TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Laura Katariina Teder 175065IDDR

Mängu 2048 lahendamine Monte Carlo puuotsinguga

Diplomitöö

Juhendaja: Peeter Ellervee
Ph.D

Tallinn 2021

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Laura Katariina Teder

17.05.2021

Annotatsioon

Antud lõputöö eesmärgiks oli mängu 2048 mängiva tehisintellekti programmi loomine ning saadud tulemuste hindamine. Mäng 2048 on ühe inimese poolt mängitav mäng, mis sisaldab endas juhuslikkuse elementi. Mängu 2048 tulemuse hindamiseks saab kasutada kahte tunnust: mänguskoori ning kõrgeima mängu lõpuks saadud klotsi väärtust – neist viimast kasutati tulemuste hindamisel antud töös.

Töö tulemusena loodi programm, mis võimaldab mängida mängu 2048 kasutades nelja erinevat viisi käikude valimiseks: juhuslikud käigud, nurga strateegia, Monte Carlo puuotsing ning Monte Carlo puuotsing kombineerituna nurga strateegiaga. Tulemustest selgus, et ehkki Monte Carlo puuotsing saavutas paremaid tulemusi kui juhuslike käikude tegemine või nurga strateegia, pikendas see samal ajal oluliselt ühe mängu mängimiseks kuluvat aega. Monte Carlo simulatsioonide arvu suurendamine parandas tulemusi, kuid koos tulemustega kasvas märkimisväärselt ka üheks mänguks kuluv aeg. Võttes arvesse ka aja aspekti, saavutati kõige paremaid tulemusi Monte Carlo puuotsingu kombineerimisel nurga strateegiaga.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 43 leheküljel, 3 peatükki, 21 joonist, 2 tabelit.

Abstract

Solving the Game 2048 with Monte Carlo Tree Search

The aim of this thesis was to develop an AI program that plays the game 2048 using Monte Carlo tree search and to evaluate the results. The game 2048 is a single-player tile-based game with random elements. The game allows to use two different features for evaluation of results: score and the highest tile achieved; the latter was used in this paper.

A program was developed that plays the game 2048 allowing to use four different ways for move selection: random moves, corner strategy, Monte Carlo tree search and Monte Carlo tree search combined with corner strategy. The results illustrate that while Monte Carlo tree search produced better results than playing random moves or using corner strategy alone, it also resulted in significantly longer computing time. Taking time aspect into consideration, better results were achieved by combining Monte Carlo tree search with corner strategy, rather than by purely increasing the Monte Carlo simulation count.

The thesis is in Estonian and contains 43 pages of text, 3 chapters, 21 figures, 2 tables.

Lühendite ja mõistete sõnastik

Expectimax	Minimaxil põhinev otsingualgoritm, mis püüab maksimeerida oodatavat kasu
Heuristiline otsing	Otsingu juhtimiseks või tõhustamiseks kasutatakse kogemuslikke teadmisi ja ainevaldkonna kohta käivat infot
MCTS	*Monte Carlo tree search*, Monte Carlo puuotsing
Minimax	Otsingualgoritm, mis püüab minimeerida võimalikku kahju halvima stsenaariumi korral
Monte Carlo meetodid	Rühm arvutialgoritme keerukate protsesside modelleerimiseks, mis kasutavad hinnanguliste tulemuste arvutamiseks korduvat juhuslikku valimit
Monte Carlo puuotsing	Otsingualgoritm teatud tüüpi otsustusprotsesside jaoks, eriti nende jaoks, mida kasutatakse lauamänge mängivas tarkvaras
Puu hargnemistegur	Maksimaalne ühest tipust väljuvate kaarte arv
Tehisintellekt	Arvutisüsteem, mis on võimeline sooritama ülesandeid, mis tavapäraselt vajavad inimintelligentsi
UCB1	*Upper Confidence Bound*, algoritm, mis tasakaalustab otsingupuus edukamate tippude eelistamist ja väheuuritud tippude uurimist

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Mängud said tehisintellekti üheks oluliseks uurimisvaldkonnaks juba eelmise sajandi keskpaigas. Kõige edukamaks tehisintellekti meetodiks keeruliste mängude puhul peetakse otsingut, mille varjuküljeks on aga sageli suur ajakulu. Levinud mängudes kasutatavaks otsingutehnikaks on pikalt olnud minimaxil põhinevad otsingutehnikad, sealhulgas expectimax, ent üha enam ka Monte Carlol põhinev Monte Carlo puuotsing. Monte Carlo puuotsingut hakati mängude lahendamisel hindama peaasjalikult alates sellest hetkest, kui osaliselt sellel algoritmil põhinev Go-d mängiv programm AlphaGo võitis 2015. aastal professionaalset Go mängijat.

Käesoleva lõputöö eesmärgiks on luua juhuslikkuse elementi sisaldava mängu 2048 lahendamise tehisintellekt, mis kasutab mängukäikude valikul Monte Carlo puuotsingut, ning saadud tulemusi ja meetodi sobilikkust hinnata. Selle jaoks kirjutatakse esmalt mängu 2048 mängiv programm, mis kasutab mängimisel juhuslikke käike, ning leitakse 10 000 juhusliku mängu tulemused ja keskmine üheks mänguks kuluv aeg. Järgmisena lisatakse programmile nurga strateegia kasutamine, mille põhimõtteks on suurema väärtusega klotside koondamine ühte nurka ning mis on üheks levinumaks heuristiliseks meetodiks mängu 2048 puhul.

Kolmanda meetodina kasutatakse Monte Carlo puuotsingut, mille puhul tuleb välja töötada ka mängu lõppseisude hinnangufunktsioon, kuna võit-kaotus skaala pole antud mängu spetsiifikat arvestades sobiv. Neljanda mängukäikude valimise meetodina kombineeritakse Monte Carlo puuotsing nurga strateegiaga. Iga eeltoodud meetodiga saadud tulemusi võrreldakse omavahel. Monte Carlo puuotsingu puhul hinnatakse täiendavalt Monte Carlo simulatsioonide arvu mõju tulemustele ja mänguks kuluvale ajale. Lõputöö tulemusena luuakse eelkirjeldatud programm ning antakse hinnang, milline kasutatud meetoditest on nii tulemusi kui ajafaktorit arvesse võttes optimaalseim mängu 2048 lahendamiseks.

Lõputöö koosneb kolmest peatükist. Esimeses peatükis antakse põgus ülevaade mängudest tehisintellekti uurimisvaldkonnana ning kirjeldatakse mängu 2048 tausta, reegleid ja võimalikke saavutatavaid tulemusi selle mängimisel. Teises peatükis kirjeldatakse puuotsingu meetodite kasutamist mängude lahendamiseks ning analüüsitakse Monte Carlo puuotsingu meetodi sobilikkust mängu 2048 jaoks. Kolmandas peatükis kirjeldatakse loodud programmi, mis sisaldab endas mängu 2048 reegleid ja mängu mängivat tehisintellekti, ning analüüsitakse saadud tulemusi.

2 Mäng 2048

Mängud on tehisintellekti uurimisvaldkonnana olnud aktuaalne aastakümneid, kuna algoritme täiustatakse pidevalt ning suuri edasiminekuid on võimaldanud ka protsessorite võimekuse kasv. Kui eelmise sajandi keskpaigas olid kesksteks uuritavateks mängudeks male ja kabe, siis 2014. aastal välja lastud mäng 2048 kogus selle väljalaskmise järel tehisintellekti valdkonnas kiiresti populaarsust.

2.1 Mängud tehisintellekti uurimisvaldkonnana

Tehisintellekt (*artificial intelligence, AI*) on loomuliku intellekti jäljendamine, s.t arvutisüsteemi võime täita funktsioone, mida üldiselt seostatakse inimmõistusega, näiteks arutleda ja õppida. Intellektitehnika ehk uurimissuund, mis tegeleb tehisintellektisüsteemide väljatöötamisega, on kujunenud laiahaardeliseks teadusalaks, mille põhiküsimused on endiselt aktuaalsed. Mitmed ideed on intellektitehnikasse tulnud just mängude mängimise analüüsimisest [1].

Mängude puhul kasutatakse tehisintellekte erinevatel otstarvetel – nii mängumaailmade haldamiseks, mingi mängu üksuse käitumise modelleerimiseks reaalajas, samuti mängija poolt antud sisendite töötlemiseks ning nende alusel mängus olevate tegelaste käitumise ja emotsioonide simuleerimiseks [2]. Peale selle on mängud levinud tehisintellekti uurimisobjektiks, kus tehisintellekti abil püütakse saavutada paremaid mängutulemusi, kui on saavutanud professionaalsed mängijad. Muuhulgas on loodud ka rahvusvaheline arvutimängude assotsiatsioon (*International Computer Games Association*), mis korraldab programmide võistlusi males, kabes ja teistes mängudes [1].

Mängude mängimine on tehisintellekti jaoks hea valdkond, sest mängude näol on tegu struktureeritud probleemidega, kus on lihtne mõõta edu ja ebaedu. Esimene märkimisväärne kabemängu mängimise programm õnnestus luua 1960. aastate algul Arthur Samuelil. 1970. aastal alustati Põhja-Ameerika maleprogrammide tšempionaate [1], ehkki tolleaegsed mängud ei mänginud eriti head malet, vaid pakkusid huvi peamiselt

tehisintellekti võimaluste uurimise seisukohalt [3]. 1997. aastal võitis aga maleprogramm Deep Blue tolleaegset maailmameistrit Garri Kasparovi [1].

