521160P Johdatus Tekoälyyn

Harjoitus #1

Pelit etsintäalgoritmien avulla

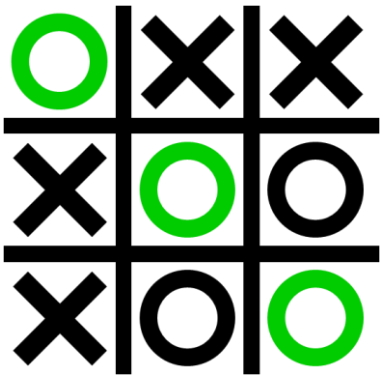
Kevät 2018

Valitse seuraavista kolmesta esitellystä pelistä kaksi ja palauta Optimaan toteuttamasi ratkaisut (python-tiedostot). Mikäli valitset sekä ristinollan, että Connect 4-pelin, toteuta ainakin toinen algoritmeista alfa-beta pruning algoritmilla.

Ristinolla

Pelin säännöt

Ristinolla on suosittu kahden henkilön pelattava kynä- ja paperipeli, jota pelataan 3x3 ruudukolla. Pelissä pelaajat laittavat vuorotellen merkkinsä (X tai O) tyhjään ruutuun. Ensimmäinen pelaajista, joka saa 3 merkkiään peräkkäin pystysuuntaisesti, vaakasuuntaisesti tai diagonaalisesti voittaa pelin. Jos kaikki ruudut ovat täynnä ja kumpikaan pelaajista ei saanut merkkejään peräkkäin, peli päättyy tasan. Ristinolla on nollasummapeli, jossa yhden pelaajan voitot ovat aina pois toiselta pelaajalta. Tästä syystä, jos pelaat täydellistä tekoälyä vastaan, se joko voittaa pelin tai peli päättyy tasan. Lisäksi ristinollaa usein käytetään pelipuita ratkaisevien algoritmien havainnollistamisessa sen yksinkertaisuuden vuoksi. Lisätietoja pelin säännöistä löydät osoitteesta http://mathworld.wolfram.com/Tic-Tac-Toe.html



Kuva 1. Ristinolla

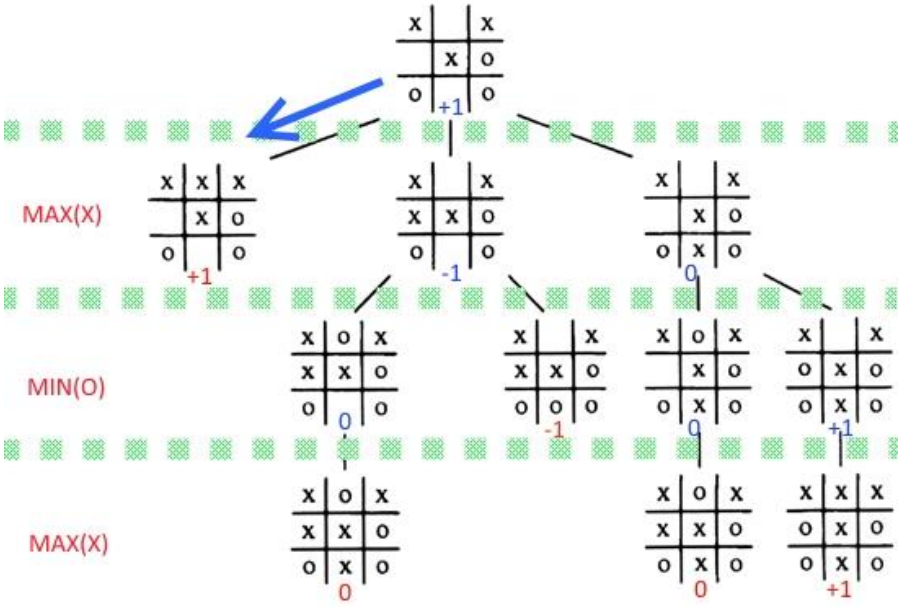
Tehtävä 1

Sinun tehtäväsi on toteuttaa etsintä-algoritmi, joka päättää ristinolla-pelissä tekoälyn seuraavan siirron. Luonnollinen valinta algoritmiksi kaksin pelattavassa nollasummapelissä on joko minimax-algoritmi tai alfa-beta pruning algoritmi mutta myös muutkin vaihtoehdot ovat mahdollisia.

