Programming Paradigms
159.272
Assignment 2 Game Tree

Amjed Tahir

a.tahir@massey.ac.nz

Example: Hughes' game trees

- a game tree is a directed graph whos:
  - ***nodes*** are positions in a game and
  - ***edges*** are moves.
- The complete game tree for a game is the game tree starting at the initial position and containing all possible moves from each position

```
case class GameTree(val gameState: GameState, val
move: List[Move])
```

in our case each node in the tree contains a configuration (position) of the game and list of the configurations that can be reached in one move from the current one

This game tree was explained in Hughes paper - Why Functional Programming Matters (available on Stream!)

Example: Hughes' game trees

To build the tree:

- root node contains the initial value x,
- the list of root values for the subtrees is generated by f x
- we map (reptree f) over this list of values to recursively build the tree

```
def gameTree(player: Player, depth: Int, game: GameState):
GameTree = {
 def subGameTree(c: ColumnNum): Move = {
      Move(c, gameTree(otherPlayer(player),  depth-1,
dropPiece(game, c, pieceOf(player))))
    }
.....
 if (depth == 0)
          GameTree(game, List())
        else
          GameTree(game,
allViableColumns(game).map(subGameTree))
```

Example: Hughes' game trees

The initial configuration of the game first in GameState

The list of children (configuration of each move – (6*7=) 42 ➔ (42-1=) 41…..)

Moves takes a position and returns a list of positions I can get to from that position

```
def gameTree(player: Player, depth: Int, game: GameState):
GameTree = {
 def subGameTree(c: ColumnNum): Move = {
      Move(c, gameTree(otherPlayer(player),  depth-1,
dropPiece(game, c, pieceOf(player)))) }
.......
 if (depth == 0)
          GameTree(game, List()) \\no childs
        else
          GameTree(game,
allViableColumns(game).map(subGameTree)) \\this is a big and
infint gametree
```

- We want to produce a function that will give us the actual value of a given position, which will depend on all reachable positions from that position.
- assume computer wants to move to child node with max true value, opponent wants min true value

```
 // Find the move that maximises the minimum utility to a player,
assuming it is that player's turn to move.

  def maxmini(player: Player, tree: GameTree): ColumnNum = {
    tree match {

      case GameTree(x, List()) => throw new Exception("The AI was
       asked to make a move, but there are no moves possible. This
       cannot happen")

      case GameTree(x, moves) => moves.maxBy(m =>
       minimaxP(player, m.tree)).move //Rating each position (the position of
       max value to me
    }
  }
```

Example: Hughes' game trees

Evaluating a position.

```
def estimate(player: Player, game: GameState): Int = {
   if (fourInALine(pieceOf(player), game))
     100
   else if (fourInALine(pieceOf(otherPlayer(player)), game))
     -100
   else
     0
 }
```

What happens if the game tree generated is infinite?
Or just very big?

- The game tree is also explained in Hughes' paper (Why FP).
- See page 12-16
- The examples are in Haskell, but the logic is the same.