High Level design and diagrams

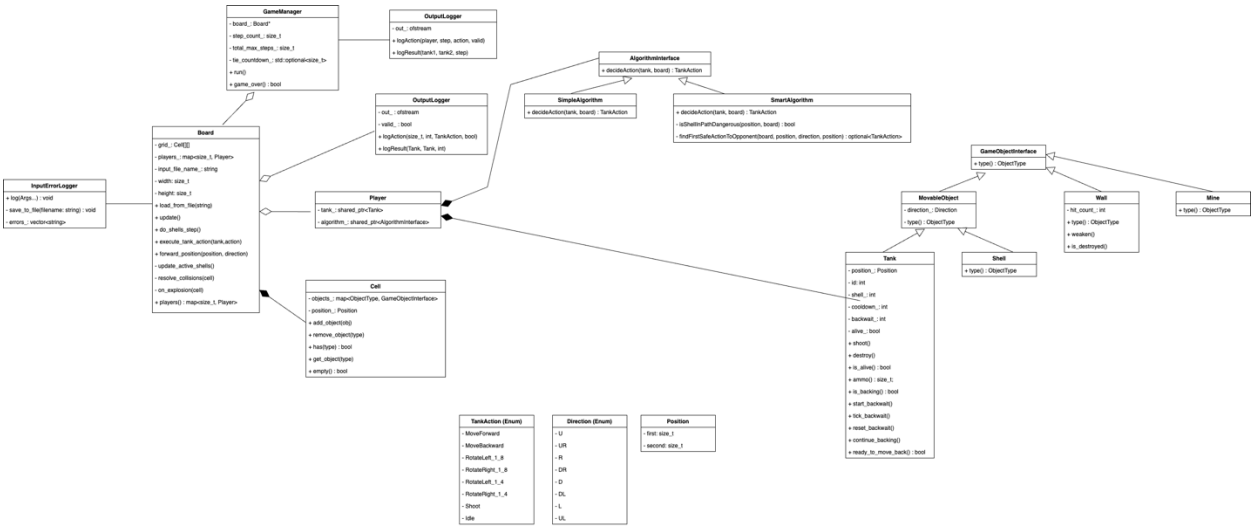
Explanations of design considerations and alternatives

The design of the Tanks Game project prioritizes modularity, readability, and future extensibility. Core components such as Tank, Wall, Mine, and Shell all inherit from a common GameObjectInterface, allowing the Board class to manage different object types uniformly. Tanks and shells are distinguished as movable entities through a MovableObject intermediate class. Player behavior is separated from the game state via an AlgorithmInterface, enabling the easy addition of new strategies without modifying the core game logic. Smart pointers are used throughout the project to ensure safe memory management without manual deallocation. Alternatives considered included using raw pointers or a centralized object manager, but smart pointers were chosen for their simplicity and safety. Additionally, by keeping the algorithms detached from the Board, the system can later support more complex behaviors or multiple tanks per player with minimal refactoring.

