IGRA ŠAH OSTVARENA METODOM MINIMAKS

Luka Terzić

Seminar 2 | FER

Sadržaj

1. Uvod 1

2. Grafičko korisničko sučelje 3

3. Metoda minimaks 4

4. Isplatne funkcije 6

5. Rezultati 8

Zaključak 9

Literatura 10

1. Uvod

Šah je jedna od najpopularnijih društvenih igara na svijetu. Igra se na ploči sa 64 polja koja su raspoređena u osam redaka i osam stupaca. Svaki igrač ima 16 figura – kralja, kraljicu, dva lovca, dva skakača, dva topa i osam pješaka. Cilj igre je napraviti šah mat na protivnikovog kralja. Šah je pozicija u kojoj je kralj pod izravnim napadom, a mat je pozicija u kojoj je kralj u šahu i u kojoj ne postoji validan potez za kralja u kojem bi izašao iz pozicije šaha. Također, postoji i mogućnost neriješenog rezultata.

Pravila šaha koja danas poznajemo pojavila su se u Europi u 15. stoljeću, ali su standardizirana tek krajem 19. stoljeća. Šah je igra koja ne uključuje nikakav faktor sreće. Stanje ploče vidljivo je svim igračima te ne postoje nikakve skrivene informacije. Igrači moraju predvidjeti poteze protivnika i pripremiti se za obranu i napad. Igra se smatra velikim intelektualnim izazovom za čovjeka jer zahtjeva visoku razinu vještine i iskustva igrača koji su ključni faktori za uspjeh u igri.

Budući da šah smatramo velikim intelektualnim izazovom za čovjeka, nije iznenađujuće da se šah pokazao vrlo zanimljivim za računarske znanstvenike koji su razvili brojne agente koji su pokušali igrati igru bolje od najboljih svjetskih velemajstora. Jedan od najznačajnijih trenutaka za igru šah dogodio se 1997. godine u kojoj je Deep Blue postao prvo računalo koje je uspjelo pobijediti svjetskog prvaka koji je u tome trenutku bio Garry Kasparov. Deep Blue je ekspertni sustav koji se pokretao na IBM-ovom superračunalu te njegov uspjeh smatramo jednim od najznačajnijih za područje umjetne inteligencije [1].

Događaja je prikazan je na slici 1.1.

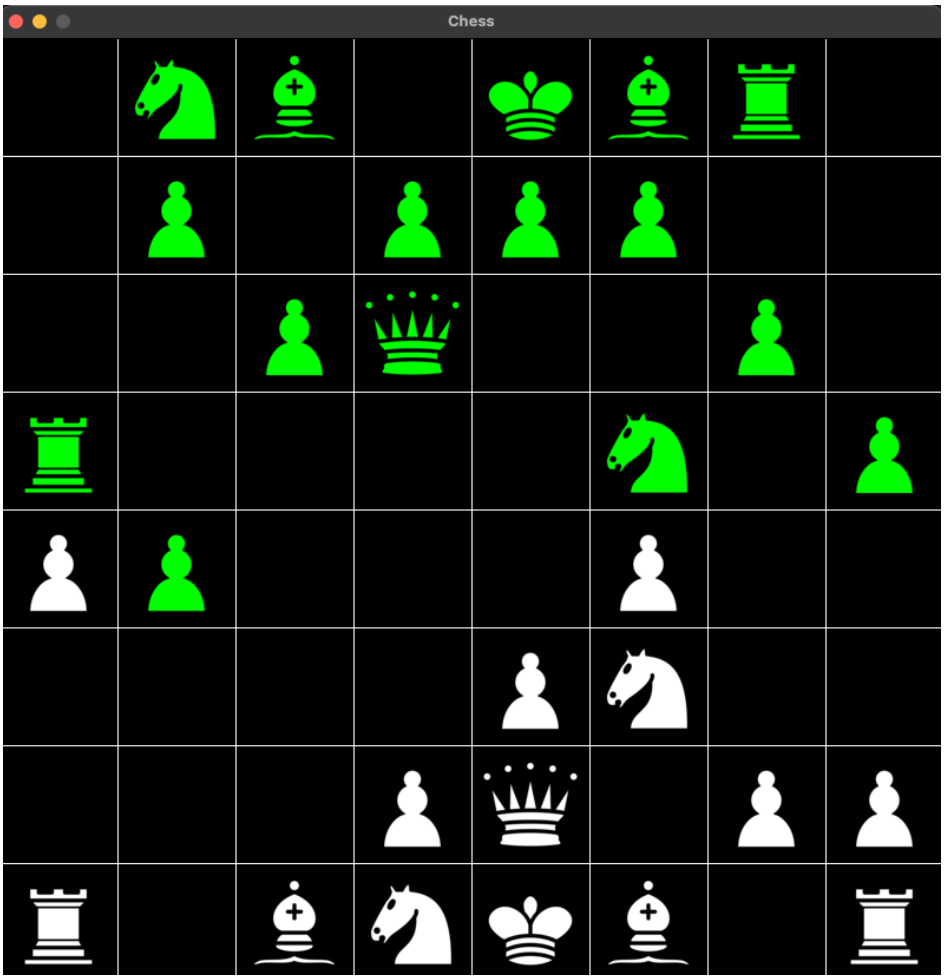
Sl. 1.1. Garry Kasparov protiv Deep Blue-a [2]

