Tic Tac Toe on STM32H750B-DK

Stefanija Atanasova
63210397

17th September 2025

1 Introduction

In this project I made a Tic Tac Toe game on the STM32H750B Discovery board. I used the LCD touchscreen to show the game board, and I used three LEDs to show the current player and game status. I also used the push-button to reset the game at any time.

For the screen and touch input I used the library `stm32_lcd.h`. With it I could easily draw the grid, text, and read touch positions. This helped me focus more on the game logic, LEDs, and interrupts.

Note: Watch the attached video to see the Tic Tac Toe game running on the board.

2 GPIO and LED Control

I used three GPIO pins for the LEDs:

- PI13 for the LED of Player X
- PJ2 for the LED of Player O
- PD3 for the LED showing the game is finished

I set them as outputs during initialization. I wrote functions for controlling them:

- `LED_SetTurn(int player)` – I use it to show the current player LED.
- `LED_GameOver()` – I use it to turn on all LEDs when the game is over.

3 Push-Button and Interrupt

The USER button on the STM32H750B Discovery board is already connected to an external interrupt line, and the BSP drivers handle the low-level interrupt function `EXTI15_10_IRQHandler()`. Because of that, I did not need to fully write my own handler.

Instead, I used the HAL callback function `HAL_GPIO_EXTI_Callback()` to add my own behavior. In this function I check if the interrupt came from the USER button pin, and if yes, I call `ResetGame()`.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    if (GPIO_Pin == BUTTON_USER_PIN) {
        ResetGame();
    }
}
```

This way, whenever I press the push-button, the game is restarted automatically. It also keeps the main game loop simple, since I don't need to constantly check for button presses there.

4 Tic Tac Toe Game Logic

4.1 CheckWinner()

I wrote this function to check rows, columns, and diagonals of the 3×3 board. It returns the winner if there is one, or 0 if nobody has won yet.

4.2 DrawBoard()

This function draws the grid and puts X and O symbols depending on the state of the board. I also call the LED update inside, so the LEDs always show whose turn it is.

4.3 HandleTouchInput()

I read the touchscreen using `BSP_TS_GetState()`.

- I convert the coordinates to a row and column.
- If the cell is empty, I put the symbol of the current player.
- I redraw the board and check if someone won.
- I change the player and update the LEDs.

4.4 ResetGame()

I reset all the cells of the board, set the current player back to Player X, redraw the grid, and turn on Player X's LED.

5 Main Program Flow

The program flow I made is simple:

- I initialize HAL, system clock, LCD, touchscreen, LEDs, and push-button interrupt.
- I draw the empty Tic Tac Toe board.
- Player X starts with the LED on PI13.
- Players play by touching the screen.

- The game checks for a winner and updates the LEDs.
- If there is a winner, all LEDs turn on.
- If I press the push-button, the interrupt resets the game.

6 Future Ideas

I want to add:

- **Single-player vs CPU:** so one player can play against the board. For this I could use the **Minimax algorithm**, which is often used in games like Tic Tac Toe to let the computer always play the best possible move.
- **Score tracking:** to count wins and draws across rounds.

7 Appendix – Important Code Snippets

7.1 LED Control

```c
void LED_SetTurn(int player) {
    if (player == PLAYER_X) {
        *GPIOI_ODR &= ~(1 << 13); // X LED on
        *GPIOJ_ODR |= (1 << 2);   // O LED off
    } else if (player == PLAYER_O) {
        *GPIOJ_ODR &= ~(1 << 2);  // O LED on
        *GPIOI_ODR |= (1 << 13);  // X LED off
    }
}

void LED_GameOver(void) {
    *GPIOI_ODR &= ~(1 << 13);
    *GPIOJ_ODR &= ~(1 << 2);
    *GPIOD_ODR |= (1 << 3);       // All LEDs on
}
```

7.2 Game Logic

```c
int CheckWinner(void) {
    for (int i = 0; i < 3; i++) {
        if (gameBoard[i][0] == currentPlayer &&
            gameBoard[i][1] == currentPlayer &&
            gameBoard[i][2] == currentPlayer) return currentPlayer;

        if (gameBoard[0][i] == currentPlayer &&
            gameBoard[1][i] == currentPlayer &&
            gameBoard[2][i] == currentPlayer) return currentPlayer;
    }

    if (gameBoard[0][0] == currentPlayer &&
        gameBoard[1][1] == currentPlayer &&
        gameBoard[2][2] == currentPlayer) return currentPlayer;
    if (gameBoard[0][2] == currentPlayer &&
        gameBoard[1][1] == currentPlayer &&
        gameBoard[2][0] == currentPlayer) return currentPlayer;

    return 0; // No winner
}
```

7.3 Reset Function with Interrupt

```c
void ResetGame(void) {
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            gameBoard[i][j] = 0;

    currentPlayer = PLAYER_X;
    DrawBoard();
    LED_SetTurn(currentPlayer);
}

// HAL callback for user button interrupt
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    if (GPIO_Pin == BUTTON_USER_PIN) {
        ResetGame();
    }
}
```