IMPLEMENTING TAKUZU IN HASKELL

Advanced Functional Programming

Emanuel Stöckli, Vahid Shirvani

December 16, 2014

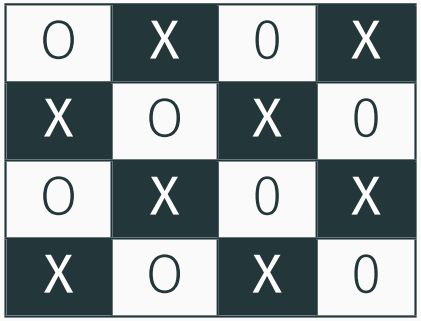
Uppsala University

THE PROBLEM

THE PROBLEM #1

THE PROBLEM #2

THE PROBLEM #3



THE PROBLEM #4

O	O	X	X
?	X	?	0
X	O	O	X
X	X	O	0

THE PROBLEM #5

O	O	X	X
?	?	?	0
X	O	O	X
X	X	O	0

OUR SOLUTION

TOWARD THE SOLUTION

- Iterative Approach

TOWARD THE SOLUTION

- Iterative Approach
- Application of a bunch of rules

TOWARD THE SOLUTION

- Iterative Approach
- Application of a bunch of rules
- Those rules might be game specific, the rest is general

TOWARD THE SOLUTION

- Iterative Approach
- Application of a bunch of rules
- Those rules might be game specific, the rest is general
- End condition? The board doesn’t change anymore

SOLUTION TO PROBLEM #1

SOLUTION TO PROBLEM #1

SOLUTION TO PROBLEM #2

SOLUTION TO PROBLEM #2

SOLUTION TO PROBLEM #3

O	X	0	X
?	X	O	?

SOLUTION TO PROBLEM #3

O	X	0	X
X	X	O	0

SOLUTION TO PROBLEM #4

O	O	X	X
?	X	?	0
X	O	O	X
X	X	O	0

SOLUTION TO PROBLEM #4

O	O	X	X
O	X	?	O
X	O	O	X
X	X	O	O

SOLUTION TO PROBLEM #5

O	O	X	X
?	?	?	0
X	O	O	X
X	X	O	0

SOLUTION TO PROBLEM #5

O	O	X	X
O	?	?	0
X	O	O	X
X	X	O	0

Design decisions and data representation

KEY DESIGN DECISIONS #1

Data Representation

1. Everything is in a list
2. Everything is row-based
3. Everything goes in one direction

KEY DESIGN DECISIONS #1

Data Representation

1. Everything is in a list
2. Everything is row-based
3. Everything goes in one direction

- Both directions?
  $\rightarrow$ Apply the rules twice

KEY DESIGN DECISIONS #1

Data Representation

1. Everything is in a list
2. Everything is row-based
3. Everything goes in one direction

- Both directions?
  $\rightarrow$ Apply the rules twice
- Columns?
  $\rightarrow$ Apply the rules to the transponded list

KEY DESIGN DECISIONS #1

Data Representation

1. Everything is in a list
2. Everything is row-based
3. Everything goes in one direction

- Both directions?
  → Apply the rules twice
- Columns?
  → Apply the rules to the transponded list
- 3x3 blocks in Sudoku?
  → Map them to rows

KEY DESIGN DECISIONS #1

Sudoku 3x3 block

	2						9	
3		1	9		6	5		2
			8		4			
	9						5	
5			2		3			6
	7						2	
			4		7			
8		2	5		1	7		3
	5						8	

KEY DESIGN DECISIONS #2

Clean Project Skeleton

1. **Main.hs**
   handles all the IO, calls the game specific functions and provides the output.
2. **AbstractGameLogic.hs**
   defines the board and a bunch of generic high-order functions that are used in both games, e.g. to apply specific functions to the board
3. **TicTacLogic.hs**
   provides Tazuku specific code
4. **SudokuLogic.hs**
   provides Sudoku specific code

KEY DESIGN DECISIONS #2

Using Cabal (Common Architecture for Building Applications and Libraries)

```
name:          Tic-Tac-Logic
author:        Emanuel Stöckli, Vahid Shirvani
version:       1.0
Build-Type:    Simple
cabal-version: >= 1.2

executable tic-tac-logic
   main-is:    Main.hs
   build-depends: base

Test-Suite test
   type:           exitcode-stdio-1.0
   main-is:        TicTacLogicTest.hs
   build-depends: base, tasty, tasty-hunit
```

KEY DESIGN DECISIONS #2

which leads to more cleanliness (everything is in a sandbox) and simplicity (install, build and run)

```
$ cabal sandbox init
$ cabal install --enable-tests

$ cabal build
$ cabal run

$ cabal build test
$ cabal test --show-details=always
```