Do 2023. godine razvijeni su brojni agenti koji igraju šah te ih smatramo značajno snažnijima od bilo kojeg čovjeka. Razvoj ovih agenata doprinio je razvoju brojnih šahovskih teorija koje se danas koriste u kompetitivnom šahu.

2. Grafičko korisničko sučelje

Za potrebe rada razvijeno je grafičko korisničko sučelje u programskom jeziku python koristeći biblioteku pygame. Razvijena su tri načina korištenja programa. Prvi pokreće igru u načinu rada u kojemu mogu igrati dva korisnika. Drugi način omogućava korisniku da proba odigrati partiju protiv proizvoljno odabranog agenta. Treći način omogućava pokretanje partije u kojoj igraju dva proizvoljno odabrana agenta.

Razvijeno grafičko korisničko sučelje prikazano je na slici 2.1.



Sl. 2.1. Grafičko korisničko sučelje

3. Metoda minimaks

Agent koji je razvijen za potrebe rada koristi metodu minimaks. Metoda minimaks definira dva igrača. Igrač MAX (računalo) i igrač MIN (protivnik). Bitno je za naglasiti kako igrač MIN može biti ili računalo ili čovjek. Igrač MAX nastoji maksimizirati svoj dobitak, dok igrač MIN nastoji minimizirati dobitak igrača MAX. Igrači igraju naizmjenično, a optimalna strategija igrača MAX je strategija koja mu donosi najveći dobitak, uz pretpostavku da igrač MIN koristi istu strategiju. Svaki igrač koristi strategiju koja minimizira maksimalan gubitak [3].

U praksi je protivnikova strategija nepoznata i najvjerojatnije različita od strategije igrača MAX, pa zbog toga nije moguće savršeno predvidjeti sve protivnikove poteze . Zbog toga, kako bi povukli optimalan potez, svaki puta kada su na redu, igrači moraju nanovo izračunati svoju optimalnu strategiju, krenuvši od trenutne pozicije igre [3].

Minimaks radi pretraživanje u dubinu što znači da je njegova prostorna složenost $O(m)$, gdje je m dubina stabla pretraživanja. Međutim, vremenska složenost je $O(b^m)$, gdje je b faktor grananja igre. To je vrlo nezgodno jer igra šah ima velik faktor grananja s vrlo velikom mogućom dubinom stabla pretraživanja. Ovo nas dovodi do zaključka da sigurno nećemo moći pretraživati cijelo stablo u dubinu jer bi to iziskivalo veliku količinu vremena. U stvarnosti nemamo vremena potpuno pretražiti stablo sve do završnih čvorova [3].

Moramo donositi vremenski ograničene i nesavršene odluke. Pretraživanje treba presjeći na određenoj dubini d i napraviti procjenu vrijednosti isplatne funkcije uporabom heurističke funkcije. Igrači tipično imaju različite heurističke funkcije [3].

Na slici 3.1. prikazano je ostvarenje minimaks agenta za potrebe rada.

```
def min_maxN(BOARD, N):
    moves = list(BOARD.legal_moves)
    scores = []

    for move in moves:
        temp = deepcopy(BOARD)
        temp.push(move)

        outcome = temp.outcome()

        if outcome is None:
            if N > 1:
                temp_best_move = min_maxN(temp, N - 1)
                temp.push(temp_best_move)

            scores.append(eval_board(temp))

        elif temp.is_checkmate(): return move
        else: scores.append(0)

        scores[-1] = scores[-1] + eval_space(temp)

    if BOARD.turn: return moves[scores.index(max(scores))]
    else: return moves[scores.index(min(scores))]
```

