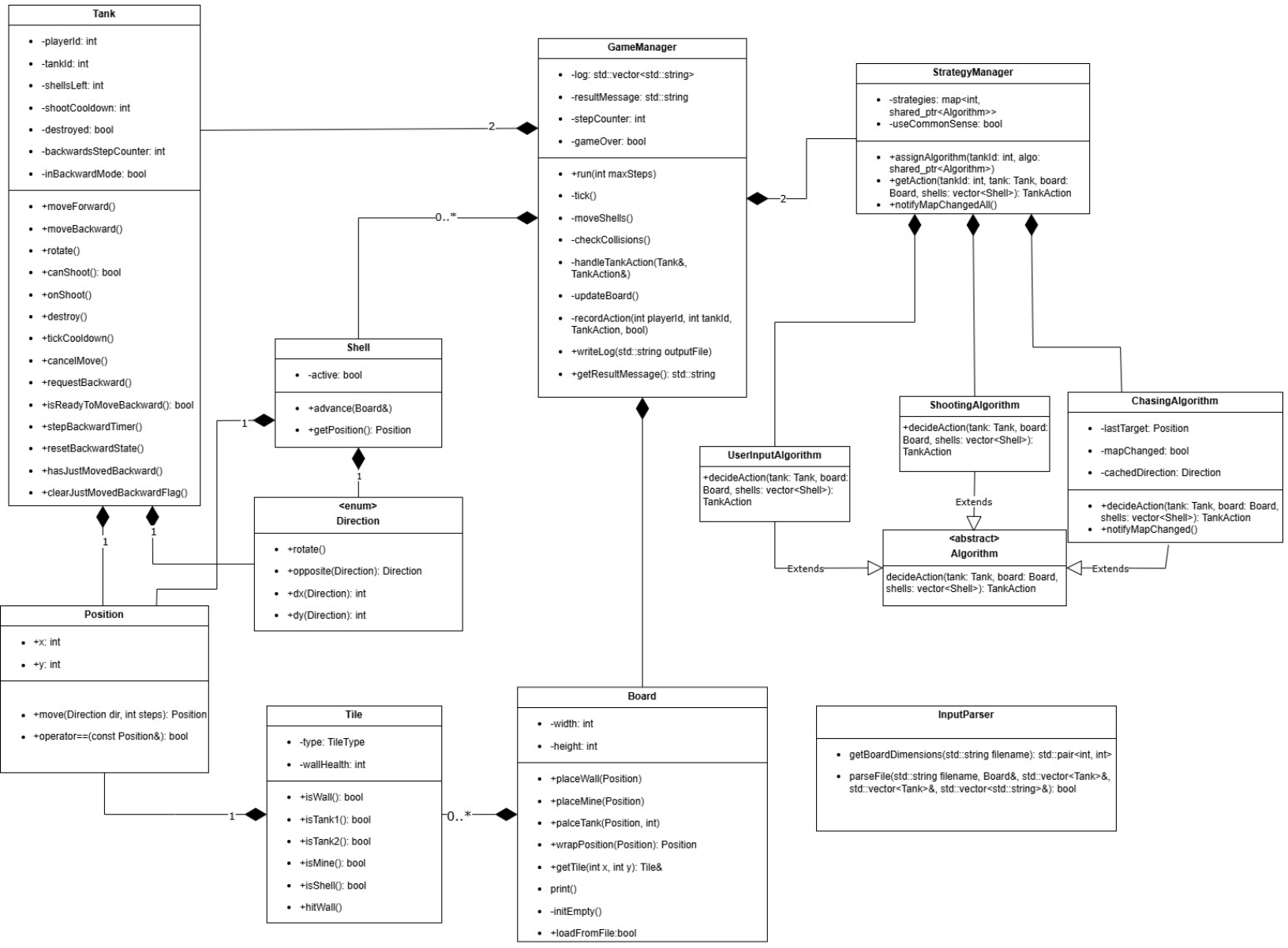
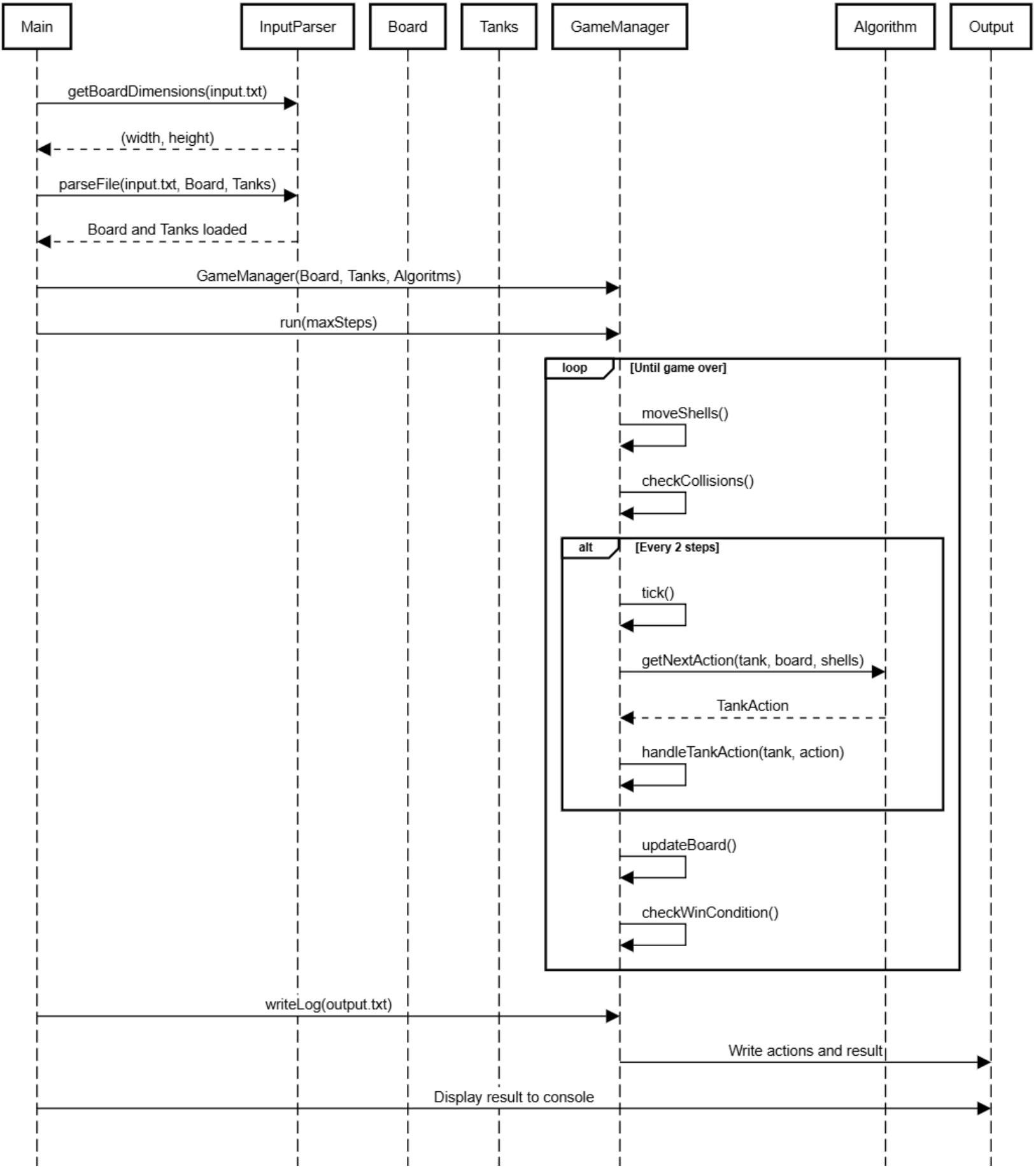
High Level Design

