**Design Considerations & Alternatives**

**A. Modular Class Design**

* The game logic is split across specialized classes: Tank, Shell, Board, Tile, etc.
* This improves maintainability and testability.

**Alternatives considered:**

* Combining shell and tank logic into GameManager – but this would violate single responsibility principle.

**B. Position Wrapping Logic**

* Wrapping is handled inside Board::wrapPosition(), ensuring consistency for both tank and shell movement.
* Ensures tanks/shells reappear on the opposite side without duplicating logic.

**Alternative:** Let tanks handle their own wrapping – rejected to prevent inconsistencies.

**C. Tile Occupancy Tracking**

* Tiles store their own TileType, used by collision logic.
* Collisions checked via checkCollisions() using tile state.

**Alternative:** Maintain separate entity maps – rejected for simplicity and performance.

**. Testing Approach**

**A. Manual Test Boards**

We designed custom .txt files with specific tank placements and objects to test:

* Tank movement and wraparound
* Shell collisions (shell-shell, shell-tank)
* Wall durability (2 hits)
* Tank on mine
* Backward movement

Each board was crafted to isolate one mechanic at a time.

**B. Test Algorithms**

* Simple algorithm scripts (TestAlgorithm) return deterministic actions based on step count.
* Ensures reproducibility for step-by-step validation.

**C. Output Logging**

* recordAction() logs every tank action with BAD STEP annotation if invalid
* writeLog() writes the full log + final result with reason to output.txt
* Each test can be auto-verified by checking this output