SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Jednostavni AI agent za igru "šah"

Leo Goršić
Voditelj: *Marko Đurasević*

Zagreb, Svibanj, 2023

Commented [MĐ1]: Ovo bi trebalo biti na kraju prosle stranice.

Sadržaj

1. *Uvod* 3
2. *Organizacija programa* 5
3. *Stanje igre i legalnost poteza* 6
4. *MiniMax algoritam za odabir optimalnog poteza* 8
5. *Zaključak* 10
6. *Sažetak* 11
7. *Literatura* 11

1. Uvod

Igra “šah” je vrlo popularna igra za dva igrača. Igrači vuku naizmjenično poteze (prvo igra igrač sa bijelim figuricama). Cilj igre je zadati šah-mat suparničkom kralju. Pravila o kretanju figurica po ploči su se kroz povijest mijenjala te su se ustalila sljedeća pravila [1]:

- Pješak se kreće isključivo prema naprijed. Iz početne pozicije može ići naprijed za jedno ili za dva polja, a nakon toga samo za po jedno polje. Uzimati pak može samo ukoso na prvo polje prema naprijed. Pješak se dolaskom do zadnjeg reda promovira.

- Kralj se može kretati u svim smjerovima za jedno polje. Nikada ne smije doći na mjesto koje napada bilo koja suparnička figura. Kralj je najvažnija figura te napad na njega (šah) treba otkloniti. Ako nije moguće otkloniti napad suparnik pobjeđuje (šah mat)

- Kraljica je najjača figura, kreće se neograničenim brojem slobodnih polja u svim smjerovima: okomito, vodoravno i dijagonalno. Ne može preskakati figure.

- Kula (naziva se i "top") se kreće neograničen broj slobodnih polja okomito i vodoravno (po stupcima i redovima). Ne može preskakati ostale figure.

- Lovac se kreće neograničen broj slobodnih polja po dijagonalama. Ne može preskakati druge figure.

- Skakač (naziva se i "konj") jedina je figura koja može preskočiti drugu figuru, svoju ili protivničku. Kreće se u obliku slova L: dva polja ravno, zatim jedno polje lijevo ili desno, na bilo koju stranu.

Commented [MĐ2]: Suparničkom?

Commented [MĐ3]: Ako

Commented [MĐ4]: Ako se vec koriste redovi, mozda bolje onda ovdje reci stupcima?

Commented [MĐ5]: Se krece

Postoje mnoge strateške igre koje su “riješene”, to jest pronađen je optimalan skup poteza koji garantirano igrača dovodi do pobjede (ili makar izjednačene igre). Primjer takvih igara su “križić - kružić” i “4 u nizu”. Međutim igra šah zbog prirode kretanja figurica i veličine ploče uzrokuje prevelik broj mogućih kombinacija igre.

Commented [MĐ6]: Bolje "dovodi do" ili "uzrokuje"

Zbog toga unatoč naglom razvoju računala (pa čak i kvantnih računala) nije izgledno da će se pronaći takav skup poteza koji bi garantirano doveli do pobjede u igri šah. Kako bi se dočarala kompleksnost ove igre spominje se vrlo poznati „Shannonov broj“ koji se smatra donjom granicom složenosti ove igre. Claude Shannon je grubom procjenom odredio kako u prosjeku igrač raspolaže 10^3 mogućih poteza. Također je zaključio da igra u prosjeku traje oko 40 parova poteza. Time se dolazi do broja od 10^{120}mogućih partija šaha [2].

Commented [MĐ7]: Partija?

Zbog ovolike kompleksnosti igre iscrpno pretraživanje nam neće dati dovoljno dobre rezultate te se moraju razmotriti sofisticiraniji postupci kojim će AI agent donositi odluke. U ovom seminarskom radu dati ću pregled implementacije jednostavnog algoritma minimax pomoću kojeg ću napraviti igru šah u programskom jeziku Python.

2. Organizacija programa

Program je podijeljen u tri datoteke radi veće preglednosti. Najvažnija datoteka je *ChessEngine* koja sadrži trenutno stanje igre i sve popratne metode koje se koriste za osiguravanje legalnosti poteza.

