CS2006 Haskell Project 2 - Gomoku

Matriculation Numbers: 160001362, 160016245, 160021429 (Group 14)

06/03/2018

Contents

1 Summary of Functionality

This practical specified the development of the board game Gomoku using the functional programming language Haskell.

The provided README file gives a detailed description of what the solution can do and instructions explaining how to configure settings through the command line as well as in game.

The following functionality has been implemented:

1.1 Basic Specification:

All requirements from the basic specification have been implemented. They are as follows:

1. **Implement the game mechanics in Board.hs -**
2. **Implement the drawWorld function in Draw.hs to display the current board state graphically -**
3. **Implement appropriate event handlers for inputs events -**
4. **Implement a move generator (in AI.hs) and an evaluation function (in Board.hs) to provide a computer opponent -** two AIs have been implemented, a random AI and a heuristic based AI. The random AI generates a random index refer to an element in a list of empty positions of the board and places a piece there. The heuristic AI uses the evaluation function in Board.hs to calculate its board score relative to its opponent for each possible move and chooses the highest scoring move.

1.2 Additional Requirements:

From the suggested additional requirements, all of the easy and medium requirements have been implemented, as have the listed hard requirements:

1.2.1 Easy

- **Easy Requirement 1 -**
- **Easy Requirement 2 -**
- **Easy Requirement 3 -**

1.2.2 Medium

- **Medium Requirement 1 -**
- **Medium Requirement 2 -**

- **Medium Requirement 3 -**
- **Medium Requirement 4 -**

1.2.3 Hard

- **Hard Requirement 2 -**

1.3 Further Features:

The following additional features have also been implemented:

- An iterator class, IteratorOfTwistedIntMatrix, has been implemented in twisted_int_matrix.py file which iterates over an instance of TwistedIntMatrix from top left to bottom right across the TwistedInt elements. As with the IteratorOfTwistedIntegers iterator, this iterator has a hasNext(), next() and __init__ functions.

2 Design and Implementation

- **Basic Specification - TwistedInt Data Structure (twisted_int.py) -**
- **Easy Requirements 1 and 2 (checker.py) -**
- **Easy Requirement 3 (twisted_integers.py) -**
- **Medium Requirement 1 (twisted_integer.py) -**
- **Medium Requirement 2 and 3 (twisted_integers.py) -**
- **Hard Requirement 1 (twisted_int_matrix.py) -**
- **Hard Requirement 2 -**

3 Evidence of Testing

The code contains basic example docTests for every function. The output of which follows on the next page [1]. The output of unit tests would have been displayed but as discussed in the Known Problems section, we could unfortunately not integrate them with the final implemenation. The results shown can also be found in results.txt.

[1] Terminal output would have been directly included within this report rather than as a screen shot but the characters disagree with latex formatting

```
$ python checker.py -v
...
1 items had no tests:
     __main__
7 items passed all tests:
   2 tests in __main__.getAllCombinations
   1 tests in __main__.isAssociativeAdd
   1 tests in __main__.isAssociativeMul
   1 tests in __main__.isCommutativeAdd
   1 tests in __main__.isCommutativeMul
   1 tests in __main__.isDistributive
   3 tests in __main__.mulEqualToOne
10 tests in 8 items.
10 passed and 0 failed.
Test passed.


$ python twisted_int.py -v
...
2 items had no tests:
    __main__
    __main__.TwistedInt
4 items passed all tests:
   6 tests in __main__.TwistedInt.__add__
   1 tests in __main__.TwistedInt.__init__
   6 tests in __main__.TwistedInt.__mul__
   3 tests in __main__.TwistedInt.__str__
16 tests in 6 items.
16 passed and 0 failed.
Test passed.


$ python twisted_integers.py -v
...
3 items had no tests:
    __main__
    __main__.IteratorOfTwistedIntegers
    __main__.TwistedIntegers
8 items passed all tests:
   3 tests in __main__.IteratorOfTwistedIntegers.__init__
   2 tests in __main__.IteratorOfTwistedIntegers.hasNext
   3 tests in __main__.IteratorOfTwistedIntegers.next
   1 tests in __main__.TwistedIntegers.__init__
   1 tests in __main__.TwistedIntegers.__str__
   2 tests in __main__.TwistedIntegers.size
   1 tests in __main__.findValAdd
   1 tests in __main__.findValMul
14 tests in 11 items.
14 passed and 0 failed.
Test passed.
```

```
$ python twisted_int_matrix.py -v
...
File "twisted_int_matrix.py", line 269, in __main__.getPossibleMatrices
Failed example:
    for m in list:
        print(m)
Expected:
    <0:2> <0:2>
    <0:2> <0:2>
Got:
    <0:2> <0:2>
    <0:2> <0:2>
    <0:2> <0:2>
    <0:2> <0:2>
4 items had no tests:
    __main__
    __main__.IteratorOfTwistedIntMatrix
    __main__.TwistedIntMatrix
    __main__.TwistedIntMatrix.__add__
12 items passed all tests:
   4 tests in __main__.IteratorOfTwistedIntMatrix.__init__
   4 tests in __main__.IteratorOfTwistedIntMatrix.hasNext
   4 tests in __main__.IteratorOfTwistedIntMatrix.next
   3 tests in __main__.TwistedIntMatrix.__init__
   8 tests in __main__.TwistedIntMatrix.__mul__
   3 tests in __main__.TwistedIntMatrix.__str__
   3 tests in __main__.TwistedIntMatrix.calcDotProduct
   4 tests in __main__.TwistedIntMatrix.getCol
   3 tests in __main__.TwistedIntMatrix.twistedIntAdd
   3 tests in __main__.TwistedIntMatrix.twistedIntMul
   6 tests in __main__.contains
   7 tests in __main__.equalMatrices
**********************************************************************
1 items had failures:
   1 of   4 in __main__.getPossibleMatrices
56 tests in 17 items.
55 passed and 1 failed.
***Test Failed*** 1 failures.
```

[2]

4 Known Problems

5 Problems Overcome

6 Summary of Provenance

- 160001362:
- 160016245:
- 160021429:

[2] The 1 failure is the due to repeated results being printed due to a bug with the contains function when passing in the same matrix multiple times. This is discussed in the Known Problems section.

- Unit Testing

7 Conclusion

In conclusion we have successfully implemented all of the basic specification, easy, medium and hard requirements without major faults. Overall this practical felt a lot more accessible than the previous practical due to the use of Python rather than Haskell. Given more time we would address the issue relating to the running of the unit tests and perhaps add more mathematical functions to the solution.