521160P Johdatus Tekoälyyn

Harjoitus #1

Pelit hakualgoritmien avulla

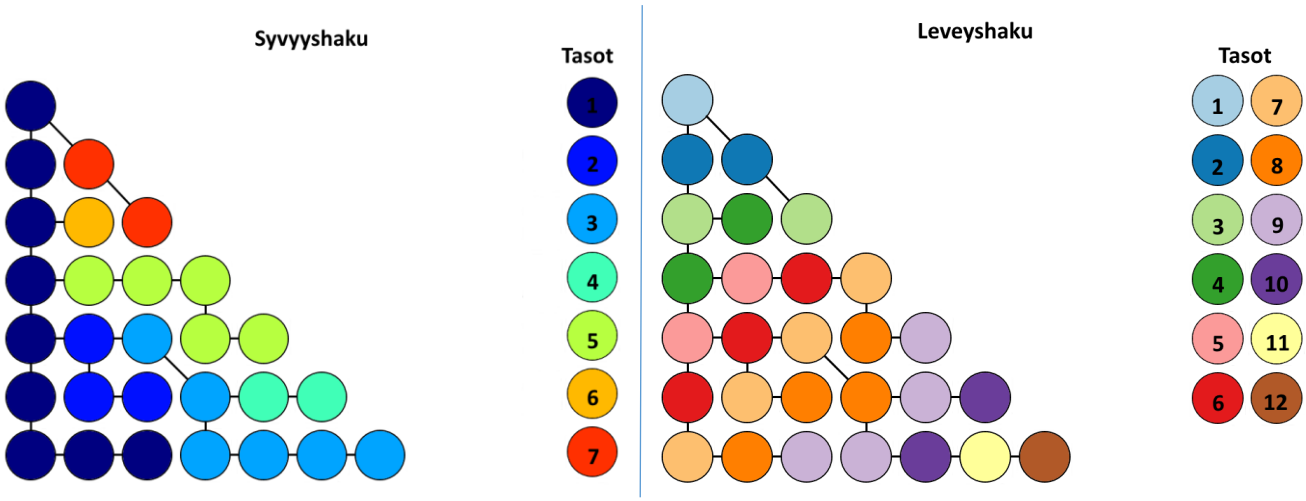
Kevät 2019

Hakualgoritmit

Monet etsimisongelmat voidaan muokata ratkaistavaksi hakualgoritmeilla. Tyypillinen esimerkki tästä on navigointi-ongelma, jossa pyritään löytämään mahdollisimman lyhyt tai nopea reitti ***alkutilasta tavoitetilaan***. Ongelman hakuavaruus voidaan kuvata suunnattuna graafina tai puukaaviona, joissa solmut kuvaavat ***sallittuja tiloja*** ja niiden väliset linkit kuvaavat siirtymiä. Usein kaikki mahdolliset siirtymät eivät ole sallittuja, sillä kyseessä voi olla **aiemmin käsitelty tila** tai tutkittava tila on jo etukäteen määritetty ***kielletyksi tilaksi***. Tällöin aiemmin käsitellyt tilat sekä kielletyt tilat jätetään kokonaan tutkimatta.

Kun tutkittavien tilojen hyvyydestä ei ole etukäteen mitään tietoa, käytetään leveyshakua tai syvyyshakua. Leveyshaussa käydään läpi puukaavion yksi taso kerrallaan, ennen kuin siirrytään tutkimaan seuraavaa tasoa. Syvyyshaussa puolestaan edetään puukaaviossa mahdollisimman syvälle ja kun tutkittavan haaran päätepiste saavutetaan, jatketaan tutkimista seuraavaksi syvimmästä tiedossa olevasta tutkimattomasta haarasta. Molemmat haut pysähtyvät, kun tavoitetila saavutetaan.

