Functional programming

Final assignment – Janine Klein Rot

Algorithm

To solve the domino board, the following algorithm is used. In the beginning, a list of all available dominos and a board consisting of the pips of the given input at an index are created. For each domino in the list of available dominos, all pairs of indices on the board are found where that domino can be placed. A list results with the pairs of indices for each of the available dominos at which the domino can be placed on the board. This list is checked on three conditions.

- This list is checked for dominos with no pair of indices to place the domino on the board. If there are one or more of those unplaceable dominos, the board is unsolvable and it is not added to the list of solutions or used for further solving.
- If the list of pairs of indices did not contain unplaceable dominos, the same list of pairs of indices is checked for dominos that can be placed at exactly one pair on indices. For all of these dominos in the list, the domino is placed on the board and the domino is removed from the list of available dominos. For the resulting board and list of available dominos, again the pairs of indices where each domino can be placed on the board are found. Thereafter the list is checked again for dominos that are unplaceable and if needed, the list is checked again for dominos that can be placed on one pair of indices. This continues until the board is solved or the board is unsolvable.
- If the board is not solved and the list of pairs of indices did not contain dominos that can be placed on one pair of indices, the same list of pairs of indices is checked for dominos that can be placed in more than one pair of indices. The first domino of this list is taken and for each of the pair of indices where this domino can be placed, a copy of the current board is made. On each copy of the board, the domino is placed on one of the pairs of indices. The placed domino is removed from the list of available dominos. At this moment, there is a list of boards that could all still result in a solution, with for each board the corresponding list of available dominos. For each of those boards, the list of pairs of indices where each of the dominos in the resulting list of dominos can be placed is created again. The list is checked as was done in the beginning, by first checking for dominos that are unplaceable, thereafter, if needed, for dominos that can be placed on one pair of indices and thereafter, if needed, for dominos that can be placed in more than one pair of indices. This continues until the board is solved or the board is unsolvable.

This loop of checking the tree conditions on the list of available dominos for the board is repeated until all boards are either unsolvable or solved. In this way, for an input board, either no solutions, one solution or more solutions can be found.

Running the applications

Both applications can be found on my GitHub repository, https://github.com/JKleinRot/NedapUniversity_TheDominoEffect. Below the instructions to running the applications on your pc are described, assuming Java and GHCI are installed on your pc.

Running the Java application

- Save the jar file 'TheDominoEffect.jar', that can be found in the folder 'TheDominoEffectJava', on your pc
- Open a terminal and go to the folder where you saved the jar file
- Enter 'java -cp TheDominoEffect.jar main.TheDominoEffect' to start the application

Running the Haskell application

- Save the Haskell file 'TheDominoEffect.hs' that can be found in the folder 'TheDominoEffectHaskell', on your pc
- Open a terminal and enter 'ghci'
- Enter ':l <path to the folder where you saved the Haskell file>'
- Enter 'TheDominoEffect' to start the application

Comparison

Throughout writing the application in both Java and Haskell, differences between both languages became clearer. Below some of the differences between both languages are discussed.

Clarity of code

For me, the Java application is clearer than the Haskell application. If you look for a function that modifies the board, in Java you start to look in the Board class. In the Haskell application you have to look through the complete code to find the function you are looking for. After I got the Haskell application working, I grouped the functions better on what they do, which may have been good to do during the process. I did add a line above each method during the process stating what each function was supposed to do and the type annotation. In Java, while working on the code, I kept updating the Javadoc which for me is a clear way to have a description of your method and its inputs and output.

Testing

In Java, I started writing unit tests in the beginning, but since writing those tests takes a long time, the number of tests I wrote kept getting less. This is not ideal, since during the process you work more and more on the algorithm and these methods are more important to test then for example the constructor and getters for the Position class. The classes contained some private classes to split the actions of a larger public method. However, these private methods cannot be testing separately, since you cannot call them. To find the problem in a larger test of the public method earlier, I sometimes wrote a separate test for a private method and made the private method temporarily public to find out which of the private methods was not doing what is was supposed to do. For the Haskell application, testing each function separately was easier. You could call each method from the GHCI and you could see if the outcome was what you expected. This became a bit harder if you wanted to test methods with a lot of inputs that all need to be correct for testing the function. The unit tests in Java give you the opportunity to once in a while run all tests and see if something fails, this option is not present in Haskell since you can only test your functions if you call them with appropriate inputs. When the total application fails, unit tests can sometimes help you find the place where things go wrong, where for the Haskell application you do not know in what functions things went wrong.

Ease of writing

In Haskell, the lines in for example pattern matching or multiple functions in a do block should be perfectly aligned. If this is not the case, you find out when the interpreter complains about the alignment of the code. In Java, alignment is not a big deal, but if you forget a closing bracket, the IDE gives a compiler error indicating that the application cannot be compiled and ran without the correct placement of brackets. Before I started to implement the algorithm, in Java I created classes that represented the board, a position on the board and a domino, with some having inner classes representing part of those classes, and its getters for its attributes. This took more time than representing the board, a position and a domino in Haskell, where I made a type declaration and created some initial values.