Arcade Volleyball Report

BARBA CARVAJAL CRISTIAN GABRIEL

Major Modifications

Volleyball arcade is a game created in assembly language in 1987. This time, the game was inspired by the code of the classic game Pong, which has very simple mechanics such as linear movements and minimal physics. Improvements were implemented for the volleyball game, such as advanced physics, a central net, and an interactive menu. The result is a more elaborate game written entirely in assembly language.

Visual and Structural Changes

Compared to classic Pong, Volleyball Arcade incorporates improved sprites for both the ball and the players, replacing the simple rectangles originally used. Using the INT 10h/0Ch interrupt, each pixel was drawn individually, allowing detailed colors and shapes to be displayed on the screen. This project takes better advantage of mode 13h, offering a color screen with a more expressive look.

Gameplay Mechanics

Several procedures were isolated from the main loop, such as physics handling, collision checking, sprite rendering, and user input. This modular structure not only improves readability, but also makes debugging and performance tuning considerably easier. The project also incorporates optimized routines for drawing and erasing sprites in order to minimize flickering, an important issue when working in assembly language with limited CPU cycles.

Physics and Collision Engine

The game's physics design has also been improved. Instead of simple rectangular comparisons, the center of the ball and the paddles are now calculated to determine the force and direction of the impact, resulting in more natural rebounds. The ball retains its previous speed and its movement depends on the collision history, not just on reversing the axis sign. Added to this is a touch counting system inspired by the real rules of volleyball, where each team has a maximum of four contacts before committing a fault. It also detects when the ball crosses the net, which automatically resets the counters, and different reappearance positions were implemented depending on the side where the ball lands.

Game Systems

A two-digit score was implemented for each player, with colors clearly differentiating each side. At the end of the game, the program displays a winner screen, options to restart, and the possibility to return to the main menu. In addition, the game now includes an interactive menu that allows you to select between two modes: Player vs. Player (PVP) and Player vs. AI (PVAI), presenting a much more complete structure. The menu also represents a departure from the usually static interfaces in assembly projects, incorporating keyboard-driven selection and clean screen transitions.

Artificial Intelligence

Instead of requiring two human players, a right paddle controlled by "AI" was implemented, capable of moving automatically following the horizontal position of the ball. In addition, the AI is designed to decide when to jump depending on the vertical speed of the ball, the horizontal distance, and whether or not it is on the ground, imitating more complex reactive behavior.

Challenges Faced

Managing Complex Physics

When implementing the game physics updates, there were several challenges. Managing gravity, jumps, and forces dependent on the point of impact required good use of the available registers. Furthermore, it was necessary to deal with signed and unsigned operations, handle negative values using two's complement, and avoid frequent overflows. Added to this was the obligation to maintain a constant execution speed without relying on advanced timers, so the stability of the game depended on the optimization of each routine.

Sprite Rendering Performance

Another major challenge involved managing sprite rendering without compromising frame rate. Since drawing in mode 13h requires the use of INT 10h, calling the BIOS repeatedly can slow down execution dramatically. To mitigate this, several optimizations were implemented, such as minimizing memory accesses, reducing redundant register manipulation, and calculating sprite boundaries in advance. These optimizations were essential to achieving fluid gameplay even under the constraints of real-mode DOS.

Collision Detection Complexity

Collision detection also became a significantly more complex problem than in Pong. Instead of simple rectangular comparisons, the new system had to check for vertical, horizontal, and overlapping collisions, adjust the ball's position to prevent it from "getting stuck" inside a sprite after

a collision, detect impacts with the net, floor, and edges, and calculate the resulting force based on the distance to the center of the paddle. This level of precision involved writing detailed and well-timed routines, as any error would result in strange bounces or jumps out of range. Debugging these routines required extensive use of memory dumps, register monitoring, and step-by-step execution in DOSBox's debugger mode.

AI Movement and Timing

Finally, designing the AI system was very challenging. The AI had to move proportionally without crossing the net, react to the position of the ball with a small degree of predictability, and jump only when appropriate, all while respecting the rules of gravity and collision like a normal player. It was necessary to precisely adjust the movement speed and jump synchronization logic. Implementing the AI behavior in the assembly required careful allocation of memory registers and variables to avoid unwanted overwrites and maintain logical consistency throughout the game loop.

Instructions for Compiling

After downloading arcade.asm from the GitHub repository, in DOSBox:

```
masm /a arcade.asm

Link arcade

arcade
```