Molemmat hakustrategiat tarvitsevat ns. avoimen listan (engl. open list), johon tallennetaan selvitetyt tutkimattomat tilat sekä ns. suljetun listan (engl. closed list), johon tallennetaan jo käsitellyt tilat. Toteutuksen kannalta menetelmien ainoa ero on, että leveyshaussa uudet sallitut tutkimattomat tilat tallennetaan jonoon, kun taas syvyyshaussa uudet sallitut tutkimattomat tilat tallennetaan pinoon. Jonossa uudet tutkimattomat tilat tallennetaan listan perälle ja seuraavaksi tutkittavaksi tilaksi valitaan aina listan ensimmäinen alkio (engl. first-in-first-out, FIFO). Pinossa uudet tutkimattomat tilat tallennetaan listan alkuun ja seuraavaksi tutkittavaksi tilaksi otetaan listan ensimmäinen alkio (engl. last-in-first-out, LIFO). Kuvassa 1 on havainnollistettu syvyyshaun ja leveyshaun eriäviä hakustrategioita eräässä puukaaviossa. Tasojen numerot kertovat tilojen tutkimisjärjestyksen.



Kuva 1. Syvyyshaun ja leveyshaun hakustrategiat eroavat toisistaan seuraavasti eräässä puukaaviossa.

Useimmissa tapauksissa hakuavaruus on niin suuri, että leveyshakua tai syvyyshakua käyttämällä tavoitetilan löytäminen vie suhteettoman kauan aikaa. Tällöin tilojen kombinatorisen räjähdyksen hillitsemiseksi voidaan käyttää heuristista arviointifunktiota tilojen paremmuuden selvittämiseksi. Arvioidut tilat tallennetaan prioriteettilistaan, josta seuraavan tutkittavan tilan poimintaan käytetään paras-ensin strategiaa (engl. best-first strategy).

Heuristinen funktio h(n) on määritelmän mukaan kustannusarvio tutkittavasta tilasta tavoitetilaan. Kun heuristiseen funktioon lisätään vielä itse tutkittavan polun kustannusarvio g(n), saadaan luotua **A*-hakualgoritmi**, joka on esitetty kaavassa 1.

$$f(n) = h(n) + g(n) \quad (1)$$

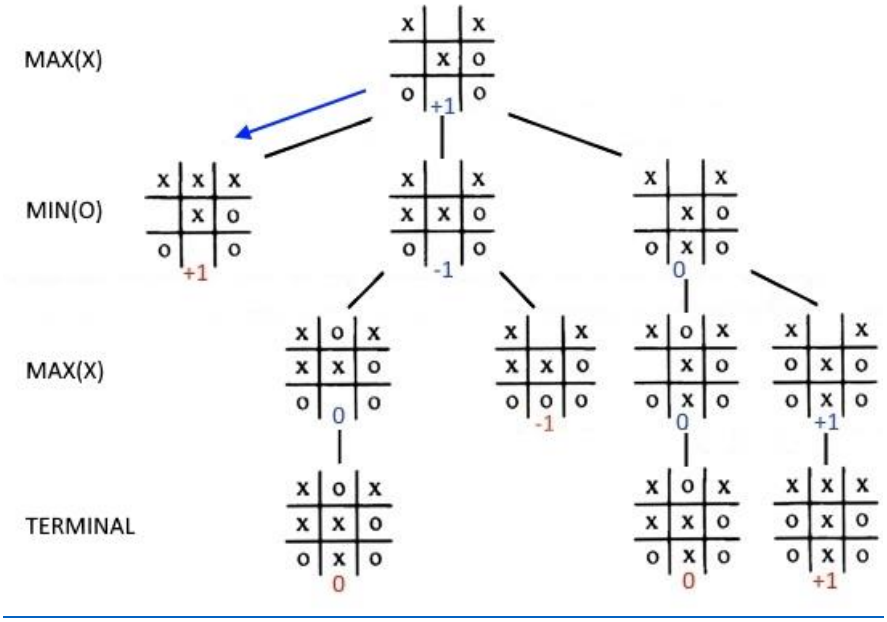
A*-hakualgoritmi löytää aina optimaalisen reitin, ***mikäli kustannusarvio h(n) tutkittavasta tilasta tavoitetilaan ei yliarvioi kustannusta*** [1]. A*-hakualgoritmin optimaalisuusehdon täyttyessä se karsii huomattavan osan hakuavaruudesta toimien varsin tehokkaasti.