Minimax algoritmi arvioi pelipuun siirtoja, jotka johtavat päätetilaan (terminal state). Algoritmissa tekoälyä kutsutaan maksimoivaksi pelaajaksi ja vastustajaa minimoivaksi pelaajaksi. Maksimoiva pelaaja yrittää saada suurimman mahdollisen tuloksen kuin mahdollista, kun taas minimoiva pelaaja yrittää minimoida tuloksen vastasiirtojen avulla. Esimerkiksi jos tekoälyllä on kaksi vaihtoehtoista siirtoa, jotka johtavat joko voittoon tai häviöön tai pelkästään tasapeliin. Silloin minimax-algoritmi olettaa, että minimoiva pelaaja pelaa täydellisesti ja tulee valitsemaan voittavan siirron, jos saa siihen mahdollisuuden. Tästä syystä tekoäly (maksimoiva pelaaja) valitsee varman siirron niin, että peli päättyy tasan.

Ristinollassa on kolmenlaisia päätetiloja, joille voidaan antaa seuraavat numeroarvot: voitto +1, häviö -1 ja tasapeli 0. Kuvassa 2 on esitetty esimerkki tilanne ristinollan pelipuusta, kun peli on lähellä loppua. Aluksi minimax-algoritmi luo koko pelipuun ylhäältä alas päätetiloihin asti. Tämän jälkeen päätetilat arvioidaan antamalla niille arvot +1, -1 tai 0 riippuen pelin tuloksesta. Seuraavaksi algoritmi alkaa siirtymään pelipuussa

ylöspäin pakaten numeroarvoja sen mukaan, onko kyseessä maksimoivan vai minimoivan pelaajan vuoro. Esimerkiksi jos kyseessä on maksimoivan pelaajan vuoro ja haarassa on kaksi vaihtoehtoa, merkataan haarautuvaan pisteeseen suurempi arvoista. Tällä tavalla pakataan numeroarvot aina pelipuun alkuhaaraan asti kerros kerrallaan. Kuvassa 2 punaiset numerot kuvaavat päätetiloille annettuja numeroarvoja ja siniset numerot pakattuja numeroarvoja. Lopulta ylimmässä haarassa on vaihtoehtoina arvot +1, -1 tai 0, ja koska kyseessä on maksimoivan pelaajan vuoro, valitsee algoritmi vaihtoehdoista arvon +1 (voiton).



Kuva 2. Ristinolla pelipuu ratkaistu minimax-algoritmilla

Pseudokoodi minimax-algoritmille:

```
01 function minimax(node, depth, maximizingPlayer)
02     if depth = 0 or node is a terminal node
03         return the heuristic value of node

04     if maximizingPlayer
05         bestValue := −∞
06         for each child of node
07             v := minimax(child, depth − 1, FALSE)
08             bestValue := max(bestValue, v)
09         return bestValue

10     else    (* minimizing player *)
11         bestValue := +∞
12         for each child of node
13             v := minimax(child, depth − 1, TRUE)
14             bestValue := min(bestValue, v)
15         return bestValue
```

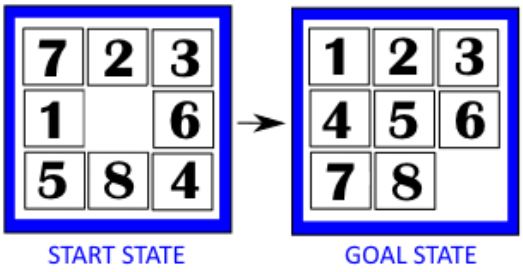
Yläpuolella on esitetty pseudokoodi minimax-algoritmille. 3x3 ristinollassa on ainoastaan 255,168 erilaista mahdollista päätetilaa, joka tarkoittaa että etsintäsyvyyttä ei tarvitse rajoittaa ja sen voi jättää huomiotta pseudokoodissa (rivien 2-3 depth=0). Päätetilojen saadessa vain arvoja +1, 0 ja -1, niin positiivista ja negatiivista ääretöntä vastaavat esimerkiksi luvut 2 ja -2. Lisäksi rivillä 6 oleva "for each child of node" voidaan toteuttaa tekemällä seuraava laillinen siirto for-silmukan avulla ja tarkistamalla, miten se vaikuttaa parhaaseen arvoon (rivit 7-8 ja 13-14). Huomaa että tämä toteutustapa toimii rekursiivisesti, joten sinun täytyy perua tehty siirto (eli korvata tyhjällä siirrolla pseudokoodissa rivien 8 ja 14 jälkeen).

