



Report for

Input-Output systems project

Student:

Stefan Makivić, 63220418

Table of Contents

1.0. Project overview	1
2.0 File main.c	2
2.1 Key functions	2
2.1.1 int main(void)	2
2.1.2 static void SystemClock_Config(void)	3
2.2 Logic behind the program	3
3.0 File touchscreen.c	4
3.1 Key functions	4
3.1.1 void display_main_screen(void)	4
3.1.2 void handle_main_screen_touch(uint16_t x, uint16_t y)	5
3.2 Key concepts	5
4.0 File tic_tac_toe.c	6
4.1 Key functions	6
4.1.1 void start_tic_tac_toe(void)	6
4.1.2 void draw_grid(void)	7
4.1.3 void draw_player_mark(int row, int col, int player)	7
4.1.4 void handle_tic_tac_toe_touch(uint16_t x, uint16_t y)	7
4.1.5 void check_game_status(void)	8
4.1.6 void tic_tac_toe_game_over(char winner)	8
4.2 Key concepts	9
5.0 File memory_game.c	9
5.1 Key functions	10
5.1.1 void start_memory_game(void)	10
5.1.2 void draw_start_button(void)	10
5.1.3 void draw_color_squares(void)	10
5.1.4 void generate_sequence(void)	11
5.1.5 void play_sequence(void)	11
5.1.6 void handle_memory_game_touch(uint16_t x, uint16_t y)	11
5.1.7 void check_player_input(Color color)	12
5.1.8 void show_game_over(void)	12
5.2 Key concepts	13
6.0 File whack_a_mole.c	13

6.1 Key Functions.....	13
6.1.1 void start_whack_a_mole(void)	13
6.1.2 void draw_score_and_lives(void).....	14
6.1.3 void draw_mole_grid_outline(void)	14
6.1.4 void spawn_mole(void)	15
6.1.5 void handle_whack_a_mole_touch(uint16_t x, uint16_t y).....	15
6.1.7 void game_over(void)	16
6.1.8 void whack_a_mole_game_loop(void)	16
6.1.9 void reset_game_state(void).....	16
6.2 Key Concepts.....	17

1.0. Project overview

The project is a multi-game application developed on the STM32H7 board, featuring **Tic Tac Toe**, **Whack-a-Mole**, **Memory Game**, and a main screen to choose between them. The project uses touch inputs and an LCD display for user interaction. Each game follows its own logic while adhering to a common state management system to switch between screens.

For each game and for the main screen I made a separate C file and in this report I will describe the logic behind the games and everything I did. I will also mention some of the more important functions from each file and briefly describe what they do.

With this report I have submitted the five files that I went through and a video showing how it works. The entire project is on my GitHub on this link:

https://github.com/smakivic/stm32h7_multi_game_app

2.0 File main.c

The main.c file is the entry point for the multi-game application on the STM32H7 board. It is responsible for initializing the system, configuring the clock, setting up the touch screen and LCD, and managing the flow of the application by handling touch inputs and game states. The file integrates different game states, handles game logic, and allows users to switch between games using touch input. It also includes functions for managing the non-blocking logic of the Whack-a-Mole game.

2.1 Key functions

2.1.1 int main(void)

The main() function is the core of the program. It initializes hardware components, such as the CPU cache, HAL, system clock, LCD, and touchscreen. It continuously runs in a loop to check for touch inputs and manage game logic based on the current state. Here's a detailed explanation:

- **CPU and Peripheral Initialization:**
 - CPU_CACHE_Enable(): Enables the CPU L1 Cache for faster memory access.
 - HAL_Init(): Initializes the HAL (Hardware Abstraction Layer), which provides a unified interface to the STM32 peripherals.
 - SystemClock_Config(): Configures the system clock to ensure the correct speed for the processor and peripherals.
- **LCD and Touchscreen Initialization:**
 - BSP_LCD_Init(): Initializes the LCD screen in landscape orientation.
 - BSP_TS_Init(): Sets up the touchscreen with its width, height, and orientation, enabling touch detection with appropriate accuracy.
- **Game State Handling:**
 - The main loop checks for touch input (BSP_TS_GetState) and dispatches the touch event to the corresponding game function based on the current state (current_state).
 - The states include MAIN_SCREEN, TIC_TAC_TOE, WHACK_A_MOLE, and MEMORY_GAME. For example, when in WHACK_A_MOLE state, handle_whack_a_mole_touch(x, y) processes the touch event.
- **Whack-a-Mole Game Logic:**