Tekoäly peleissä

Tekoälyä käytetään nykyään yhä enenevissä määrin erilaisissa tietokone- ja videopeleissä. Yksinkertaisissa deterministisissä kahden henkilön pelattavissa nollasumma-peleissä hakualgoritmeista ei ole apua ja on turvauduttava erilaiseen lähestymistapaan. Tällöin voidaan käyttää **Minimax-algoritmia.**

Minimax-algoritmi arvioi pelipuun siirtoja, jotka johtavat päätetiloihin. Algoritmissa tekoälyä kutsutaan maksimoivaksi pelaajaksi ja sen vastustajaa minimoivaksi pelaajaksi. Maksimoiva pelaaja yrittää saavuttaa suurimman mahdollisen tuloksen, kun taas minimoiva pelaaja yrittää minimoida lopputuloksen vastasiirtojen avulla. Tarkastellaan esimerkkinä tilannetta, jossa tekoälyllä on kaksi vaihtoehtoista siirtoa, jotka johtavat joko voittoon tai häviöön tai pelkästään tasapeliin. Nyt Minimax-algoritmi olettaa, että minimoiva pelaaja pelaa täydellisesti ja tulee valitsemaan voittavan siirron, jos saa siihen mahdollisuuden. Tästä syystä maksimoiva pelaaja (tekoäly) valitsee varman siirron niin, että peli päättyy tasan.

Käytetään toisena esimerkkinä jätkänshakkia (3x3 ristinolla), jossa on kolmenlaisia päätetiloja ja joille voidaan antaa seuraavat numeroarvot: voitto +1, häviö -1 ja tasapeli 0. Kuvassa 2 on esitetty esimerkkitilanne jätkänshakin pelipuusta, kun peli on lähellä päätetiloja. Aluksi Minimax-algoritmi luo koko pelipuun ylhäältä alas päätetiloihin asti. Tämän jälkeen päätetilat arvioidaan antamalla niille arvot +1, -1 tai 0 pelin lopputuloksesta riippuen. 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 solmun haaroissa on kaksi vaihtoehtoa, merkataan solmuun suurempi arvoista. Tällä tavalla pakataan numeroarvot aina pelipuun ylimpään solmuun asti kerros kerrallaan. Kuvan 2 ylimmässä solmussa on lopulta vaihtoehtoina arvot +1, -1 ja 0, ja koska kyseessä on maksimoivan pelaajan vuoro, valitsee algoritmi vaihtoehdoista suurimman arvon +1 eli voiton.



Kuva 2. Jätkänshakin pelipuu saadaan ratkaistua Minimax-algoritmilla.

Jätkänshakin koko hakuavaruus koostuu 255 168 tilasta ja ottamalla pelilaudan symmetriat huomioon, niin ainoastaan 26 830 tilasta. Monissa muissa peleissä kuten shakissa tai Go:ssa hakuavaruus on niin iso, että normaalilta Minimax-algoritmilta menee käytännössä äärettömän kauan koko pelipuun tutkimiseksi. Jotta mahdollisimman hyvä siirto löydettäisiin siedettävässä ajassa (vaikkakaan ei optimaalisin), on yksi ratkaisu katkaista haku ennalta määrätyltä syvyydeltä ja antaa tämän syvyyden päätetiloille arvo erillistä arviointifunktiota käyttäen. Lisäksi voidaan käyttää hakuavaruuden rajaamiseksi Alpha-Beta karsintaa (engl. alpha-beta pruning), joka karsii pois haarat, jotka eivät vaikuta tekoälyn lopulliseen päätökseen. Alpha-Beta karsinta tuottaa saman lopputuloksen kuin Minimax-algoritmi mutta nopeammassa ajassa. Parhaassa tapauksessa samalla määrällä laskutoimituksia Alpha-beta karsinnalla voidaan tutkia kaksi kertaa syvemmältä kuin pelkällä Minimax-algoritmilla [1].

