Haskell: Kalah

Obiective

- utilizarea mecanismelor **funcționale** și de **tipuri** ale limbajului Haskell pentru implementarea unui **joc** (*Kalah/Mancala*)
- pentru bonus: exploatarea evaluării **lenețe** pentru explorarea controlată a spațiului stărilor într-un algoritm de inteligență artificială (*Minimax*).

Descriere

Scopul temei este implementarea jocului *Kalah* (denumit și *Mancala*). Modul de joc este descris aici. Se va implementa varianta de Kalah cu **48 de piese**.

În final, programul vostru:

1. va permite jocul între **doi utilizatori umani**
2. îi va da posibilitatea unui jucător uman să joace **împotriva calculatorului**, AI-ul fiind bazat pe o euristică simplă
3. pentru **bonus**, AI-ul va implementa o euristică mai complexă, bazată pe algoritmul *minimax*.

Cerințe

Rezolvarea temei este structurată pe etapele de mai jos. Începeți prin a vă familiariza cu structura **arhivei** deresurse. În rezolvare, exploatați **testele** drept cazuri de utilizare a funcțiilor pe care le implementați.

Tabla și mutările (90p)

Implementați tipurile de date și funcțiile din fișierul `Board.hs`, conform indicațiilor locale. Testele aferente se găsesc în fișierul `BoardTest.hs`. După implementarea (eventual parțială) a tablei și a mutărilor, puteți utiliza funcția `twoHumans` din fișierul `Interactive.hs`: introduceți numărul casei (de la `1` la `6`, numerotat de la stânga la dreapta), urmat de `Enter`, pentru precizarea mutărilor.
Pentru implementarea tablei de joc, aveți în vedere utilizarea unei structuri de date care să permită parcurgerea și actualizarea ușoară a numărului de semințe din fiecare casă. De asemenea, gândiți structura`Board` astfel încât să vă faciliteze implementarea funcțiilor exportate de `Board.hs`.

Euristică simplă (10p)

Implementați funcția `step` din fișierul `AISimple.hs`, conform specificațiilor locale. Euristica propusă este de aplicare a acelei mutări care conduce la **scorul maxim** al calculatorului în pasul următor. Testele aferente se găsesc în fișierul `AISimpleTest.hs`. Pentru testare, utilizați funcția `userMode` din fișierul `AISimple.hs`.

Bonus: Minimax (20p)

Abordarea precedentă are neajunsul că este o soluție **lacomă** de explorare a posibilităților de joc. O soluție care mărește șansele de câștig al jucătorului artificial este aceea care ia în calcul posibila evoluție ulterioară a jocului. Astfel, algoritmul Minimax explorează mutările posibile **viitoare** (până la un număr dat), atât ale jucătorului curent cât și ale adversarului, căutând să îi maximizeze celui dintâi șansele de câștig. Dat fiind faptul că dimensiunea spațiului de explorare poate fi intractabilă, deseori algoritmul include o componentă de limitare (*pruning*) a adâncimii de căutare.

În cazul Kalah, o posibilă abordare implică aplicarea Minimax pentru maximizarea **câștigului** AI-ului atunci când acesta mută, și minimizarea **pierderii** atunci când jucătorul uman mută. O particularitate a arborelui de stări generat în cazul acesta este faptul că un jucător poate juca două ture consecutive, caz în care vor fi efectuați doi pași consecutivi de maximizare sau de minimizare, în funcție de jucătorul curent.

Astfel, implementarea voastră va avea în vedere următoarele funcționalități:

- **Generarea** arborelui (posibil infinit) Minimax de stări (*expand*)
- **Limitarea** numărului de niveluri ale arborelui la o valoare dată (*prune*)
- **Evaluarea** trade-off-ului câștig/pierdere după o euristică dată și alegerea pierderii minime în condițiile în care adversarul joacă să își maximizeze câștigul, conform Minimax (*pick*)

În cazul particular al Kalah, funcția de evaluare euristică se va folosi atât de scorul AI-ului, cât și de cel al jucătorului uman, pentru a determina **câștigul** adus de stare dată celui dintâi. Spre exemplu, o stare în care scorul jucătorului uman este mai mare decât cel al AI-ului ar putea fi considerată ca aducând un câștig negativ celui din urmă.

Pentru o modularizare și reutilizare mai bună a codului, implementarea va înrola tipul de date `Board` în clasa de tipuri `Consecutive`, definită în `Consecutive.hs`. Este cerută apoi implementarea generică a algoritmului Minimax în fișierul `AIMinimax.hs`, cât și particularizarea acestei implementări în funcția `step`. Pentru a observa pas cu pas evoluția jocului, utilizați funcția `userMode` din același fișier. Testele aferente se găsesc în fișierul `AIMinimaxTest.hs`.

Precizări

- Unde este adecvat, se vor utiliza **funcționale**, în locul recursivității explicite. Cazurile evidente vor fi**depunctate**.
- Încercați să exploatați la maxim **bibliotecile** `Data.List` și `Data.Sequence`. Ambele biblioteci vă pot fi utile atât pentru construirea tablei de joc, cât și pentru proiectarea algoritmilor de parcurgere a acesteia.