Mänge lahendavad programmid on toonud mitmeid edukaid tulemusi sellistes mängudes nagu male, kabe ja triktrakk. Sageli on uuritavates mängudes kombineeritud mitmed keerukust lisavad tunnused nagu mitu mängijat, varjatud informatsioon ja juhuslikud elemendid ning lisaks sellele ka vähesed avaldatud ekspertteadmised antud teema kohta. Selleks, et lihtsustada tehisintellekti loomist, kasutavad paljud avaldatud tööd piiratud alamhulka mängu reeglitest. Monte Carlo puuotsing, mis kogus palju kuulsust seoses mänguga Go, on selles kontekstis huvipakkuvaks meetodiks [4].

Mäng 2048 kogus maailmas pärast selle välja laskmist kiiresti populaarsust. Inimesed üle maailma on kulutanud miljoneid tunde püüdmaks jõuda klotsini väärtusega 2048. Peale sõltuvusttekitava mängu mängimise annab mäng huvitava võimaluse uurida seda matemaatilise poole pealt, kasutades näiteks matemaatilist induktsiooni, numbriteooriat ja topoloogiat, püüdes samal ajal leida ka optimaalset võidustrateegiat [5].

2.2 Mängu 2048 eellugu

Threes! (leitav veebiaadressil http://asherv.com/threes/) on populaarne mäng, mille lõid Asher Vollmer, Greg Wohlwend ja Jimmy Hinson. Mäng lasti välja 2014. aasta 23. jaanuaril Sirvo poolt iOS operatsioonisüsteemile ning see pälvis märkimisväärse tähelepanu nii mängijate, mängukriitikute kui mängudisainerite poolt. Vaid mõned nädalad pärast selle väljalaskmist ilmus Androidile mõeldud kloon Fives ning seejärel iOS-i kloon 1024!, millel olid pisut muudetud reeglid [6].

Veidi aega pärast seda - 2014. aasta märtsis - lasti samal päeval GitHubis välja kaks vabavaralist veebimängu versiooni, mis mõlemad kandsid nimetust 2048 – üks Samingi poolt, ning teine, mida käsitleb käesolev töö, Gabriele Cirulli poolt [6], [7]. 19-aastane Itaalia veebiarendaja Gabriele Cirulli disainis mängu ühe nädalavahetusega, toetudes Veewo Studio analoogsele mängule 1024. Cirulli ise oli üllatunud, et just tema disain mängust 2048 sai populaarseimaks ja levis kiiresti massidesse [7].

Mängu originaalversioon on mängitav veebiaadressil https://play2048.co/, ent mängust võib leida ka hulganisti kloone ja mobiiliäppe. Mängu levinumad edasiarendused kasutavad suuremat mängulauda ja seavad eesmärgiks suurema väärtusega klotsini jõudmise [6]. Ent näiteks mängul 2048 baseeruv variatsioon nimega Evil 2048 muudab mängija jaoks mängimise raskemaks, püüdes genereerida uusi klotse mängija jaoks kõige ebasoositavamatesse kohtadesse.

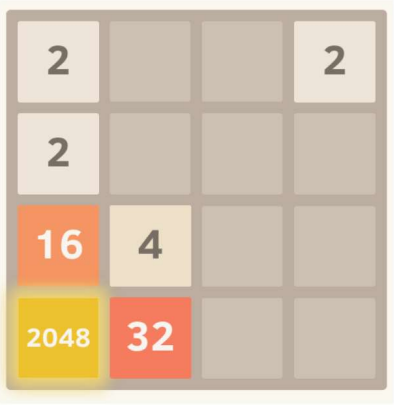
2.3 Mängu 2048 reeglid

Mäng 2048 on ühe mängija poolt mängitav veebimäng 4x4 mängulaual, mille iga lahter on kas tühi või täidetud klotsiga, mille väärtuseks on kahe aste [8]. Mängu 2048 alguses antakse mängijale kaks klotsi juhuslikes asukohtades. Klotsid võivad omada väärtust kas 2 või 4 (s.t kas kaks klotsi väärtusega 2, kaks klotsi väärtusega 4 või üks klots väärtusega 2 ja teine väärtusega 4), sealjuures väärtuse 4 esinemise tõenäosus on 1/10-le [5]. Iga mängukorra ajal teeb mängija käigu, valides ühe neljast suunast - üles, alla, vasakule või paremale. Pärast suuna valimist liiguvad kõik klotsid laual selles suunas nii kaugele kui võimalik ehk kuni nad kas puutuvad vastu mängulaua äärt või kõrval olevat klotsi.

Kui klots väärtusega v liigub mängija poolt valitud suunas vastamisi sama väärtusega klotsiga, siis need kaks klotsi liituvad omavahel ning klotsi uueks väärtuseks saab kahe omavahel liitunud klotsi väärtuste summa ($2v$). Samal ajal lisandub mängija punktisummale $2v$ punkti. Pärast seda, kui mängija on teinud lubatava käigu, genereerib mäng uue klotsi, mille väärtuseks on jällegi 9/10 tõenäosusega 2 ja 1/10 tõenäosusega 4. Uus klots genereeritakse juhuslikult valitud tühja lahtrisse [8]. Mänguväljal saavad olla mängu jooksul vaid väärtused, mis on kahe astmed. Selle saab matemaatiliselt ära tõestada kasutades matemaatilist induktsiooni [5].

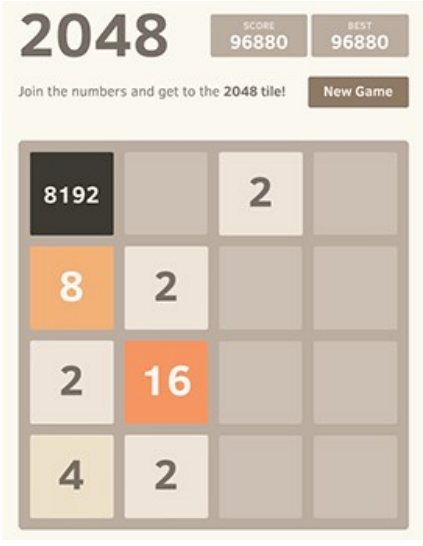
2.3.1 Mängu eesmärk

Mängu peamine eesmärk on klotside liitmise teel jõuda klotsini väärtusega 2048 (2^{11}) nagu on näidatud joonisel 1 [9].



Joonis 1. Mängulaud, kus on saavutatud eesmärgiks olev klots väärtusega 2048 [9].

Ehkki kokkuleppeliselt võidab mängija mängu, kui ta on jõudnud klotsi väärtusega 2048 loomiseni, võimaldab see siiski mängu edasi mängida. Mängu teiseks eesmärgiks on koguda võimalikult palju punkte [8]. Lõplik skoor on mängu jooksul kogutud punktisumma. Kahe madalama astmega klotsi ühendamine lisab mänguskoorile klotside ühendamise käigus saadud klotsi väärtuse jagu punkte (näiteks kahe 4se klotsi ühendamine lisab skoorile 8 punkti). Joonisel 2 on näha mängu seis pärast umbes 4000 käiku, mil on jõutud klotsini väärtusega 8192. Mängu skooriks on sellel hetkel 96880 punkti [10].



Joonis 2. Mängu 2048 seis, milles on saavutatud klots väärtusega 8192 [10].

2.3.2 Mängu lõpp

Kokkuleppeliselt on mäng võidetud, kui saadakse klots väärtusega 2048, ehkki pärast seda võib mängimist jätkata senikaua, kuni ühtegi lubatud käiku pole enam võimalik teha,

lugedes seda mängu lõpuks [5]. Käik on lubatud, kui vähemalt ühte klotsi saab sellega liigutada [8]. Mängu mängides saab kiiresti selgeks, et on võimatu mängida ühte 2048 mängu lõputult [10].

2.3.3 Maksimaalne võimalik klotsi väärtus

Teoreetiliselt on suurimaks võimalikuks saavutatavaks klotsi väärtuseks 131 072, mis on võimalik juhul, kui viimasena mängulauale tekkiva klotsi väärtuseks on 4. Juhul, kui viimasena mängulauale tekkiva klotsi väärtuseks on 2 (tõenäosus 9/10), on suurimaks võimalikuks saavutatavaks klotsi väärtuseks 65536, milleks peavad laual olema klotsid väärtustega 32768, 16384, 8192, 4096, 2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 2 [11].

2.3.4 Minimaalne võimalik käikude arv mängu võitmiseks

Nagu mängu lõppemise kirjelduses on välja toodud, on kokkuleppeliselt mäng võidetud, kui saadakse klots väärtusega 2048. Kuna uued klotsid on juhuslikult genereeritud – väärtusega 2 tõenäosusega 0,9 ja väärtusega 4 tõenäosusega 0,1 – pole võimalik öelda täpset minimaalset käikude arvu mängu võitmiseni, ent saab välja arvutada käikude arvu vahemiku.