- The main() function includes non-blocking logic to manage mole spawning and timeouts. It uses HAL_GetTick() to track elapsed time for spawning and despawning moles without interrupting other game functions.
- check_mole_timeout() checks if a mole has been on screen for too long, while spawn_mole() creates a new mole at regular intervals.
- **Performance Optimization:**
 - A small delay (HAL_Delay(100)) is added at the end of each loop iteration to reduce CPU usage, ensuring the system runs efficiently without unnecessary resource consumption.

2.1.2 static void SystemClock_Config(void)

This function configures the system clock settings for the STM32H7 board. It sets up the High-Speed External (HSE) oscillator, the PLL (Phase-Locked Loop) settings, and the clock dividers for different peripheral buses.

The system clock is sourced from the PLL with a frequency determined by the settings provided in the function (e.g., PLLM = 5, PLLN = 160). The function ensures that the processor runs at optimal speeds for high-performance applications.

2.2 Logic behind the program

The main.c file governs the overall structure and flow of the application. It is responsible for:

- **System Initialization:** Ensuring that the processor, LCD, and touchscreen are correctly set up.
- **State Management:** Using the current_state variable to track which game is active and dispatch touch events accordingly.
- **Touch Input Handling:** Continuously reading touch input and dispatching it to the appropriate game logic based on the active state.
- **Non-blocking Game Logic:** Especially in the Whack-a-Mole game, time-based events like mole spawning and timeouts are handled without blocking the main loop.
- **Game State Switching:** Ensuring smooth transitions between different games and the main menu using touch inputs.

By combining touch input handling, game logic, and real-time event management, this file serves as the backbone of the multi-game project, making it responsive and efficient.

3.0 File touchscreen.c

This file, touchscreen.c, handles the user interface for the main screen of the project and manages touch input when the player selects one of the three games. It includes functions for drawing the main screen, responding to user touch input, and starting games. Here's a breakdown of the main components:

Overview:

1. **display_main_screen()**: Draws the main screen with three sections, each representing a different game (Memory game, Tic Tac Toe, and Whack-a-Mole). Each section has a unique color and displays the name of the respective game.
2. **handle_main_screen_touch(uint16_t x, uint16_t y)**: Handles touch input on the main screen. Depending on where the user touches, it transitions the state to the selected game and starts it.

3.1 Key functions

3.1.1 void display_main_screen(void)

Purpose: This function draws the main menu, dividing the screen into three sections for the games and labeling them accordingly.

Logic:

- It first clears the screen to a white background using UTIL_LCD_Clear().
- The screen is then split into three vertical sections. Each section is filled with a different background color and displays the name of one of the games in the center.
- For example, the left section is colored dark blue and labeled "Memory game", the middle section is labeled "Tic Tac Toe", and the right section is labeled "Whack-a-Mole".

Key Functions Used:

- UTIL_LCD_Clear(): Clears the screen.
- UTIL_LCD_FillRect(): Fills a rectangle with a specific color.
- UTIL_LCD_SetTextColor() and UTIL_LCD_SetBackColor(): Sets the text and background colors.
- UTIL_LCD_DisplayStringAt(): Displays text at a specified location.

3.1.2 void handle_main_screen_touch(uint16_t x, uint16_t y)

Purpose: This function handles the user's touch input on the main screen. It checks where the user touched and starts the corresponding game (Memory game, Tic Tac Toe, or Whack-a-Mole).

Logic:

- First, it checks if the current state is MAIN_SCREEN. If not, the function returns without doing anything.
- Based on the x-coordinate of the touch input:
 - If the touch is in the left section ($x < 160$), it starts the **Memory game**.
 - If the touch is in the middle section ($160 \leq x < 320$), it starts **Tic Tac Toe**.
 - If the touch is in the right section ($x \geq 320$), it starts **Whack-a-Mole**.
- The game will only start if there isn't a game currently in progress (game_in_progress is false), preventing users from accidentally starting a game twice.

