Универзитет ,,Св. Кирил и Методиј’’ Скопје

Факултет за информатички науки

и компјутерско инженерство

Анализа на шаховски агент: Важноста на длабочина кај Minimax алгоритам, и негова имплементација

|  |  |
| --- | --- |
| Ментор:  проф д-р Соња Гиевска | Студент:  Филип Гавриловски  231136 |

Скопје, 2025

Содржина

[Вовед 3](#_Toc203669015)

[Minimax и евристички функции 3](#_Toc203669016)

[Minimax 3](#_Toc203669017)

[Функција на проценка 4](#_Toc203669018)

[Евристики 4](#_Toc203669019)

[Оптимизација 9](#_Toc203669020)

[Alpha-Beta поткастрување 9](#_Toc203669021)

[Редослед на потези 9](#_Toc203669022)

[Табела на транспозиции 10](#_Toc203669023)

[Quiescence пребарување 10](#_Toc203669024)

[UCI протокол – Поставување околина за тестирање 11](#_Toc203669025)

[Резултати 12](#_Toc203669026)

[Заклучок 14](#_Toc203669027)

[Препораки и забелешки 14](#_Toc203669028)

[Референци 16](#_Toc203669029)

# Вовед

Шахот е една од најпознатите игри во светот и како стратегиска и детерминистичка игра долго време претставува инспирација за развојот на вештачката интелигенција. Шаховски агенти се развиваат повеќе од 50 години и со напредокот на хардверот постојано бележат значителен раст во перформансите. Deep Blue на IBM во 1997г. успеал да го победи тогашниот светски шампион, а денес Stockfish е убедливо посилен од најдобрите човечки играчи.

Еден од алгоритмите кој останал присутен низ сите генерации на шаховски агенти и сè уште е во широка употреба е Minimax алгоритмот. Природата на шахот, која бара од играчите постојано да размислуваат повеќе потези однапред, совршено се совпаѓа со начинот на работа на Minimax алгоритмот, кој ги разгледува сите различни сценарија.

Можеме да препоставиме дека длабочината на дрвото што се пребарува правопропорционално влијае на квалитетот на одлуките на агентот. Но, со зголемување на длабочината се појавува експоненцијален раст на бројот на потези кои треба да се разгледаат, што значително влијае на временските перформанси. Целта на овој проект е да се анализираат току тие промени, поточно како промената на длабочината влијае на перформансите и временската сложеност на шаховскиот агент.

# Minimax и евристички функции

## Minimax

Minimax е основен алгоритам што се користи во игри со совршена информација, како шах. Функционира со претпоставката дека двата играчи играат оптимално така што, едниот, вообичаено страната која ја евалуира агентот, се обидува да ја максимизира вредноста на позицијата, додека другиот да ја минимизира.

Алгоритмот симулира потези наизменично за двата играчи земајќи ги предвид сите можни потези и формирајќи дрво на можни позиции. На краевите од дрвото, наречени листови, се користи **евристичка функција за евалуација на тековната состојба**. Потоа, вредностите се враќаат наназад, се избира максимум кога е потег на максималниот играч, и минимум кога е потег на минималниот.

Длабочината всушност одредува колку потези нанапред ќе гледа агентот.

## Функција на проценка

Функцијата на проценка се користи кога алгоритмот ќе пристигне до лист во дрвото и треба да даде соодветна оценка на тековната состојба - шаховската табла. Лист може да претставува пристигнување на одредената длабочина или пристигнување до состојба каде нема повеќе легални потези.

Оценката може да биде пресметана на два начина: така што максималниот играч секогаш ќе биде базиран на одредена боја (пример белите фигури), или максималниот играч да е секогаш агентот, без оглед на бојата, пристап познат како агентно-базирана евалуација. Во мојата имплементација јас го користам вториот пристап.

Функцијата на проценка ги зема предвид сите предодредени евристики и резултатот кој го враќа е сума на вредностите добиени од секоја од нив. Со подобрување на опфатот и презицноста на евристиките, како и со внимателно подесување на поените кои тие ги даваат може да се зголеми ефикасноста и точноста на евалуацијата.

## Евристики

Функцијата на проценка се состои од повеќе независни евристики, секоја од кои анализира специфичен аспект од шаховската позиција. Евристиките можат да ја намалуваат или зголемуваат оценката така што даваат пенали, или награди, за позиции кои не се поволни, или се поволни, соодветно. Тие се комбинираат за да дадат слика за тоа колку е поволна позицијата за агентот.