Datoteka *ChessMain.py* sadrži implementaciju iscrtavanja ploče i glavne petlje u kojima igrači naizmjenično vuku poteze. Njena se implementacija uvelike oslanja na biblioteku *PyGame* te ranije spomenutu datoteku *ChessEngine*. Naposlijetku datoteka *ChessAIEngine* sadrži algoritme za odabir optimalnog poteza iz skupa legalnih poteza.

Commented [MĐ8]: Numerirati sekcije i staviti da svakak pocinje na zasebnoj stranici. Izgleda ljepse. Takoder staviti dvostruko poravnavanje teksta

Commented [MĐ9]: kojima

3. Stanje igre i legalnost poteza

Razmotrimo najprije ChessEngine datoteku. Stanje ploče modelirano je 2D poljem stringova:

```
[
    ["bR", "bN", "bB", "bQ", "bK", "bB", "bN", "bR"],
    ["bp", "bp", "bp", "bp", "bp", "bp", "bp", "bp"],
    ["--", "--", "--", "--", "--", "--", "--", "--"],
    ["--", "--", "--", "--", "--", "--", "--", "--"],
    ["--", "--", "--", "--", "--", "--", "--", "--"],
    ["--", "--", "--", "--", "--", "--", "--", "--"],
    ["wp", "wp", "wp", "wp", "wp", "wp", "wp", "wp"],
    ["wR", "wN", "wB", "wQ", "wK", "wB", "wN", "wR"]
]
```

Svaki element polja predstavlja jedno polje na šahovskoj ploči. Prvo slovo označava boju figurice „b“ za crnu te w za bijelu. Drugo slovo označava vrstu figurice (p za pješaka, R za kulu, N za skakača, B za lovca,…). Ako na tom se ne nalazi figura to polje je označeno sa "--". Svakim potezom igrača stanje ovog polja automatski se ažurira.

Funkcija *get_valid_moves* sadrži algoritam dohvaćanja legalnih poteza. Zbog specifičnosti situacije kad je kralj napadnut (šah) osmišljen je „naivni algoritam“ koji radi u nekoliko koraka:

Commented [MĐ10]: Polje

Commented [MĐ11]: Ako

Commented [MĐ12]: Ne nalazi

Commented [MĐ13]: Staviti u italic

Rad naivnog algoritma:

1. Generiraj sve moguće poteze[1]
2. Odigraj svaki generirani potez te s protivničke perspektive generiraj sve njegove legalne poteze
3. Ako protivnik može napasti (tj. pojesti) mog kralja tada ukloni taj potez iz skupa legalnih poteza

Iako ovaj algoritam je ispravan vrlo je neučinkovit zbog velikog broja generiranih poteza.

Commented [MĐ14]: S

Commented [MĐ15]: Ako

[1] Ovdje razmatramo sve legalne poteze figurica te pri tome zanemarujemo činjenicu je li kralj u opasnosti (šah).

4. MiniMax algoritam za odabir optimalnog poteza

Prilikom igre agent mora moći donositi odluke o odabiru najboljeg poteza. Metoda *find_best_move* je zadužena za implementaciju algoritma minimax koji pretražujući stablo igre donosi odluku o optimalnom potezu. Problem koji se nameće pri definiranju ovakvog algoritma jest opis trenutnog stanja igre, točnije moramo znati koliko koja strana u određenom trenutku igre gubi ili pobjeđuje kako bismo evaluirali „dobrotu“ mogućih poteza. Zbog toga razvijen je „pohlepni algoritam“ koji će stanje evaluirati isključivo prema materijalnom kriteriju. Takva evaluacija nije sasvim točna jer položaj figurica na ploči čini nezanemarivo veliku razliku između igara. Ova strategija ima i određenih prednosti. Prva i očigledna prednost jest jednostavnost implementacije, a druga prednost je objektivnost pri evaluaciji. Ideja algoritma jest definirati koliko svaka figurica vrijedi te zatim nastojati igrati takve poteze koji bi maksimizirali ukupnu vrijednost igračevih figurica. Vrijednost figurica definirana je cijelim brojem koji se može proizvoljno odabrati no pri tome je važno da omjer reflektira stvarni omjer jačine figurica. U skladu s tim većina *chess engina* koristi omjer jačina kakav je prikazan u tablici 1 [3]:

Commented [MĐ16]: italic

Commented [MĐ17]: Igracevih, ili nesto drugo

Commented [MĐ18]: Definirana je

Commented [MĐ19]: Direktno referencirati tavlicu u tekstu (jacine figurica prikazane su u tablici xy)

