Prolog: Ultimate Tic-Tac-Toe

- Data publicării: 10.05.2022
- Data ultimei modificări a enunțului: 19.05.2022 Etapa 1: 15.05.2022 (vezi changelog)
- Data ultimei modificări a scheletului: 27.05.2022 (Etapa 2), 11.05.2022 (Etapa 1)
- Deadline hard: ziua laboratorului 12 + 2 zile
- Forum temă [https://curs.upb.ro/2021/mod/forum/view.php?id=225449]
- vmchecker [https://vmchecker.cs.pub.ro/ui/#PP]

Obiective

- Aplicarea mecanismelor oferite de Prolog pentru implementarea unor funcții clasice, dar și pentru fluxul bidirecțional al controlului.
- Exploatarea mecanismului de backtracking pe care îl oferă motorul de execuție Prolog.

Organizare

Tema este împărțită în **2 etape**:

- una pe care o veți rezolva după laboratorul 10, cu deadline dependent de ziua în care aveți laboratorul 11:
  - laborator marți ⇒ deadline 19 mai
  - laborator miercuri ⇒ deadline 20 mai
  - laborator joi ⇒ deadline 21 mai
  - laborator vineri ⇒ deadline 22 mai
  - laborator luni ⇒ deadline 25 mai
- una pe care o veți rezolva după laboratorul 11, cu deadline la o săptămână după deadline-ul etapei 1, în a doua zi după ziua laboratorului 12.

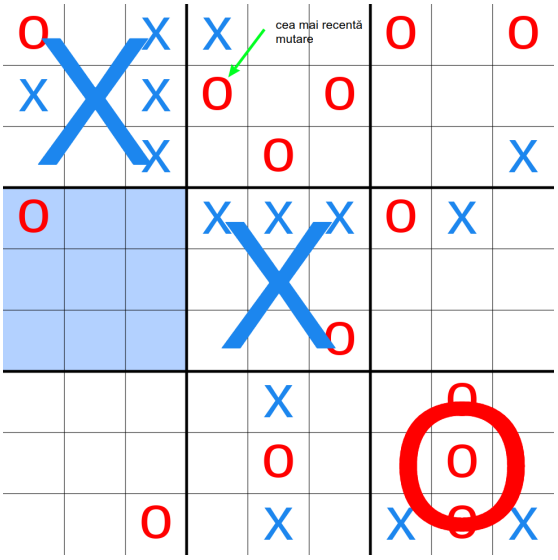
Așa cum se poate observa, **ziua deadline-ului variază în funcție de semigrupa în care sunteți repartizați**. **Restanțierii care refac tema și nu refac laboratorul beneficiază de ultimul deadline** (deci vor avea deadline-uri în zilele de 25.05, 01.06).

Rezolvările tuturor etapelor pot fi trimise până în ziua laboratorului 12 (**deadline hard pentru toate etapele**). Orice exercițiu trimis după un **deadline soft** se punctează cu **jumătate** din punctaj. Cu alte cuvinte, nota finală pe etapă se calculează conform formulei: **n = (n1 + n2) / 2** (n1 = nota obținută înainte de deadline; n2 = nota obținută după deadline). Când toate submisiile sunt înainte de deadline, nota pe ultima submisie este și nota finală (întrucât n1 = n2).

În fiecare etapă, veți folosi ce ați învățat în săptămâna anterioară pentru a dezvolta aplicația.

Descriere generală

Veți implementa în Prolog anumite componente dintr-un joc de Ultimate Tic-Tac-Toe [https://en.wikipedia.org/wiki/Ultimate_tic-tac-toe]. Vom utiliza aceste reguli [https://ultimate-t3.herokuapp.com/rules]. Puteți juca jocul aici [https://ultimate-t3.herokuapp.com/local-game]. Vom numi jocul de Ultimate Tic Tac Toe "UTTT" și un joc obișnuit de Tic-Tac-Toe (X și 0) "TTT". Vom face referire mai jos la următorul exemplu:



Vom identifica tablele individuale, ca și pozițiile din tablele individuale, prin "pozițiile" (în ordinea parcurgerii de la stânga la dreapta și de sus în jos): **NW, N, NE, W, C, E, SW, S, SE**:

```
nw | n | ne
---+---+---
w  | c |  e
---+---+---
sw | s | se
```

Tabla de UTTT (o vom mai numi și "U-board") este formată 9 table obișnuite de TTT (numite în cod "board"). O mutare a unui jucător va fi o mutare obișnuită de TTT pe una dintre tablele disponibile pentru mutări (despre asta mai târziu). În exemplul de mai sus, ultima mutare a fost o jucătorului `0` în poziția indicată de săgeata verde, deci jucătorul `0` a mutat în poziția `w` din tabla individuală care este în poziția `n` din tabla `e` UTTT.