Key Functions Used:

- start_memory_game(), start_tic_tac_toe(), start_whack_a_mole(): These functions are called to initialize and start the corresponding games.
- UTIL_LCD_FillRect(): Used again in the right section to update the color dynamically when starting the Whack-a-Mole game.

3.2 Key concepts

State Management: The variable current_state is used to track which part of the program is active. It ensures that the touch input is only processed when the main screen is active.

Game Progress Flag: The game_in_progress flag prevents multiple games from starting simultaneously. It is set to true once a game starts and reset when the player returns to the main screen.

4.0 File tic_tac_toe.c

This file handles the implementation of the Tic Tac Toe game in my STM32H7 project. It includes logic for drawing the game grid, handling player input via touch, checking the game status (win, lose, or draw), and transitioning back to the main screen when the game ends.

Overview:

1. **start_tic_tac_toe()**: Initializes the Tic Tac Toe game by clearing the screen, drawing the grid, and setting up the initial state.
2. **draw_grid()**: Draws the Tic Tac Toe grid and the "Back" button on the screen.
3. **handle_tic_tac_toe_touch(uint16_t x, uint16_t y)**: Handles touch input during the Tic Tac Toe game and places player marks based on where the user taps.
4. **check_game_status()**: Checks if either player has won or if the game is a draw.
5. **tic_tac_toe_game_over(char winner)**: Displays the game result (win or draw) and returns to the main screen after a short delay.

4.1 Key functions

4.1.1 void start_tic_tac_toe(void)

Purpose: This function sets up the game state and the visual representation of the Tic Tac Toe grid. It is called when the player selects the Tic Tac Toe option from the main menu.

Logic:

- The screen is cleared with a dark blue background using UTIL_LCD_Clear().
- The function then draws the Tic Tac Toe grid using draw_grid().
- It initializes the 3x3 grid to represent empty cells (denoted by 0).
- Player 1 (X) is set to start the game.
- A small delay is added with HAL_Delay() to avoid processing residual touch input from previous states.

Key Functions Used:

- UTIL_LCD_Clear(): Clears the screen.
- draw_grid(): Draws the Tic Tac Toe grid and the "Back" button.
- HAL_Delay(): Adds a delay to avoid touch carryover.

4.1.2 void draw_grid(void)

Purpose: This function draws the visual layout of the Tic Tac Toe grid on the screen.

Logic:

- It draws four vertical and horizontal white lines to form the 3x3 grid.
- The "Back" button is displayed in the top left corner to allow the player to return to the main screen.

Key Functions Used:

- UTIL_LCD_DrawVLine() and UTIL_LCD_DrawHLine(): Draw vertical and horizontal lines to form the grid.
- UTIL_LCD_DisplayStringAt(): Displays the "Back" button text.

4.1.3 void draw_player_mark(int row, int col, int player)

Purpose: This function draws the player's mark ('X' for Player 1 and 'O' for Player 2) at the specified grid cell.

Logic:

- The x and y positions for the mark are calculated based on the grid's layout.
- If the player is X, it draws an 'X' in white at the specified position; if the player is O, it draws an 'O'.

Key Functions Used:

- UTIL_LCD_DisplayStringAt(): Displays the 'X' or 'O' on the screen at the correct position.

4.1.4 void handle_tic_tac_toe_touch(uint16_t x, uint16_t y)

Purpose: This function handles the touch input during the Tic Tac Toe game. It determines which grid cell was touched, places the current player's mark, and checks the game status.

Logic:

- First, it checks if the "Back" button was pressed (if the touch coordinates are within the button's area) and, if so, returns to the main screen.
- Then, it maps the touch coordinates to the grid rows and columns. If the touch is within a valid grid cell and the cell is empty, the current player's mark is placed.

- It calls `check_game_status()` to see if there is a winner or if the game ends in a draw.
- The current player is then switched.