Explanations of testing approach

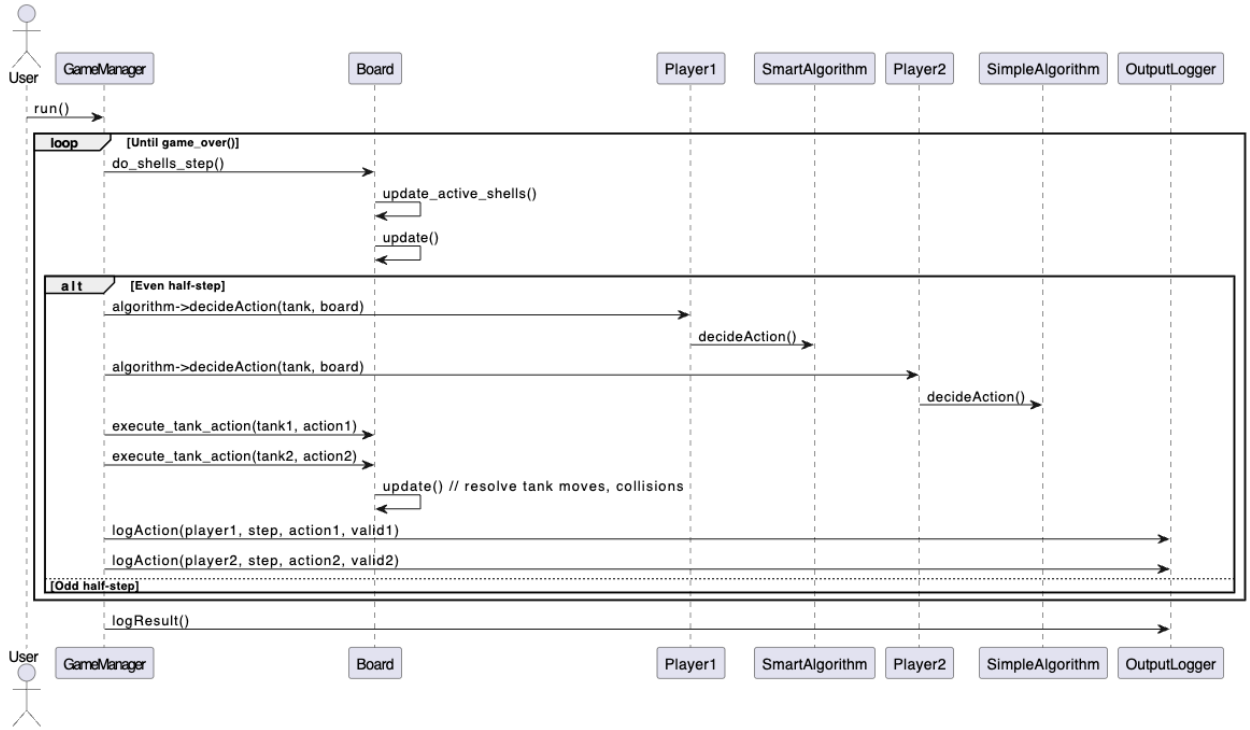
The project includes both automatic and manual testing strategies. For automated testing, the Google Test (gtest) framework was integrated into the CMake build system. Unit tests cover key classes such as Board, and Tank, ensuring that game actions behave as expected under various scenarios. Special attention was given to edge cases like collision handling, shell movements, and tank destruction. In addition to gtest, manual testing was conducted by running full games with different input maps, verifying visual board output, log files, and the final game results. Manual validation was also used to catch integration issues between components, such as timing mismatches between shell movement and tank actions. This combined approach provides confidence in both the correctness of individual modules and the overall game behavior.

Class Diagram:



This class diagram describes the object-oriented design of the Tanks Game project.
At the core is the Board class, which manages a two-dimensional grid of Cell objects, a list of active shells, and the players. Each Player owns a Tank and an AlgorithmInterface, allowing different AI behaviors.
Game objects such as Wall, Mine, and MovableObject inherit from GameObjectInterface, ensuring a unified interface for all entities on the board. Tank and Shell are special movable objects that inherit from MovableObject.
The GameManager class controls the main game flow, interacting with the board and logging outputs using OutputLogger.
In addition, there are supporting types like Position, Direction, and TankAction.

Sequence Diagram:



This sequence diagram describes the main game loop in the Tanks Game project.
The user starts the game by invoking GameManager::run().
The game then enters a loop that continues until a game-over condition is met.
In each iteration, the game first handles shell movements by calling Board::do_shells_step(), which updates all active shells.
At every even half-step, meaning every full step, the GameManager requests an action from each player's algorithm, SmartAlgorithm for Player 1 and SimpleAlgorithm for Player 2.
The chosen actions are then executed on the board via Board::execute_tank_action(), and the board resolves tank moves and collisions.
Each action's success or failure is logged using the OutputLogger.
On odd half-steps, only shells continue moving without tank actions.
At the end of the game, the final result, whether a win, loss, or tie, is recorded in the output log.