Chapter 1

Example game

As an example to demonstrate the

1.1 Battleship – Match

For the demonstration

The game has two phases – the setup of the battlefield and the shooting.

1.1.1 Design

As the shooting part of the game is

1.1.2 Game API

The api of the game is fairly simple, mostly consists of model classes that help describe the game model in a type–safe manner. For a more complex game the creation of more utilities would make sense as to make bot development easier.

Domain classes

These include any models that are used in the engine–bot communication or may be useful for bot development.

`Position` A simple $(x; y)$ coordinate pair, also lets one check whether it is a valid position on the game field. *(data class)*

`Direction` The orientation of a ship, may be vertical or horizontal. *(enumeration)*

`Ship` Describes the properties of a ship, including its (start) position, direction and size. *(data class)*

`ShootResult` the result of an shot, may be a miss, a hit or a sink. *(enumeration)*

`TargetMap` the view of the opponent's field. Shows all the tried positions as shoot results. When a ship is sunk, its bordering water is marked missed, and the ship itself as sunk. Each position can be queried individually and it also contains a helper method to list all non–tried targets.

Bot interface

The bot interface (`BotInterface`) is based on the design of the communication between the engine and the bots, therefore it has two abstract methods: `setupField` and `nextShoot`.

`setupField` takes the sizes of the ships to be placed as its parameter and must return a valid list of ships. It is called at the beginning of the game once.

`nextShoot` is called on each turn and is used to determine the target of the next shot. Its arguments are the current value of the bot's target map, the previous target of the bot, and the result of the previous shot. The latter two are `null` on the first turn of the bot. The method must return a valid, not yet tried target to shoot at.

BattleshipBot.java

```java
public interface BattleshipBot extends BotInterface {

    List<Ship> setupField(final List<Integer> shipSizes);

    Position nextShoot(
        final TargetMap targetMap,
        final Position previousTarget,
        final ShootResult previousResult
    );
}
```

Utility

As this game is fairly simple, I have create a simple utility class for shared usage – `ShipUtil`. This helper has a simple public method, `randomSetup`, which places all the ships at random – but valid – positions at random orientations. I have found this rather practical, as the placement of the ships – if not following some elaborate scheme – is often done randomly by the players and is a non–trivial task (due to the rules of ship placement).

1.1.3 Engine

The engine module contains the actual game engine class, as well as other resources that class uses, such as `ShipMap` and `ShipState` which mutably hold information of the state of a player's field.

The game engine class' structure is quite typical. For per–bot data keeping it defines a static inner class to hold the field of the bot with its last target and the result of that shot (if any). For each bot an instance of this gets created on the duel game engine's bot initialization phase, based on given bot's placement of its ships.

After that, the turn–based shooting starts. A turn here – by the duel game engine's definition – represents a set of operations that take place while one of the bots is working. This may include multiple shots, until a it finally misses or wins the game by sinking all enemy ships. After each shot attempt the engine first validates the target by checking it is a not yet tried position on the board. Based on the validation result the engine either declares the game lost due to an error, or actually processes the changes which results in a shot result. The bot's previous shot data is always updated to the current target and result, then based on the shot result some action may follow:

- If the shot was a miss, the bot's turn is over.
- If the shot was a hit, the turn continues with a next shot.
- If the shot sank a ship, the engine checks if all ships are sunk. If that is the case, the game is won by the current bot, otherwise the turn continues as with simple hits.

1.1.4 Random bot

I have found it often useful when implementing a game to first create a