Loogiline programmeerimine, sügissemester 2013

Kursusetöö

Ülesande kirjeldus

Koostada Prologis kabeprogramm, mis sooritab korraga ühe käigu/võtmise(d). Programm peab võistlema vastase programmiga. Predikaat "arbiiter" annab programmidele korda-mööda käiguõiguse. Mäng lõpeb, kui ühel mängijatest ei ole enam võimalik teha käike. Võitja on programm, mis sooritas viimase käigu. Arbiiter kontrollib käikude õigsust ja diskvalifitseerib sohki teinud programmi.

Kõik programmid peavad järgima järgmisi kokkuleppeid:

1. Kabelaua seis esitada faktidega ruut/3:

```
ruut(X,Y, Status).     % kus X,Y ∈ [1,8]

      Status = 0       % tühi
      Status = 1       % valge
      Status = 2       % must
      Status = 10      % valge tamm
      Status = 20      % must tamm
```

NB! Valged alustavad väiksemate X-koordinaadi väärtusega ruutudest, mustad – suuremate X-koordinaadi väärtusega ruutudest st. valge nupu jaoks leidub algseisus fakt: `ruut(1,1,1).`

2. Mänija programmi vormistamise reeglid:

2.1. Käiku sooritav programm peab olema vormistatud mooduli kujul
Näide:

```
:- module(mooduli_nimi, mooduli_peapredikaat/1).
```

2.2. Mooduli peapredikaat peab olema kujul

```
mooduli_peapredikaat(Color).
```

kus `Color` – tähistab nuppude värvi, millega antud programm mängib.

2.2. Moodulis ei tohi esineda staatilisi fakte ruut/3 ja definitsiooni

```
:- dynamic ruut/3.
```

2.3. Mängija programmist ei tohi väljuda fail-ga

2.4. Backtrack-iga ei tohi minna teise mängija programmi.

Seetõttu on soovitav kasutada mängija peapredikaadi järgmist struktuuri:

```
mooduli_peapredikaat(Color):-
....., !.
mooduli_peapredikaat(_).
```

3. Arbiiteri kohandamine konkreetsetele programmidele
   3.1 Faktis `players_turn(1,2,Nimi1).` tuleb 3. parameetri väärtuseks kirjutada programmi nimi, mis mängib MUSTADE nuppudega.

3.2 Fakti `players_turn(2,1,Nimi2)`. tuleb 3. parameetri väärtuseks kirjutada programmi nimi, mis mängib VALGETE nuppudega.

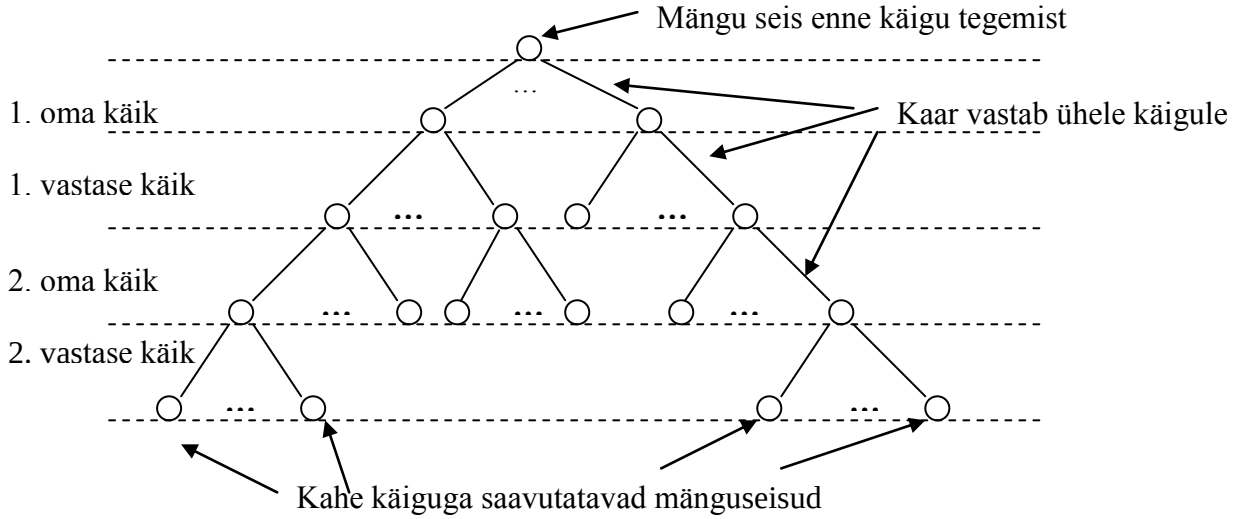
4. Mängu käivitamine

4.1. Laadida esmalt mällu programm arbiter.pl ja seejärel mängijate programmide failid

4.2. Kutsuda välja mängu käivitav predikaat "turniir/0"

Abistavaid näpunäiteid käikude planeerimiseks

Käikude planeerimiseks on otstarbekas genereerida saavutatavate mänguseisude puu. Igale mänguseisule vastab puu üks tipp ja igale käigule kaar tippude vahel. Puu sügavus on määratud sellega kui mitu sammu antud käiku ette planeeritakse.