Advanced Coding Topics HW 1



Tank Game - Main Flow Sequence



Explanations of design considerations and alternatives:

The project follows a modular structure with clear responsibility division: Board and Tile manage the world state, GameManager handles game logic, and InputParser processes input files. Tanks and shells are managed independently by GameManager, avoiding tight coupling with tiles and enabling flexible movement.

Backward movement is managed internally by each Tank through counters and flags, providing a delay for initial backward steps but allowing immediate subsequent moves. Managing backward logic externally was rejected to keep tank behavior self-contained.

Collision detection occurs after movement to separate concerns. Immediate collision checking was rejected for simplicity and to prevent nested logic complexity.

The project defines a base Algorithm class for AI behavior, with derived strategies like UserInputAlgorithm, ShootingAlgorithm, and ChasingAlgorithm. We initially experimented with a limited two-step BFS for pathfinding, but due to its inaccuracy, we adopted a full BFS that caches only the initial direction toward the enemy. Recalculation occurs only when the map changes (enemy moves or walls break), balancing efficiency and intelligence.

A CommonSense utility layer was introduced to consistently enforce survival heuristics across all strategies, such as evading incoming shells and preferring movement over risky standing stillCommonSense. This design prevents code duplication across AI algorithms and ensures basic survival instincts without each algorithm reimplementing defensive checks.

InputParser was designed to recover from minor errors (extra tanks, wrong characters) by logging issues into input_errors.txt and continuing execution when possible, enhancing robustness.

Future-proofing was achieved by managing multiple tanks through vectors (player1Tanks and player2Tanks), allowing easy expansion to team battles or variable tank counts without structural changes. This will be helpful in upcoming assignments, as stated by the faculty.

Explanations of testing approach:

Our testing approach began by verifying the correctness of basic functionalities such as tank movement, rotation, collision with walls, and interaction with mines. We did this using simple single-action algorithms and maps, accompanied by a printed visualization of the map.

Once these fundamental actions worked reliably, we moved on to testing more complex and obscure edge cases, such as simultaneous collisions between multiple objects (e.g., three shells colliding at the same position). At this point, we had created a user-input “algorithm” that allowed us to test out anything we could think of.

After ensuring the core mechanics behaved as intended even under extreme scenarios, we focused on testing the behavior of the different algorithms. For each algorithm, we designed specific board setups with a desired outcome and checked whether the tank's behavior matched the expectation.

We also tested the “common sense” function we pass our algorithm’s decisions through. We did so by using it with our user input capability to see if it would prevent us from deadly situations and intervened when it thought we could shoot the enemy.

This methodical progression from basic features to edge cases and then to strategic behavior allowed us to build confidence in the robustness and correctness of the system.