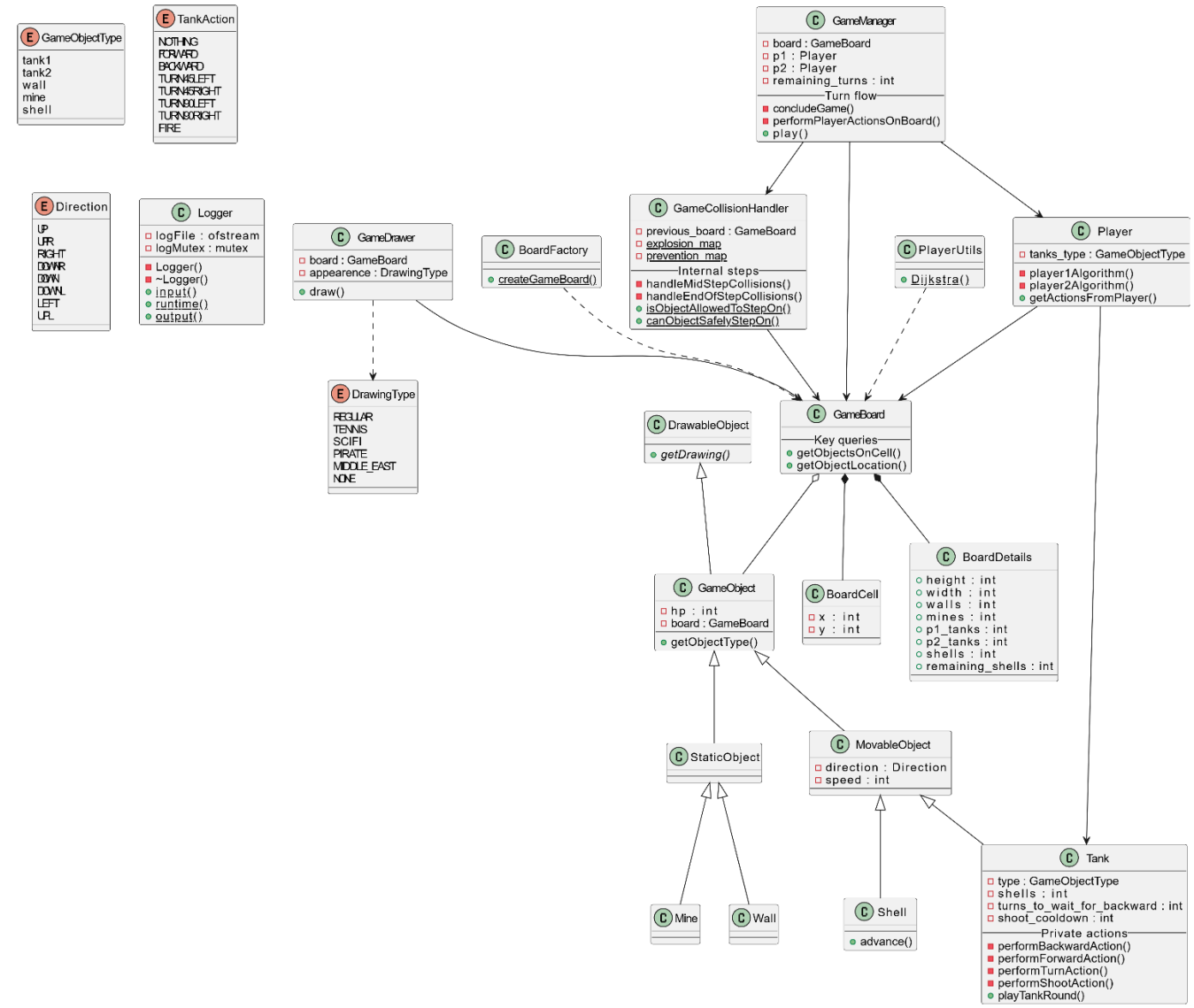
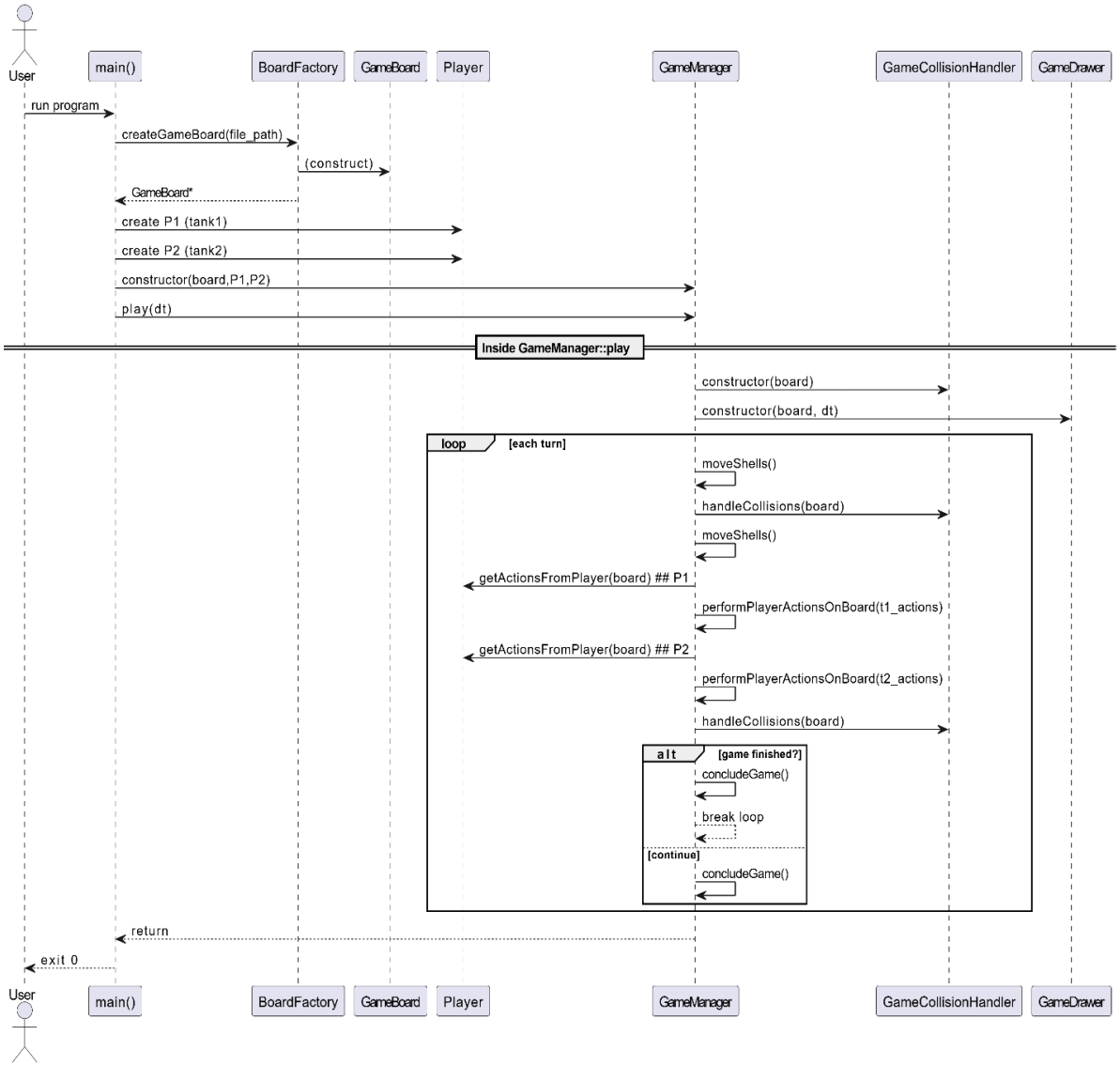
HLD Document