Mikäli toteutat ratkaisun Minimax-algoritmilla, tee muutoksesi python koodissa **tictactoe.py** funktioon Minimax(board, maximizingplayer, player). Funktio toimii rekursiivisesti ja ottaa argumentteina pelilaudan (board), onko kyseessä maksimoiva pelaaja True tai False totuusarvona (maximizingplayer) sekä kyseessä olevan pelaajan merkin (player). Jos käytät jotain muuta ratkaisua, toteuta korvaava funktio käyttämällesi algoritmille.

8-Puzzle

Pelin säännöt

8-puzzle koostuu kahdeksasta liukuvasta laatasta, jotka ovat numeroitu ykkösestä kahdeksaan. Numeroidut laatat ovat sijoitettu 3x3 ruudukolle. Näin ollen yksi ruudukon kohdista on tyhjä, johon voi siirtää minkä tahansa horisontaalisesti tai vertikaalisesti tyhjän paikan vieressä olevan laatan. Siirtämällä jokin laatoista tyhjään paikaan, jää laatan aiempi paikka tyhjäksi. 8-puzzlen tarkoitus on saavuttaa tavoitetila mahdollisimman vähin siirroin satunnaisesti sekoitetusta alkutilasta. 8-puzzle on ongelmapeli 15-puzzlen pienennetty versio. Lisätietoja pelin säännöistä löydät osoitteesta https://en.wikipedia.org/wiki/15_puzzle



Kuva 3. Alkutila ja päätetila 8-puzzle-pelissä

Tehtävä 2

Sinun tehtäväsi on luoda etsintäalgoritmi, joka ratkaisee 8-puzzle-pelin. 8-puzzle on yksi varhaisimmista heuristisista etsintäalgoritmi-ongelmista ja sen ratkaisussa on yleisesti käytetty A*-algoritmia. Tyypillinen ratkaisu on noin 20 siirtoa pitkä ja sen haarautumiskerroin (kuinka monta mahdollista siirtoa keskimäärin) on 2.66. Tämä tarkoittaa, että perusteellinen etsintä 20 siirron syvyydestä käy läpi (2.66)^20 = 3.3*10^8 tilaa. Tästä syystä on erityisen tärkeä löytää mahdollisimman hyvä heuristinen funktio.

Heuristinen funktio h(n) tekee kustannusarvion päästäkseen tavoitetilaan. Jos haluamme löytää lyhimmän mahdollisen polun ratkaisuun, seuraavat kaksi heuristista funktiota ovat varteenotettavia vaihtoehtoja:

Hamming etäisyys: Laskee väärillä paikoilla olevien tiilien lukumäärän. Tila, jolla on vähän tiiliä väärillä paikoilla, on lähempänä tavoitetilaa ja se on parempi kuin tila, jolla on monta tiiltä väärällä paikalla.

Manhattan etäisyys: laskee horisontaalisten ja vertikaalisten etäisyyksien yhteenlasketun summan tavoitetilasta.

Esimerkiksi hamming etäisyys kuvalle 3 on h1(n) = 4, sillä sekoitetun alkutilan tiilistä neljä on väärällä paikalla. Manhattan etäisyys voidaan laskea kuvalle 3 seuraavasti: h2(n) = 1 + 0 + 0 + 3 + 2 + 0 + 2 + 0 = 8.

A*-algoritmi tarvitsee heuristisen funktion lisäksi kustannusarvion g(n) nykyisestä tilasta. Voimme käyttää polun kustannusarviona g(n) siirtojen lukumäärää päästäkseen nykyiseen tilaan, mikä on sama asia kuin

pelipuun syvyys. Nyt voimme yhdistää heuristisen funktion h(n) ja nykyisen polun kustannusarvion g(n) summaamalle ne yhteen f(n) = h(n) + g(n).