Key Functions Used:

- `draw_player_mark()`: Draws the player's mark on the screen.
- `check_game_status()`: Checks if the game has ended in a win or draw.

4.1.5 void check_game_status(void)

Purpose: This function checks the current state of the game to see if there is a winner or if the game ends in a draw.

Logic:

- It checks all rows, columns, and diagonals to see if either player has filled a line with their mark.
- If a player wins, the `tic_tac_toe_game_over()` function is called with the winning player's identifier.
- If no player wins and all grid cells are filled, the game is declared a draw.

Key Functions Used:

- `tic_tac_toe_game_over()`: Displays the result and resets the game.

4.1.6 void tic_tac_toe_game_over(char winner)

Purpose: This function displays the result of the game (whether 'X' won, 'O' won, or if it was a draw) and returns to the main screen after a delay.

Logic:

- The screen is cleared, and the appropriate message ("X player won!", "O player won!", or "It's a draw!") is displayed in the center of the screen.
- After a 3-second delay, the game transitions back to the main screen by calling `display_main_screen()`.

Key Functions Used:

- `UTIL_LCD_Clear()`: Clears the screen.
- `UTIL_LCD_DisplayStringAt()`: Displays the result message.
- `HAL_Delay()`: Adds a delay before returning to the main screen.
- `display_main_screen()`: Returns to the main screen after the game ends.

4.2 Key concepts

Touch Input Mapping: The touch input coordinates are mapped to grid cells based on the layout. This allows the player to place their mark by tapping the corresponding cell.

State Management: The `current_state` is used to track whether the Tic Tac Toe game is active or if the player has pressed the "Back" button to return to the main screen.

Game Logic: The game checks for a win or draw after each move and transitions to an end state if either condition is met.

5.0 File memory_game.c

This file contains the implementation of the Memory Game for the STM32H7 project. The game involves displaying a sequence of colors that the player must replicate by tapping the appropriate squares. The sequence grows with each successful attempt, and the game ends when the player fails to match the sequence. The file manages game setup, sequence generation, player input handling, and game state transitions.

Overview:

1. `start_memory_game()`: Initializes the Memory Game, draws the color squares, and sets up the initial game state.
2. `draw_start_button()`: Draws the "Start game" button on the screen.
3. `draw_color_squares()`: Draws the four color squares used in the game.
4. `generate_sequence()`: Creates a random sequence of colors that the player must follow.
5. `play_sequence()`: Displays the sequence of colors on the screen for the player to memorize.
6. `handle_memory_game_touch(uint16_t x, uint16_t y)`: Handles player input during the game and verifies if the player taps the correct sequence of squares.
7. `check_player_input(Color color)`: Compares the player's input with the generated sequence to check for correctness.
8. `show_game_over()`: Displays the player's final score and transitions back to the main screen after a short delay.

5.1 Key functions

5.1.1 void start_memory_game(void)

Purpose: This function sets up the Memory Game when selected from the main menu. It draws the initial screen layout and prepares the game state.

Logic:

- Clears the screen and draws the 'Start game' button.
- Draws the four color squares on the screen.
- Resets game variables such as the score and sequence length.
- Initializes the random seed for generating sequences.
- Waits for the player to press the 'Start game' button to begin.

Key Functions Used:

- UTIL_LCD_Clear(): Clears the screen.
- draw_start_button(): Draws the "Start game" button.
- draw_color_squares(): Draws the color squares.

5.1.2 void draw_start_button(void)

Purpose: This function draws the 'Start game' button on the screen, which the player presses to start the game.

Logic:

- The button is drawn at the center of the screen with the label "Start game".
- Touch detection is set up to listen for the player's input.

Key Functions Used:

- UTIL_LCD_DisplayStringAt(): Displays the "Start game" text on the button.

5.1.3 void draw_color_squares(void)

Purpose: This function draws the four color squares used for player input in the game.

Logic:

- Four squares are drawn in fixed positions, each with a different color: Red, Blue, Green, and Yellow.

- The squares serve as input areas for the player during gameplay.