Sl. 3.1. Programsko ostvarenje algoritma minimaks za igru šah

4. Isplatne funkcije

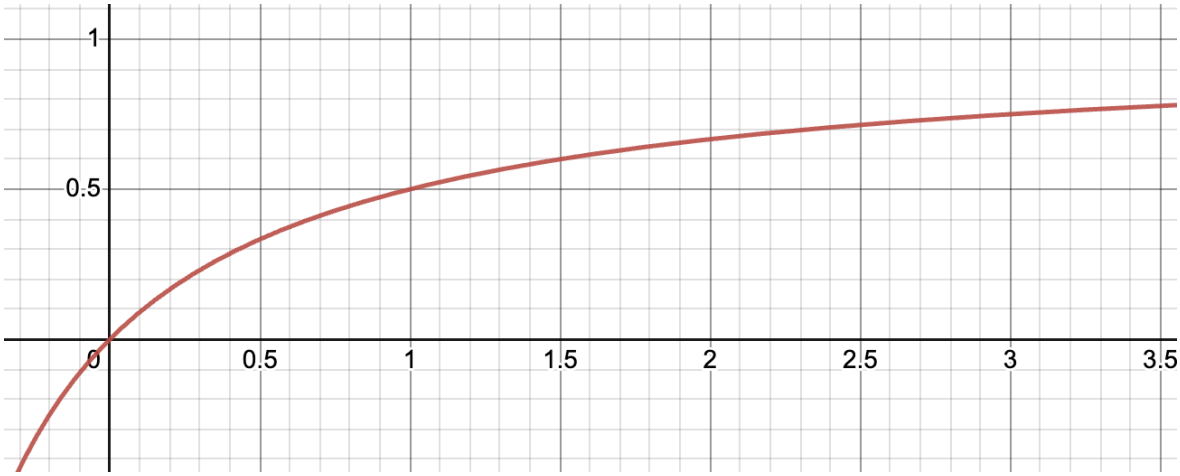
Za potrebe minimaksa za šah iskorištene su dvije isplatne funkcije. Prva isplatna funkcija izvedena je kao zbroj vrijednosti igračevih figura. Pješak ima vrijednost jedan, lovac i skakač imaju vrijednost tri, top ima vrijednost pet, a kraljica ima vrijednost devet.

Nažalost, korištenje samo zbroja vrijednosti svih igračevih figura nije dovoljno za ostvarenje dobrog minimaks igrača jer ovakva isplatna funkcija ne uzima u obzir poziciju figura na ploči. Pozicija figura ima krucijalan značaj u evaluaciji trenutnog statusa igrača.

Iz ovog razloga uvedena je dodatna isplatna funkcija koja pretpostavlja da ako igrač ima velik broj mogućih validnih poteza, da je onda i u dobroj poziciji. Pretpostavljamo da što je veći broj mogućih poteza, da je i bolja pozicija igrača.

$$vrijednost = \frac{x}{a + x}, \qquad x = broj\ svih\ mogućih\ poteza, \qquad a > 0$$

Odabrali smo ovu dodatnu funkciju isplate jer je strogo rastuća i ograničena na interval $< 0{,}1 >$. Također, ova funkcija ima važno svojstvo da ne može dodati ili oduzeti veću vrijednost od vrijednosti pješaka koji je figura koji donosi najmanju vrijednost u primarnoj isplatnoj funkciji. Dodatna isplatna funkcija prikazana je na slici 4.1.



Sl. 4.1. Dodatna isplatna funkcija uz $a = 1$

Kako bi se uveo nedeterminizam u izračunu isplatne funkcije, za vrijednost a uzima se nasumični cijeli broj na intervalu $[1, 100]$.

Ostvarenje isplatnih funkcija prikazano je na slici 4.2.

```python
def eval_board(BOARD):
    score = 0
    pieces = BOARD.piece_map()
    for key in pieces:
        score += scoring[str(pieces[key])]
    return score


def eval_space(BOARD):
    no_moves = len(list(BOARD.legal_moves))
    value = (no_moves / (random.randint(1, 100) + no_moves))
    if BOARD.turn: return value
    else: return -value
```