A*-algoritmin pseudokoodi on esitetty alhaalla. Se käyttää avointa listaa (**openSet**) solmuille, joita ei ole vielä tutkittu. Haluamme arvioida jokaisen solmun vain kerran, jonka takia pistämme suljettuun listaan (**closedSet**) solmut, jotka on jo tutkittu.

Alkutilanteessa avoin lista sisältää alkutilan ja suljettu lista on tyhjä. Lisäksi meidän täytyy ottaa ylös ns. emosolmu listaan nimeltä **cameFrom**, sillä kun lopulta löydämme ratkaisun, voimme tämän listan avulla rakentaa polun alkutilasta päätetilaan. Polun kustannusarviona g(n) käytetty pelipuun syvyys voidaan tallentaa listaan **gScore**. gScore listan alkutila on alustettu arvoon 0. Vastaavasti listan **fScore** arvo on heuristisen funktion ja kustannusarvion summa. Koska ensimmäisen solmun syvyys on 0, niin sen f(n) arvo on täysin heuristinen. Toteuta algoritmi perustuen alla olevaan pseudokoodiin.

Pseudokoodi A algoritmille:*

```
function A*(start, goal)
    closedSet := {}
    openSet := {start}
    cameFrom := the empty map
    gScore := {}
    gScore[start] := 0
    fScore := {}
    fScore[start] := heuristic_cost_estimate(start, goal)
    while openSet is not empty
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        closedSet.Add(current)

        for each neighbor of current
            if neighbor in closedSet
                continue            // Ignore already evaluated neighbors.

            if neighbor not in openSet      // Discover a new node
                openSet.Add(neighbor)

            // The distance from start to a neighbor
            tentative_gScore := gScore[current] + 1
            if tentative_gScore >= gScore[neighbor]
                continue            // This is not a better path.

            // This path is the best until now. Record it!
            cameFrom[neighbor] := current
```

```
            gScore[neighbor] := tentative_gScore
            fScore[neighbor] := gScore[neighbor] +
heuristic_cost_estimate(neighbor, goal)
    return failure

function reconstruct_path(cameFrom, current)
    total_path := [current]
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.append(current)
    return total_path
```

Mikäli toteutat ratkaisun A*-algoritmilla, tee muutoksesi python koodissa **eightpuzzle.py** funktioon Astar(self, state). Funktio ottaa argumentteina luokan EightPuzzle kaikki attribuutit (self) sekä pelilaudan nykyisen tilan (state). Jos käytät jotain muuta ratkaisua, toteuta korvaava funktio käyttämällesi algoritmille.

Connect 4

Pelin säännöt

Connect Four (Neljänsuora) on kahden henkilön pelattava strategia peli, joka perustuu ristinollaan. Pelissä molemmat pelaajat pudottavat vuorotellen värillisiä kiekkoja (tässä harjoituksessa käytetään kiekkojen sijasta merkkejä X tai O) ylhäältä pelilautaan, joka on 7 saraketta leveä. Jokaiseen sarakkeeseen mahtuu maksimissaan 6 kiekkoa. Kiekot putoavat suoraan alaspäin sarakkeen pohjaan asti. Pelaaja, joka saa ensimmäisenä 4 hänen kiekoistaan vierekkäin pystysuuntaisesti, vaakasuuntaisesti tai diagonaalisesti voittaa pelin. Peli päättyy tasan, jos kaikki kiekot ovat sijoitettu pelilaudalle (42 siirron jälkeen) niin, että kummastakaan merkistä ei muodostu neljän peräkkäisen kiekon muodostamaa suoraa. Lisätietoja pelin säännöistä löydät osoitteesta: http://mathworld.wolfram.com/Connect-Four.html

Kuva 4. Connect Four (Neljänsuora)

Tehtävä 3

Sinun tehtäväsi on luoda etsintä-algoritmi, joka päättää connect four-pelissä tekoälyn seuraavan siirron. Yleisesti käytettyjä valintoja algoritmeiksi ongelman ratkaisemiseksi ovat minimax tai alfa-beta pruning, mutta muutkin vaihtoehdot ovat mahdollisia. Pelissä on 4,531,985,219,092 mahdollista tilaa, kun pelilaudalle sijoitetaan 0-42 kiekkoa. Tämän takia kaikkia mahdollisia tiloja ei voi tutkia ja etsintäsyvyyttä täytyy rajoittaa. Uusien tilojen tutkiminen tulee lopettaa, kun valittu etsintäsyvyys saavutetaan. Alapuolella on esitetty pseudokoodi alfa-beta pruning algoritmille. Minimax algoritmin pseudokoodi löytyy ristinollan tehtäväohjeista.