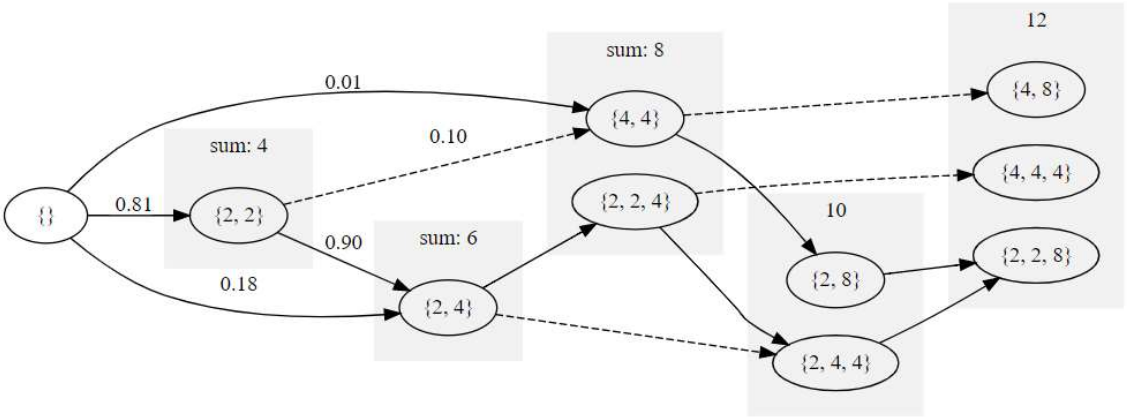
Eeldades parima võimaliku stsenaariumi realiseerumist, mille korral oleksid kõik uued klotsid väärtusega 4 ning iga käigu ajal liituksid kaks klotsi üheks, oleks klotsi 2048 saamiseks vaja vähemalt 512 klotsi (512x4=2048) ning sooritada tuleks 519 käiku. Kui eeldada aga, et saadakse vastupidiselt kõik klotsid väärtusega 2, siis oleks vaja mängulauale saada 1024 uut klotsi ning sooritada vähemalt 1032 käiku. Seega minimaalne mängukäikude arv, mis on vajalik mängu võitmiseks, eeldusel, et mängitakse täiuslikku mängu, on vahemikus [519, 1032] [5].

Kasutades klotside 2 ja 4 esinemise realistlikke tõenäosuseid, saab leida minimaalse võimaliku käikude arvu mängu võitmiseni modelleerides mängu 2048 Markovi ahelana. Et mängida täiuslikku mängu, mille korral saab iga mängukäigu ajal kaks klotsi ühendada, jäetakse Markovi ahela koostamisel välja mänguväli ning pannakse klotsid nii-

öelda ühte kotti kokku, kus samade väärtustega klotsid liituvad sõltumata nende paiknemisest.

Selleks, et esitada selliselt lihtsustatud 2048 mängu Markovi ahelana, tuleb määratleda ahela olekud ja üleminekute tõenäosused. Iga olek sümboliseerib mänguväljal olevaid klotse mingil ajahetkel ning ülemineku tõenäosused täpsustavad, milline olek tuleb millise tõenäosusega järgmisena. Iga olek defineeritakse kotis olevate klotside kaudu, klotside järjestust arvesse võtmata. Selleks, et esitada seda Markovi ahelas, peab välja arvestama ülemineku tõenäosused esialgsest olekust igasse võimalikku järgnevasse olekusse. Kuna klotside paiknemist mängulaual arvesse ei võeta, on tõenäosuste arvestamisel olulised vaid klotside 2 ja 4 esinemise tõenäosused. Kui samade väärtustega klotside paarid on ühendatud, lisab mäng uue juhusliku klotsi järgnevasse olekusse juurde, milleks tõenäosusega 0,9 on 2 ja tõenäosusega 0,1 on 4 [12].

Joonisel 3 on välja toodud esimesed käigud, kus ilma sildita pideva joonega märgitud noolte puhul on antud ülemineku tõenäosus 0,9 ning katkendliku joonega märgitud üleminekute tõenäosus on 0,1. Grupeerides olekud klotside väärtuste summa järgi kihtidesse joonistub välja, et iga ülemineku korral (v.a üleminek kõige esimesest olekust) minnakse üle kas järgmisesse või ülejärgmisesse kihti tõenäosusega vastavalt kas 0,9 või 0,1 [12].



Joonis 3. Mängu 2048 Markovi ahela algus, mis ei võta arvesse klotside asetust mänguväljal [12].

Keskmine minimaalne käikude arv leitakse mängu simulatsioonide läbiviimise teel, kus iga käivitatud simulatsiooni korral genereeritakse üks kindel ahelat läbiv trajektoor, valides olekuid juhuslikkuse alusel proportsionaalselt ülemineku tõenäosustega. Miljoni simulatsiooni tulemusena saadi keskmiseks minimaalseks mängu võitmiseks vajalikuks

käikude arvuks 938,8 (standardhälbega 8,3 käiku) – seda eeldusel, et mängitakse täiuslikku mängu ehk võtmata arvesse klotside asukohti mängulaual. Keskmine minimaalne käikude arv 938,8 on märksa lähemal 1032le kui 519le, kuna klotsid väärtusega 2 esinevad oluliselt suurema tõenäosusega kui klotsid väärtusega 4 [12].

Eelnevad arvutuskäigud võimaldavad anda parema hinnangu töö praktilises osas loodava mängu 2048 lahendamise programmi keerukusele ja lahendamise jaoks loodava otsingupuu suurusele. Järgmises peatükis kirjeldatud otsingupuude puhul on suurt hulka võimalikke mängulaua olekuid sisaldavate mängude juures üheks peamiseks probleemiks otsingupuu eksponentsiaalne kasv ning sellest tingitud vajadus piirata otsingut seatud tingimuste abil, näiteks seades piiri otsingu sügavusele puus või läbides tippe valikuliselt.

3 Mängude lahendusalgoritmid

Mängu 2048 Javascriptis kirjutatud lähtekood on vabalt kättesaadav GitHubis aadressil https://github.com/gabrielecirulli/2048 ning see on motiveerinud paljusid looma mängust erinevaid kloone, ent ka mängu heal tasemel mängivaid tehisintellekte. Nii professionaalsed kui hobiprogrammeerijad on loonud erinevaid strateegiaid ja algoritme kasutavaid programme mängu 2048 mängimiseks. Peamiseks eesmärgiks on saavutada suuri klotsi väärtusi ja kõrgeid skoore, milleks tuleks leida võimalikult optimaalne lahendusalgoritm ja strateegia antud mängu jaoks.

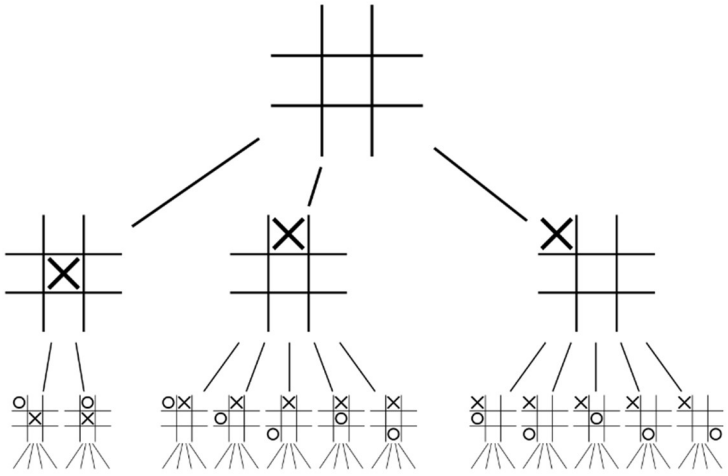
3.1 Mängude lahendamine kasutades puuotsingut

2000ndate aastate alguses oli sagedasemaks arvutimängudes kasutatavaks tehisintellektitehnikaks lõplike olekumasinate (*Finite State Machines*) kasutamine, mille puhul mingi mänguobjekti käitumine jagatakse loogilisteks olekuteks selliselt, et iga erineva käitumise jaoks on antud objektil üks olek, mis selle käitumiseni viib. Otsustuspuid, närvivõrke ja geneetilisi algoritme hakati tollal arvutimängude valdkonnas alles kasutusele võtma, kuna varasemalt polnud nende tehnikate kasutamine võimalik protsessorite võimekuse piirangute tõttu [2].

Lõplikke olekumasinaid on küll lihtne mõista, kuid olekute arvu kasvades raske hallata ja struktureerida [2]. Kuna juba definitsiooni järgi on intelligentsete süsteemide puhul tegemist ebakorrapärase ja halvasti struktureeritava probleemalaga, on siin millegi leidmiseks või autonoomse tegutsemise kavandamiseks vaja sageli kasutada otsingu algoritme. Näiteks viib otsingu algoritmini see, kui süsteemi määramispiirkond liigendatakse seisunditeks, määratakse seisundite vahelised üleminekud ja taandatakse lahenduse leidmine tee otsingule lähteseisundist soovitud seisundini [13]. Selliselt saab kirjeldada ka mängukäike, mis ühest mänguseisust järgmisesse viivad.

Puuotsingut kasutavad programmid nõuavad üldjuhul palju rohkem aega kui otsesed meetodid, kuna enne iga käigu tegemist tuleb sooritada otsing puul, mis esitab kõikvõimalikke käikude järjendeid. Puuotsingu eelisteks on aga võimalus kasutada seda ka keerulisemate mängude puhul ning lisaks saab puuotsingut kasutavat programmi täiendada, kasutades teadmisi konkreetse mängu kohta. Näiteks selle asemel, et vaadata kõiki võimalikke järgmisi käike, võib vaadata käikude alamhulka, mis määratakse ära mingi lihtsa algoritmiga. Lisaks, selle asemel, et vaadelda käikude järjendeid kuni ühe mängija võiduni, võib otsida kuni fikseeritud sügavuseni ja seejärel arvutada mängulaua seisu väärtuse, kasutades seatud hinnangukriteeriume. Väga väikeste ülesannete jaoks on puuotsing üldjuhul aga vähem efektiivne kui otsesed meetodid [1].