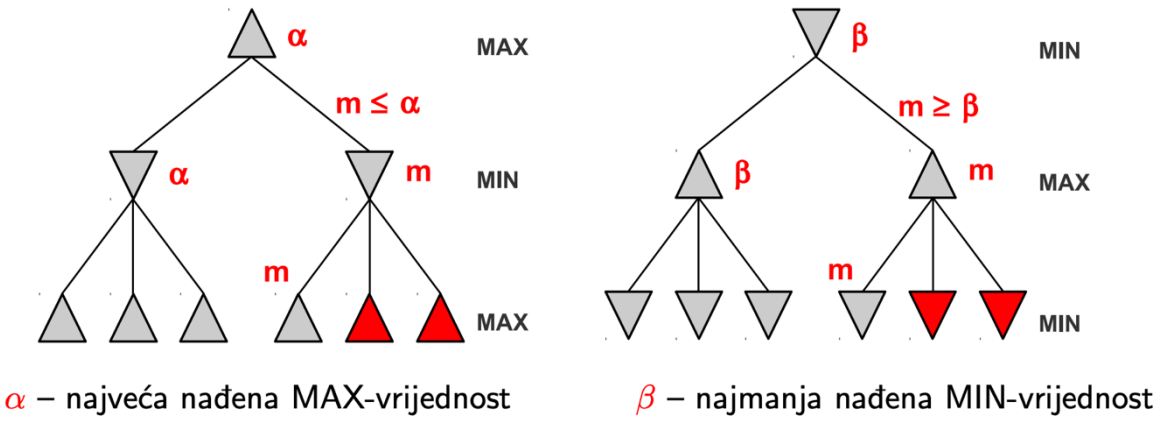
Sl. 4.2. Ostvarenje isplatnih funkcija

5. Rezultati

Postignuti minimaks agent koji podrezuje stablo pretraživanja na dubini razine dva postiže odlične rezultate te uspješno može pobijediti nove igrače šaha. Također, agenti koji rade podrezivanje na razini dva ili tri daju poteze gotovo instantno. Nažalost, kada se poveća dubina podrezivanja na četiri prosječno vrijeme davanja poteza mjereno na 10 igara iznosi 22.13 sekunde. Ako čovjek igra protiv ovakvog agenta to mu već može stvoriti neugodnosti jer za svaki potez agenta mora čekati značajnu količinu vremena.

Kako bi se ostvario bolji agent koji će brže davati poteze moglo bi se implementirati alfa-beta podrezivanje. Alfa-beta podrezivanje podrezuje kad god se za neki čvor ustanovi da potezi ni u kojem slučaju ne mogu biti povoljniji od nekog već istraženog poteza. Ako podrezujemo ispod MIN čvora, riječ je o alfa podrezivanju. Suprotno, ako podrezujemo ispod MAX čvora, riječ je o beta podrezivanju [3].

Postupak alfa-beta podrezivanja prikazan je na slici 5.1.



Sl. 5.1. Alfa-beta podrezivanje

Kao buduće unaprjeđenje za algoritam minimaks koji je implementiran za potrebe ovoga rada, poželjno je implementirati alfa-beta podrezivanje kako bi se skratilo vrijeme čekanja na potez agenta. Također, kao moguće poboljšanje, moguća je implementacija isplatne funkcije koja radi bolju evaluaciju pozicije igračevih figura na ploči.

Zaključak

U sklopu rada implementiran je agent koji koristi metodu minimaks za određivanje poteza u igri šah. Također, ostvareno je grafičko korisničko sučelje preko kojeg je moguće igranje igre te pregled rada agenta. Za potrebe algoritma, ostvarene su dvije isplatne funkcije. Prva isplatna funkcija uzima u obzir sumu vrijednosti igračevih figura na ploči dok dodatna isplatna funkcija uzima u obzir poziciju figura.

Ostvarena metoda minimaks postiže dobre rezultate. Međutim, ako se stablo pretraživanja podrezuje na dubini većoj ili jednakoj četiri, algoritam postaje prespor za ugodno igranje čovjeka protiv agenta. Kao moguće unaprjeđenje na trenutnu izvedbu, može se implementirati alfa-beta podrezivanje te bolja isplatna funkcija za evaluaciju pozicije figura na ploči.

Literatura

[1] https://en.wikipedia.org/wiki/Chess

[2] https://www.forbes.com/sites/davidewalt/2011/05/03/kasparov-vs-deep-blue/?sh=7978635930f8

[3] Šnajder J., Igranje igara.

Igra šah ostvarena metodom minimaks

Sažetak

Ovaj rad bavi se razvojem agenta koji koristi metodu minimaks za određivanje poteza u igri šah. Daje se opis rada i implementacija metode minimaks. Kao dio implementacije metode minimaks ostvarene su isplatne funkcije koje uzimaju u obzir sumu svih vrijednosti i poziciju igračevih figura na ploči. Prikazani su rezultati implementacije i dana su moguća poboljšanja u izvedbi. Razvijeno je grafičko korisničko sučelje kojim se može prikazati rad algoritma.

Ključne riječi: igra šah, metoda minimaks, isplatna funkcija, grafičko korisničko sučelje

The game of chess achieved using the minimax method

Abstract

This thesis deals with the development of an agent that uses the minimax method to determine moves in the game of chess. A description of how minimax works and the implementation of the algorithm is provided. As a part of the implementation of the minimax method, payoff functions were created that consider the sum of all values and the position of the player's pieces on the board. The results of the implementation are presented, and possible improvements in performance are given. A graphical user interface has been developed to display the algorithm's operation.
Keywords: the game of chess, minimax algorithm, payoff function, graphical user interface