Pseudokoodi alfa-beta-pruning algoritmille:

```
01 function alphabeta(node, depth, α, β, maximizingPlayer)
02      if depth = 0 or node is a terminal node
03          return the heuristic value of node
04      if maximizingPlayer
05          v := -∞
06          for each child of node
07              v := max(v, alphabeta(child, depth - 1, α, β, FALSE))
08              α := max(α, v)
09              if β ≤ α
10                  break (* β cut-off *)
11          return v
12      else
13          v := +∞
14          for each child of node
15              v := min(v, alphabeta(child, depth - 1, α, β, TRUE))
16              β := min(β, v)
17              if β ≤ α
18                  break (* α cut-off *)
19          return v
```

Tässä pelissä etsintäsyvyyden rajoittaminen on keskeisessä roolissa. Kun ennalta määrätty etsintäsyvyys tai päätetila saavutetaan, lasketaan arviointifunktiolla pelilaudalle lukuarvo, jota maksimoiva pelaaja (tietokone) yrittää siirroillaan kasvattaa ja minimoiva pelaaja pienentää. Hyväksi havaittu arviointifunktio connect four-pelille tullaan esittämään myöhemmin näissä ohjeissa.

Pseudokoodissa äärettömänä ja miinus äärettömänä voidaan käyttää mahdollisimman isoja lukuarvoja (esimerkiksi lukuja 1000000000 ja -1000000000). Lisäksi rivillä 6 oleva “for each child of node” voidaan toteuttaa for-silmukan avulla tekemällä seuraava laillinen siirto ja tarkistamalla, miten se vaikuttaa minimax-algoritmilla parhaaseen arvoon ja alfa-beta-pruning algortimilla alfaan ja betaan. Molemmat pseudokoodit toimivat rekursiivisesti eli sen jälkeen kun olet tehnyt seuraavan laillisen siirron ja vertaillut for-silmukan sisällä parasta arvoa tai alfaa ja betaa, niin siirto tulee perua (korvata tyhjällä).

Jos käytät alfa-beta pruning algoritmia, niin maksimoivan ja minimoivan pelaajan toiminta kannattaa erottaa kahdeksi eri funktioksi. Voit ensiksi toteuttaa ratkaisun minimax-algoritmilla ja sen jälkeen laajentaa ratkaisu alfa-beta pruning algoritmiksi.

Sinun täytyy lisäksi luoda arviointifunktio, joka arvioi nykyisen tilan arvon ja palauttaa sen. Esimerkiksi jos 3x3 ristinollassa käytettäisiin arviointifunktiota, se yksinkertaisesti palauttaisi tilan arvoksi +1 voitosta, -1 häviöstä ja 0 tasapelistä. Hyvä arviointifunktio connect four-peliin (mutta ei täydellinen) on esitetty kirjassa (R. L. Rivest, Game Tree Searching by Min/Max Approximation, AI 34 [1988], pp. 77-96) ja jota myös käytetään tässä tehtävässä seuraavasti (X on maksimoiva pelaaja):

Aluksi tarkastetaan, onko päätetila saavutettu. X:n voittaessa pelin funktio palauttaa arvon +512 pistettä. O:n voittaessa pelin funktio palauttaa arvon -512 pistettä. Tasapeli tilanteessa funktio palauttaa arvon 0.

Muussa tapauksessa funktio arvioi kaikki mahdolliset 4 peräkkäisen ruudun muodostamat segmentit pelilaudalla (pystysuuntaiset, vaakasuuntaiset ja diagonaaliset). Vaakasuuntaisia neljän ruudun segmenttejä on 24, pystysuuntaisia on 21 ja diagonaalisia on 24. Jokainen segmentti arvioidaan erikseen alla olevien sääntöjen perusteella ja lopuksi palautetaan kaikkien segmenttien saamien arvojen summa.