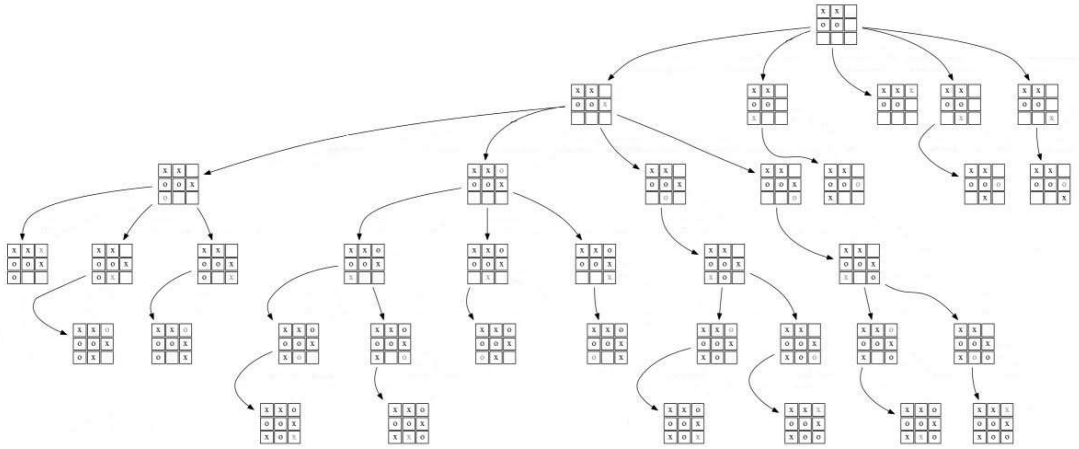
Puuotsingu puhul modelleeritakse mäng puuna, mille tipud tähistavad olekuid (mänguseisu) ja kaared mängukäike, mis ühest olekust teise viivad. Joonistel 4 ja 5 on toodud välja trips-traps-trulli mängupuu osa. Joonisel 4 on nulltasemel juurtipp, mis tähistab mängu algolekut, milleks on tühi 3x3 trips-traps-trulli mänguväli. Esimesel tasemel on toodud välja kolm võimalikku olekut pärast seda, kui X on teinud avakäigu. Sellest järgmisel tasemel on toodud välja mängu võimalikud olekud pärast seda, kui O on teinud vastukäigu. Tasemeid saab välja tuua kuni mängu lõpu olekuni välja [14].



Joonis 4. Trips-traps-trulli mängupuu esimesed kolm taset [14].

Puuotsingu algoritm võimaldab leida kõik mingist mänguseisust tehtavad käigud ning nende käikude järel tekkinud uutest mänguseisudest omakorda võimalikud edasised käigud. Trips-traps-trull on lihtne ja võrdlemisi väheste valikutega mäng, kuid joonisel 5 on näha, et isegi selle mängu puhul on võimalikke käikude järjendeid üksjagu [15]. Võiks

arvata, et programm saab mängu mängida, sooritades täieliku otsingu mänguseisude puul lähteseisust kuni võiduni, ent sageli pole see isegi päris lihtsate mängude puhul võimalik või optimaalne [1].



Joonis 5. Trips-traps-trulli puu suurus pärast seda, kui mõlemad mängijad on teinud kaks käiku [15].

Sundides eksponentsiaalselt kasvavat puud probleemi lõpliku lahenduse leidmiseks arvestama kõigi järglastega nõuab palju arvutusvõimsust, mis suure hargnemisteguriga mängude korral tingib äärmiselt aeglase väljundini jõudmise [15]. Näiteks malemängus on keskmine hargnemistegur 35 ja keskmiselt teeb kumbki mängija 50 käiku. Seega sisaldab puu 35^{100} tippu [1]. Teine äärmus on genereerida korraga ainult jooksva seisu vahetud alluvad, mida kohe testida, et nende hulgast valida parim. Viimane eeldab hindamisfunktsiooni olemasolu, mille alusel saab mänguseisudele hinnanguid anda.

Kiirema puuotsingu saab saavutada metoodika väljatöötamisega, mis annab osadele tippudele suurema tähtsuse, otsides nende tippude järglastest eelisjärjekorras [15]. Lisaks sellele võib otsingu suunamiseks anda kogemuslikku ehk heuristilist lisainfot. Omad heuristilised reeglid on olemas paljudes mängudes. Selline otsing ei garanteeri lühimat ega parimat teed, kuid annab hea lahendi paljudel juhtudel, võimaldades seejuures otsinguruumi suurel määral vähendada [13]. Heuristikud on tehisintellektis väga tähtsad, sest tüüpilised probleemid on nii rasked, et algoritmiline lahend kas ei eksisteeri või on praktikas ebaefektiivne [1].

3.2 Mängu 2048 lahendusalgoritmid

Käesoleva töö praktilises osas kasutatud algoritmi valikut alustati nii professionaalsete kui hobiprogrammeerijate seas populaarselt küsimuste ja vastuste lehelt Stack Overflow. Lehel on arutelu mängu 2048 jaoks kõige optimaalsema algoritmi väljaselgitamiseks [16], sealjuures on tehtud mitmeid antud mängu lahendavaid programme ning jõutud ka muljetavaldavate tulemusteni. Edasi jätkati mängu 2048 lahendamise kohta käivate teadustööde tulemuste analüüsimise ning levinumate meetodite edasiuurimisega.

Uurides varasemalt tehtud 2048 lahendamise tehisintellekte, jäävad peamiselt silma minimaxi, expectimaxi ning Monte Carlo puuotsingu algoritmid, mis kõik kasutavad puuotsingut ning võimaldavad simuleerida erinevate käikude valimise tulemusi. Lisaks neile on üheks populaarseimaks lahendamisstrateegiaks nn nurga strateegia, mille puhul püütakse kasutada võimalusel klotside liigutamiseks vaid kahte-kolme suunda, eesmärgiga suurema väärtusega klotsid ühte nurka kokku koguda. Peale selle võib näiteks igale veerule ja igale reale anda punkte selle eest, kui rea või veeru suurima väärtusega klots paikneb mänguvälja servas [9].

Taiwanis 2014. aastal peetud mängu 2048 arvutiturniiril, kus iga programm pidi mängima 100 mängu keskmise kiirusega üle 100 käigu sekundis, kasutasid kõik esikolmikusse jõudnud programmid expectimaxi otsingut (vt tabel 1). Täiendavalt kasutati mänguseisude hindamiseks mitmeid heuristilisi tehnikaid [8]. Üldjuhul kasutatakse expactimaxi puhul sügavuse piirangut, seda just suure hargnemisteguri tõttu [9].

Tabel 1. 2048 lahendamise boti turniiri esikolmik [8].

Programm	Maksimaal-ne klotsi väärtus	2048 klots	4096 klots	8192 klots	16384 klots	32768 klots
CWU	32768	100%	100%	96%	67%	2%
CGI-2048	16384	100%	100%	94%	59%	0%
OWENLIN	16384	99%	99%	83%	18%	0%

Minimax on sarnaselt expectimaxile küllaltki populaarne mängu 2048 lahendamise algoritm, mille puhul püütakse minimeerida käigust tuleneda võivat maksimaalset kahju. Algoritmi idee on genereerida jooksvast positsioonist lähtudes võimalike järglaste hulk kuni terminaalsete tippudeni, nendele positsioonidele rakendada staatilist

hinnangufunktsiooni ja, liikudes puus tase-haaval kõrgemale tagasi lähtepositsiooni, arvutada selle hinnang [1]. Minimax püüab leida käigu, mis garanteeritult viib parima tulemuseni, ent keerulisemate mängude puhul ei pruugi see olla võimalik, kuna mängupuu läheks väga suureks [14].

Minimax puhul sooritatakse otsing kõikvõimalike mänguseisude puul, eeldades, et igal tasemel püütakse maksimeerida oma võidu tõenäosust, aga vastane püüab seda minimeerida [1]. Stack Overflow kasutajad on toonud välja asjaolu, et kuigi mängus 2048 võetakse vastasena arvutit (ehk pärast igat käiku juhuslikku tühja lahtrisse lisatavat klotsi), siis arvuti vastasena ei püüa mängija võidutõenäosust minimeerida nagu minimaxi algoritm eeldab.

3.2.1 Monte Carlo puuotsing

Monte Carlo puuotsing on tõenäosuslik otsingualgoritm, mis sobib hästi kasutamiseks nende mängude puhul, kus on palju erinevaid valikuvõimalusi. Monte Carlo puuotsingut esitleti 2008. aastal avaldatud uurimustöös uudse, ühtse raamistikuna mängude tehisintellekti jaoks. See kasutab juhuslikke otsinguruumi uurimisi, et ennustada kõige lootustandvamaid mängukäike. Monte Carlo puuotsingut saab kasutada iga lõpliku pikkusega mängu jaoks - nii klassikaliste lauamängude, tänapäevaste lauamängude kui ka videomängude puhul [17].

Mänguteooria algoritmides nagu minimax ja sellel põhinev expectimax on vajalik hinnangufunktsiooni olemasolu, mis hindaks hetkeseisu ja võimaldaks teha kindlaks, millised käigud on head [14]. Selliste mängude puhul nagu Sudoku, male ja Go, samuti 2048, kus on kõrge hargnemistegur, kasvab võimalike käikude hulk mängu edenedes eksponentsiaalselt, mis tähendab omakorda seda, et kõigi käikude väljaarvutamiseks ja hindamiseks vajalik arvutusvõimsus läheb väga suureks. Kuigi kõigi võimalike käikude tulemuste hindamine suurendaks võiduvõimalusi, ei ole see ressursside kasutamise osas väga suure hargnemisteguriga mängude puhul optimaalne [15].