Tablele individuale de TTT și tabla de UTTT se câștigă după regulile obișnuite din X și 0. Câștigarea unei table individuale de TTT reprezintă o mutare (un X sau 0) pe tabla mare de UTTT. În exemplul de mai sus, tablele din pozițiile `nw`, `c` și `se` din U-board au fost deja câștigate, de jucătorul `x`, `x` și `0`, respectiv. Jucătorul `0` este, de asemenea, aproape de a câștiga tablele `n` și `ne`.

Când un jucător câștigă pe tabla mare de UTTT, este câștigător al jocului și jocul se termină. În exemplul de mai sus, dacă jucătorul `0` câștigă și tablele `ne` și `e` atunci câștigă jocul (dacă nu îl câștigă `x` înainte).

Tabla sau tablele individuale de TTT pe care poate muta un jucător se decid(e) astfel:

- inițial, primul jucător poate muta (pune un X sau un 0) pe oricare dintre cele 9 table individuale.
- în mod obișnuit, un jucător este obligat să își realizeze mutarea pe tabla cu aceeași poziție (în cadrul U-board-ului) cu poziția pe care tocmai a mutat oponentul său în cadrul unei table individuale. În exemplul de mai sus, cum jucătorul `0` tocmai a mutat pe poziția `w` a unei table individuale, jucătorul `x` va trebui acum să mute într-una din pozițiile libere din tabla `w`.
- în cazul în care tabla pe care ar trebui să joace un jucător, atunci când îi vine rândul, a fost deja câștigată (și deci nu se mai pot face mutări pe ea), jucătorul care este la rând va putea muta pe oricare dintre tablele individuale disponibile (care nu au fost deja câștigate). În exemplul de mai sus, dacă jucătorul `0` ar fi mutat în centrul tablei (tabla individuală din poziția `n` a U-board), ar fi câștigat tabla `n`, dar, cum tabla `c` este deja finalizată, jucătorul `x` ar fi putut la următoarea mutare să mute în oricare dintre tablele `ne`, `w`, `e`, `sw`, `s`, ceea ce ar fi fost un avantaj pentru `x`.

Etapa 1

În această etapă vom implementa câteva predicate care lucrează cu liste. Trebuie implementate construcția și accesul la o stare a jocului și efectuare unei mutări în joc. Pentru bonus, se vor implementa două strategii foarte simple.

Reprezentarea concretă a unei stări este **la alegerea fiecăruia**. Puteți folosi liste, perechi, sau structuri (compounds).

Vom avea 3 grupuri de predicate (ordinea recomandată de implementare a predicatelor este cea din fișierul sursă):