Tehtävä 1 (2.5p)

Eräässä uudessa tosi-tv sarjassa kolme pariskuntaa (mies ja nainen) on kuljetettu saarelle. Pariskuntien tehtävänä on siirtyä pienen veneen avulla saarelta takaisin mantereelle. Veneeseen mahtuu maksimissaan kaksi henkilöä kerrallaan ja vene tarvitsee vähintään yhden henkilön liikkuakseen. Sarjan käsikirjoittajat pyrkivät luomaan konfliktitilanteita, joiden välttämiseksi pariskunnan mies ei saa jättää vaimoaan toisen miehen seuraan niin, että hän ei ole itse läsnä. Pariskunnat päättävät alkaa pelaamaan sarjan käsikirjoittajia vastaan ja yrittävät siirtyä saaren ja mantereen välillä niin, että kaikki mahdolliset konfliktitilanteet jäävät syntymättä. He päättävät siis käyttää ongelman selvittämiseksi joko leveyshakua tai syvyyshakua.

Tiedostossa **rivercrossing.py** on valmiiksi toteutetut hakualgoritmit leveyshaku ja syvyyshaku kyseiselle joen-ylitys-ongelmalle (engl. river-crossing puzzle). Ongelman mallintamisessa on käytetty listoja meritien molemminpuolisista tiloista. Saaren puoleisen tilan listasta käytetään nimeä *startingside* ja mantereen puoleisen tilan listasta nimeä *goalside*. Peliin osallistuvista objekteista käytetään nimiä: H1, H2, H3, W1, W2, W3 ja B, missä H viittaa mieheen (engl. husband), W viittaa vaimoon (engl. wife) ja B viittaa veneeseen (engl. boat). Pelin alkutilanteessa kaikki kolme pariskuntaa ja vene ovat saarella, jolloin *startingside* = [”H1”, ”H2”, ”H3”, ”W1”, ”W2”, ”W3”, ”B”] ja *goalside* = []. Koska kaikki peliin osallistuvat objektit kuuluvat jompaankumpaan listaan, voidaan *startingside*:n ja *goalside*:n katsoa olevan toistensa vastatiloja. Yhdellä puolella erilaisia mahdollisia tiloja on yhteensä 2^7 = 128 kappaletta, mutta kun venettä ei oteta huomioon, niin tiloja on 64 kappaletta. Kuvassa 3 on havainnollistettu ongelman alkutilanne.

Kuva 3. Joen-ylitys ongelman alkutilanne.

Sinun tehtävänä on selvittää kielletyt tilat listana sääntöpohjaisesti kyseiselle joen-ylitys ongelmalle. Vene jätetään kokonaan huomioimatta, koska sillä ei ole vaikutusta kiellettyihin tiloihin. Kaikista 64 tilasta kiellettyjä tiloja on yhteensä 30 kappaletta. Nämä 64 tilaa on lueteltu tila-vastatila jaottelulla taulukossa 1. Keltainen väri tilan numeron kohdalla kertoo, että kyseessä on kielletty tila.

Taulukko 1. Kolme pariskuntaa – joen ylitysongelman tilat

	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.
Tila		H1	H2	H3	W1	W2	W3	H1 H2	H1 H3	H1 W1	H1 W2	H1 W3	H2 H3	H2 W1	H2 W2	H2 W3
	17.	18.	19.	20.	21.	22.	23.	24.	25.	26.	27.	28.	29.	30.	31.	32.
Vasta-tila	H1 H2 H3 W1 W2 W3	H2 H3 W1 W2 W3	H1 H3 W1 W2 W3	H1 H2 H3 W1 W2 W3	H1 H2 H3 W2 W3	H1 H2 H3 W1 W3	H1 H2 H3 W1 W2	H3 W1 W2 W3	H2 W1 W2 W3	H2 H3 W2 W3	H2 H3 W1 W3	H2 H3 W1 W2	H1 W1 W2 W3	H1 H3 W2 W3	H1 H3 W1 W3	H1 H3 W1 W2