Suure otsingupuuga mängude jaoks on sageli keeruline kirjutada head hindamisfunktsiooni, mis arvutaks välja hinnangu iga mänguseisu jaoks. Monte Carlo puuotsingus rakendatakse Monte Carlo meetodit mängu puuotsingu jaoks. Kuna see põhineb mänguseisudest juhusliku valimi koostamisel, ei nõua see iga mänguseisu

hindamist ega hinnangufunktsioonide kasutamist [18]. Monte Carlo puuotsing tegeleb tõhusalt mängudega, millel on kõrge hargnemistegur. Kogudes informatsiooni soosib see üha enam neid käike, mis on suurema tõenäosusega head, muutes otsingu asümmeetriliseks [14].

Suure hulga võimalike mänguolekutega mängude puhul nagu male ja Go võib ammendav otsing osutuda selle mahu tõttu problemaatiliseks. Monte Carlo puuotsingu toob küll välja üha tugevamaid käike, mida kauem see töötab, ent otsingu saab igal hetkel peatada. Kui minimax on sobilik algoritm lihtsate mängude lahendamiseks, siis Monte Carlo puuotsing on paremaks alternatiiviks keerulisemate mängude jaoks, hoolimata asjaolust, et see pakub ligikaudseid lahendusi absoluutsete asemel [14].

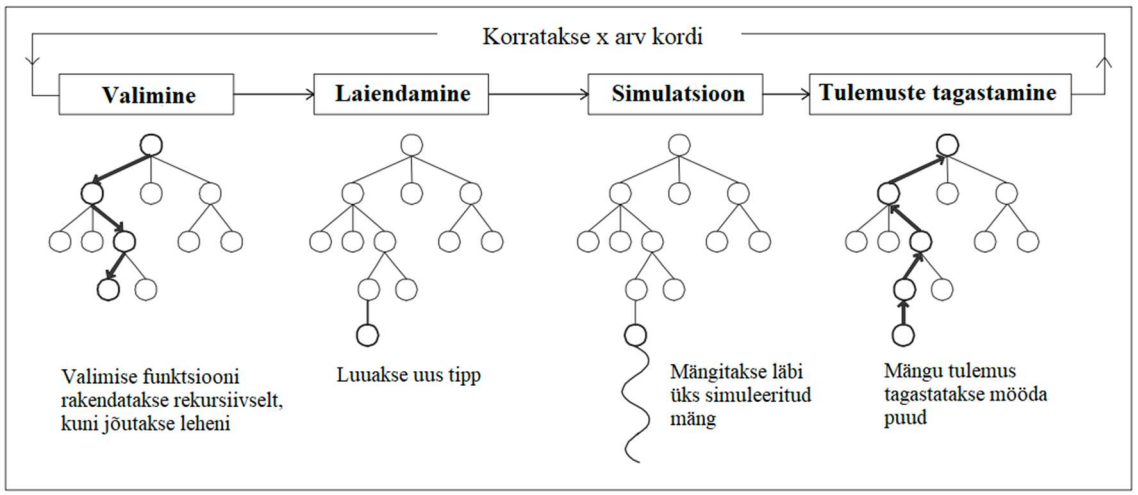
Monte Carlo puuotsingut on edukalt kasutatud paljude determineeritud täiusliku informatsiooniga mängude puhul. Ent samamoodi on Monte Carlo puuotsingu meetodeid kasutatud ka mängude jaoks, milles on varjatud informatsioon ja määramatuse aspekt [19]. Determineeritud katse korral on katse tulemus katsetingimustega üheselt määratud, juhusliku katse puhul võib katse tulemus olla samade katsetingimuste korral erinev [20]. Mängus 2048 esineb juhuslikkuse element, kuna pärast iga käiku lisatavate klotside väärtus ja asukoht mängulaual on juhuslik.

Monte Carlo puuotsingu algoritm, peaasjalikult just koos ülemise usalduspiiri (*Upper Confidence Bounds*, *UCB1*) valemiga, andis paljude mängude lahendamise tehisintellektide puhul suuri edasiminekuid [4]. Neller [9] tõi enda uurimuses välja, et Monte Carlo simulatsioone kasutades saadi mängu 2048 mängimisel keskmiseks klotsi väärtuseks 1024 ja suurimaks 2048, seejuures tehti katsetes iga lubatava käigu kohta 1000 Monte Carlo simulatsiooni piiratud sügavusega 3. Yarasca ja Nguyen [21] leidsid võrdlevas uurimises, et võrreldes expectimaxiga saadi Monte Carlo puuotsingut kasutades mängus 2048 paremaid tulemusi, ent viimane oli siiski ajakulukam.

3.2.2 Monte Carlo puuotsingu etapid

Monte Carlo puuotsing on efektiivne meetod suurte puude korral, simuleerides korduvalt paljude - ent mitte kõigi - teede läbimist mängupuus. Mida rohkem erinevaid teid läbitakse hetkeseisust kuni mängu lõppseisuni, seda paremini suudab algoritm hinnata,

millised teed on head. Selleks, et säilitada simulatsioonide käigus kogutud informatsiooni, ehitab Monte Carlo puuotsing simulatsioonide käigus enda eraldiseisva otsingupuu. Pärast hulga simulatsioonide läbiviimist saab Monte Carlo puuotsing piisavalt informatsiooni, et tagastada hea käik antud mänguseisust [14]. Joonisel 6 on välja toodud Monte Carlo puuotsingu protseduuri kirjeldus.



Joonis 6. Monte Carlo puuotsingu etapid [14].

Valiku faasis (*selection*) alustatakse juurtipust ja valitakse olemasoleva info alusel järjestikuseid järglastippe, kuni jõutakse leheni ehk veel laiendamata tipuni [14]. Tippude valimisel arvestatakse nii võitude määra (ehk antud tipust võidetud mängude arvu jagatist kõigi antud tipu läbimise kordadega) kui ka püüdu vahepeal läbida neid tippe, mis on jäänud kõrvale [18]. Selleks, et alguses vähem edukad tipud täiesti kõrvale ei jääks, kasutatakse UCB1 valemit:

$$\frac{\omega_i}{s_i} + c\sqrt{\frac{\ln s_p}{s_i}}$$

milles

ω_i = antud tipu võitude arv

s_i = antud tipu simulatsioonide arv

s_p = vanemtipu simulatsioonide arv

c = uurimise parameeter, milleks üldjuhul valitakse $\sqrt{2}$ [14].

Laiendamise (*expansion*) faasis laiendatakse otsingupuud, lisades sellele ühe uue tipu. Simulatsiooni (*simulation*) faasis tehakse läbi üks mängusimulatsioon kuni mängu lõpuni,

et selgitada välja võitja. Tulemuste tagastamise (*backpropagation*) faasis uuendatakse kõigi läbitud teel olnud tippude skoore kuni juurtipuni välja, sõltuvalt simuleeritud mängu tulemusest [14]. Skoori uuendamisel suurendatakse tipu külastamise arvu ühe võrra ning võidu korral ka võitude arvu ühe võrra [18].

Seda neljafaasilist algoritmi jooksutatakse korduvalt läbi, kuni on saadud piisavalt infot, et tagastada hea käik antud mänguseisust. Monte Carlo otsingupuu on oma struktuurilt identne mängupuu ühe alamosaga, ent Monte Carlo otsingupuu sisaldab täiendavalt simulatsioonide käigus saadud statistilist informatsiooni [14].

4 Monte Carlo puuotsingu rakendamine mängu 2048 mängimiseks

Järgnevalt on välja toodud lõputöö käigus saavutatud tulemused ja nendeni jõudmise protsess lähtuvalt töö alguses püstitatud uurimiseesmärgist. Kood antud lõputöö jaoks on kirjutatud Java programmeerimiskeeles, kasutades Eclipse IDEt. Java programmeerimiskeel osutus valituks lähtudes autori eelistustest.

4.1 Mängu programmi loomine

Mängu 2048 võib esitada 4x4 maatriksina. Alg- ja lõppolekute kirjeldusi saab mõlemaid olla mängus 2048 rohkem kui üks. Algolekuks on mängu esialgne seis, kus mängulaual paiknevad mistahes kohtadel kaks klotsi väärtusega 2 ja 2, 2 ja 4 või 4 ja 4. Mängu kaotusega lõppevateks lõppolekuteks on mänguseisud, mille puhul ei ole jõutud klotsini väärtusega 2048 ning mängija ei saa teha ühtegi lubatud käiku. Kaotusega lõppeva lõppoleku puhul on täidetud kolm tingimust:

1. mänguväljal ei ole ühtegi tühja ruutu;
2. mitte üheski reas ega veerus ei paikne kõrvuti kahte sama väärtusega klotsi;
3. mänguväljal ei ole ühegi klotsi väärtus >= 2048.

Mängu võiduga lõppenud lõppolekuteks loetakse mängulaua seisu, kus on jõutud klotsini väärtusega 2048. Kuna 2048 klotsini jõudmine võimaldab mängu valiidse käigu olemasolul edasi mängida, on kokkuleppeliselt võimalik mängu eesmärki muuta. Peale suurima klotsi väärtuse võib eesmärgi püstitamisel lähtuda ka mängu skoorist. Käesolevas töös seatakse võidu kriteeriumiks klotsini väärtusega 2048 jõudmine.

Klotsid, mis tekivad liitumise teel antud mängukäigu ajal, ei saa sama käigu ajal uuesti liituda. Näiteks liigutades paremale klotsirida väärtustega [2][2][2][2], on rea väärtus antud käigu järel [0][0][4][4] (mitte [0][0][0][8]). Antud tingimus välistas mängukoodi kirjutamisel kõrvutiolevate klotside võrdsuse kontrolli tsükli kujul, kuna sel juhul ei olnud klotsid, mis olid tekkinud klotside liitumisel antud mängukorra ajal, eristatavad neist, mis

olid mängulaual enne käigu tegemist. Alternatiivselt oleks olnud võimalik antud mängukäigu ajal liitumise teel tekkinud klotside indeksid salvestada ning enne igat klotside ühendamist teha vastav kontroll. Rohkem kui kahe sama väärtusega klotsi ühendamist alustatakse valitud suuna poolt. Näiteks klotside [0][2][2][2] paremale liigutamisel saadakse tulemuseks [0][0][2][4] (mitte [0][0][4][2]).

