## Faza lll

## U poslednjoj fazi dodate su funkcije koje obezbedjuju odigravanje partije izmedju čoveka i računara.

## Funkcije koje ovo obezbedjuju su: covek\_protiv\_racunara(), proceni\_stanje(stanje), max\_value(stanje,dubina,alpha,beta), min\_value(stanje,dubina,alfa,beta) kao i funkcije koje su obradjene u prethodnim fazama.

## Najpre bi bilo dobro objasniti način na koji vrednujemo jedno konkretno stanje na tabli. Ovo obezbedjuje funkcija proceni\_stanje kojoj se kao argument prosledjuje stanje za koje treba izračunati vrednost.

## Vrednost stanja izračunavamo oslanjajući se na koncepciju koja je obrazložena u prvoj fazi: svaki potez menja stanje na tabli time što smanjuje broj mogućih poteza kako za protivnika tako i za samog igrača koji odigrava potez. Od najveće važnosti za igrača jeste smanjiti broj mogućih poteza protivnika za najveću moguću vrednost ALI u isto vreme taj potez treba da što manje utiče na broj mogućih poteza igrača koji ga odigrava.

## Imajući ovo u vidu, za izračunavanje stanja na tabli koristimo obrazac:

## protivnikValue – igračValue / 6

## protivnikValue – broj mogućih poteza protivnika

## igračValue – broj mogućih poteza onoga ko odigrava potez

## proj mogućih poteza igrača delimo sa 6 da bi smo obezbedili prioritet

## (smanjiti broj mogućih poteza protivnika ima veći prioritet u odnosu na smanjivanje broja mogućih vlastitih poteza. Uzimanjem većeg ili manjeg broja za konstantu delioca biramo koliko ćemo uzimati u obzir smanjenje vlastitih mogućnosti nakon odigranog poteza)

## Ukoliko imamo situaciju na tabli da je broj mogućih poteza za nekog igrača 0 (odnosno da je neko pobedio i da je igra gotova), vraćamo najveću (odnosno najmanju) moguću vrednost (recimo 999 i -999). Na ovaj način smo obezbedili da poednička pozicija ima najveću vrednost i da se uvek krećemo ka njoj ako je to moguće.

## Sada kada imamo definisan način vrednovanja table, ostaje da ovo primenimo u min max algoritmu sa alfa beta odsecanjem.

## MinMax algoritam implementiran je funkcijama min\_value i max\_value.

## Kao pomoćna promenljiva, stanju na tabli (pri generisanju novih stanja) se dodeljuje i potez koji je odigran da bismo stigli u to stanje. Na ovaj način ćemo na kraju moći lako da preuzmemo potez za koji smo utvrdili da je najbolji u obliku (int) (char) da bi bio kompatibilan sa igraj\_potez funkcijom koja potez prihvata u ovom formatu.

## Na kraju funkcija koja sve ovo spaja je covek\_protiv\_racunara()

## Na početku funkcije nalazi se inicijalizacija igre (inicijalizacija tabele, odabir prvog igrača itd...).

## Nakon svakog poteza proveravamo da li igra još uvek traje (da li imamo pobednika).

## U zavisnosti od toga ko igra prvi, obezbedjeno je grananje na deo koji se izvršava kada igra čovek i deo koji se izvršava kada je potez računara.

## Kod odigravanja poteza čoveka tražimo unos poteza sve dok potez na ulazu ne bude validan.

## Kod odigravanja poteza računara pozivamo max\_value funkciju za izračunavanje najboljeg poteza za trenutno stanje na tabli. Potez koji treba odigrati preuzimamo lokalnom promenljivom „potez“ i pozivamo funkciju igraj\_potez za tu vrednost koju smo dobili.

## Kada više nema mogućih poteza za jednog od igrača, igra se prekida i proglašavamo pobednika.