	33.	34.	35.	36.	37.	38.	39.	40.	41.	42.	43.	44.	45.	46.	47.	48.
Tila	H3 W1	H3 W2	H3 W3	W1 W2	W1 W3	W2 W3	H1 H2 H3	H1 H2 W1	H1 H2 W2	H1 H2 W3	H1 H3 W1	H1 H3 W2	H1 H3 W3	H2 H3 W1	H2 H3 W2	H2 H3 W3
	49.	50.	51.	52.	53.	54.	55.	56.	57.	58.	59.	60.	61.	62.	63.	64.
Vasta-tila	H1 H2 W2 W3	H1 H2 W1 W3	H1 H2 W1 W2	H1 H2 H3 W3	H1 H2 H3 W2	H1 H2 H3 W1	W1 W2 W3	H3 W2 W3	H3 W1 W3	H3 W1 W2	H2 W2 W3	H2 W1 W3	H2 W1 W2	H1 W2 W3	H1 W1 W3	H1 W1 W2

Muokkaa funktiota *illegalStates(roles)*, joka ottaa argumenttinaan peliin osallistuvat objektit *roles*. Funktion tulee palauttaa lopuksi kaikki kielletyt tilat. Funktion sisällä sinulle on jo annettu kaikki taulukossa 1 luetellut 64 tilaa listana *allstates*. Lisäksi sinulle on annettu tyhjät listat laillisille tiloille (*legalstates*) sekä laittomille tiloille (*illegalstates*). Käy aluksi läpi kaikki tilat (*allstates*) silmukassa (eng. for loop) seuraavien ohjeiden mukaisesti:

Aivan silmukan alussa sinun tulee erottaa läpikäytävä tila erillisiin *husbands* and *wifes* listoihin. Esimerkiksi, jos läpikäytävä tila on *state* = [”H1”, ”H2”, ”W1”, ”W3”], niin se voidaan jakaa niin, että *husbands* = [”1”, ”2”] ja *wifes* = [”1”, ”3”].

Seuraavaksi tallenna tila lailliset tilat listaan *legalstates*, jos:

1. tilassa ei ole yhtään miestä tai ei yhtään vaimoa (len(husbands) == 0 or len(wifes) == 0)

(Ehdon täyttävät taulukon tilat: 1,2,3,4,5,6,7,8,9,13,36,37,38,39,55)

2. tilassa on vain samoja pariskuntia (husbands == wifes)

(Ehdon täyttävät taulukon tilat: 1,10,15,17,26,31,35,51)

3. tilassa on kaikki kolme miestä (len(husbands) == 3)

(Ehdon täyttävät taulukon tilat: 17,21,22,23,39,52,53,54)

4. tilassa on kaksi miestä ja yksi vaimo, joista vaimo ja toinen miehistä ovat pariskunta (len(husbands) == 2 and len(wifes) == 1 and wifes[0] in husbands)

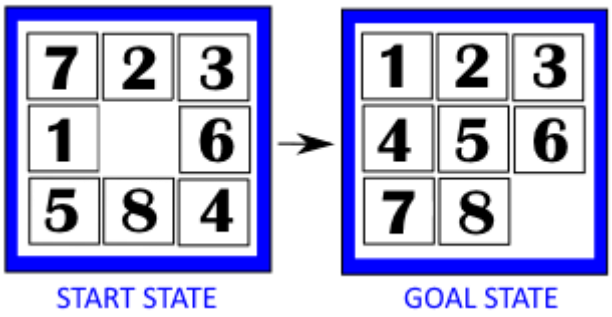
(Ehdon täyttävät taulukon tilat: 40,41,43,45,47,48)

Lopuksi käy läpi aivan uudessa silmukassa kaikki tilat (*allstates*) ja mikäli jokin silmukan tiloista ei kuulu laillisiin tiloihin eli *legalstates* listaan, on sen pakko kuulua laittomiin tiloihin eli *illegalstates* listaan. Funktio palauttaa lopuksi laittomat tilat.

Tämän jälkeen voit halutessasi kokeilla sekä leveyshakua, että syvyyshakua ongelman ratkaisemiseksi. Tämä onnistuu muuttamalla alkuperäisen kooditiedoston rivercrossing.py riviä numero 109.