Pärast iga lubatud käiku ehk käiku, mille järel on toimunud mängulaual muudatus (vähemalt ühe klotsi asukoha muutus või vähemalt kahe sama väärtusega klotsi ühendamine), lisatakse mänguväljale suvalisse tühja lahtrisse uus klots väärtusega 2 või 4 (vt joonis 7). Kui kasutaja valib klotside liigutamiseks suuna, mis mängulaual muudatust ei too, uut klotsi ei lisata (käik ei olnud lubatud).

Game.java klassis:

```
void addTile(int[][] board) {
	ArrayList<Integer> column = new ArrayList<Integer>();
	ArrayList<Integer> row = new ArrayList<Integer>();
	for (int j = 0; j < 4; j++)
		for (int i = 0; i < 4; i++)
			//Tühja lahtri koordinaadid salvestatakse:
			if (board[j][i] == 0) {
				column.add(i);
				row.add(j);
			}
	//Valitakse juhuslik indeks:
	int rnd = (int) (Math.random() * column.size());
	//Indeksi alusel leitakse tühja lahtri y- ja x-koordinaat:
	int y = row.get(rnd);
	int x = column.get(rnd);
	//Tõenäosusega 0,9 lisatakse tühja lahtrisse 2, tõenäosusega 0,1 4:
	board[y][x] = Math.random() < 0.9 ? 2 : 4;
}
```

Joonis 7. Uue 2 või 4 väärtusega klotsi lisamine juhuslikku tühja lahtrisse pärast iga mängukäiku.

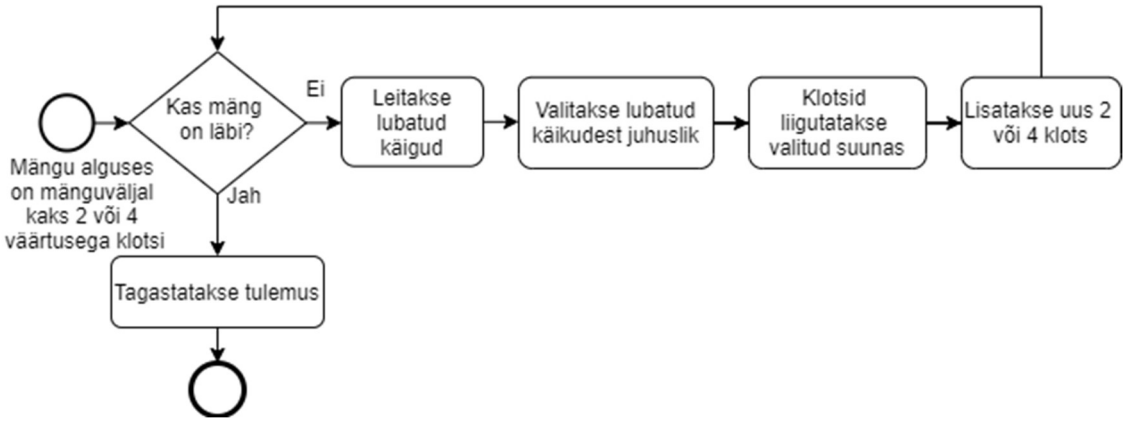
Meetod *legalMoves* (vt joonis 8) kontrollib, kas käigu tegemisel mingis suunas toimub muutus mänguväljal ning tagastab lubatavad käigud meetodi sisendiks olevast mänguseisust. Meetodit *legalMoves* kasutab ka meetod *gameOver*, mis kontrollib pärast iga uue klotsi lisamist, kas mängijal on võimalik teha vähemalt üks lubatud käik. Kui ühtegi käiguvõimalust ei ole, on mäng läbi. Samuti on antud töö jaoks koostatud mängukoodis mäng läbi juhul, kui on jõutud klotsini väärtusega 2048.

Game.java klassis:

```
ArrayList<Integer> legalMoves(int[][] board) {
	ArrayList<Integer> legalMoves = new ArrayList<Integer>();
	//Muutujasse initialBoard salvestatakse sisendiks olev mängulaua seis:
	int[][] initialBoard = new int[4][4];
	for (int j = 0; j < 4; j++)
		for (int i = 0; i < 4; i++)
			initialBoard[j][i] = board[j][i];
			//Käik 1 ehk „üles“ lisatakse lubatava käiguna, kui selle
			//tegemisel toimub muutus mängulaual (meetod moveUp
			//tagastab null, kui klotside liigutamisel üles muutust
			//mänguväljal ei toimunud, ning uue mängulaua seisu, kui
			//muutus toimus):
			if (moveUp(board) != null) {
				legalMoves.add(1);
				//Taastatakse sisendiks olnud mängulaua seis
				//järgmise suuna kontrollimiseks:
				for (int j = 0; j < 4; j++)
					for (int i = 0; i < 4; i++)
						board[j][i] = initialBoard[j][i];
			}
			//Kontrollitakse ka ülejäänud kolme suunda sama loogika
			//alusel
			/---/
		}
	//Tagastatakse kõik lubatud käigud kontrollitavast mänguseisust:
	return legalMoves;
}
```

Joonis 8. Lubatavate käikude leidmine sisendparameetriks olevast mänguseisust.

Nii testimise kui andmete analüüsimise ja võrdlemise eesmärgil lisati meetod, mille puhul arvuti mängib mängu valides antud seisust tehtavate lubatud käikude seast käike juhuslikkuse alusel. See võimaldas hiljem näha, kas kasutatavad meetodid parandavad saadavaid tulemusi ning kui suur on nende mõju tulemustele. Juhusliku mängu protsessidiagramm on toodud välja joonisel 9.



Joonis 9. Juhusliku mängu protsessidiagramm.

Mängimisel juhuslikke käike valiv meetod *chooseRandomMove* (vt joonis 10) kontrollib esmalt, et mäng ei oleks läbi, leiab seejärel kõik antud mänguseisust tehtavad lubatud käigud ning valib neist juhuslikkuse alusel tehtava käigu. Pärast käigu tegemist lisatakse mänguväljale juhuslikku tühja lahtrisse klots väärtusega 2 või 4 ning tehakse uus juhuslik käik senikaua, kuni mäng on võidetud või ühtegi käiku pole enam võimalik teha.

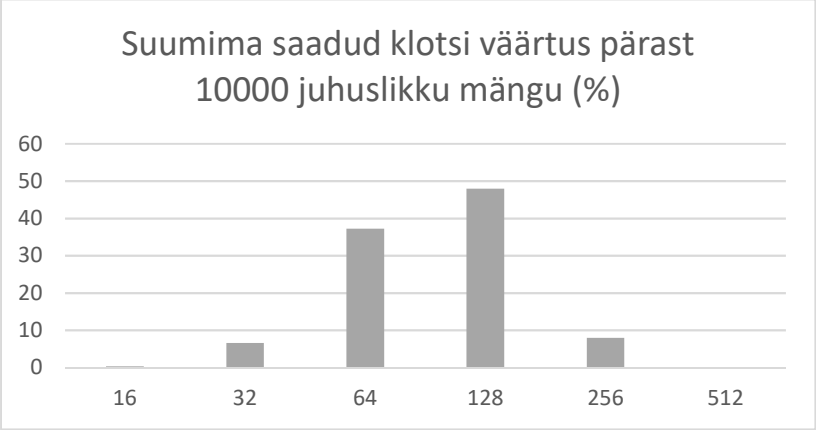
Game.java klassis:

```
void chooseRandomMove() {
	//Kontrollitakse, et mäng ei ole läbi:
	if (gameOver(board) == null) {
		//Leitakse lubatud käigud antud mänguseisust:
		ArrayList<Integer> legalMoves = legalMoves(board);
		//Lubatud käikude massiivist tagastatakse juhuslik indeks:
		int rnd = (int) (Math.random() * legalMoves.size());
		//Tagastatakse juhuslikule indeksile vastav käik:
		int move = legalMoves.get(rnd);
		//Sooritatakse valitud käik:
		board = makeMove(board, move);
		//Lisatakse juhuslikku tühja lahtrisse klots 2 või 4:
		addTile(board);
		chooseRandomMove();
	}
}
```

Joonis 10. Juhuslike käikudega mängimine.

Juhuslikke mänge mängitakse Main.java klassis ette antud arv kordi ning iga mängu lõpuks suurima mänguväljal oleva klotsi väärtus salvestatakse ArrayList-tüüpi muutujasse, milles olevad väärtused prinditakse mängude mängimise lõppedes välja koos esinemise sagedustega.

Mängides läbi 10000 juhuslikku mängu, saadi suurimaks klotsi väärtuseks kõige sagedamini 128 (48% mängudest). Järgnesid 64 (37,2% mängudest) ja 256 (7,9% mängudest). Kõige madalamaks klotsi väärtuseks jäi 16 ning kõige suuremaks 512. 10000-st mängust vaid kahe puhul jõuti klotsini 512. Protsentuaalne jaotus on toodud välja joonisel 11.



Joonis 11. Suurima mängu lõpuks saadud klotsi väärtuste jaotus pärast 10000 juhuslikku mängu.