KEY DESIGN DECISIONS #3

Unit Testing with HUnit and Tasty

- We wanted to ensure that changes on our code do not negatively affect properly working functionality that has already been implemented
- Especially because we had to implement a Sudoku solver in the second step. Therefore we often increased abstraction and used the same code for both games. No chance without tests.
- One test group with multiple tests for every rule
- 43 unit tests for Tukuzu and 3 for Sudoku

KEY DESIGN DECISIONS #3

```
  Avoiding triples 2
    Testing 1 Row with empty middle cell (X -1 X):                          OK
    Testing 1 Row with empty middle cell (X -1 X):                          OK
    Testing 1 Row with empty middle cell (X -1 X):                          OK
    Testing 1 Board with empty middle cell (X -1 X), ONLY 1 iteration:      OK
    Testing 1 Board with empty middle cell (X -1 X), ONLY 1 iteration:      OK
    Testing 1 Board with empty middle cell (X -1 X), ONLY 1 iteration:      OK
    Testing 1 Board with empty middle cell (X -1 X), with RECURSION:        OK
  Avoiding triples 3
    Testing 1 Row to see if it is in our interest:                          OK
    Testing 1 Row to see if it is in our interest:                          OK
    Testing 1 Row to find the index of element:                             OK
    Testing 1 Row to find the index of element:                             OK
    Testing 1 Row to replace an element:                                    OK
    Testing 1 Board to replace a row:                                       OK
    Testing 1 Row to find indices of an element occurrences:                OK
    Testing 1 Row to find indices of an element occurrences:                OK
    Testing 1 Row to find all possible combinations:                        OK
    Testing 1 several rows to find index of verified row:                   OK
    Testing 1 several rows to find index of verified row:                   OK
    Testing 1 Board that can be filled up, ONLY row-wise:                   OK
    Testing 1 Board that can be filled up:                                  OK
  Completing a row or a column
    Testing 1 Row with various empty cells (1,1,-1,-1):                     OK
    Testing 1 Row with various empty cells (1,1,-1,-1):                     OK
    Testing 1 Board that can be filled up (1,1,-1,-1), ONLY 1 iteration:    OK
  Avoiding row or column duplication
    Testing 1 Row against another row (1,-1), (1,0):                        OK
    Testing 1 Row against another row (1,-1), (1,0):                        OK
    Testing 1 Row against another rows (1,-1), ((1,0)):                     OK
    Testing 1 Row against another rows (1,-1), ((1,0)):                     OK
    Testing 1 Board to return complete rows ((1,-1), (1,0)):                OK
    Testing 1 Board to return complete rows ((1,-1), (1,0)):                OK
    Testing 1 Board that can be filled up ((1,-1), (1,0)), ONLY row-wise:   OK
    Testing 1 Board that can be filled up ((1,-1), (1,0)), ONLY row-wise:   OK
    Testing 1 Board that can be filled up ((1,-1), (1,0)):                  OK
  Advanced technique 1
    Testing 1 Board that can be filled up, ONLY row-wise:                   OK
    Testing 1 Board that can be filled up:                                  OK
  Advanced technique 2
    Testing 1 Board that can be filled up:                                  OK

All 43 tests passed (0.01s)
```

KEY DESIGN DECISIONS #3

Black-box system testing with shUnit2

- Ensure that our program in its entirety works as required
- How will the program be used? (see code snippet below)
- The shell testing framework shUnit2 fits those requirements
- We don't care about the implementation of Takuzu/Sudoku. It could be written in Erlang or in any other programming language.

```
$ ghc tic-tac-logic.hs
$ cat tic-tac.txt | ./tic-tac-logic
```

KEY DESIGN DECISIONS #3

How does the code look like?

```
#! /usr/bin/env sh

function test_Takuzu2x2 () {
   > result.txt ## clear old test results
   cat input.txt | ./tic-tac-logic > result.txt
   diff output.txt result.txt
   assertTrue '2x2 example from project description
       fails' $?
}

## Call and Run all Tests
. "shunit2"
```

KEY DESIGN DECISIONS #4

Complexity

- The performance efficiency was as important as the functionality itself
- Avoid functions that had quadratic complexity

KEY DESIGN DECISIONS #5

Single responsibility principle

- Many small functions
- Functions become black boxes
- Debugging and unit testing become easier

LIMITATIONS

Our solutions are limited to

- Takuzu puzzles that have one unique solution $\rightarrow$ no ambiguity
- Sudoku puzzles that can be solved by completing rows, columns and 3x3 sub boards iteratively as long as the game board changes

CONCLUSION

CONCLUSION

- We learned a lot about rule-based approaches, about testing, about Haskell and of course about those board games
- Both advanced rules were tricky to implement
- Experienced some technical issues (ended up installing Ubuntu for Cabal)
- Very rewarding once it works
- Successful pair programming
- Successful collaboration based on the useage of a GitHub repository and issues

Questions?