Key Functions Used:

- UTIL_LCD_FillRect(): Fills the rectangle representing each square with its corresponding color.

5.1.4 void generate_sequence(void)

Purpose: This function generates a random sequence of colors for the player to memorize and replicate.

Logic:

- Randomly selects colors from a predefined list (Red, Blue, Green, Yellow).
- Ensures no color appears more than three times consecutively.
- The sequence length increases after each successful round.

Key Functions Used:

- rand(): Generates random numbers used to select colors.

5.1.5 void play_sequence(void)

Purpose: Displays the color sequence on the screen for the player to memorize.

Logic:

- Each color in the sequence is shown for a set duration, with a short delay between colors.
- The message "Watch closely!" is displayed during the sequence display phase.
- After the sequence ends, the message "Begin!" signals the player to start their turn.

Key Functions Used:

- show_color(): Fills the square with the corresponding color.
- HAL_Delay(): Adds a delay between colors.

5.1.6 void handle_memory_game_touch(uint16_t x, uint16_t y)

Purpose: Handles the player's touch input during the game and checks if it matches the sequence.

Logic:

- Maps the touch coordinates to the correct square.

- Calls `check_player_input()` to verify if the tapped square matches the sequence.
- If the input is correct, it moves to the next color in the sequence. If incorrect, the game ends.

Key Functions Used:

- `check_player_input()`: Compares the player's input to the sequence.
- `show_game_over()`: Displays the final score and ends the game.

5.1.7 void check_player_input(Color color)

Purpose: Verifies whether the player's touch input matches the current step in the sequence.

Logic:

- Compares the color tapped by the player with the expected color in the sequence.
- If correct, proceeds to the next step; if incorrect, ends the game and displays the score.
- If the player completes the sequence, the score increases, and the next level starts.

Key Functions Used:

- `play_sequence()`: Replays the new, longer sequence after a successful round.
- `show_game_over()`: Ends the game if the player taps the wrong square.

5.1.8 void show_game_over(void)

Purpose: Displays the player's final score and returns to the main screen after a delay.

Logic:

- Clears the screen and shows a message with the player's score.
- After a 3-second delay, it returns to the main menu.

Key Functions Used:

- `UTIL_LCD_Clear()`: Clears the screen.
- `UTIL_LCD_DisplayStringAt()`: Displays the game over message and score.
- `HAL_Delay()`: Delays before returning to the main menu.

5.2 Key concepts

Touch Input Handling: The game maps touch input to specific color squares, allowing players to interact with the game by tapping.

Sequence Generation: A random sequence of colors is generated, and the player must replicate it. The sequence grows as the player progresses.

Game State Management: The game tracks the player's progress, handling transitions between different states like sequence display, player input, and game over.

6.0 File whack_a_mole.c

This file manages the implementation of the Whack-a-Mole game in the STM32H7 project. It includes logic for mole spawning, touch input handling, score and life tracking, and transitioning back to the main screen when the game ends.

Overview:

1. `start_whack_a_mole()`: Initializes the Whack-a-Mole game, drawing the grid and setting up the initial game state.
2. `spawn_mole()`: Randomly places a mole on the grid and starts the timer for mole appearance.
3. `handle_whack_a_mole_touch(uint16_t x, uint16_t y)`: Handles touch input, checking if the player hits a mole or the "Back" button.
4. `check_mole_timeout()`: Checks if moles have been on the screen too long and removes them if the player misses.
5. `game_over()`: Ends the game, displaying the final score and transitioning back to the main screen after a delay.
6. `whack_a_mole_game_loop()`: Controls the game flow, continuously spawning moles and updating the game state.
7. `reset_game_state()`: Resets the game variables for a fresh start.

6.1 Key Functions

6.1.1 void start_whack_a_mole(void)

Purpose: Initializes the Whack-a-Mole game, drawing the grid and resetting the game state. Called when the player selects the Whack-a-Mole option from the main menu.

Logic:

- The screen is cleared with a dark blue background using UTIL_LCD_Clear().
- Score and lives are drawn at the top, and the mole grid outline is displayed.
- The game starts by calling whack_a_mole_game_loop().