- predicate care construiesc reprezentarea unei stări (care va fi accesată folosind predicatele de acces):
  - `initialState` construiește reprezentarea stării inițiale a jocului.
  - `buildState` construiește reprezentarea unei stări pe baza configurației tablei de joc și pe baza poziției în care a mutat jucătorul anterior.
- predicate care accesează reprezentarea unei stări:
  - `getBoards`, `getBoard`, `getPos/3`, `getPos/4` obțin informații despre tablele de joc individuale. Tablele individuale sunt văzute ca liste de 9 celule, puse în ordinea pozițiilor dată mai sus. Fiecare celulă poate avea ca valoare atomul (literalul) `x`, numărul `0`, sau atomul vid `''`. Pentru starea din exemplul de mai sus, `getBoard(State, n, B)` trebuie să lege `B` la lista `[x,'','', 0,'',0, '',0,'']`.
  - `getUBoard` obține configurația tablei de UTTT, văzută ca o tablă individuală. În plus față de tablele individuale, tabla de UTTT poate avea și celule cu valoarea `r`, pentru tablele individuale remizate. Pentru starea din exemplul de mai sus, `getUBoard(State, UBoard)` trebuie să lege `UBoard` la lista `[x,'','', '',x,'', '','',0]`.
  - `getNextPlayer` obține jucătorul care urmează la rând (`x` sau `0`). Acesta poate fi determinat numărând celulele cu `x` și cu `0` din tablele individuale. Primul jucător este `x`. În exemplul de mai sus, următorul jucător este `x` pentru că sunt 14 mutări ale lui `x` și 14 mutări ale lui `0`.
  - `getNextAvailableBoards` obține tablele individuale (ca poziții în tabla de UTTT) disponibile pentru următoarea mutare.
    - în starea inițială, este întreaga listă de poziții (jucătorul poate muta în orice tablă), deci pentru `initialState(S0), getNextAvailableBoards(S0, Boards)`, `Boards` trebuie legat la lista completă de poziții: `[nw, n, ne, w, c, e, sw, s, se]`;
    - de obicei, este o listă conținând o singură poziție, aceeași cu poziția dintr-o tablă individuală unde a mutat jucătorul precedent. În exemplul de mai sus, cum jucătorul `0` tocmai a mutat în poziția `w` a tablei `n`, jucătorul `x` trebuie să mute obligatoriu în tabla `w`, deci `getNextAvailableBoards(State, Boards)` trebuie să lege `Boards` la lista `[w]`;
    - atunci când jucătorul precedent a mutat într-o poziție care corespunde unei table individuale în care jocul s-a terminat, sunt disponibile pentru următoarea poziție toate tablele care nu sunt încă finalizate (nu au fost câștigate sau remizate). Dacă în exemplul de mai sus `x` mută în centrul tablei `w`, pentru următoarea mutare, a lui `0` vor fi disponibile tablele `[n, ne, w, e, sw, s]`.
  - `getBoardResult` obține rezultatul pentru configurația unei table individuale. Rezultatul poate fi `x`, `0`, `r`, sau `''`, acesta din urmă pentru cazul în care jocul pe această tablă individuală continuă.
- predicate pentru efectuarea unei mutări. Dacă pentru mutare este disponibilă o singură tablă, atunci mutarea este poziția pe această tablă unde se va pune `x` sau `0`; dacă pentru mutare sunt disponibile mai multe table, atunci mutarea este o pereche `(UPos, Pos)` între poziția tablei individuale (în cadrul U-board) unde se va face mutarea, și poziția din tablă unde se va pune `x` sau `0`.
  - `validMove` verifică validitatea unei mutări. O mutare este validă dacă:
    - jocul de UTTT nu s-a terminat;
    - în starea curentă este o singură tablă individuală disponibilă pentru a muta, mutarea este o poziție, și poziția este validă în tabla disponibilă, sau
    - în starea curentă sunt mai multe table individuale disponibile, mutarea este o pereche de poziții, tabla aleasă pentru a muta nu este finalizată (câștigată sau remizată), și poziția din tablă aleasă pentru mutare este liberă.
  - `makeMove` determină starea următoare după efectuarea unei mutări.

Pentru BONUS în această etapă se vor implementa două strategii foarte simple:

- `dummy_first`, care alege întotdeauna prima (în ordinea pozițiilor) mutare disponibilă. În exemplul de mai sus, strategia va întoarce `n`, prima poziție disponibilă din tabla `w` unde trebuie să mute `x`. Dacă din starea din exemplu, `x` ar muta, conform unei alte strategii, în `c`, pentru mutarea lui `0` strategia `dummy_first` ar întoarce `(n, n)`, pentru că `n` este prima tablă disponibilă, și `n` este prima poziție disponibilă din acea tablă.
- `dummy_last`, care alege întotdeauna ultima (în ordinea pozițiilor) mutare disponibilă. În exemplul de mai sus, strategia va întoarce `we`, ultima poziție disponibilă din tabla `w` unde trebuie să mute `x`. Dacă din starea din exemplu, `x` ar muta, conform unei alte strategii, în `c`, pentru mutarea lui `0` strategia `last` ar întoarce `(s, se)`, pentru că `s` este ultima tablă disponibilă, și `se` este ultima poziție disponibilă din acea tablă.

NOTĂ: pentru majoritatea testelor de la predicatele de acces este necesar ca `buildState` să fie implementat. Pentru restul (mai puține), este necesar ca `initialState` să fie implementat. Ideal este să implementați predicatele de acces la reprezentarea stării în același timp cu părțile corespunzătoare din predicatele care construiesc reprezentarea stării.

Hints

Pe parcursul implementării temei, veți găsi foarte utile predicatele nth0/3 [https://www.swi-prolog.org/pldoc/doc_for?object=manual] și nth0/4 [https://www.swi-prolog.org/pldoc/doc_for?object=nth0/4].

Pentru a afișa o stare a jocului, folosiți predicatul `printBoards/1`, iar pentru a afișa o tablă individuală folosiți predicatul `printBoard/1`. De exemplu, pentru a vizualiza starea inițială (odată ce ați implementat construcția sa), puteți folosi interogarea:

```
initialState(S), printBoards(S).
```

Iar pentru a afișa o stare utilizată în teste, puteți folosi, de exemplu (odată ce ați implementat predicatul `buildState`:

```
uttt(2, S), printBoards(S).
```

Testele sunt disponibile în fișierul `checker.pl`, iar jocurile și listele de mutări folosite în teste sunt disponibile în fișierul `input.pl`.

