

Toteutusdokumentti

Ohjelman rakenne

Ohjelma sisältää yksinkertaisen tekstimuotoisen Reversi-pelin, sekä siihen liittyvän tekoälyn. Tekoäly koostuu useista erillisistä moduuleista, jotka yhdessä muodostavat kokonaisuuden, jossa keskipeliä arvioidaan heuristikalla ja loppupeli ratkaistaan täydellisellä minimax-haulla.

Pelin toimintalogiikka ja sääntöjen tarkistus sijaitsevat erillisessä moduulissaan (board.py) ja tämä on täysin erillään tekoälystä. Tässä moduulissa suoritetaan laillisten siirtojen tarkistus ja pelaus, nappuloiden käantö, vuoron vaihto sekä pelin loppumisehtojen tarkistus.

Keskipelin heuristiikka ja arvointifunktio (evaluate.py), koostuu kolmesta osasta:

1. Matriisista, jossa jokaiselle ruudulle on täsmennetty arvo ruudun strategisen tärkeyden perusteella. Käytetty matriisi perustuu yleisesti käytettyyn Reversi-malliin mutta sitä on muokattu huomioiden muut ohjelman heuristiikan erityispiirteet.
2. Mobility-score, joka on erotus omien ja vastustajan siirtojen määrästä. Mobilityn tavoitteena on pitää omien siirtovaihtoehtojen määrä suurena, jotta löydetään arvokkaita polkuja ja estetään se, että vastustaja voi pakottaa siirtoihin. Lisäksi mobility auttaa välttämään tilannetta, jossa hävittäisiin keskipelissä siitä syystä, että vastustaja saa kaikki napit itselleen.
3. Kolmas osa heuristiikkaa ottaa huomioon sen, että arvomatriisi ei enää päde, kun kulma on saatu haltuun, silloin kulman viereisten nappien arvo nousee koska ne muuttuvat vakaaksi ja niitä ei voi enää käantää, tämä toteutetaan lisäämällä arvobonus kulman viereisiin ruutuihin.

Nämä kolme osa-alueita yhdistetään valitun painotuksen mukaan muodostamaan lopullinen arvio pelitilanteesta.

Edellistä arvointifunktiota käyttää Minimax-keskipeli (minimax_midgame.py) sisältää minimax-haun alpha-beta-karsinnalla, jota on tehostettu siirtojen esijärjestämällä (reorder.py) sekä transpositio-kirjastolla, joka tallentaa aiemmin löydettyjä parhaita siirtoja ja nostaa ne ensiksi arvioitavaksi, tehostaen näin minimaxin toimintaa ja mahdollistaen suuremman hakusyvyyden.

Iteratiivinen syventäminen (iterative_deepening_midgame.py) puolestaan mahdollistaa minimax_midgamen tehokkaan käytön varmistamalla, että hakua jatketaan aina asetettuun aikarajaan asti.

Reversin loppupelissä on kyse vain pisteen maksimoinnista eikä mikään heuristiikka voi tarjota siihen lisäarvoa, jos samalla syvyydellä pystytään ratkaisemaan parhaat loppupisteet. Tämän vuoksi loppupelissä käytetään erillistä minimaxia (minimax_endgame.py) pisteen maksimointiin ja ai.move.py joka kontrolloi AI-pelaajan siirtoja, siirtää päättöksen siirroista sille, kun peliä on jäljellä 10 siirtoa.

Siirronvalintalogiikka (`ai_move.py`), sisältää kaikki tarvittavat haut: midgamen, endgamen, random_moven varmistamaan, että poikkeustilanteissakin jokin siirto palautetaan (vaikkakaan tätä ei pitäisi ikinä tapahtua). Lisäksi se kutsuu myös erillistä riskimoodia, jos endgame-arvo on tappiollinen, pyrkien pitämään näin voiton mahdollisuudet suurempana sen sijaan että pyrittäisiin häviämään mahdollisimman pienin luvuin.

Aikavaativuudet ja suorituskyky

Minimaxin teoreettinen aikavaativuus $O(b^d)$, missä b on haarautumisluku ja d on syvyys. Tätä voidaan ennaltaan tehostaa alpha-beta-karsinnalla parhaimillaan tasolle $O(b^{d/2})$, koska voimme jättää haarat, joihin emme pääse, jos vastustaja pelaa täydellisesti kokonaan huomioimatta. Ohjelmassa Alpha-beta-karsintaa on tehostettu siirtojen transpositio-kirjastolla, joka tallentaa aiemmin parhaaksi valittuja siirtoja ja siirtää ne ensin käsiteltäväksi, ja myösken muissa tapauksissa esijärjestämällä siirrot niin että oletettavasti parhaat siirrot arvioidaan ensin. Käytännössä `minimax_midgame` laski keskimäärin viiden tai kuuden siirron syvyyteen, kun aikarajana oli 2 sekuntia, mutta parhaimillaan pääsi yli 10 siirron syvyyteen. Tämä tietenkin riippuu siirtovaihtoehtojen määrästä, joka ei pelin edetessä ole vakio. Loppupelin `minimax_endgame` toimii poikkeuksetta tehokkaasti 10 siirron syvyydellä koska myös siirtovaihtoehdot pelin lopuksi ovat usein rajalliset. Useimmiten myös 11 siirron syvyys toimi testatessa viiveettä mutta tässä kohtaa hidastumista voi alkaa jo tapahtua.

Puutteet ja parannusehdotukset

Heuristiikkaa on mahdollista kehittää vielä huomattavasti pelkästään hienosäättämällä pelin sisäistä mekanikkaa, tämä tosin on hyvin hidasta ja vaatii testausta, jota on vaikea toteuttaa esimerkiksi tilastollisilla metodeilla koska tekoäly tekee aina samat siirrot samassa tilanteessa oman mallinsa mukaan. Myösken heuristiikka voisi hyötyä vielä tarkemmin määritellystä "varmojen" ruutujen laskennasta myös keskipelin aikana. Minimaxien osalta on myös mahdollista parantaa laskentasyvyyttä entisestään.

Laajojen kielimallien käyttö

Ohjelman tai dokumenttien toteutuksessa ei ole käytetty laajoja kielimalleja. ChatGPT 5.1 mallia on käytetty aiheeseen perehtymiseen yleisellä tasolla, kuten esimerkiksi pyytämällä selitystä alpha-beta-karsinnan toiminnasta erilaisissa skenaarioissa.

Lähteet

<https://www.chessprogramming.org/Minimax>

https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/RUSSIA/Final_Paper.pdf

<https://kartikkukreja.wordpress.com/2013/03/30/heuristic-function-for-reversiothello/>

https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

<https://playstrategy.org/analysis/flipello>