Tehtävä 2 (2.5p)

Sinun tehtävä on luoda heuristinen arviointifunktio h(n) 8-puzzle peliin. 8-puzzle koostuu kahdeksasta liukuvasta laatasta, jotka ovat numeroitu ykkösestä kahdeksaan ja numeroidut laatat ovat sijoitettu 3x3 ruudukolle. Yksi ruudukon paikoista jää tyhjäksi, johon voi siirtää minkä tahansa horisontaalisesti tai vertikaalisesti tyhjän kohdan vieressä olevan laatan. Kun jokin laatoista siirretään tyhjään paikkaan, jää laatan aiempi paikka tyhjäksi. 8-puzzlen tarkoitus on saavuttaa tavoitetila mahdollisimman vähin siirroin satunnaisesti sekoitetusta tai ennalta määrätystä alkutilasta. Kuvassa 4 on esitetty 3x3 8-puzzle pelin eräs alkutila ja tavoitetila.



Kuva 4. Eräs alkutila ja tavoitetila 3x3 8-puzzle-pelille.

Tyypillinen ratkaisu pelille on noin 20 siirtoa pitkä ja sen haarautumiskerroin (eli kuinka monta siirtoa keskimäärin on mahdollista suorittaa yhden vuoron aikana) on 2,66. Tämä tarkoittaa sitä, että perusteellinen haku 20 siirron syvyydestä käy läpi $(2,66)^{20}$ = $3,15*10^8$ tilaa. Tästä syystä on erityisen tärkeä löytää mahdollisimman hyvä heuristinen funktio kyseiselle ongelmalle. Hyviä vaihtoehtoja heuristisiksi funktioiksi ovat joko **hamming-etäisyys** tai **city-block-etäisyys** (tunnetaan myös nimellä manhattan-etäisyys).

Hamming-etäisyys: Laskee väärillä paikoilla olevien laattojen lukumäärän. Tila, jolla on vähän laattoja väärillä paikoilla, on lähempänä tavoitetilaa kuin esimerkiksi tila, jolla on kaikki laatat väärillä paikoilla.

City-block-etäisyys: Laskee kaikkien laattojen horisontaalisten ja vertikaalisten etäisyyksien yhteenlasketun summan tavoitetilasta.

Esimerkiksi hamming-etäisyys kuvan 4 tilanteelle on $h_1(n) = 4$, sillä alkutilan laatoista neljä on väärillä paikoilla tavoitetilaan nähden. City-block-etäisyys lasketaan kuvalle 4 seuraavasti: $h_2(n) = 1 + 0 + 0 + 3 + 2 + 0 + 2 + 0 = 8$. Lopputilan tyhjällä laatalla ei ole vaikutusta kummankaan heuristisen funktion arvoon.

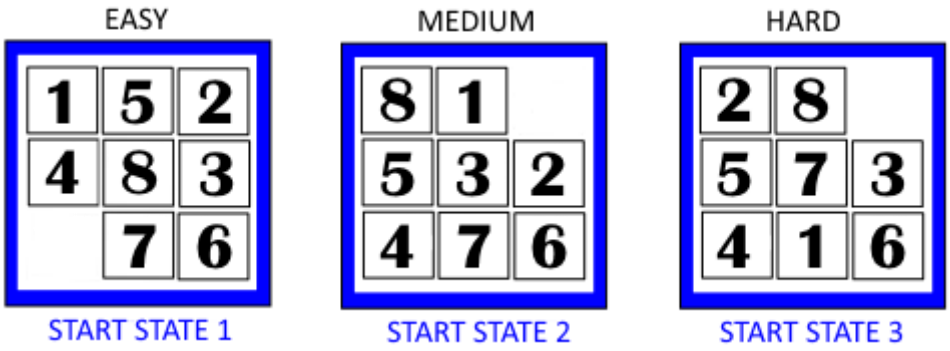
A*hakualgoritmissa polun kustannusarviona g(n) on käytetty pelipuun syvyyttä. Lisäksi algoritmissa tutkimattomat tilat tallennetaan avoimeen listaan *openlist* ja aiemmin käsitellyt tilat tallennetaan suljettuun listaan *closedlist*. Kaavassa 1 esitetyn funktion f(n) perusteella algoritmi valitsee seuraavaksi tutkittavaksi tilaksi aina edullisimman vaihtoehdon. Lopuksi algoritmin löydettyä ratkaisu, voidaan rakentaa polku alkutilasta tavoitetilaan.

