

Määrittelydokumentti

Ohjelmointikieli

Aiheeni on toteuttaa shakkipelin tekoäly, ja käytän siihen Pythonia. Hallitsen Pythonin hyvin. Opettelen myös JavaScriptia, mutta en vielä siinä määrin, että voisin tehdä vertaisarviointeja kyseisellä kielellä.

Algoritmit ja tietorakenteet

Aion käyttää työssäni minimax-algoritmia, jossa tekoäly valitsee siirron, joka maksimoi sen oman edun olettaen, että vastustaja pelaa aina optimaalisesti. Algoritmia tehostan alpha–beta-karsinnalla, joka vähentää pelipuun haarojen määrää ja nopeuttaa hakua. Lisäksi toteutan heuristiikkafunktion, joka arvioi pelitilanteen laatua materiaalin määrän ja nappuloiden sijoittumisen perusteella. Tietorakenteina käytän 8x8 kokoista taulukkoa shakkilaudan esittämiseen sekä erillistä siirtorakennetta, joka sisältää alkuruudun, kohderuudun ja mahdollisen kaapatun nappulan tiedot.

Ongelma ja ratkaisu

Ratkaisen ongelman, jolla tekoäly voi tehdä parhaan päätöksen, käymättä läpi kaikkia mahdollisia vaihtoehtoja. Tämä kaikki pitää tehdä tietyssä ajassa. Aion ratkaista ongelman hyödyntämällä yllä mainittuja algoritmeja.

Ohjelman syötteet ja käyttö

Ohjelma saa syötteitä useasta eri lähteestä: tekstikäyttöliittymästä, komentorivi argumenteista ja tekoälyn sisäisistä syötteistä. Lisäksi ohjelmassa suoritetaan syötteiden validointi ja virhetilanteiden käsittely, jotta toiminta pysyy virheettömänä.

Tekstikäyttöliittymässä pelaaja antaa siirtonsa merkkijonona, esimerkiksi muodossa e2e4. Tämä teksti muutetaan tietorakenteeksi/koordinaattipariksi, jota ohjelma pystyy käsittelemään. Siirron jälkeen ohjelma tarkistaa, onko se laillinen. Jos siirto on sallittu, pelilauta päivitetään ja vuoro siirtyy toiselle pelaajalle. Jokaisen siirron jälkeen tarkistetaan myös, onko peli päättynyt esimerkiksi kuninkaan menetykseen.

Komentoriviargumentit määrittävät pelin asetuksia käynnistysvaiheessa. Niiden avulla voidaan valita, esimerkiksi kumpi pelaaja aloittaa, millä värillä pelataan, mikä on minimax-algoritmin hakusyvyys, tai halutaanko peli aloittaa jostakin tietyistä asemasta. Näin käyttäjä voi säätää tekoälyn tasoa ja pelin alkuasetelmia ennen pelin alkua.

Tekoälyn sisäiset syötteet muodostuvat pelitilan tiedoista, esimerkiksi nykyisestä laudasta, vuorossa olevasta pelaajasta ja mahdollisista siirroista. Minimax-algoritmi käyttää näitä tietoja rakentaakseen pelipuun, jossa se käy läpi eri siirtovaihtoehtoja ja arvioi niiden tuottamat asemat heuristiikkafunktion avulla.

Lopuksi ohjelma suorittaa syötteiden validoinnin ja virhetilanteiden tarkistuksen. Virheelliset tai väärin kirjoitetut syötteet hylätään ja käyttäjältä pyydetään uusi syöte. Samoin komentoriviargumenttien arvot validoidaan, jotta ohjelma käynnistyy oikeilla asetuksilla.

Aika- ja tilavaativuudet

Shakin tekoälyn ydin on minimax-algoritmi, jota tehostetaan alpha–beta-karsinnalla. Minimax-algoritmin aika- ja tilavaativuus riippuvat mahdollisten siirtojen määrästä (b) ja haun syvyydestä (d). Ilman karsintaa minimaxin aikavaativuus on $O(b^d)$, koska jokaisessa haun tasossa tutkitaan kaikki mahdolliset siirrot.

Alpha–beta-karsinta vähentää tutkittavien haarojen määrää, jos siirrot tutkitaan optimaalisessa järjestyksessä. Käytännössä saavutettu parannus riippuu kuitenkin asemasta ja siirtojärjestyksestä, mutta se pienentää huomattavasti laskennan määrää.

Tilavaativuus minimax-algoritmissa on $O(d)$, koska rekursiivinen haku tarvitsee pino tilaa yhtä monta tasoa kuin haun syvyys on. Lisäksi muistia käytetään pelilaudan tilojen ja siirtojen säilyttämiseen. Tekoälyn suorituskyky riippuu siitä, kuinka syvälle peli puuhun voidaan edetä annetussa ajassa.

Lähteet

Aion käyttää lähteinä Wikipediaa sekä GeeksforGeeksin artikkeleita, jotka käsittelevät minimax-algoritmia, alpha–beta-karsintaa ja heuristiikkafunktion toteutusta. Hyödynnän myös Pythonin virallista dokumentaatiota kielen ominaisuuksiin ja perusrakenteisiin liittyen. Lisäksi käytän shakin teoriaan liittyviä lähteitä.

Harjoitustyön ydin

Harjoitustyöni ydin on shakkipelin tekoäly, joka osaa tehdä järkeviä siirtoja oikeata (ihmis) pelaajaa vastaan. Tekoälyn päätöksenteko perustuu minimax algoritmiin, jota tehostetaan alpha–beta-karsinnalla, jotta haku olisi riittävän nopea. Työn tärkein osa on heuristiikkafunktion suunnittelu, joka osaa arvioida pelitilanteen materiaalin määrän ja nappuloiden sijoittumisen perusteella. Varsinainen pelilogiikka ja käyttöliittymä toimivat tukena, joiden tarkoitus on mahdollistaa tekoälyn testaaminen ja pelaaminen, mutta suurin osa kehitystyöstä keskittyy tekoälyn toiminnan ja arviointimenetelmien toteuttamiseen ja parantamiseen.

Kuulun tietojenkäsittelytieteen kandidaatin opinto-ohjelmaan.