COMP3000 Programming Languages

Assignment 1

Chaz Lambrechtsen

45426317

Table of contents

1. Introduction ---------- 1
    a. Aim ---------- 1
2. Implementation ---------- 1
    a. Helper functions ---------- 2
    b. Final solution ---------- 2
3. Testing ---------- 3
    a. Helper functions ---------- 3
    b. Final solution ---------- 3

Introduction

This report will include implementation details of the code used to solve the given assignment, as well as additional testing after the implementation of each function. Testing will include descriptions of different tests and what inputs these tests for.

Aim

The aim of this assignment is to complete the given skeleton code bundle in order to solve the "Snake" puzzle game. Solving the snake puzzle game will align all sections to create a 3x3 cube.

Implementation

Implementing the goals of this assignment required research on functional programming and Scala's built in functions and how these functions are used to implement loops, and operations.

Helper functions

Initially it is best to solve smaller problems related to the larger problem. In the given skeleton code, we have a group of helper functions that are utilised in the other functions to solve the puzzle.

getIncrement() was the first function to be implemented. This function simply increments (or decrements) the corresponding value in a 3-tuple integer representing X, Y, Z on a 3D plain.

prependDir() required some further research on what functional characters to use in order to prepend to a list. The solution was simple.

For every dirList => prepend dir to dirList

```
solutions.map {
  case dirList => dir :: dirList
}
```

prependStart() is a similar problem to prependDir however requires more research in how Scala handles tuples from Java. A tuple can be created by enclosing the values in brackets.

For every dirList => create a 2Tuple with (xyz, dirList)

```
solutions.map {
  case dirList => (xyz, dirList)
}
```

getPossibleDirections() will return a list of possible directions that can be used to make a move in another function.

In the assignment specifications, the first and second moves must always be the same direction, which is XPos and YPos, respectively

Final solution (solveSnake, getSolutions, tryMove)

tryMove() tries to move in a specified direction given the current CubeState. When the CubeState is in StartState, this means it needs to make the first move, which is always XPos with XYZ values 1, 0, 0.

For OtherState we can match to the specified move input and find the correct direction to move using if statements as shown below.

```
move match {
  case XPos => if (move1._1 >= 3) None
               else if (segments(numMoves) == 2) Some(OtherState(segments, occupied + (move1) + (move2), move1, XPos, numMoves + 1))
               else Some(OtherState(segments, occupied + (move1), move1, XPos, numMoves + 1))
```

Move1 and move2 values are calculated using the helper function combineXYZ

```
val move1 = combineXYZ(end, getIncrement(move), 1)
val move2 = combineXYZ(end, getIncrement(move), 2)
```

This helper function will add m (input) to each direction and multiply the corresponding direction

```
def combineXYZ (dir1: (Int, Int, Int), dir2: (Int, Int, Int), m: Int): (Int, Int, Int) =
  (dir1._1+m.*(dir2._1),dir1._2+m.*(dir2._2),dir1._3+m.*(dir2._3))
```

Testing

Helper functions

getIncrement() – All 6 possible outputs are tested using 6 unique tests (For all axis X, Y , Z and both negative and positive for each axis)

prependDir() – Tests different input values and different input amounts

> I have created each test with different values to test different inputs, I have also added extra lists in another test and a test with larger lists in another as well as empty lists. These tests simulate different inputs that this function can handle.

prependStart() - Tests different input values and different input amounts

> Similarly, to prependDir, I have created each test with different values to test different inputs including larger lists, many lists and empty lists. Tests are similar to prependDir()

getPossibleDirections() – Testing based on different states.

> Using st3, st4 and st5 to test for expected output based on the state's direction and numMoves.

Final solution (solveSnake, getSolutions, tryMove)

tryMove() – Test result of different movement directions in different states

The initial test that only test for a single state (st0). I have included many more tests to ensure this function is working correctly with in different OtherState scenarios utilising the initially included states st1-st5 that have not otherwise been tested using tryMove.