Figurica	Jačina
Pješak	1
Lovac	3
Skakač	3
Kula	5
Kraljica	10
Kralj	Bilo koji broj jer ga obje strane nužno imaju

Tablica 1: Prikaz korištenih jačina figurica

Osim ovih vrijednosti definiramo dva specifična slučaja u igri. Ako se pronađe potez koji matira protivnika našoj sumi dodijeliti ćemo vrijednost konstante checkmate koja je

Commented [MĐ20]: Ako

postavljena na 1000 (kako bi simulirala beskonačnu prednost nad protivnikom). Ako se pronađe potez koji dovodi do izjednačenja (pat) tada naša suma postaje 0 kako bi uvijek bila veća od gubitka (što bi bili negativni brojevi). Trenutno implementirani algoritam najprije iz liste svih legalnih poteza prolazi kroz svaki potez, odigra ga te gleda sve protivničke poteze te ih odigra. Za svaki takav par poteza evaluira materijalno stanje te odabire onaj potez koji maksimizira igračevu materijalnu dobit. Obzirom da je pretraživanje dubine 2 dosta skromno algoritam ponekad ne vidi nijedan dobar potez koji bi odmah maksimizirao njegovu materijalnu prednost. U tom slučaju algoritam povlači nasumični potez iz skupa validnih poteza.

Commented [MĐ21]: Ako

Commented [MĐ22]: ?

5. Zaključak

Kroz ovaj seminarski rad prikazan je rad jednostavne implementacije igre šah u programskom jeziku Python. Igra šah odabrana je zbog njene kompleksnosti i nemogućnosti iscrpnog pretraživanja stabla igre, osim toga, igra je i jako popularna. Implementirani program koristi jedino *PyGame* biblioteku kako bi prikazao figurice i iscrtavao šahovsku ploču. Trenutno implementirani agent mogao bi pobijediti osobu koja samo zna osnovna pravila šaha. Igrači koji imaju iskustva s velikom lakoćom mogu pobijediti ovako implementiranog agenta. Glavni problem sa trenutnom implementacijom jest što ona ne omogućuje jednostavnu nadogradnju na pretraživanje s većim razinama. Ideja poboljšanja jest implementacija rekurzivnog algoritma koji bi omogućio pretraživanje sa većim dubinama. Također unaprjeđenja su moguća u vrednovanju stanja ploče. Algoritam bi morao osim brojnosti figurica (materijalni kriterij) gledati i njihove međusobne položaje (pozicijski kriterij). Nadalje, algoritmu se može dodavati i određena preferencija za pojedine poteze. Trenutno jedina preferencija je pobjeda (šah mat), no trebalo bi se dodati i preferencija za napad kralja (šah) i ostale poteze koji protivniku smanjuju broj legalnih poteza. Daljnja unaprjeđenja su moguća ugradnjom otvaranja, završnica i ostalih komponenti šahovske teorije koji mogu skratiti vrijeme pretraživanja te dati agentu dodatne informacije o stanju igre.

Commented [MĐ23]: s

6. Sažetak

Igra “šah” je vrlo popularna igra za dva igrača. Cilj igre je zadati šah-mat suparničkog kralju. Igra zbog velikog broja kombinacija (od 10^{120}mogućih igara šaha) nije pogodna za iscrpno pretraživanje već je neophodan algoritam poput *MiniMax-a* koji selektivno pretražuje poteze koji su vjerojatniji da dovedu igrača pobjedi. Implementirani algoritam je „pohlepni“ MinMax algoritam koji maksimizira materijalnu dobit. Kako bi agent mogao vrednovati različite poteze potrebno je napraviti tablicu vrijednosti figurica. Algoritam trenutno pretražuje samo dva poteza u dubinu te ukoliko ne pronađe potez koji maksimizira njegovu materijalnu dobit povlači nasumičan potez. Trenutna implementacija može pobijediti početnika u šahu no iskusniji igrači s lakoćom pobjeđuju ovakvu implementaciju zbog male dubine pretraživanja te nemogućnosti razmatranja pozicija figurica na ploči.

7. Literatura

[1] Šah - Wikipedia
[2] Shannonov broj
[3] Anatomija AI agenta za Šah - članak