Key Functions Used:

- UTIL_LCD_Clear(): Clears the screen.
- draw_mole_grid_outline(): Draws the grid outline.
- draw_score_and_lives(): Displays the player's score and remaining lives.

6.1.2 void draw_score_and_lives(void)

Purpose: Displays the current score and remaining lives at the top of the screen.

Logic:

- The function first draws the outline of the grid using draw_mole_grid_outline().
- The "Back" button is drawn in the top-left corner.
- The player's score and lives are displayed using UTIL_LCD_DisplayStringAt().

Key Functions Used:

- draw_mole_grid_outline(): Draws the grid outline.
- UTIL_LCD_DisplayStringAt(): Displays text for the score, lives, and "Back" button.

6.1.3 void draw_mole_grid_outline(void)

Purpose: Draws the white rectangle outlining the grid area for the moles.

Logic:

- A rectangle is drawn from coordinates (80, 32) to (440, 270) using vertical and horizontal lines.

Key Functions Used:

- UTIL_LCD_DrawHLine() and UTIL_LCD_DrawVLine(): Draw the horizontal and vertical lines for the grid outline.

6.1.4 void spawn_mole(void)

Purpose: Spawns a mole at a random position within the grid.

Logic:

- The function checks for any inactive mole in the moles array.
- If an inactive mole is found, its position is set randomly within the grid, and its appearance time is recorded.
- The mole is drawn on the screen as a brown circle, and its appearance duration is set randomly between 2 to 5 seconds.

Key Functions Used:

- UTIL_LCD_FillCircle(): Draws the mole on the screen.

6.1.5 void handle_whack_a_mole_touch(uint16_t x, uint16_t y)

Purpose: Handles touch input during the game, checking if a mole is hit or if the "Back" button is pressed.

Logic:

- The function first checks if the "Back" button was pressed; if so, the game state is reset, and the player is returned to the main menu.
- It then checks if the player touched any active mole. If a mole is hit, it increments the score and clears the mole from the screen.

Key Functions Used:

- reset_game_state(): Resets the game before going back to the main screen.
- clear_mole(): Clears the mole from the screen after it is hit.

6.1.6 void check_mole_timeout(void)

Purpose: Checks if any active mole has timed out and removes it if the player missed it.

Logic:

- For each active mole, the function compares the current time with the mole's appearance time. If the mole has been on the screen longer than its duration, it is removed, and the player loses a life.
- If the player has no remaining lives, game_over() is called.

Key Functions Used:

- clear_mole(): Clears the mole from the screen.

- game_over(): Ends the game if the player has no lives left.

6.1.7 void game_over(void)

Purpose: Ends the game, displays the final score, and transitions back to the main screen after a delay.

Logic:

- The screen is cleared, and a message with the player's final score is displayed.
- After a 3-second delay, the game resets, and the main screen is shown.

Key Functions Used:

- UTIL_LCD_Clear(): Clears the screen.
- UTIL_LCD_DisplayStringAt(): Displays the game over message.
- HAL_Delay(): Adds a delay before transitioning back to the main screen.

6.1.8 void whack_a_mole_game_loop(void)

Purpose: The main game loop, continuously spawning moles and updating the game state.

Logic:

- The game state is first reset.
- The first mole is spawned, and the game continues based on touch input and mole timeouts.

Key Functions Used:

- reset_game_state(): Initializes the game state.
- spawn_mole(): Spawns the first mole.

6.1.9 void reset_game_state(void)

Purpose: Resets all game variables to prepare for a new game.

Logic:

- Resets the score, lives, and deactivates all moles.

Key Functions Used:

- None (directly manipulates variables).

6.2 Key Concepts

Mole Spawning: Moles are randomly placed in a 9x6 grid and have a random appearance duration between 2 to 5 seconds.

Touch Input Mapping: Touch inputs are mapped to grid cells to allow players to hit moles by tapping them.

State Management: Game states like score, lives, and mole activity are managed to track player progress and game status.

Game End Handling: The game ends when the player has no remaining lives, displaying a final score and returning to the main screen after a delay.