Pentru testele de `validMove` care ies din timp, vedeți acest thread [https://curs.upb.ro/2021/mod/forum/discuss.php?d=14713#p40496].

Etapa 2

În această etapă vom evalua calitatea mutărilor după un algoritm dat și vom implementa două strategii care folosesc aceste măsuri de calitate.

Calitatea mutărilor într-o tablă individuală

Într-o tablă individuală, vom avea următoarele *priorități* pentru mutări (mutarea cu cea mai mare calitate va fi cea de prioritate minimă):

- orice mutare care duce la câștigarea tablei de către jucătorul aflat la mutare are prioritate 0.
  - în exemplu, în tabla ne, pentru jucătorul 0, mutarea n are prioritate 0.
- orice mutare care blochează oponentul de la a câștiga (este într-o poziție în care dacă oponentul mută, câștigă) are prioritate 1.
  - în exemplu, în tabla ne, pentru jucătorul x, mutarea n are prioritate 1.
- dacă tabla este goală, mutările în colțuri au prioritate 2.
- dacă jucătorul curent nu a mutat de loc în această tablă, iar oponentul a mutat deja în centru, mutările în colțuri au prioritate 3.
- dacă jucătorul curent nu a mutat de loc în această tablă, iar oponentul nu a mutat deja în centru, mutarea în centru are prioritate 3.
- orice mutare care duce tabla într-o stare din care jucătorul curent poate câștiga cu o singură mutare are prioritate 4.
- în orice alt caz în afară de cele de mai sus, o mutare într-un colț are prioritate 5, iar celelalte mutări au prioritate 6.

Predicatul `movePriority/4` evaluează prioritatea unei mutări, pentru un jucător, pentru o tablă individuală.

Predicatul `bestIndividualMoves/3` ordonează mutările disponibile într-o tablă individuală în funcție de prioritatea lor pentru jucătorul curent. Ordinea apriori a mutărilor este cea din lista positions. Două mutări cu prioritate egală își păstrează ordinea apriori. Folosiți pentru sortare `sortMoves/2`.

În exemplu, pentru tabla w, cele mai bune mutări sunt, în ordine, `[c, ne, sw, se, n, w, e, s]`, pentru că:

- oponentul a mutat deja în acea tablă, iar centrul nu e încă ocupat
- apoi, cele 3 colțuri în afară de nw nu sunt ocupate
- apoi mijlocurile laturilor.

Strategia `narrowGreedy` va întoarce o mutare bazată pe următorul algoritm: dacă este o singură tablă disponibilă, se va alege mutarea cu cea mai mică prioritate (sau prima mutare cu cea mai mică prioritate); dacă sunt mai multe table disponibile, se alege tabla cu cea mai mică prioritate, și în ea mutarea cu cea mai ică prioritate.

Calitatea mutărilor în jocul UTTT

Pentru a evalua mutările la nivelul întregului joc de UTTT, vom considera următoarea ordonare a priorităților:

- mutări care fac ca jucătorul curent să câștige jocul
- mutări care îl duc pe oponent într-o tablă în care nu a mutat de loc

- mutări care îl duc pe oponent într-o tablă în care a mutat o singură dată
- mutări care îl duc pe oponent într-o tablă în care a mutat de cel puțin 2 ori, cu prioritate pentru tablele unde jucătorul curent are mai multe mutări
  - nu intră aici tablele în care oponentul câștigă, tablele în care jucătorul curent mai are o mutare pentru a câștiga tabla individuală
- mutări care nu se încadrează în alte cazuri din această listă, inclusiv cele de mai jos
- mutări care duc oponentul într-o tablă în care jucătorul curent este la o mutare de a câștiga
- mutări care duc oponentul într-o stare în care oponentul este la o mutare de a câștiga, dar acea mutare duce jucătorul curent într-o tablă în care este la o singură mutare de a câștiga sau este deja finalizată
- mutări care duc oponentul într-o stare în care oponentul este la o mutare de a câștiga, iar acea mutare duce jucătorul curent într-o tablă în care **nu** este la o singură mutare de a câștiga
- mutări care duc oponentul într-o tablă deja finalizată
- mutări care duc oponentul într-o tablă în care poate muta pentru a câștiga direct întreg jocul.