Class Design UML



Main Flow UML



Design Explanations

- Separation of responsibilities- GameManager orchestrates turns, while Tank objects encapsulate their own action logic. This complies with the assignment rule that algorithms decide but never mutate game state; only the manager applies changes.
- Two-level object hierarchy- `GameObject → {StaticObject, MovableObject}` reduces duplication (shared HP/direction/speed) and leverages polymorphism for `getDrawing()` and collision policies.
- Data-driven collision handling- GameCollisionHandler uses two static maps (`explosion_map`, `prevention_map`) instead of hardcoded `switch` blocks, simplifying addition of new object types.
- Factory for reproducible boards- BoardFactory decouples I/O parsing from gameplay logic, enabling deterministic unit & integration tests from text fixtures.
- Logger singleton- centralised logging avoids scattered `stdout` and supports thread-safe writes via an internal mutex.

Alternatives considered:

- Entity Component System (ECS)- compose behaviour instead of inheritance. higher flexibility but excessive boilerplate for a small game.
- Observer pattern- for rendering, let GameDrawer subscribe to board-update events instead of repeatedly polling the board. This would further decouple rendering from game logic, but it would also require adding an event-dispatch infrastructure.
- A* search instead of Dijkstra in `PlayerUtils::Dijkstra()` - faster in practice, but plain Dijkstra met the performance requirements and is easier to reason about.

Testing Approach Explanation

- Unit tests- focus on isolated logic such as BoardCell operators, `Tank::playTankRound` validation rules, and collision map lookups.
- Deterministic integration tests- each text fixture processed by BoardFactory creates a board with a known layout; test cases run multiple full turns and compare the resulting board snapshot to an expected baseline.
- Scenario coverage- fixtures cover corner cases; simultaneous shell collisions, invalid tank moves, mine detonation chains, and game-end conditions.
- Logging validation- output log files are parsed to ensure illegal moves are flagged correctly and winner/tie messages are produced.