CS/SE2340.502 Term Project
Professor Alice Wang
Fall (November) 2024

Group Members:
-Omer Tariq
-Parsa Bazrpash
- Jaena Orozco

Project Report: Math-Match Game

A) Program Description

The goal of this project was to use MIPS assembly language to create an interactive Math-Match game. Each card in the game has two sides: one side shows a multiplication equation (for example, 2x3), while the other side is hidden as a question mark (?). The game is played on a 4x4 grid.

Matching card pairs that yield the same multiplication result is the aim. Each turn, players disclose two cards. The cards remain face-up if the outcomes match; if not, they are momentarily shown before being hidden once more. Until every card matches and there are no more question marks on the board, the game is over.

The game incorporates music effects and basic visual components made using ASCII characters to improve the user experience. (See the user manual for the game for further information.)

B) Challenges and Solutions

Challenge 1: Second Card Display Error

- **Problem**: The second card would not flip back over when no match was found.
- **Solution**: We debugged the logic by analyzing how the "check" function processed unmatched pairs. Writing pseudocode helped us pinpoint logical errors, and a fixed conditional statement was implemented by Jaena.

Challenge 2: Infinite Loops

- **Problem**: The display continuously printed, causing an infinite loop.
- **Solution**: We revised the game loop and identified that incorrect termination conditions were being used. Collaborative debugging resolved the issue by clarifying exit conditions in the loop.

Challenge 3: Timer Implementation

- **Problem**: Implementing a timer with constant refresh without affecting performance.
- **Solution**: Using online resources like Stack Overflow, Parsa successfully integrated a timer that refreshed periodically without freezing the game.

Challenge 4: Separation of Files

- **Problem**: Dividing game logic into modular files while maintaining cohesion.
- **Solution**: Team brainstorming led to a clearer structure, separating the board display, sound effects, and game logic into distinct modules.

C) Lessons Learned

- Importance of planning: Writing pseudocode significantly improved debugging and game logic clarity.
- Arrays in MIPS
- Setting up meetings with a group
- Team collaboration: Combining our skills allowed us to tackle issues effectively.
- Modularity: Separating files made the codebase more manageable and easier to debug.

D)Algorithms and Techniques Used

- Random number generation for card placement
- Input validation for player moves
- Display formatting using syscalls
- Card state tracking (face-up, face-down, matched)
- Match checking logic

Game Components

1. **Data Structures:**
    - equations[]: Contains mathematical equations as strings, e.g., { "2x2", "2x3", ..., "5x5" }.
    - answers[]: Stores the results of these equations, e.g., { 4, 6, ..., 25 }.
    - positions[16]: Tracks the shuffled positions of equations and answers.
    - revealed[16]: A boolean array indicating whether a card is currently face-up.
    - matched[16]: A boolean array indicating whether a card has been matched.
2. **Card Representation:**
    - **Key**: Identifies the card type and position.
        - Keys < 16 represent equations.
        - Keys >= 16 represent answers.
    - **Value**: Stores the content of the card.
        - E.g., "2x3" for an equation or 6 for an answer.

Display Logic

The game board consists of a 4x4 grid where cards are displayed based on their state: matched, revealed, or hidden.

Code Representation of Display Logic:

```
for (int row = 0; row < 4; row++) {
    for (int col = 0; col < 4; col++) {
    print "|"; // Card border

    // Check card status: matched, revealed, or hidden
    bool show = false;
    if (card[row][col].status == matched || card[row][col].status == revealed) {
    show = true;
    }

    // Display card content based on its type and state
    if (show) {
    if (key < 16) {
        print equation[key]; // Display equation
    } else {
        print answer[key - 16]; // Display answer
    }
    } else {
    print "?"; // Card is face-down
```

```
        }

        print "|"; // Card border
        }
}
```

Sound Logic Overview

The game uses a MIDI-based system call to play a sound whenever a card is flipped, enhancing user experience with auditory feedback.

- **Sound Parameters:**
    - flip_instrument: Specifies the instrument (e.g., Glockenspiel, value 9).
    - flip_duration: Sound duration (e.g., 150ms).
    - flip_volume: Maximum volume (127).
    - flip_pitch: Note pitch (72 corresponds to C5).
- **Implementation:**
    - The sound is triggered using syscall 31, which plays a MIDI note based on the parameters loaded into registers.
    - A brief delay is added using syscall 32 (sleep) to ensure the sound is audible.
    - The logic runs whenever a card is flipped, adding an interactive element to gameplay.

E) Contributions (Peer Evaluation)

- **Jaena**: Developed the board display and fixed a critical bug in the "check" function.
- **Parsa**: Implemented sound effects, integrated the timer, and created the demo.
- **Omer**: Authored the user manual.
- **All**: Collaborated on editing, debugging, game loop logic, and brainstorming solutions.

  In conclusion everyone did their part. Parsa worked on sound effects and integrating the timer, and Jaena focused on developing the display. I focused on debugging as well as creating the manual. Everyone pulled their weight

F) Suggestions

1. Future versions could include enhanced visuals or animations for matches.
2. Introducing different difficulty levels by altering the timer and board size.
3. Providing detailed logs for easier debugging during development.

- ○