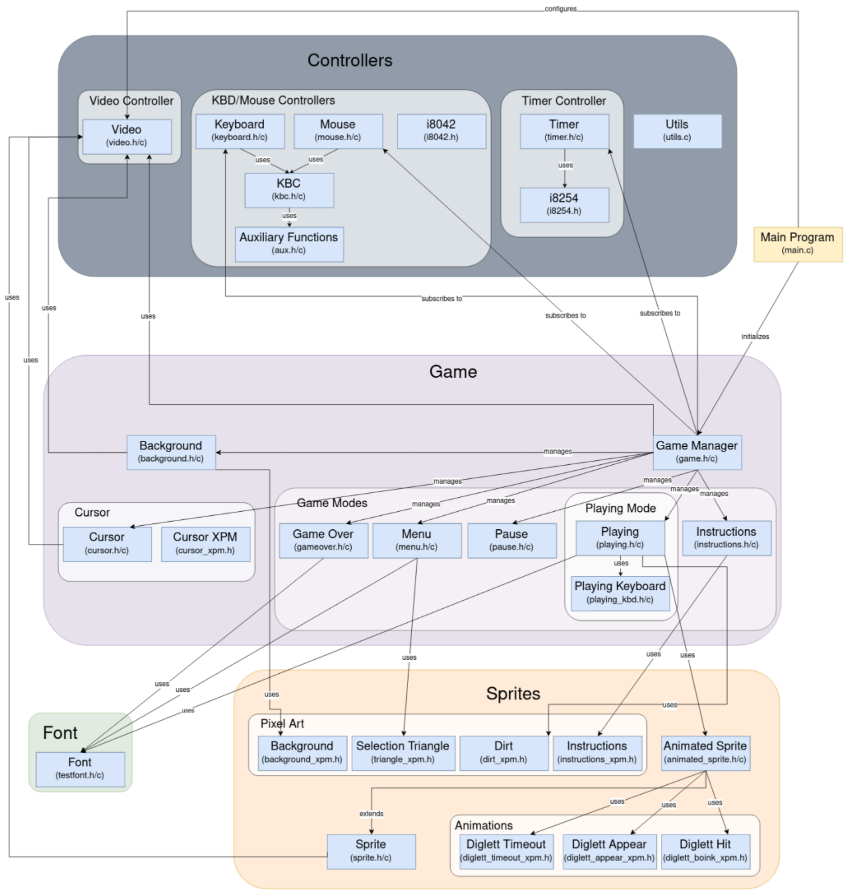
The game uses a multi-buffer system:
- **Static buffer**: For unchanging elements
- **Back buffer**: For dynamic content
- **Buffer swapping**: For smooth rendering

2.5. File Organization

The codebase is organized into logical modules:
- **controllers**: Hardware abstraction layer
- **game**: Game logic and modes
- **sprites**: Visual asset management
- **modes**: Different game states
- **fonts**: Text rendering

This modular architecture ensures clean separation between hardware interaction, game logic, and visual representation, making the code maintainable and well-structured.

2.6. Architecture Image



3. What devices did we use and for what purpose?

Timer:

The timer device, configured to generate interrupts at a frequency of 60Hz, was essential for managing time-dependent operations in our game.

- **Game Loop Timing** - The timer is used to control the main game loop, ensuring that updates to the game state (such as updating scores and handling animations) occur at regular intervals.

- **Diglett Visibility and Animation** - Each diglett in the game has associated timers that determine how long it remains visible or hidden, as well as the duration of animations (appear, whack, etc). The timer interrupts are used to decrement these counters, triggering state changes when the counters reach zero.

- **Game duration** - To manage the duration of a game (60 seconds), we use the system clock (with *clock_gettime*). This allows us to accurately track the elapsed real-world time, independent of the timer interrupt frequency, which was a problem initially. By recording the start time and checking the current time, we calculate the remaining time for important features like the countdown and game over state.

Keyboard:

The keyboard is used as the main input device for player actions in the game.

- **Player Input** - The keyboard is used for player input, allowing users to "whack" digletts by pressing keys mapped to each diglett position.

- **Game Control** - The keyboard also manages game functions such as starting, pausing, and navigating menus, making it essential for both gameplay and overall control.

Mouse:

The mouse is used as an alternative input device for navigation in the game.

- **Game Control** – Just like the keyboard, the mouse can be used for navigating menus and interacting with UI elements, such as starting the game or selecting options. The cursor is updated in real-time, and clicks are handled to trigger menu actions.

Video/Graphics:

The video device is used to render and display all visual elements of the game.

- **Screen Drawing** - The video card was used to draw all game graphics (backgrounds, sprites, animations, text, and UI elements). Double buffering was implemented to avoid flickering and tearing, by drawing everything to a back buffer and then swapping it to the screen for smooth updates.

- **Dynamic UI Updates** - The video card allowed us to update the score, timer bar, and other on-screen information in real time.

- **Game Mode Transitions** - Switching between game modes (menu, playing, pause, instructions, game over) was managed by clearing and redrawing buffers, so each mode showed the correct graphics. Using the video card, we achieved smooth graphics and transitions.

4. What are the differentiating features of our project?

Diglett System:
Each diglett has individual timers with randomized show and hide durations, along with a multi-state approach that allows realistic diglett behaviour.

- **Individual Timer Management** - Digletts operate on independent timer scheduling for unique show and hide cycles, randomized duration algorithms to prevent predictable patterns, and interrupt coordination to manage multiple concurrent timers without conflicts.
- **Multi-State Behaviour Implementation** - The system implements a five-state lifecycle (hidden->visible->boinking->timing_out->hidden), ensuring smooth transitions and concurrent processing of multiple digletts of different states.

Real-time Progress:
A dynamic time bar tracks remaining game time, updating every frame using timer interrupts for instant visual feedback. A real-time points counter displays the player's current score throughout gameplay.

- **Time tracking** - The tracking of remaining game time is done by clock_gettime that ensures accurate real-time independent of interrupt frequency, enabling reliable countdowns and game-over triggerings.
- **Score Management** - The points counter updates immediately upon diglett hits and misses, providing instant feedback to players. Score tracking persists throughout the game session and is displayed prominently during gameplay and on the game-over screen.
- **Dynamic Display** - The dynamic time bar updates every frame via interrupts and featuring color transitions, while the points counter refreshes with each scoring event. Optimized interrupt handling maintains smooth gameplay without lag for both timing and scoring elements.