Joonis 1. Käikude planeerimispuu

Käigu sooritamiseks vajalikud planeerimistegevused:

1. Planeerimiseks vajalike faktide loomine, näiteks musti ruute iseloomustav abifakt ruut/7 omab järgmist vormingut:

```
ruut(X,Y,Color,Plan_step,Prev_state, Present_state,Cost).
```

kus

`X,Y` -- nupu koordinaadid [1,..,8]
`Color` -- ruudul oleva nupu värv [1,2]
`Plan_step` -- planeerimispuu korrus, millele fakt vastab [0,..,n]
`Prev_state` -- eelmise seisu ID, millest tekib jooksev seis
`Present_state` -- käiguga tekkiva seisu ID
`Cost` -- käigu tulemusena tekkiva seisu hind (vastase nupu võtmisel: `Cost:=Cost+1`, oma nupu kaotamisel: `Cost:=Cost-1`)

2. Planeerimispuu genereerimine:
   a. Planeerimispuu koosneb ruut/7 faktidest, mille parameeter `Prev_state` võimaldab puud läbida terminal-tipust juur-tipu suunas.

3. Parima käigu valimine:
   a. Kasutades fakti ruut/7 parameetri `Cost` väärtusi, leida planeerimispuu terminaalsetele tippudele vastavate (, kus parameeter `Plan_step =` max planeerimissügavus, näiteks `Plan_step = 2`) faktide ruut/7 hulgast niisugune, mille parameeter `Cost` omab suurimat väärtust.
   b. Kasutades fakti ruut/7 parameeterit `Prev_state` liikuda planeerimispuu juurtipuni ja kuulutada sellele teele jääva esimese käigu tulemus mängu uueks seisuks.
4. Kopeerida valitud käiguga tekkiv uus seis vastasele nähtavaks faktide hulgaks ruut/3 ja anda juhtimine tagasi arbiiterprogrammile.

Tammi programmeerimine (rekursiivne rakendus kuni jõuab nuppude arvu mõttes püsipunktini):

- Antud tammi positsioon (X_0, Y_0)
- Samm1: 2 diagonaali määramine võrranditega:
  - $X = Y + X_0 - Y_0$ (Rel*)
  - $X = -Y + (X_0 + Y_0)$ (Rel**)
    mis lõikuvad positsioonis (X_0, Y_0).
- Samm2: FORALL Rel $\in$ {Rel*, Rel**} Tamm saab võtta, kui diagonaalil Rel leidub vastase nupuga ruut (X_v, Y_v), mis rahuldab võrrandit Rel, st.
  $\exists P(X_v, Y_v) \models Rel \wedge \text{COLOR}(X_v, Y_v) =/= \text{COLOR}(X_0, Y_0) =/= 0$
  - tammi ja vastase nupu vahel on kõik ruudud tühjad st.
    $\forall P(X_i, Y_i) \models Rel$: $\text{sign}(X_v-X_0)=\text{sign}(X_i-X_0) \wedge d(P_0,P_i) < d(P_0,P_v) \Rightarrow \text{COLOR}(X_i, Y_i) = 0$
  - sellel diagonaalil vahetult peale vastase nuppu leidub mittetühi vabade ruutude hulk

  $T = \{(X_t, Y_t) \models Rel, 0 < X_t, Y_t < 9 \wedge \text{COLOR}(X_t, Y_t) = 0 \wedge \text{sign}(X_v-X_0) = \text{sign}(X_t-X_0) \wedge \exists!(X_t, Y_t) \models Rel \wedge$
- Samm3: Võtmisel tekib igast vabast järelruudust uus planeerimisharu
- Samm4: Kas Sammud 1 – 3 andsid püsipunkti?

- Kui “jah”, siis return;
- Kui “ei”, siis goto Samm1

Kasulikke linke

www.dobrev.com/games/Checkers/checkers.html