**Slide 1 – Introduction (~0:35)**

“Good [morning/afternoon] everyone.  
We are Group 2 — Avishek Dutta, Md. Ishrak Mashroor, Ashab Mahmud Raseen, and Sneha Nandy.  
Our project is called *TicTacX* — a console-based take on the classic Tic-Tac-Toe game, built entirely in C.

Now, I know Tic-Tac-Toe might sound simple, but our focus was on making it run *everywhere* — Windows, Linux, macOS — with a clean terminal interface, fast responsiveness, and game logic that simply does not break no matter what you throw at it.”

**Slide 2 – Description (~0:40)**

“This isn’t just a copy of the old game. We reimagined it for the terminal.  
We wanted something modern for CLI — colorful output, intuitive controls, and most importantly, rock-solid stability.  
Whether it’s a casual game between friends or a quick match against the computer, the core of our work is in **modular architecture, strong input validation**, and OS compatibility.

In short — you can run it anywhere, and it will just work.”

**Slide 3 – What We Built (~0:45)**

“So, what exactly did we build?  
A fully interactive Tic-Tac-Toe game with two ways to play: human vs. human, or human vs. AI.  
We split the code into separate files for the UI, game logic, and AI.  
And here’s something we’re proud of — it looks good, even in a terminal window.

We use ANSI escape codes to add color, Unicode for neat borders, and clear screen handling so the display updates instantly without flicker — no graphical libraries, just precise control over the console.”

**Slide 4 – Game Modes (~0:45)**

“We’ve got two main modes.

The first is **Player vs Player**, the classic — two people taking turns.

The second is **Player vs AI**. In this mode, the human always plays X and the computer plays O. The AI isn’t trying to be a grandmaster — it’s an easy level that plays random valid moves, but it follows all the same rules as a human.

And here’s the important design choice: both modes use *exactly the same core game loop*. That means if we upgrade the AI later, we don’t have to rewrite the win/draw detection — it just works.”

**Slide 5 – Features (~0:50)**

“Let’s talk features.

* The board is color-coded — X in red, O in blue — easy to see at a glance.
* You can replay instantly without restarting the program.
* Every input is validated — wrong formats, numbers outside 1–3, or moves in occupied cells are all caught.
* We handle the quirks of different operating systems with preprocessor directives and UTF-8 checks.

These aren’t flashy features — they’re the things that make a program *reliable*.”

**Slide 6 – Code Structure (~0:50)**

“Under the hood, here’s how it’s organized:

* main.c runs the menu, checks the OS, seeds the random number generator.
* game.c has the main loop, win detection, and turn handling.
* board.c handles rendering with ANSI colors and Unicode borders.
* GameAI.c holds the AI’s move logic.
* .h header files store constants like X, O, and EMPTY, plus function prototypes.
* The Makefile compiles the whole thing with modularity — if we change one file, it only rebuilds that part.”

**Slide 7 – Game Logic (~0:50)**

“In run\_game(), we start by creating a 3×3 array filled with EMPTY spaces.  
We alternate turns between X and O using a simple toggle.  
Every move is validated before it touches the board.  
For win detection, we loop through all rows, all columns, and both diagonals.  
We also have a turn counter — when it reaches 9 without a win, we declare a draw.

In Player vs AI mode, we just replace one player’s turn with ai\_move() — the rest of the loop stays exactly the same.”

**Slide 8 – AI Implementation (~0:50)**

“Our AI lives in ai\_move().  
It starts with a short delay — one second — just to make it feel like the computer is thinking.  
Then it picks a random row and column until it finds an empty cell.  
It plays O in that cell, prints the move coordinates, and passes control back to the main loop.

Crucially, the AI doesn’t have its own win-checking logic — it uses the same one as human players. That means less duplicated code and fewer bugs.”

**Slide 9 – AI Key Challenges (~0:55)**

“Now, making the AI wasn’t without its challenges.

1. **Inefficient RNG Seeding** — At first, we had srand(time(NULL)) inside GameAI.c. That meant reseeding every time the AI played, and if moves were close together in time, we got the same ‘random’ result. Moving it to main() seeds just once, giving better randomness.
2. **Near-Full Board Looping** — Early on, when the board was nearly full, the AI could waste time looping over filled cells. We fixed that by breaking out as soon as a valid cell is found.
3. **Redundant AI Win Checks** — Our first AI version repeated the PvP win logic inside AI code. We refactored to reuse the exact same win/draw detection used in Player vs Player mode.”

**Slide 10 – Conclusion (~0:30)**

“So to wrap it up — TicTacX is a small game, but it’s built with the same care you’d expect in a much bigger project.  
It’s modular, portable, stable, and easy to extend.

And of course, the best part — it’s actually fun to play.”