**Startup path (program boot)**

**main.c**

* **Seed RNG once**: srand(time(NULL)); — done at program start so AI randomness is good for the whole run (and not reseeded per move).
* **OS/UTF-8 checks**:
  + On Windows, switches code page to UTF-8 if needed; on Linux/macOS checks LANG contains UTF-8. This ensures borders/colors render correctly.
* **Menu loop**: prints options, reads choice, flushes stdin, clears screen, and dispatches to run\_game(mode) or to help/about. The loop only breaks on “Exit.”

**Entering the game**

**run\_game(int mode) in game.c**

* **Board state**: local char board[3][3] initialized to EMPTY (from board.h), current\_player = X, turn = 0.
* Branches by **mode**:
  + **Mode 1 (PvP)** → reads human input every turn.
  + **Mode 2 (PvAI)** → human plays X, AI plays O; human path uses scanf; AI path calls ai\_move(board).

**Board rendering (every turn)**

**board.c**

* **print\_board(board)** draws a 3×3 grid with row/col headers, Unicode borders, and colored cells (X=red, O=blue). Uses print\_cell() to color each cell and prints separators between rows.
* Color/char constants (X, O, EMPTY, ANSI codes) are defined in **board.h**.

**Human turn (input → validation → apply move)**

Inside **PvP** or the **human branch of PvAI**:

1. **Read**: scanf("%d %d", &row, &col); if it fails, flush input buffer and re-prompt.
2. **Range-check**: must be 1..3 for both row/col; otherwise, message + continue.
3. **Normalize**: convert to 0-indexed with row--; col--;.
4. **Occupancy**: if board[row][col] != EMPTY, reject.
5. **Apply**: board[row][col] = current\_player; then turn++.

**Note on I/O hygiene**: after every scanf, the code **flushes stdin** (while (getchar() != '\n');) to prevent stale input from breaking the next read.

**Win detection (same for both modes)**

After a valid move, **three checks** run:

1. **Rows/Columns** — loop i = 0..2
   * Row win: board[i][0]==board[i][1]==board[i][2]!=EMPTY
   * Col win: board[0][i]==board[1][i]==board[2][i]!=EMPTY  
     On hit: print board + winner, return;.
2. **Diagonals** — two fixed checks: main diagonal [0,0]→[2,2] and anti-diagonal [0,2]→[2,0]. On hit: board + winner, return;.
3. **Draw** — if turn == 9, print board + “draw!”, then break;.

**Winner printing**:

* PvP uses win\_message(board, current\_player) (prints board + green “Player X wins!”).
* PvAI annotates when **Computer (O)** wins.

**Player switching**

* PvP: simple toggle: current\_player = (current\_player == X) ? O : X; or equivalent.
* PvAI: same toggle after each applied move; AI/human roles are determined by current player.

**AI turn (how ai\_move() works)**

**GameAI.c**

* **Pacing**: sleep(1) before and after the move to simulate “thinking.”
* **Selection loop**: repeatedly generate row = rand()%3, col = rand()%3 until board[row][col] == EMPTY. Then place **O** and print the chosen coordinates (1-indexed for user clarity).
* **Why RNG seeding in main() is optimal**: seeding once prevents low-entropy sequences from repeated, close-timed reseeding; this was fixed by moving srand(time(NULL)) out of AI and into main().

**Headers & contracts**

* **board.h**: declares print\_board, defines X, O, EMPTY, and color codes; it’s the **single source of truth** for glyphs and styling.
* **game.h**: declares run\_game(int mode) for main.c to call.
* **GameAI.h**: declares ai\_move(char board[3][3]) for game.c to call.

**Make/build flow**

* **Makefile** (overview): compiles each .c to .o with proper flags and links them into the final executable; incremental builds only recompile changed files. (You’ll show this target-by-target briefly when you scroll the repo’s root.)

**One realistic trace (PvAI example you can narrate live)**

1. **main()** → seed RNG → pass UTF-8/OS checks → show menu → user picks “Human vs AI” → run\_game(2).
2. **run\_game init** → clear 3×3 board to EMPTY, set current\_player='X', turn=0.
3. **Human(X) turn** → read row/col → validate (format/range/occupancy) → write X → turn++.
4. **Win/draw checks** → if none, toggle to O.
5. **AI(O) turn** → ai\_move(board) → sleep → random empty cell → write O → announce coordinates.
6. **Back to run\_game** → turn++ → same win/draw checks → toggle to X.
7. Repeat steps **3–6** until **row/col/diag win** or **turn==9** ⇒ draw.

**How to present this smoothly**

* When you show **main.c**, literally point out srand(time(NULL)) and the OS checks. Then say “menu → run\_game(mode).”
* Jump to **game.c** and trace one loop: “read → validate → apply → check rows/cols/diags → draw check → toggle.”
* Open **board.c** for 5 seconds to show the pretty printing and colors.
* Open **GameAI.c** to show the sleep(1) and the random loop with the EMPTY check. Say one line about why seeding moved to main().

If you want, I can compress this into a **one-page presenter cheat sheet** (with bolded file names and 1–2 lines under each), but this tracing above is the “to-the-metal” narrative you asked for.