4.2 Nurga strateegia kasutamine

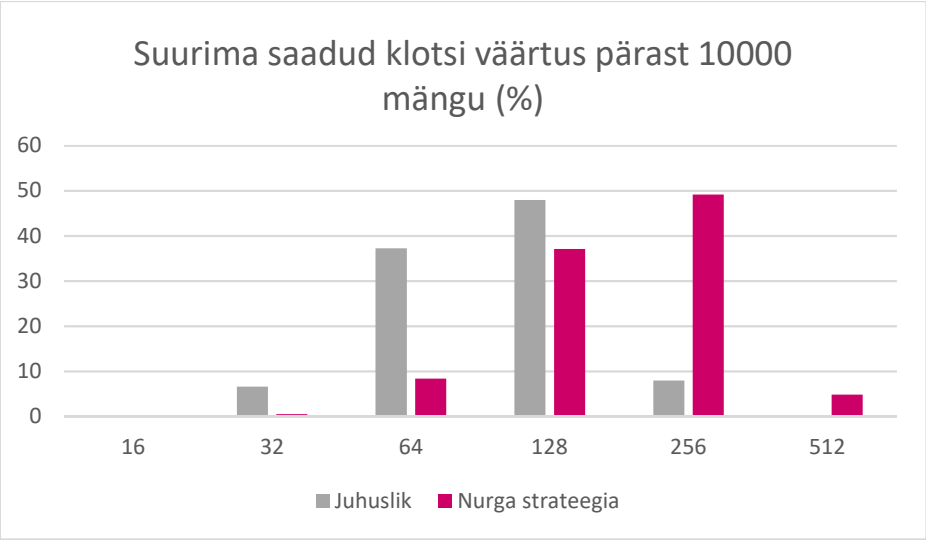
Teooria osas on põgusalt välja toodud üks levinumaid mängu 2048 mängimise strateegiaid, milleks on suuremate klotside kogumine ühte nurka, kasutades võimalusel vaid kahte käigusuunda. Antud strateegiat kasutav kood on välja toodud joonisel 12, kus eelistatud suundadeks on käigud üles (1) ja paremale (4). Klotse liigutatakse alla (2) vaid juhul, kui klotside liigutamine üheski muus suunas pole võimalik.

Game.java klassis:

```
void chooseMoveCorner() {
	//Kontrollitakse, et mäng ei ole läbi:
	if (gameOver(board) == null) {
		//Leitakse lubatud käigud antud mänguseisust:
		ArrayList<Integer> legalMoves = legalMoves(board);
		int move;
		if (legalMoves.contains(1) && legalMoves.contains(4)) {
			if (Math.random() < 0.5) {
				move = 1; //Käik üles
			} else {
				move = 4; //Käik paremale
			}
		} else if (legalMoves.contains(1)) {
			move = 1; //Käik üles
		} else if (legalMoves.contains(4)) {
			move = 4; //Käik paremale
		} else if (legalMoves.contains(3)) {
			move = 3; //Käik vasakule
		} else {
			move = 2; //Käik alla
		}
	//Sooritatakse valitud käik:
	board = makeMove(board, move);
	//Lisatakse juhuslikku tühja lahtrisse klots 2 või 4:
	addTile(board);
	chooseMoveCorner();
	}
}
```

Joonis 12. Nurga strateegia kasutamine käikude valimisel.

Nurga strateegiat kasutades saadi selgelt paremaid tulemusi kui täiesti juhuslikke käike tehes. Suurimaks saadud klotsi väärtuseks jäi endiselt 512. Protsentuaalselt on 10000 simulatsiooni tulemusena mängu lõpuks saadud suurima klotsi väärtused toodud välja joonisel 13.

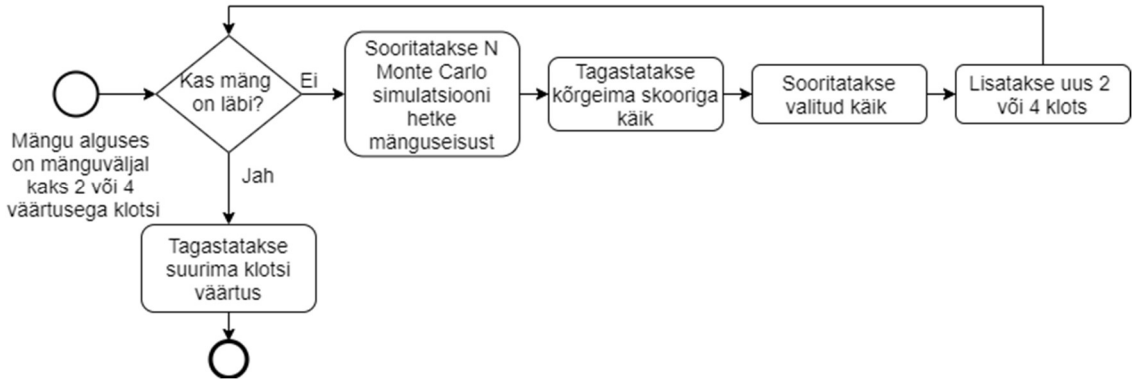


Joonis 13. Suurima saadud klotsi väärtus protsentides pärast 10000 juhuslikku ja 10000 nurga strateegiat kasutanud mängu.

4.3 Monte Carlo puuotsingu kasutamine

Monte Carlo puuotsingu kasutamiseks lisati kaks klassi: MCTS.java, mis koondab endas Monte Carlo puuotsingu meetodeid, ning Node.java, mis sisaldab Monte Carlo otsingupuu tipu objekti parameetreid ja meetodeid. Monte Carlo puuotsingu koodi kirjutamisel on juhendava materjalina kasutatud Michael Liu (2018) artiklit *„Implementing Monte Carlo Tree Search in Node.js“* [22], milles Monte Carlo puuotsingut rakendati mängule *Connect Four*.

Monte Carlo puuotsingu kasutamisel mängu 2048 mängimiseks kasutab programm kahte mänguvälja. Üheks on mänguväli, mis sisaldab parasjagu mängitava 2048 mängu hetkeseisu. Teine mänguväli on kasutusel Monte Carlo simulatsiooni jaoks, millel mängitakse läbi juhuslikku mängu simulatsiooni alustades käimasoleva mängu hetkeseisust. Pärast etteantud arvu Monte Carlo simulatsioonide tegemist valitakse statistiliselt kõige paremate tulemusteni viinud mängulaua hetkeseisust tehtud esimene käik. Valitud käik sooritatakse mängitavas mängus. Enne järgmise käigu tegemist viiakse uuest mänguseisust läbi etteantud arv Monte Carlo simulatsioone, et leida kõrgeima potentsiaaliga järgmine käik. Eeltoodud protsess on toodud välja joonisel 14.



Joonis 14. Mängu 2048 mängimine kasutades käikude valimisel Monte Carlo puuotsingut.

Enne iga mängukäigu tegemist kutsutakse Main.java klassis välja MCTS.java klassi meetod *runSearch* (vt joonis 15), mille sisenditeks on mängulaua seis ja tehtavate Monte Carlo simulatsioonide arv, ning meetod *bestMove* (vt joonis 16), mis tagastab simulatsioonide tulemusena saadud parima tehtava käigu. Meetodi *runSearch* eesmärgiks on statistika loomine, mille põhjal hiljem parim käik valitakse.

MCTS.java klassis:

```
void runSearch(int[][] board, int mcSimulations) throws Exception {
	//Lisatakse Monte Carlo puu juurtipuks hetke mänguseis:
	this.makeNode(board);
	int completedSimulations = 0;
	while (completedSimulations < mcSimulations) {
		//Käivitatakse valikufunktsioon:
		Node node = this.select(board);
		Integer gameOver = this.game.gameOver(node.board);
		Integer score = this.game.score(node.board);
		//Kontrollitakse, et tegu pole lehega ning et mäng ei ole läbi:
		if (node.isLeaf() == false && gameOver == null) {
			//Laiendatakse antud tippu:
			node = this.expand(node);
			//Lisatud tipust alates mängitakse mäng lõpuni:
			score = this.simulate(node);
		}
	//Tagastatakse simuleeritud mängu tulemus üles mööda Monte Carlo puud:
	this.backpropagate(node, score);
	//Suurendatakse tehtud Monte Carlo simulatsioonide arvu ühe võrra:
	completedSimulations += 1;
	}
}
```

Joonis 15. Monte Carlo simulatsioonide käivitamine.

Meetod *runSearch* loob esmalt antud mänguseisust Monte Carlo otsingupuu juurtipu, kasutades selleks *makeNode* meetodit. Muuhulgas lisab *makeNode* meetod juurtipu parameetri *unexpandedMoves* ehk tipust tehtavate ja veel laiendamata käikude väärtusteks kõik antud mänguseisust tehtavad legaalsed käigud, kasutades selleks jällegi Game.java klassi *legalMoves* meetodit. Seejärel sooritatakse Monte Carlo simulatsioone *runSearch* meetodi sisendina etteantud arv kordi.

Esimeses, valimise etapis (joonis 16) liigutakse mööda otsingupuud, alustades juurtipust, valides sellest lubatava käigu ning liikudes järglastipuni, milleni antud käik viib. Käike tehakse seni, kuni jõutakse sellise järglastipuni, mis on kas leht või mida pole täielikult laiendatud. Käikude valimisel kasutatakse ülemise usalduspiiri (*UCB1*) valemit, valides igal korral kõrgeima *UCB1* väärtusega järglase.

