Toteutusdokumentti

Ohjelma rakentuu graafiseen käyttöliittymään ja pelilogiikkaan liitetystä tekoälystä, joka puolestaan sisältää tekoälyn logiikan, Siirto-luokan, sekä Siirto-olioita säilövän keon. Tekoäly kommunikoi pelilogiikan kanssa, korvaten näin ihmispelaajan. Sen vastuulla on siirtojen luominen ja parhaan siirron suorittaminen. Siirto tietää itsestään oleellisia asioita, kuten käsiteltävän muodostelman lopullisen x-sijainnin, korkeuden, kiertojen määrän, sekä muodostuneiden täysien rivien sekä kolojen määrät. Siirto myös pisteyttää itsensä sisäisen laskukaavansa mukaan. Luodut siirrot lisätään kekoon, jonka huipulla oleva, ”paras” eli suurimman numeroarvon saanut siirto valitaan suoritettavaksi. Tekoäly tekee muodostelmalle siirron mukaiset muutokset ja pudottaa sen palikkapinon päälle. Tämä prosessi toistetaan kunnes peli on ohi.

Yhden kierroksen suorituksen aikavaativuus rakentuu seuraavasti: tekoäly päivittää tietonsa operaatiossa, jonka aikavaativuus on O(x*y), jossa x ja y ovat pelikentän ulottuvuudet. Mahdollisten siirtojen etsimisen vaativuus on O(n), jossa n, siirtojen määrä, on aina välillä 9 … 40. Siirtojen luominen itsessään on aikavaativuudeltaan O(1) (sivujen etsintä) + O(rivit * x) (täysien rivien etsintä, $1 \leq$ rivit ≤ 4) + O(n) (kolojen etsintä, n = kolojen syvyys) + O(log n) (siirtojen lisäys kekoon). Nämä kaikki suoritetaan siis jokaiselle siirrolle, ts. n kertaa, jolloin siirtojen etsimisen aikavaativuus on karkeasti $O(n^2)$. Siirron haku keosta ja sen suorittaminen ovat vakioaikaisia toimintoja, lopuksi suoritettu keon tyhjennys taas vaativuudeltaan O(n). Siirtojen etsimisen (ja myös tietojen päivittämisen) $O(n^2)$-vaativuus dominoi tätä, jolloin koko kierroksen aikavaativuus on sama, $O(n^2)$.

Keon tilavaativuus kullakin kierroksella on siirtojen määrä, joka on aina vähintään 9 ja korkeintaan 40. Tekoälyn käyttämä apumuuttujien ja attribuuttien määrä puolestaan on aina sama, siirtojen määrästä tai muista syötteistä riippumatta. Tällöin yleiseksi tilavaativuudeksi jää keon O(n), ainakin teoriassa. Todellisuudessa tekoälyn luomat kopiot erilaisista olioista jäävät rasittamaan muistia ainakin hetkeksi ennen kuin Javan roskienkerääjä korjaa ne pois, joten tilanne ei ole aivan niin optimoitu kuin voisi toivoa. Tilankäytön tehokkuus onkin yksi niistä asioista, joita projektissa olisi vielä voinut kehittää tehokkaammaksi.

Huomattavin työhön loppujen lopuksi jäänyt ongelma on bugi, joka lopettaa pelin hieman liian aikaisin. Tämä johtuu tekoälyn putoavalle muodostelmalle suorittamista operaatioista, joissa jokin muodostelman mahdollisista kierroista ei enää mahdu pelikentälle, mutta tekoäly suorittaa sen siitä huolimatta. Rajallisen ajan takia en ehtinyt korjata ongelmaa, vaikka onnistuinkin selvittämään sen syyn. Se jää siis jatkokehittelyn ensimmäiseksi pähkinäksi. Muita paranneltavia seikkoja ovat lähinnä algoritmien tilan- ja ajankäytön optimoiminen tehokkaimmaksi mahdolliseksi, sekä tekoälyn että pelin itsensä suhteen. Jatkan todennäköisesti näiden kanssa työskentelyä.

Muita parannuksia voisivat olla tekoälyn seuraamista helpottavat toiminnot: paremmat nopeudensäätövaihtoehdot, visuaalinen esitys tekoälyn heuristiikasta, sekä varsinaisen logiikan parantaminen siten, että se pelaa peliä paremmin. Tarkoituksenani on myös luoda pelistä kaksinpeliversio, jossa ihmispelaaja voi pelata tekoälyä vastaan ja molemmat voivat häiritä toistensa suoritusta erilaisilla tavoilla. Tätä tarkoitusta varten tekoälyn vaikeustasoa olisi voitava säätää jotenkin, esim. muuttamalla sen sisäisiä painotuksia kovakoodatuista arvoista muuttujiksi ja luotava jonkinlainen mahdollisuus virheille tai ”tyhmille” siirroille.