Toteutusdokumentti

1. Yleisrakenne

Ohjelman kolme tärkeintä luokkaa ovat **Game**-, **Statistics**- ja **StrategyHandler**-luokat. Luokat toimivat seuraavasti: Game-luokka pyytää **TextUI**-luokan avulla pelaajalta tarvittavat asetukset ja käynnistää pelin. Game-luokka pyytää joka kierroksella kaikilta **Player**-luokan toteuttavilta pelaajilta käden, minkä jälkeen tarkistetaan voittaja. Voittaja ja pelatut siirrot tallennetaan **Statistics**-luokkaan, minkä jälkeen Pelaajat (lähinnä StrategyHandler) päivittävät omien strategioiden sisäiset mallit Statistics-luokan tarjoamien tilastojen avulla.

Lisäksi on olemassa **abstrakti luokka Strategy,** jonka kaikki ennustusalgoritmit (mm. MarkovFirstOrder, MarkovSecondOrder) toteuttavat. Player-luokka, jonka StrategyHandler (tekoäly) ja **TestPlayer** (lukee vanhojen pelien siirrot ja pelaa niiden mukaan) toteuttavat.

Ohjelma sisältää myös Enum-luokan **Hand** ja ArrayList-kopion **Lista**.
Strategy-luokan toteuttavat algoritmiluokat: **MarkovFirstOrder, MarkovSecondOrder, PatternMatching, RandomAi, StupidAi**

2. Saavutetut aika- ja tilavaativuudet

Kaikki algoritmit(PatternMatchingia lukuunottamatta) ovat kokonaisaikavaativuudeltaan O(n). Kun peliä pelataan n kierrosta, niin algoritmit päivittävät itseään n-kertaa, mutta tämän jälkeen itse ennustus on vakioaikainen toimenpide.

PatternMatching-algoritmissa jokainen käden ennustus on O(n*m), missä n on tarkasteltavien siirtojen määrä ja m on halutun patternin maksimipituus. Kun ennakointioperaatiota suoritetaan joka syötteen jälkeen, on kokonaisaikavaativuus vähintään $O(n^2 * m)$.
StrategyHandlerin metastrategioiden päivitysnopeus on riippuvainen sen sisältämien algoritmien nopeudesta, mutta itse parhaimman metastrategian valitseminen ja metastrategioiden pisteytys ovat O(1) aikavaativuudeltaan.

Markovin malleihin perustuvat algoritmit ovat tilavaativuudeltaan O(1), koska niiden käyttämät matriisit ovat vakiokokoisia. "Tyhmempien" algoritmien StupidAi ja RandomAi aika ja tilavaativuus ovat vakioiaikaisia.

Lista-luokan lisäys ja poisto ovat pahimmillaan O(n), mutta koska lisäyksen pahin tapaus tapahtuu vain kerran per n-lisättyä alkiota, niin keskimääräinen aikavaativuus on käytännössä vakioaikainen.

3. Työn mahdolliset puutteet ja parannusehdotukset

Tehokkaampi pattern-matching-algoritmi. Vähemmän riippuvuuksia luokkien välillä. Vaikeustasot eivät välttämättä ole parhaat mahdolliset, johtuen lähinnä valinnanvaikeudesta algoritmien suhteen ja testauksen vaikeudesta.