<u>Säännöt segmenttien arvioimista varten:</u>

-50 pistettä, kun segmentissä on kolme O merkkiä ja ei yhtään X merkkiä

-10 pistettä, kun segmentissä on kaksi O merkkiä ja ei yhtään X merkkiä

-1 piste, kun segmentissä on yksi O merkki ja ei yhtään X merkkiä

(0 pistettä, kun segmentissä ei ole yhtään merkkiä tai siinä on sekä X että O merkkejä sekaisin)

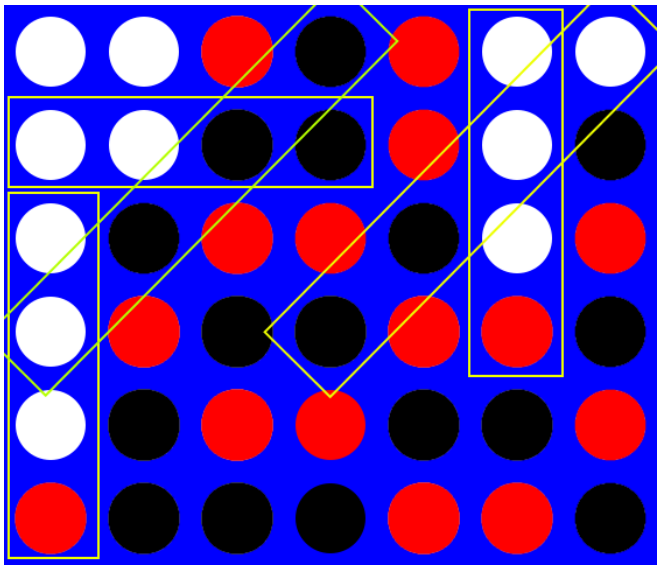
+1 piste, kun segmentissä on yksi X merkki ja ei yhtään O merkkiä

+10 pistettä, kun segmentissä on kaksi X merkkiä ja ei yhtään O merkkiä

+50 pistettä, kun segmentissä on kolme X merkkiä ja ei yhtään O merkkiä

Lisäksi annetaan 16 bonuspistettä sen mukaan, kenen vuoro on pelata (X:n vuorosta +16 pistettä ja O:n vuorosta -16 pistettä)

Kuvassa 5 on esitetty esimerkki tilanteelle segmentit neljälle peräkkäisille ruuduille, joissa ei ole kuin yhtä väriä. Ainoastaan ne segmentit, joissa on vain yhtä väriä siis vaikuttavat arvioinnin lopputulokseen. Arviointifunktio palauttaisi kuvan 5 pelilaudan arvoksi +68 pistettä, kun musta merkki on maksimoiva pelaaja.



Kuva 5. Esimerkki arviointifunktion segmentöinnistä

Arviointifunktion toimiessa yllä olevien sääntöjen mukaan ei kuitenkaan tarkoita, että se olisi paras mahdollinen vaihtoehto. Se ei myöskään suosi nopeaa voittamista, eli jos algoritmilla on mahdollisuus voittaa esimerkiksi sekä 23:lla ja 17:sta siirrolla, arviointifunktiossa ei ole kannustinta voittamaan nopeammin. Yllä esitetty arviointifunktio on toteutettuna funktiossa evaluation(), mutta voit esimerkiksi pisteyttää arviointifunktion eri numeroarvoilla ja tarkastella paraniko algoritmin suorituskyky. Toteuta valitsemasi algoritmi tiedostoon **connect4.py**.

Palauta

Valitse tehtävistä kaksi ja palauta muokkaamasi python tiedostot (.zip tai .rar tiedostoon pakattuna) Optiman palautuslaatikkoon Harjoitus 1 **26.3.2018 klo 23:59** mennessä. Mikäli teet algoritmit sekä ristinollaan, että connect four-peliin, toteuta vähintään toinen alfa-beta pruning algoritmilla. Tästä harjoituksesta on mahdollisuus tienata maksimissaan 5 pistettä (2.5p + 2.5p).