Predicatul `bestMoves/2` ordonează mutările disponibile în ordinea prezentată mai sus. Pentru stările în care sunt mai multe table disponibile pentru jucătorul curent, mutările vor fi luate apriori (înainte de sortarea după priorități) după ordonearea tablelor individuale conform cu `bestIndividualMoves/3`.

NOTĂ: când calculați cele mai bune mutări, în moemntul evaluării unei anumite mutări M, atunci când facem referire la ce va face mai departe un alt jucător (sau același jucător), faceți evaluarea pe starea în care mutarea M s-ar fi aplicat deja.

În exemplu, pentru x, cele mai bune mutări (după această strategie) sunt, în ordine, `[sw, w, e, s, n, ne, c, se]`, pentru că:

- sw este un colț, iar acolo oponentul are puține mutări
- în w oponentul are puține mutări
- în e oponentul are puține mutări, dar x are deja o mutare
- în s, oponentul are puține mutări, dar x are deja 2 mutări
- în n, oponentul va câștiga tabla, dar îl trimite pe
- în ne, oponentul va câștiga tabla, iar câștigând nu va duce într-o tablă în care x va câștiga direct
- în c, tabla este deja finalizată
- în se, la fel.

Strategia `greedy` va întoarce cea mai bună mutare (prima) din mutările întoarse de `bestMoves/2`.

Bonus etapa 2

Pentru puncte bonus, implementați cât mai elegant, folosind `findall`, `forall`, și având reguli diferite doar atunci când este neapărat nevoie (e.g. pentru calculul priorităților pe diferite cazuri).

Hints

- folosiți `findall/3`
- folosiți `sortMoves/2` din `utils.pl`

Precizări

- se va lucra numai în fișierul `uttt.pl`
- pentru rularea testelor se va apela predicatul vmcheck/0 (apela de la consolă ca `vmcheck.`)
  - puteți apela teste individuale cu `vmtest(<nume_test>)`, e.g. `vmtest(narrowGreedy)`
- pentru rezultate mai detaliate ale testelor, **decomentați** linia `detailed_mode_disabled :- !, fail.` din fișierul `checker.pl`. Modul detaliat (unde este posibil să primiți câteva puncte în plus cu implementările implicite) **nu** este cel folosit pe vmchecker, dar în acest mod testerul oferă mai mult detalii despre testele eșuate.
- în `input.pl` există și starea `uttt(enunt, S)`, care este exemplul de mai sus din acest enunț.

Resurse

- Schelet etapa 1 [https://ocw.cs.pub.ro/courses/_media/pp/22/teme/prolog/etapa1.zip]
- Schelet etapa 2 [https://ocw.cs.pub.ro/courses/_media/pp/22/teme/prolog/etapa2.zip]

Changelog

- 10.05 - publicarea temei (încă nu sunt disponibile testele pentru bonusul etapei 1)
- 11.05 - modificare a scheletului - în `uttt.pl` trebuie inclus `files.pl` în loc de `checker.pl`
- 11.05 - adăugare teste pentru bonusul etapei 1
- 11.05 - mai multe exemplificări în enunț
- 11.05 - activare vmchecker
- 12.05 - adăugat "NOTĂ" la etapa 1
- 15.05 - corecție în enunț unde apărea getAvailableBoards în loc de getNextAvailableBoards.
- 16.05 9:15 - clarificare cazuri validMove în enunț.
- 16.05 - modificare deadline.
- 18.05 - adăugare la `printBoards` (în scheletul pentru etapa 2) a afișării jucătorului curent și a tablelor disponibile pentru următoarea mutare.
- 18.05 - corecție teste `bestIndividualMoves`
- 18.05 - vmchecker etapa 2
- 18.05 - adăugare
  - informații bonus etapa 2
  - mai multe exemple în enunț
  - `uttt(enunt, S)` în `input.pl` și mențiune în enunț
  - mențiune tastare teste individuale
  - NOTĂ la `bestMoves`
- 19.05 - pentru prioritate 5 la tabla individuală erau mutările *în colțuri*, nu *în centru*.
- 19.05 - îmbunătățire a testelor pentru movePriority
- 19.05 - îmbunătățire a testelor pentru a reduce diferența dintre modul detaliat și modul nedetaliat (de pe vmchecker) de testare.
- 25.05 - adăugare afișări ajutătoare (comentate) pentru `play_strategies` în `utils.pl`
- 27.05 - acceptare a soluției (c,se) pentru testul greedy|c (vezi această discuție [https://curs.upb.ro/2021/mod/forum/discuss.php?d=15374])