Во продолжение следи објаснување на секоја користена евристика:

**Материјална вредност**

Материјалната предност е едноставна, но претставува еден од најзначајните фактори во шаховската проценка. Секоја фигура има одредена вредност и оваа евристика пресметува нето-разлика во материјалот помеѓу двете страни.

Во мојата имплементација се користат следните тежини[1](#ref1):

**Пешак - 100**

**Коњ - 320**

**Ловец - 330**

**Топ - 500**

**Кралица – 900**

**Крал – 20 000**

За собирање на материјалната вредност се итерира низ сите фигури на таблата, и за оние кои припаѓаат на страната што се евалуира (агентот), се додава нивната тежина, додека за другата страна се одзема. Постојат и имплементации кои доделуваат тежина 1 за пешаци, и соодветно за останатите фигури, но овој пристап, наречен пристап со **„центи-пешаци“** овозможува поголема прецизност при правење на евалуации и појасна интерпретација на добиениот резултат. Разлика од +100 значи предност од еден пешак, +350 значи приближно 3.5 пешаци, што интуитивно ја прикажува позиционата супериорност на едната страна.

Оваа евристика ни е доволна за да направиме еден базичен шаховски агент, но секако дека не е доволно, бидејќи некогаш вредноста на материјалот не е толку важен и може да има ситуации кога ќе сакаме да изгубиме на материјална вредност, со цел да ја ослабнеме позицијата на противникот од некој друг аспект, како „отварање“ на кралот, удвојување на пешаци во една колона и слично.

**Табели за фигури (Piece-square tables)**

Кога евалуираме фигури на шаховската табла не ни е доволно само да ги гледаме нивните материјални вредности. Важен фактор е и каде на таблата се наоѓаат, на пример коњ кој се наоѓа на работ на таблата е многу поограничен од коњ кој се наоѓа поблиску до центарот.

За да се опфати овој аспект, се воведуваат **табели за фигурите** (Piece-square tables) коишто даваат одредено зголемување или намалување на проценката, согласно квадратот на кој се наоѓа секоја фигура.

Дополнително, како што движиме од почетокот на играта кон завршницата се менуваат овие оптимални позиции каде што може една фигура да се наоѓа. На почетокот и средината, најдобро е кралот да ни е заштитен и по можност да ни е на првиот ранг (гледајќи од наша страна), но кон крајот на играта кога бројот на фигури е многу помал, кралот може да ни е многу корисен како активна фигура.

Затоа користиме две посебни табели, една за средишната игра и една за крајната игра, а соодносот меѓу нив се менува според фазата на партијата.

Одредување на фазата на партијата не е бинарно, туку постепено се менува. За ова, се користи **PeSTO** функција на проценка, каде транзицијатата е претставена како број од 0 до 24, согласно фигурите присутни на таблата. Тежините што беа користени за табелите беа преземени директно од PeSTO[2](#ref2).

**Контрола на центарот**

Централните полиња (d4, d5, e4, e5) се клучни за флексибилност и контрола на таблата. Оваа евристика брои колку фигури го напаѓаат центарот и колку противнички фигури го бранат. Колку е поголема контролата на агентот врз овие полиња, толку е повисок резултатот.

**Развој на фигури**

Во отворањето, правилно развиените фигури играат огромна улога во стабилизирањето на позицијата. Оваа евристика казнува коњи и ловци кои сè уште се на почетните позиции на почетокот на играта, како и топови кои не се поврзани или се блокирани. Така, агентот се поттикнува кон активна игра уште од самиот почеток.

**Структура на пешаците**

Пешаците често ја дефинираат природата на позицијата. Евристиката ги анализира следниот тип на пешаци:

**Дуплирани** пешаци (два на иста колона) добиваат казна поради намалена подвижност.

**Изолирани** пешаци (без поддршка од соседи) добиваат казна поради ранливост.

**Поминат** пешак (без противнички пешак пред себе на истата или соседна колона) добиваат бонус, бидејќи има потенцијал за промоција.

Ова овозможува агентот да го разбира долгорочниот потенцијал на структурата на пешаците.

**Развој на пешаците**

Во отворањето, агресивното и правилно позиционирање на пешаците е од клучно значење. Оваа евристика доделува поени на пешаците кои се насочуваат кон центарот (особено на колоните d и e), и ја казнува пасивноста, особено ако пешаците остануваат на втората редица долго време.

