Izvestaj projekta

TRECI DEO

Zadatak III – Min-max algoritam i heuristika

- Implementirati Min-Max algoritam sa alfa-beta odsecanjem za zadati problem (igru):
  - Na osnovu zadatog stanja problema
  - Na osnovu dubine pretraživanja
  - Na osnovu procene stanja (heuristike) koja se određuje kada se dostigne zadata dubina traženja
  - Vraća potez koji treba odigrati ili stanje u koje treba preći
- Realizovati funkcije koje obezbeđuju odigravanje partije između čoveka i računara

Zadatak III – Min-max algoritam i heuristika

- Implementirati funkciju koja vrši procenu stanja na osnovu pravila zaključivanja
- Funkcija za procenu stanja kao parametre treba da ima igrača za kojeg računa valjanost stanja, kao i samo stanje za koje se računa procena.
- Procena stanja se mora vršiti isključivo korišćenjem mehanizma zaključivanja nad prethodno definisanim skupom pravila. Zadatak je formulisati skup pravila i iskoristiti ih na adekvatan način za izračunavanje heuristike.
- Za izvođenje potrebnih zaključaka (izvršavanje upita nad skupom činjenica kojima se opisuje stanje) koristiti mašinu za zaključivanje.
- Implementirati funkciju koja prevodi stanje u listu činjenica ...

1. Dodate su min_value i max_value funkcije kojima implementiramo minmax algoritam

```
def max_value(stanje, dubina, alpha, beta):

    lista_novih_stanja = nova_stanja(stanje)
    if(dubina == 0 or len(nova_stanja) == 0):
        return (stanje,proceni_stanje())
    else:
        for s in lista_novih_stanja:
            alpha = max(alpha,min_value(s,dubina-1,alpha,beta), key = lambda x: x[1])
            if(alpha[1] >= beta[1]):
                return beta
    return alpha

def min_value(stanje, dubina, alpha, beta):

    lista_novih_stanja = nova_stanja(stanje)
    if(dubina == 0 or lista_novih_stanja is None):
        return (stanje,proceni_stanje())
    else:
        for s in lista_novih_stanja:
            beta = min(beta,max_value(s,dubina-1,alpha,beta), key = lambda x: x[1])
            if(beta[1] <= alpha[1]):
                return alpha
    return beta
```

2. **nova_stanja** – pravimo listu i zatim za svako polje na tabli pozivamo funkciju novipotez i ukoliko ona ne vrati false dodajemo u listu

```
def nova_stanja(xv, ov, prva_Matrica, naPotezu):
    Lista = list()
    for row in range(0, n):
        for col in range(0, m):
            ns = NoviPotez(xv, ov, prva_Matrica, naPotezu,row,col)
            if(ns):
                Lista.append(ns)
    return Lista
```

3. **proceni_stanje** – Ako je broj mogucih poteza za nekog igraca nula onda vracamo najvecu ili najmanju vrednost (u ovom primeru 999 i -999). Svaki potez menja stanje na tabli tako sto smanjuje broj mogucih poteza kako za protivnika tako i za samog igraca koji odigrava potez. Igracu je cilj da smanji broj mogucih poteza protivnika ali da prilikom toga taj potez sto je manje moguce utice na broj mogucih njegovih poteza.

   Vrednostprotivnika - Vrednost igraca/6

```
def proceni_stanje():
    global naPotezu, xv, ov

    if(naPotezu):
        if(xv == 0):
            return 999
        if(ov == 0):
            return -999
        ps = xv - ov/6
    else:
        if(ov == 0):
            return 999
        if(xv == 0):
            return -999
        ps = ov - xv/6
    return ps
```

4. **CovekProtivRacunara** – nakon svakog poteza proveravamo da li smo dosli do kraja, ako je na potezu X onda igra covek dok ako je na potezu O igra racunar. Imamo grananje u zavisnosti od toga ko igra prvi, kod dela za coveka trazimo od korisnika da unese kordinate i upisujemo potez dok kod dela za racunar max_value racuna koji je najbolji potez i onda ga prosledjujemo funkciji UpisiRacunar

```
def CovekProtivRacunara():
    global naPotezu, prva_Matrica, xv, ov
    showTable(prva_Matrica)

    while(Kraj(naPotezu)==0):
        if(naPotezu=="X"):
            Upisi(prva_Matrica)
            showTable(prva_Matrica)
        else:
            state = max_value(xv, ov, prva_Matrica, naPotezu,3,[xv, ov, prva_Matrica, naPotezu,-9999],[xv, ov, prva_Matrica, naPotezu,9999])[0]
            p = {
            "broj": state["broj"],
            "slovo": state["slovo"]
            }

            UpisiRacunar(prva_Matrica,p)
            showTable(prva_Matrica)
    return True
```