Toteutusdokumentti

Ohjelman yleisrakenne:

Sovellus on 2048-peli, jota pelataan oikean 2048 pelilogiikan mukaisesti. Pelillä on käytössä 4x4 matriisi, johon ilmestyy painotetusti satunnaisesti 4 ja 2 arvoisia ”tiiliä”. Peliä pelataan liu’uttamalla tiiliä laudatta joko vasemmalle, oikealle, ylös tai alas. Liu’utuksen yhteydessä kaikki tiilet, jotka ovat saman arvoisia ja liukuvat toisiinsa kiinni yhdistyvät ja plussaantuvat arvossaan yhteen. Eli kaksi vierekkäistä kakkosta liukuessaan toisiinsa kiinni muuttuvan yhdeksi neloseksi. Pelin tavoitteena on saada tiili 2048, mutta peliä voi jatkaa vielä sen jälkeenkin. Peli päättyy joko itse lopettamalla tai kun ruudukko on täynnä.

Ohjelma koodi on tehty pythonilla ja jaettu selkeästi eri osiin:

- Main-luokka: Vastaa pelin käynnistyksestä, siirtovuorojen jakamisesta (satunnais- ja minimax-siirrot) sekä käyttöliittymän päivittämisestä.
- Game-luokka: Sisältää pelin logiikan, kuten ruudun tilan, tiilien liu'uttamisen ja yhdistämisen.
- Minimax-algoritmi: Toteutettu erillisessä moduulissa, jossa sekä algoritmi että sen heuristiset funktiot laskevat yhdessä parhaan seuraavan siirron
- UI-luokka: Vastaa pelilaudan visuaalisesta esityksestä.

Tämä algoritmi versio pelistä vuorottelee niin että joka toinen siirto on minimax algoritmin tekemä, ja joka toinen on vain siirto random suuntaan. Pelin käynnistys ja siirtovuoron jako tapahtuu main luokassa, ulkonäkö ui- luokassa ja pelilogiikka game luokassa. Algoritmin siirto vuorolla siirron suunta lasketaan minimax luokassa, johon on toteutettu minimax tavalla toimiva algoritmi, joka vertaa millä siirrolla saa varmiten toisen liikkeistä huolimatta pidettyä pienimmän määrän tiiliä laudalla kerrallaan.

Saavutetut aika- ja tilavaativuudet:

Pelin algoritmin vaatima aika ja tilavaativuus muuttuu hieman pelin edetessä, aluksi meillä on valmiiksi kaksi ruutua aina varattuna, sillä peli alkaa kahdella tiilellä, eli alkusiirtoja, kun suuntia, joihin on mahdollista liu’uttaa on 4, tulisi 2 x 14 = 28. Pelin aikana aikavaativuus vaihtelee vapaiden paikkojen määrästä riippuen, mutta on jotakuinkin O((4216)^y) = O(128^y)

4. Työn mahdolliset puutteet ja parannusehdotukset

- Tehdä Fallback functio kun minimax ei keksi yhtäkään siirtoa.

- Tehdä niin että käyttäjä voi itse tehdä siirtoja.

Laajojen kielimallien (ChatGPT yms.) käyttö
Työssä on käytetty ChatGPT-mallia apuna päättämään kuinka jakaa ohjelman rakenne eri funktioihin ja eri sivuille. Ja mallin avulla on saatu ideoita minimaxin heurististen funktioiden tasapainottamiseen.

Viitteet:

Lähteinä käytettiin:

[1] Edwards, S. – *Minimax and Alpha-Beta Pruning*
https://www.cs.columbia.edu/~sedwards/classes/2020/4995-fall/reports/Minimax.pdf

[2] Nie, H., Hou, A. – *AI Plays 2048 Report*
https://cs229.stanford.edu/proj2016/report/NieHouAn-AIPlays2048-report.pdf