**Отворени и полуотворени колони за топови**

Топовите се најмоќни кога се поставени на колони без пешаци (**отворени**) или само со противнички пешаци (**полуотворени**). Оваа евристика ги наградува таквите поставувања, бидејќи тие овозможуваат поголема контрола и притисок врз позицијата на противникот.

**Безбедност на кралот**

Оваа евристика ја анализира позицијата и заштитата на кралот. Ако кралот има изведено рокада и е зад бариера од пешаци, добива бонус поени. Ако нема рокада, или пешачкиот штит е разбиен, се доделува соодветна казна. Истото се пресметува и за противничкиот крал со цел и оваа евристика, како и останатите, да даде релативна оценка.

**Напредок при предност**

Кога агентот има значајна материјална предност, оваа евристика го поттикнува да ја конвертира таа предност во победа. Таа ја наградува активноста на кралот (особено во крајната игра), напредокот на пешаците, и поставувањето на фигури поблиску до кралот на противникот. Целта е да се избегне пасивност и да се продолжи со притисок.

Целта на сите евристики е агентот да ги зема предвид различните аспекти од играта и да носи калкулирани одлуки. На овој начин, агентот „размислува“ повеќедимензионално и рационално, наместо да избира потези кои на прв поглед може да изгледаат поволно, но всушност водат кон полоша позиција кога се разгледа целата слика.

# Оптимизација

## Alpha-Beta поткастрување

**Alpha-Beta** поткастрување е класична техника за оптимизација на Minimax алгоритмот која овозможува значително намалување на бројот на разгледани состојби, без да се жртвува точноста на резултатот. Наместо да се анализираат сите можни потези на дадена длабочина, алгоритмот користи два параметри – **α** , најдобра вредност за максималниот играч, и **β**, најдобра вредност за минимизирачот, за да отсече поддрвја кои немаат влијание врз финалната одлука. На пример, ако веќе сме сигурни дека еден потег ќе биде подобар од другите достапни опции, нема потреба да ги проверуваме преостанатите.

## Редослед на потези

Alpha-Beta поткаструвањето може драматично да го намали времето на пребарување, но во најлош случај кога најдобрите можни потези се блиску до крајот на пребарувањето, поткаструвањето има минимален ефект. Поради тоа се јавува потреба од распоредување на потезите. Доколку според некои евристики успееме најдобрите потези да ги разгледаме први, шансите за рано отсекување растат, со што се штеди значително време. Во имплементацијата се користат повеќе техники за сортирање и приоретизирање на потези.

Се применува **MVV-LVA** (Most Valuable Victim - Least Valuable Attacker), каде се дава приоритет на потези кои земаат поскапа фигура со поевтина. Покрај тоа, потезите што даваат шах, рокада, унапредување на пешак или контрола на центарот се исто така наградени со повисоки поени[3](#ref3).

Се користат и **потези убијци (**killer moves), потези кои претходно резултирале со отсекување на иста длабочина, за уште подобар редослед.

Дополнително се дава предност на потези кои даваат шах, рокада, промоција и слично. Ова сортирање е клучно за подобрување на брзината и квалитетот на пребарувањето, и во одредени тестови даваше и ~40% подобрување во времето.

## Табела на транспозиции

Во шахот, иста позиција може да се појави на повеќе начини (т.н. транспозиции). За да се избегне повторна анализа на веќе позната позиција, се користи **табела на транспозиции** – кеш структура која ги чува резултатите од претходно разгледани позиции, идентификувани преку **Zobrist hashing**[**4**](#ref4)**,**[**5**](#ref5). Оваа табела содржи информации како евалуација, длабочина на пребарување, тип на јазол (точна вредност, горна или долна граница), и најдобар потег од таа позиција. Пред секоја анализа, се проверува дали моменталната позиција е веќе обработена. Доколку е, и доколку претходно е разгледана на иста или поголема длабочина, резултатот директно се користи. На овој начин се елиминираат дупликати и се добива големо забрзување, особено во длабоки пребарувања или во комплицирани позиции.

## Quiescence пребарување

Класичниот Minimax често страда од проблем наречен **„horizon effect”**, каде евалуацијата е прекината на граница на длабочина, и не ја гледа катастрофата или добивката што доаѓа со само неколку потези подлабоко.

