Programming Languages (Project 1)

Jeff Gyldenbrand (jegyl16)

November 13, 2017

Abstract

The goal of this project is ... to eat cake

Contents

1 The Kalaha game with parameters (n, m)

```haskell
module Kalaha where

import Data.List

type PitCount   = Int
type StoneCount = Int
data Kalaha     = Kalaha PitCount StoneCount deriving (Show, Read, Eq)

type KPos       = Int
type KState     = [Int]
type Player     = Bool
```

1.1 The function **startStateImpl**

```haskell
startStateImpl :: Kalaha -> KState
startStateImpl (Kalaha n m) = replicate n m ++ [0] ++ replicate n m ++ [0]
```

1.2 The function **movesImpl**

```haskell
movesImpl :: Kalaha -> Player -> KState -> [KPos]
movesImpl (Kalaha n m) p s
 | p == False = last(snd(splitAt(n+1) s))
 | p == True = last(fst(splitAt(n+1) s))
```

1.3 The function **valueImpl**

```haskell
valueImpl :: Kalaha -> KState -> Double
valueImpl (Kalaha n m) s = fromIntegral (vT - vF)
  where
    vT = last(fst(splitAt(n+1) s))
    vF = last(snd(splitAt(n+1) s))
```

1.4 The function **moveImpl**

```haskell
moveImpl :: Kalaha -> Player -> KState -> KPos -> (Player,KState)
moveImpl g p s xs = undefined
```

1.5 The function **showGameImpl**

```haskell
showGameImpl :: Kalaha -> KState -> String
showGameImpl g@(Kalaha n m) xs = undefined
```

2 Trees

```haskell
data Tree m v  = Node v [(m,Tree m v)] deriving (Eq, Show)
```

2.1 The function **takeTree**

```haskell
takeTree :: Int -> Tree m v -> Tree m v
takeTree = undefined
```

3 The Minimax algorithm

```haskell
data Game s m = Game {
    startState    :: s,
    showGame      :: s -> String,
    move          :: Player -> s -> m -> (Player,s),
    moves         :: Player -> s -> [m],
    value         :: Player -> s -> Double}

kalahaGame :: Kalaha -> Game KState KPos
kalahaGame k = Game {
    startState = startStateImpl k,
    showGame   = showGameImpl k,
    move       = moveImpl k,
    moves      = movesImpl k,
    value      = const (valueImpl k)}

startTree :: Game s m -> Player -> Tree m (Player,Double)
startTree g p = tree g (p, startState g)
```

3.1 The function **tree**

```haskell
tree      :: Game s m -> (Player, s) -> Tree m (Player, Double)
tree = undefined
```

3.2 The function **minimax**

```haskell
minimax   :: Tree m (Player, Double) -> (Maybe m, Double)
minimax = undefined
```

3.3 The function **minimaxAlphaBeta**

```haskell
type AlphaBeta = (Double,Double)

minimaxAlphaBeta :: AlphaBeta -> Tree m (Player, Double) -> (Maybe m, Double)
minimaxAlphaBeta = undefined
```

4 Testing and sample executions