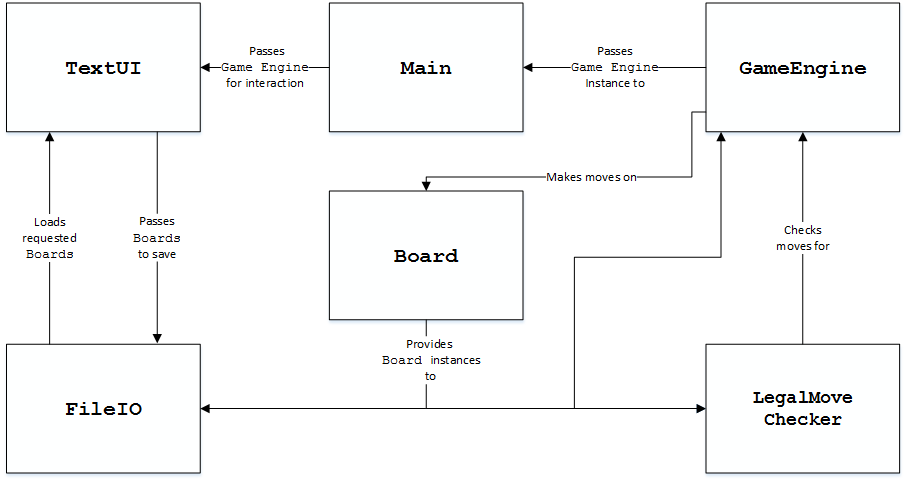
**3.3 Game Engine Design & Implementation**

Upon having established the requirements the first development step was the representation of a board and being able to make exclusively legal moves on it. Further, for the sake of easier interaction, testing and future development, a board loading/saving mechanism and a command-line based UI were also included. This lead to the following basic interaction structure:



*Figure X –* Basic program class structure

Notably, each of these was implemented as a Java class, designed to function as fairly independent components. Their respective details will be discussed henceforth.

**Main**

This class mainly serves as a runner and was incorporated as it was considered that some initial settings may have to be set prior to launching the UI. It for example, allows one to choose between the TextUI and later developed GUI. Additional settings were considered, but were added elsewhere later on.

**Board**

The Board class holds the main representation of the board, which is essentially a 2D integer array (type was later adjusted for memory reasons). As each intersection could either be empty or hold a black or white stone, these states could effectively be represented numerically as 0, 1 and 2. Here is an example of this representation:

|  |  |
| --- | --- |
|  | 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0  0 2 2 2 2 2 2 2 0  0 2 1 1 1 1 1 2 0  0 2 1 0 0 0 1 2 0 |

*Figure X –* Board representation

Notably, a special empty space type (set to 3) was added later on to allow AIs to better determine relevant search areas (See *Section 3.5 – Search Scope*). The major functions of the board are the setting/getting of intersections, testing for board equality and producing deep copies (clones) upon request.

**FileIO**

The FileIO is responsible for reading/writing board states for later usage/testing. Reading in a board state involves verifying its integrity and producing the resulting Board objects, whereas writing converts Boards into writable strings. The file representation of a board does not use integers, but rather characters for the benefit of manual editing (which was necessary prior to the GUI). Accordingly, black, white and empty were represented as ‘b’, ‘w’, ‘.’. Here the previous example:

|  |  |
| --- | --- |
| 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0  0 2 2 2 2 2 2 2 0  0 2 1 1 1 1 1 2 0  0 2 1 0 0 0 1 2 0 | 9 9  .........  .........  .........  .........  .........  .........  .wwwwwww.  .wbbbbbw.  .wb...bw. |

*Figure X –* Board file representation

The two initial numbers at the start denote the dimensions of the board. This mechanism was further enhanced as the program grew, having files hold further details such as AI search spaces (denoted by ‘-‘) as well as game objectives (E.g “White to kill 2 7”) that instruct the AI as to what group is relevant. Due to the increasing growth, the translation/integrity verification was later moved to a separate translating class.

**GameEngine**

The GameEngine was designed to be the heart of the program, holding the current Board representation and allowing players to make legal moves on it. In order to check the legality it holds an instance of the LegalMoveChecker that it asks to analyse each move attempted and will only modify the board when said test returns true. Other functions entail undoing moves, restarting the board and other user-oriented features. A further addition that was made later in the project is the addition of AI play and representation.

**LegalMoveChecker**