На пример, во последното ниво на пребарувањето може да има потег каде што кралицата зема слободен ловец, но доколку би разгледале едно ниво подоле може да дознаеме дека всушност ловецот не е слободен, и сме ја предале кралицата на противникот. Потег што на прв поглед ни дал +330 материјална предност, во следниот потег преминува во -570.

За да се ублажи овој проблем, се користи **quiescence search**, дополнително пребарување кое продолжува од „нестабилни” позиции, односно оние каде што има достапни потези кои значајно би ја смениле позицијата[6](#ref6). Со користење на оваа техника, можеме да даваме посигурни резултати, така што ќе ги евалуираме само „**тивките**“ потези, поточно оние кои не водат до материјална вредност, шах и/или промоција.

Во мојата имплементација, класичното negamax quiescence пребарување е малку модифицирано, така што се користи minimax со Alpha-Beta, кој гледа потези до одредена максимална длабочина. Ова не е класична оптимизација на време, баш напротив значително додава на временската комплексност, но практично ги подобрува резултатите на пребарувањето и ја прави евалуацијата многу постабилна.

# UCI протокол – Поставување околина за тестирање

За да се извршат објективни тестирања на шаховскиот агент, потребно е да се воспостави стандардизирана комуникација помеѓу агентот и други шаховски агенти или графички интерфејси. За таа цел се користи **UCI (Universal Chess Interface)** протоколот, кој претставува широко прифатен протокол за комуникација помеѓу шаховски агенти и интерфејси[7](#ref7). Протоколот дефинира начин на испраќање на позиции, потези и команди, како и на враќање на потези и евалуации од страна на агентот. За тестирање користев алатка наречена **Cutechess-cli**, верзија на Cutechess наменета за употреба во командна линија, која овозможува автоматизирано одигрување на игри помеѓу агенти користејќи го UCI протоколот[8](#ref8). Cutechess-cli се грижи за контролирање на времето и текот на играта, како и запишување на резултатите и генерирање на **.pgn** фајлови (Portable Game Notation), кои содржат опис на целата игра, потег по потег, заедно со метаинформации како имиња на агентите, резултат, време и слично[9](#ref9). Овие PGN фајлови се исклучително корисни за анализа на однесувањето на агентот. Во следниот дел ќе се анализираат резултатите од повеќе тестирања направени со Cutechess-cli, со цел да се оцени влијанието на длабочината на пребарување врз силата и стабилноста на агентот.

# Резултати

За да се добие објективна проценка на силата на агентот во различни конфигурации, направив серија на тестирања против различни верзии на **Stockfish**, конфигурирани на различни Elo нивоа, почнувајќи од 1600. Агентот беше тестиран со длабочини на пребарување од 1 до 4(означени како Д1, Д2, Д3, Д4), каде што секоја конфигурација играше 6 игри против дадената верзија на Stockfish.

Начинот на тестирање беше ако мојот агент победи барем една партија во тој сет, продолжуваше на следна, посилна верзија на Stockfish (+200 Elo). Ако не победи ниту една партија, тестирањето за таа конфигурација се прекинува.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Stockfish 1600 | Stockfish 1800 | Stockfish 2000 | Stockfish 2200 |
| Д1 | 0W 0D 6L | / | / | / |
| Д2 | 2W 0D 4L | 0W 1D 5L | / | / |
| Д3 | 3W 1D 2L | 2W 1D 3L | 1W 0D 5L | 0W 0D 6L |
| Д4 | 4W 0D 2L | 3W 0D 3L | 2W 0D 4L | 0W 0D 6L |

(W – победа, D – реми, L – пораз)

Резултатите покажаа очекуван тренд, зголемувањето на длабочината значајно ја подобрува силата на агентот. Д1 не успеа да победи ниту еднаш, дури ни против Stockfish 1600. Д2 постигна 1 победа против 1600, но беше сопрен на 1800. Д3 беше поконзистентен, со победи против 1600, 1800 и дури 2000, но падна на 2200. Д4 се покажа најдобро, со повеќе победи од Д3 против сите противници до 2000, но сепак, и покрај подобрената игра, неговиот “плафон” остана сличен како кај Д3, бидејќи и тој убедливо беше елиминиран на 2200.

Дополнително, може да се погледне просечното времетраење на една партија за секоја длабочина:

|  |  |
| --- | --- |
| Д1 | 3,9 минути |
| Д2 | 6,20 минути |
| Д3 | 11,47 минути |
| Д4 | 28,06 минути |

**Забелешка**: во времето е земено и времето кое му е потребно на Stockfish за да направи потег, но и ваквата информација ни дава добар генерален преглед на потребното време на различни длабочини.

