# Haskell: Monte Carlo Tree Search

- Responsabil: Mihnea Muraru
- Deadline soft: **30.04.2017**, apoi depunctare 0.5p/ zi
- Deadline hard: 03.05.2017 ora 23.59
- Data publicării: 08.04.2017
- Data ultimei modificări: 08.04.2017
- Tema se va încărca pe **vmchecker**

## Obiective

- Utilizarea mecanismelor **funcționale**, de **tipuri**, și de **evaluare leneșă** din limbajul Haskell pentru rezolvarea unei probleme de **căutare** în spațiul stărilor.
- Exploatarea evaluării lenește pentru **decuplarea** etapelor de construcție și de explorare a arborelui de căutare.
- Utilizarea unor mecanisme eficiente, ***zipper***-e, pentru alterarea unor structuri nemodificabile, caracteristice limbajelor funcționale.

## Descriere

Secțiunea prezintă câteva noțiuni de bază despre *Monte Carlo Tree Search* și relevanța acestuia pentru tema de față.

### Monte Carlo Tree Search

*Monte Carlo Tree Search* (MCTS) este un algoritm de **căutare** în spațiul stărilor unei probleme, având drept scop alegerea unei acțiuni de realizat, pornind dintr-o stare oarecare. Este un algoritm des utilizat, fiind implicat și în sistemul *Alpha Go*, care a obținut în 2015 prima victorie semnificativă a unui jucător artificial de Go împotriva unui expert uman.

Spre deosebire de algoritmul Minimax, care utilizează o funcție de evaluare a unei stări oarecare, și presupune cunoștințe despre specificul jocului respectiv, MCTS nu necesită astfel de informații. În schimb, el implică **simularea** unui număr mare de jocuri aleatoare, pornind din starea curentă, **până la final**, și alegerea ulterioară a acelei acțiuni care maximizează o anumită măsură, precum numărul mediu de victorii obținute realizând acea acțiune, ținând cont de numărul total de simulări ale acțiunii respective.

O explicație excelentă a algoritmului, pentru cazul unui **singur** jucător, alături de un exemplu de aplicare, se găsește în acest **filmuleț**, pe care vă rugăm să îl urmăriți.

După ce înțelegeți algoritmul, putem realiza următoarele remarci:

- Faza a doua, ***expansion***, va fi implicită în implementarea noastră, întrucât, exploatând evaluarea **leneșă**, arborele va putea fi construit „în totalitate” de la bun început, iar explorarea lui la cerere va fi echivalentă cu procesul de expandare.

- Faza a patra, ***backpropagation***, presupune **actualizarea** unor parametri (scor și număr de vizitări) ale nodurilor pe o cale din arbore, de la o anumită stare înapoi la radăcină. Cum structura reprezentând arborele este **nemodificabilă** (asumpție fundamentală în paradigma funcțională), este necesară utilizarea unor mecanisme specifice care să permită aceste modificări. Este vorba de ***zipper***-e, menționate în secțiunea următoare.

### Zippers

Așa cum am menționat în secțiunea anterioară, *zipper*-ele reprezintă mecanisme care permit alterarea eficientă a structurilor **nemodificabile**, specifice limbajelor funcționale.

Vă rugăm să parcurgeți acest **tutorial** despre *zipper*-e. De interes sunt secțiunile:

- *Taking a walk*
- *A trail of breadcrumbs*
- *A very simple file system*

# Cerințe

Rezolvarea temei este structurată pe etapele de mai jos. Începeți prin a vă familiariza cu structura **arhivei** de resurse. Va fi de ajutor să parcurgeți indicațiile din enunț **în paralel** cu comentariile din surse. În rezolvare, exploatați testele drept **cazuri de utilizare** a funcțiilor pe care le implementați.

### MCTS pentru un singur jucător (80p)

Cerința principală a temei vizează implementarea MCTS pentru cea mai simplă situație, în care este vorba de un **singur** jucător, conform indicațiilor din filmuleț. Pentru aceasta, veți implementa funcțiile din fișierul `MCTS.hs`.

Clasa **`GameState`** enumeră funcțiile pe care orice reprezentare a stării de joc trebuie să le expună. Detaliile se găsesc în fișierul `GameState.hs`. Pentru această cerință, nu este necesar să instanțiați această clasă, acest lucru fiind realizat în teste. Însă, majoritatea funcțiilor din fișierul `MCTS.hs` conțin în tipul lor constrângerea `GameState s a`, ceea ce înseamnă că puteți utiliza direct funcțiile din clasa `GameState`pentru orice valoare de tipul `s`. Mai precis, pentru acestă cerință, sunt necesare **doar** funcțiile `successors` și `outcome`.

### MCTS pentru oricâți jucători (20p)

Această cerință solicită ca implementarea voastră să fie suficient de flexibilă, încât să permită existența **oricâtor** jucători. **Singura diferență** față de cazul unui singur jucător este întâlnită în cadrul pasului de ***backpropagation***. Spre deosebire de situația anterioară, în care erau actualizate scorurile tuturor nodurilor de pe calea către rădăcină, acum, vor fi actualizate **selectiv** doar scorurile nodurilor aferente mutărilor jucătorului care a câștigat, sau ale tuturor nodurilor, în caz de remiză. Toate nodurile vor fi marcate ca vizitate. Explicațiile detaliate se găsesc direct în surse.

Nici pentru această cerință nu este necesar să instanțiați clasa `GameState`, cu toate că veți utiliza **și** funcțiile `playerIndex` și `maxPlayers` definite de ea.

### Bonus: Aplicare MCTS pentru jocul „X și 0” (20p)

Pentru bonus, vi se propune să **aplicați** algoritmul MCTS implementat în cazul jocului **„X și 0”**, astfel încât utilizatorul (X) să poată juca cu calculatorul (0). Pentru aceasta, veți implementa funcțiile din fișierul `TicTacToe.hs`, dar, de data aceasta, veți instanția şi clasa **`GameState`**.

Funcțiile din fișierul `Interactive.hs` pot fi folosite pentru a **interacționa** cu programul. Odată ce ați implementat jocul de „X și 0”, utilizați `twoHumans` pentru doi utilizatori umani, sau `humanVsAI step`, pentru jocul utilizatorului cu calculatorul, unde `step` este funcția de alegere a următoarei mutări, pe care trebuie să o implementați pe baza MCTS.

## Precizări

- Este indicată utilizarea **funcționalelor**. Folosirea adecvată a acestora sau nefolosirea acestora aduc modificări în punctajul temei (în limita a 1 punct). Implementarea se poate realiza **în întregime fără** recursivitate explicită, utilizând funcționale și alte funcții de bibliotecă.
- Având în vedere că multe dintre funcțiile din fișierul `MCTS.hs` presupun repetarea unei operații până la validarea unei condiții, puteți utiliza funcționala **`until`**.
- Utilizați **`fromIntegral`** pentru conversia de la `Int` la `Float`.