**Functional Programming – Final assignment**

**Comparison Haskell and Java implementation**

Both the Haskell and Java implementation implement the same algorithm (as described in the README’s of both implementations). However, the amount of code differs greatly. The Haskell implementation uses only 127 lines of code[[1]](#footnote-1). This is a lot less compared to the Java implementation: this includes 6 classes (excluding test-classes) of which the Game- and Board-class already has 123 and 163 lines of code[[2]](#footnote-2) respectively.

A difference in the amount of code could be explained by the fact that Haskell is declarative, which makes the amount of code needed to define a calculation a lot smaller. Also, in Java you have to declare a class including the constructor and getters and setters, and to use it you have to set it up. Therefore, more lines of code are needed.

During the writing of the code, it was difficult that it was not possible to (easily) test the written functions in Haskell. It had to be done manually. Therefore, if you test the result of a function stays the same if you alter the code, you had to type in the functions over and over. While on the other hand, it is possible to set up tests in Java to test the outcome of a function without constantly typing the test. However, this takes time to set up.

During the implementation of the algorithm in Java, I ran into the problem that the you have to create a (correct) new copy of the board each time you want to save this state in the tree because it otherwise overwrites the previous states of the board.

**Possible improvements of the algorithm**

The current algorithm takes the first bone on the list and tries to place it on the board. It does so until the board is full (all bones are placed) or there are no pairs left on the board. However, this is a brute force solution: there are many ways to optimize the algorithm. For example, when a new move is made there could be first looked if there is a bone that could be placed at only one pair and that bone is placed. Or, if a field has only one neighbour, there could be looked if a bone corresponding to that pair is still available and if so, the bone is placed and if not, the calculations on that branch are stopped since no solution will ever be achieved. Or, if a field is not part of a pair, it could be concluded that this board will also no longer have a solution and the calculation on that branch could be stopped as well.

1. including the print function, excluding the comments and declaration of four boards [↑](#footnote-ref-1)
2. excluding comments [↑](#footnote-ref-2)