Memory Game

The **Memory Game** is a hardware-based Simon Says–style game. The objective is simple: players must replicate an ever-growing sequence of randomly generated LED flashes using physical button inputs. The game challenges memory, timing, and reflexes, and offers multiple difficulty levels.

Game Features

Datapath

The game architecture is built around a modular design that includes:

- **LFSR-based pseudo-random number generator** for pattern generation
- **Shift register** to store and update the pattern
- **Comparator module** to verify user input
- **FSMs** for game control flow across different modes
- **Score counter** and pattern counter
- **LED output logic** and **button input encoder**

Game Modes

1. **Classic Mode**: Players observe a sequence of LED flashes and must repeat it correctly. If the player succeeds, a new random flash is added to the sequence. The game continues until a mistake is made.
2. **Time Challenge Mode**: Similar to Classic Mode, but players must respond within a fixed time window. A timer counts the time since the sequence was displayed, and failure to respond in time results in game over.
3. **Reverse Mode**: After seeing the generated sequence, players must enter it in reverse order.

Pseudo-Random Number Generator

The game originally attempted to use a truly random seed via $urandom for the Linear Feedback Shift Register (LFSR). However, this only works in simulation. For synthesis it was replaced with a fixed-seed pseudo-random generator using the polynomial x^32 + x^22 + x^2 + x + 1, which is a maximal-length LFSR. This approach still produces a long, repeatable pattern and keeps hardware logic simple and compact.

One-Hot Encoding for I/O

- **Pattern Input**: The 8 input buttons use a one-hot encoding scheme, where each button represents a unique 3-bit value.
- **Pattern Output**: A single LED is lit at a time to represent a 3-bit pattern, again using one-hot encoding for clarity and simplicity.

Challenges and Changes Made

- **Removed Speed Mode**: Initially, a speed-increasing mode was implemented where the display delay shortened over time. However, this caused timing instability and problems with GitHub Actions, so the mode was removed for simplicity and tapeout reliability.

- **Modified Random Generator**: The original plan to use `$urandom` for seeding a random pattern generator was abandoned due to synthesis incompatibility. A deterministic pseudo-random generator was used instead to ensure functionality after tapeout.

Peripherals

Onboard FPGA Resources

- **4 Directional Buttons:** Used for game mode selection (Classic, Time, Reverse).
- **2 Extra Buttons:** One is used to **start the game**, and the other to **trigger replay** after a game ends.
- **8 Onboard LEDs:** Used to display the current pattern. Each LED corresponds to one of 8 possible values.

Added Components

- **8 Input Buttons:** Added to allow the player to enter the pattern using one-hot encoding.
- **2 Extra LEDs**
    - One LED indicates the **game is currently active**
    - The other signals **game over** status
- **LCD Display:** Used to show current **round number or score**.