Tiedostossa **8puzzle.py** sinulle on jo annettu 8-puzzle peli, pelin ratkaiseva A*-hakualgoritmi satunnaisesti sekoitetulle tilalle sekä heuristinen arviointifunktio city-block-etäisyys. Sinun tehtäväsi on luoda pelille heuristisena funktiona myös hamming-etäisyys sekä vertailla näiden kahden heuristisen funktion suorituskykyä erilaisille alkutiloille.

Heuristinen funktio *calculateHamming(state)* ottaa tiedostossa argumenttina tutkittavan tilan *state*. Tila on toteutettu 9 alkiota sisältävänä listana, missä arvo 0 kuvaa tyhjää ruutua. Listan arvot kulkevat pelilaudan vasemmasta ylälaidasta oikeaan alalaitaan. Esimerkiksi kuvan 3 alkutila listana on *startstate* = [7,2,3,1,0,6,5,8,4] ja tavoitetila on *goalstate* = [1,2,3,4,5,6,7,8,0]. Tavoitetilaa tulee käyttää arviointifunktiossa ja sitä voidaan kutsua funktion sisällä komennolla *self.targetstate*.

Hamming-etäisyys voidaan toteuttaa käymällä silmukassa läpi indeksit 0-8. Aluksi muuttujan *hammingvalue* arvo alustetaan arvolla 0. Mikäli tutkittavan tilan listassa *state* oleva alkio tietyllä indeksillä on eri kuin tavoitetilan *self.targetstate* alkio samalla indeksillä, lisätään *hammingvalue* muuttujaan arvo 1. Lisäksi tutkittavan tilan *state* läpikäytävän alkion ollessa 0 eli tyhjä ruutu, jätetään lisäys tekemättä.

Kun saat toteutettua hamming-etäisyys arviointifunktion, testaa molempien heurististen arviointifunktioiden suorituskykyä *main()* funktion sisällä kuvan 5 eri vaikeusasteisille alkutiloille. Tavoitetila on sama kuin kuvan 4 tavoitetila. *main()* funktion sisällä on testattuna esimerkkinä satunnaisesti sekoitettua tilaa city-block-etäisyys arviointifunktiolla sekä alkutilaa *startstate1* molemmilla arviointifunktioilla. Sinun tehtävä on siis testata lisäksi alkutiloja *startstate2* ja *startstate3* molemmilla arviointifunktioilla.



Kuva 5. Kolme eri vaikeusasteista alkutilaa 3x3 8-puzzle-pelille.

Lähteet

[1] Russell S. & Norvig P. (2009) Artificial Intelligence : A Modern Approach (third edition). Prentice-Hall

Harjoituksen 1 oppimistavoitteet

- ymmärtää leveyshaun, syvyyshaun ja heuristisenhaun toimintatavat ja niiden käyttämät tietorakenteet.
- osaa mallintaa yksinkertaisen pelin tai ongelman puukaaviona
- osaa luoda optimaalisuusehdon täyttäviä heuristisia funktioita.
- osaa ratkaista Minimax-algoritmia käyttäen yksinkertaisen pelipuun

Palauta

Palauta muokkaamasi python tiedostot (.zip tai .rar tiedostoon pakattuna) Optiman palautuslaatikkoon Harjoitus 1 **24.3.2019 klo 23:59** mennessä. Tästä harjoituksesta voit tienata 5 pistettä (2.5p + 2.5p). Voit antaa palautetta tästä harjoituksesta liittämällä pakattuun tiedostoon erillisen tekstitiedoston palautetta varten. Harjoituksia tullaan kehittämään palautteen pohjalta tuleville vuosille vastaamaan paremmin oppimistavoitteita.