Како и очекуваното, со зголемувањето на длабочината драстично се зголемува и времетраењето на играта. Подобрувањето на перформансите доаѓа со цена во форма на зголемена временска сложеност.

**Забелешка**: 6 партии се релативно мал примерок, но бројката беше избрана поради практични причини, бидејќи со зголемување на длабочината игрите стануваат многу подолги. Секако дека со поголем број на партии ќе се добијат многу попрецизни резултати, и ќе може да се донесе посигурен заклучок.

Сите тестови се извршуваат на MacBook Pro со M3 Pro чип. Резултатите во однос на времетраење може да варираат на машина со друг хардвер, но самите исходи, всушност перформансите на агентите, не би требало да се менуваат.

# Заклучок

Заклучокот од овие тестови е дека зголемувањето на длабочината има големо влијание на перформансите, но со секој следен скок ефектот станува помал. Д4 е неспорно посилен од Д3, но разликата веќе не е толку драматична како меѓу Д2 и Д3. Ова сугерира дека по одредена точка, подобрувањата на агентот треба да се насочат кон усовршување на евристиките, како и можеби додавање нови со цел опфаќање детали од играта кои не се земени предвид, наместо само зголемување на самата длабочина.

# Препораки и забелешки

Кодот за имплементирање на овој агент е напишан во python. Позитивното за неговата употреба е што веќе има постоечка python-chess библиотека која нуди многу различни функции кои можат да се корисни за претставување на состојбата, генерирање легални потези и слично. Истото ова доколку би требало да се пишува самостојно, може да земе многу време и многу да ја зголеми комплексноста на проектот, а нема корелација со неговата цел - анализа и имплементација на алгоритмите. Едноставната синтакса на python, во комбинација со готовите функции кои ги нуди python-chess, кои се веќе оптимизирани со користење на битмапи, бит операции и слично, овозможуваат многу побрзо да се стигне до делот релевантен за овој проект. Користејќи ја библиотеката pygame направив и графички интерфејс (GUI) за играње против агентот, кој дополнително многу ми помогна кога сакав сам да тестирам одредени аспекти.

Од друга страна, бидејќи python е интерпретиран јазик, перформансите се многу полоши од компајлираните јазици како C++, Rust и слично, и тоа многу се забележува кога се извршуваат вакви програми каде перформансите се клучни. Доколку истиот код се напише во некој друг јазик, длабочината од 4 би се разгледувало многу побрзо и тоа би отворило можност да се проба и некоја поголема длабочина, а со тоа и перформансите на агентот би требало да се подобрат. Поради тие причини, доколку некој сака да прави шаховски агент, но целта му е да добие оптимални перформанси и има доволно слободно време, препорачано е да користи некои од претходно наведените јазици.

Како што напоменав, minimax алгоритмот има широка употреба кај шаховските агенти, но со развојот на технологијата и напредокот на хардверските ресурси, во последно време има многу помодерни техники за правење на шаховски агенти, кои можат да донесат и многу подобри перформанси. Некои од нив се учење со поттикнување, длабоко учење и слично.

Во репозиториумот на овој проект е достапен целиот изворен код, како и .pgn датотеките со сите симулирани игри[10](#ref10).

# Референци

1. <https://www.chessprogramming.org/Simplified_Evaluation_Function>

1. <https://www.chessprogramming.org/PeSTO%27s_Evaluation_Function>

1. <https://www.chessprogramming.org/MVV-LVA>

1. <https://www.chessprogramming.org/Transposition_Table>

1. <https://www.chessprogramming.org/Zobrist_Hashing>

1. <https://www.chessprogramming.org/Quiescence_Search>

1. <https://www.chessprogramming.org/UCI>

1. <https://github.com/cutechess/cutechess>

1. <https://en.wikipedia.org/wiki/Portable_Game_Notation>
2. <https://github.com/gavro081/fichess>

Дополнително користени ресурси:

* <https://en.wikipedia.org/wiki/Chess_engine#Limiting_an_engine's_strength>
* <https://github.com/healeycodes/andoma>
* <https://github.com/JanaAngjelkoska/Mini-vs-Expecti>
* <https://healeycodes.com/building-my-own-chess-engine>
* <https://en.wikipedia.org/wiki/Stockfish_(chess)>