-- Report completed by:

-- Name: Sukai Huang

-- UID: u6492211

-- Tutor: Debashish Chakraborty @u4610248

-- Lab Time: Thursday 3pm

There are four functions I need to complete, the first two functions shipLength and coordInBound are quite simple to write. The last two function are more essential because the purpose of this Batteship Haskell file is to first of all generate a valid Ship board by using placeShip function, and then play the game using transitionState function.

When I was building the transitionState, I encountered a technical issue. My algorithms of transitionState function is:

1. Check the input Coordinate data, if the coordinate is out of the bound, return the original State. If the Coordinate data is valid, then move to the next stage.
2. Check the condition of input State data, if the condition is Playing, then move to the next stage, otherwise output the original State

3. Divide into two conditions: one is that the coordinate has a ship body, the other one is that the coordinate does not contain a ship body. Then continue to analyse the changes in board, condition as well as the move counter.

The problem I met is that in building the condition part, I need to count the number of Hit in the Ships matrix so as to determine the game condition. I combine the Ships matrix into one big list with all the Cell element and then build a list comprehension code.

```
condition = if length([allboardelement | allboardelement <- ((board state) !! 0 ++
(board state) !! 1 ++ (board state)
!! 2 ++ (board state) !! 3 ++ (board state) !! 4 ++ (board state) !! 5 ++ (board
state) !! 6 ++ (board state) !! 7 ++
(board state) !! 8 ++ (board state) !! 9), allboardelement == Hit]) >= 16 then (if
numMoves state < 20 then Won else Lost)
else (if numMoves state < 20 then Playing else Lost),
```

This code is in line 210 to 213, but there is an issue in this code. Since initially data Cell only derives Show and Read, allboardelement == Hit will return error! Therefore I thoroughly read the requirement of the assignment and find out that there is no restriction on adding deriving unit in data type definition. So I add ‘deriving (Eq)’ in Cell data definition so that I can complete the list comprehension code in line 210.

However, after I learned higher order function lesson in week 5, I realise that I can simply use ‘filter’ built-in function to complete the task. In that case we do not need to add deriving (Eq) any more.

I think I have considered many cases during the assignment, including the case when the input of the coordinate element is negative number. In this assignment I do not need to put many efforts in considering the exceptional case due to the presence of validPlacement function. However, one assumption I can think of is the initial input of the Ship matrix as well as the Board matrix are of 9*9 size. Also the input should match the type indicated in the function signature. I run the program and I found out that if my input is not a coordinate. The program will stop and exit.

One thing I need to mention is that if I do the assignment again I will definitely use higher order function to simplify my steps. It is good to use higher order function especially when many data in the assignment contains lists and matrix. Using higher order function can also make my code not so confusing when other people read. For example, in the file I write 4 sub function for each direction in placeShip function to update the Ships matrix. Due to the situation that I need to use and reuse an ‘updateList’ function again and again, I write a recursive function to do the task. But in fact it is very confusing and complicated! Furthermore personally I think writing recursive function is more challenging than writing higher order function. Thus it has high chances leading to mistakes and errors. (Those codes are in line 178 to 199)

```
updatesimpledown :: Integer -> Ships -> Integer -> Integer  -> Ships
updatesimpledown lengthofship shipsmap coordx coordy
    | lengthofship == 1 = updateList shipsmap (fromIntegral coordy) (updateList
(shipsmap !! (fromIntegral coordy)) (fromIntegral coordx) True)
    | lengthofship > 1  =  updatesimpledown 1 (updatesimpledown (lengthofship - 1)
shipsmap coordx coordy) coordx (coordy + lengthofship - 1)
    | otherwise = error "length is bigger than 1"
```

Higher order function is much simpler! Thus if I start to write code after week 5. I will definitely use ‘foldl’ function to simply complete the task of

updating the Ships matrix. There might be an improvement of algorithms of the code too next time.

Using the validPlacement function at the first order in the placeShip function made encoding much easier. Also since Katya always says “Think before you write”, I made a simple logic flow before I started to write codes. It indeed helps me encoding as writing logic flow helps me to avoid repetition or some redundant steps.

The biggest learning point for this assignment is that it re-highlight that Haskell is a pure functional programming language which is free from side effect. For example when I count the number of move counter in transitionState function I need to be aware of the function input and usually all the guards and conditional expressions are using the input value! Well in other words, we need to consider ‘Won’ condition when the input of the counter number of Hit reached 16 instead of 17, and consider ‘Lost’ condition when input of the number of moves reached 19 instead of 20.

To make my code clear, I write some comments in the file to show steps as well as the purpose of the sub-functions. I also write some doctest comments to test my program. They all passed. I also planned to write property function so as to use quickCheck test. Unfortunately when I import Test.QuickCheck and compile the file in Intellij IDE, it gives me warning that the import of Test.QuickCheck is redundant. Since the requirement of the ‘High Distinction’ is that there is no warning when compiling the file, I just leave the property function in my file and remove the QuickCheck import command in my file. Nevertheless I know the steps to run quickcheck test.

I use ‘:set +s’ command to check the time of the main function by entering some random test variables. The time output is about 0.02 sec. I think it is quick fast.

Factors which affects the speed of running a program:

1. Code itself. If the code is concise then it will speed up the running time.
2. Hardware capability. The speed of CPU

Factors in the above column is what I believe the main factors affecting the speed of a program. Through this assignment task I realise that the algorithms of a code is really important. The order of the code will significantly determine the speed of the program.