MCTS.java klassis:

```
Node select(int[][] board) throws Exception {
	Node node = this.nodes.get(board);
	//Monte Carlo puus liigutakse kõrgeima UCB1 väärtusega järglaseid
	//valides seni, kuni jõutakse tipuni, mis on kas täielikult
	//laiendamata või leht:
	while (node.isFullyExpanded() && !node.isLeaf()) {
		//Leitakse kõik antud tipust sooritatavad lubatud käigud:
		ArrayList<Integer> moves = node.allMoves();
		int bestMove = 0;
		double bestUCB1 = 0;
		//Leitakse järglaste UCB1 väärtused:
		for (int move : moves) {
			double childUCB1 =
				node.childNode(move).getUCB1(this.UCB1Param);
			//Leitakse kõrgeima UCB1 väärtusega järglane:
			if (childUCB1 > bestUCB1) {
				bestMove = move;
				bestUCB1 = childUCB1;
			}
		}
		node = node.childNode(bestMove);
	}
	return node;
}
```

Joonis 16. Monte Carlo valikufunktsioon.

Valikufunktsioon tagastab (täielikult) laiendamata tipu, millele luuakse MCTS.java klassi laiendamise (*expand*) meetodiga otsingupuusse juurde üks juhuslikult valitud võimalik järglastipp (vt joonis 17).

MCTS.java klassis:

```
Node expand(Node node) throws Exception {
	//Leitakse antud tipu laiendamata järglasteni viivad käigud:
	ArrayList<Integer> moves = node.unexpandedMoves();
	//Tagastatakse käikude massiivist juhuslikult valitud indeks:
	int index = (int) Math.floor(Math.random() * moves.size());
	//Leitakse indeksile vastav käik:
	int move = moves.get(index);
	//Leitakse mängulaua seis pärast valitud käigu tegemist:
	int[][] childState = this.game.chooseMove(node.board, move);
	//Leitakse saadud mängulaua seisust tehtavad lubatud käigud:
	ArrayList<Integer> childUnexpandedMoves =
		this.game.legalMoves(childState);
	//Uuest mänguseisust luuakse uus tipp:
	Node childNode = node.expand(move, childState, childUnexpandedMoves);
	//Loodud tipp lisatakse Monte Carlo puule:
	this.nodes.put(childState, childNode);
	//Tagastatakse loodud Monte Carlo puu tipp:
	return childNode;
}
```

Joonis 17. Monte Carlo laiendamise funktsioon.

Lisatud tipust alates mängitakse mäng simulatsiooni etapis lõpuni, tehes juhuslikult valitud käike. Simulatsiooni etapi kood on toodud välja joonisel 18. Kuna programmi testimisel ei jõudnud mäng ühelgi korral võiduni ehk klotsini väärtusega 2048, ei kasutata programmis mängu tulemuse hindamisel võit-kaotus skaalat. Selle asemel määrati skoorid suurima mängulaual oleva klotsi järgi mängu lõppedes, mille puhul alates klotsist 128 (k.a) on skooriks suurima klotsi väärtuse jagu punkte; alla selle on skooriks 0. Selline hindamise skaala saadi katsetamiste tulemusena.

MCTS.java klassis:

```
int simulate(Node node) {
	int[][] board = node.board;
	Integer gameOver = this.game.gameOver(board);
	//Mängitakse kuni simuleeritud mängu lõppemiseni:
	while (gameOver == null) {
		ArrayList<Integer> moves = this.game.legalMoves(board);
		int move = moves.get((int) Math.floor(Math.random() *
			moves.size()));
		//Tehakse juhuslik lubatud käik:
		board = this.game.chooseMove(board, move);
		gameOver = this.game.gameOver(board);
	}
	int score = this.game.score(board);
	//Tagastatakse simuleeritud mängu lõpuskoor:
	return score;
}
```

Joonis 18. Monte Carlo simulatsiooni funktsioon.

Monte Carlo simulatsiooni viimases etapis uuendatakse kõigi läbitud tippude statistikat kuni juurtipuni välja (vt joonis 19). Tipust tehtud simulatsioonide arvu suurendatakse ühe võrra ning skoori vastavalt suurima klotsi väärtusest tulenevale skoorile.

MCTS.java klassis:

```
void backpropagate(Node node, Integer score) {
	//Tulemused tagastatakse kuni juurtipuni välja:
	while (node != null) {
		//Antud tipust tehtud simulatsioonide arvu suurendatakse ühe
		//võrra:
		node.plays += 1;
		//Kui saavutati klots väärtusega vähemalt 128 suurendatakse
		//antud tipu skoori suurima saadud klotsi väärtuse võrra:
		if (score != null)
			node.totalScore += score;
		//Liigutakse antud tipu vanema juurde:
		node = node.parent;
	}
}
```

Joonis 19. Monte Carlo tulemuse tagastamise funktsioon.

Määratud skoorid mõjutavad järgmise Monte Carlo simulatsiooni valiku faasi. Monte Carlo simulatsioone tehakse *runSearch* meetodi sisendina etteantud arv kordi.

Simulatsioonide järel leitakse meetodiga *bestMove* käik, mis viis parimate tulemusteni (joonis 20, valitakse *policy „max“*).

MCTS.java klassis:

```
int bestMove(int[][] board, String policy) throws Exception {
	//Kui kõiki lubatud käike ei katsetatud simulatsioonide käigus,
	//antakse veateade:
	if (this.nodes.get(board).isFullyExpanded() == false) {
		throw new Exception("Not enough information!");
	}
	Node node = this.nodes.get(board);
	ArrayList<Integer> allMoves = node.allMoves();
	int bestMove = 0;
	//Kõrgeima tulemusega (tipu skoor jagatud sellest tipust mängitud
	//mängude arvuga) käigu leidmine:
	if (policy == "max") {
		double max = 0;
		for (int move : allMoves) {
			Node childNode = node.childNode(move);
			double ratio = childNode.totalScore / childNode.plays;
			if (ratio > max) {
				bestMove = move;
				max = ratio;
			}
		}
	}
	//Nurga strateegia rakendamine:
	else if (policy == "corner") {
		double max = 0;
		for (int move : allMoves) {
			Node childNode = node.childNode(move);
			double ratio = childNode.totalScore / childNode.plays;
			if (move == 1) { //Käik üles
				ratio = ratio * 1.2;
			} else if (move == 4) { //Käik paremale
				ratio = ratio * 1.2;
			} else if (move == 3) { //Käik vasakule
				ratio = ratio * 1.1;
			}
			if (ratio > max) {
				bestMove = move;
				max = ratio;
			}
		}
	}
	//Tagastatakse parimate tulemusteni viinud käik:
	return bestMove;
}
```

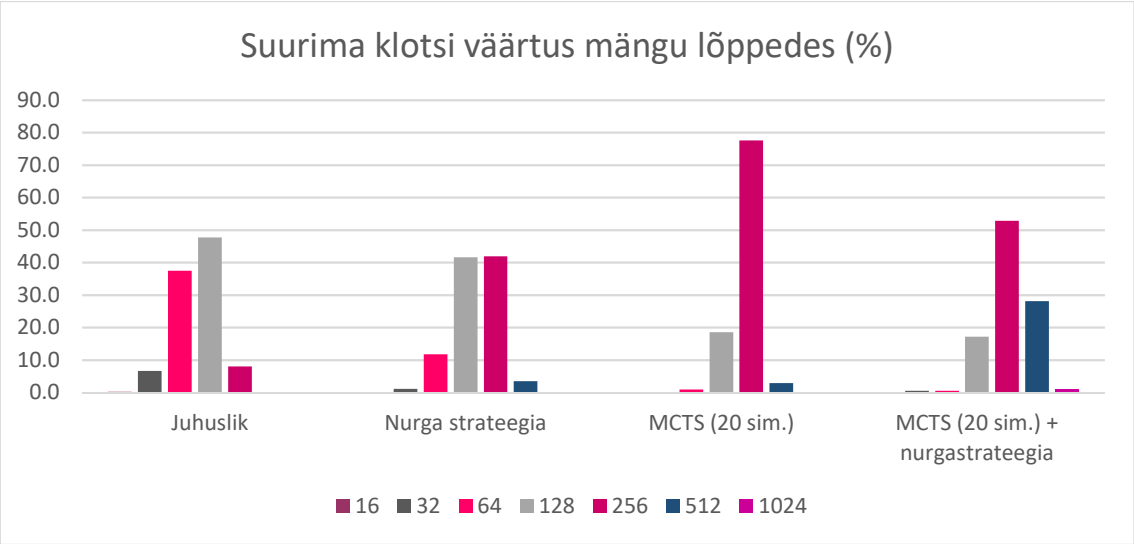
Joonis 20. Monte Carlo simulatsioonide alusel mängu hetkeseisust tehtava parima käigu tagastamine.

4.4 Monte Carlo puuotsingu kombineerimine nurga strateegiaga

Otsingu efektiivsemaks muutmiseks on vähemalt kaks teed: (1) kasutada paremat algoritmi ja (2) kasutada suunavat lisainfot; seejuures võib kasutada ühte, teist või mõlemat korraga. Heuristiline otsing ehk kogemusel ja arvamustel põhinev probleemilahendus, mida kasutatakse otsingu tõhustamiseks, võib oluliselt vähendada otsinguoperatsioonide arvu, leides mitte tingimata parima, ent üldjuhul siiski hea lahendi [13]. Heuristilise otsingu korral püütakse avatud tippe järjestada nii, et otsinguprotsess jätkuks perspektiivikamates suundades [1].

Otsingu efektiivsemaks muutmiseks kombineeriti järgnevalt omavahel Monte Carlo puuotsing ning nurga strateegia. Selle jaoks täiendati MCTS.java klassi meetodit *bestMove*, mis leiab Monte Carlo simulatsioonide järgselt parima mängukäigu antud mänguseisust. Kui varasemalt valiti käik kõrgeima skoori ja simulatsioonide arvu jagatise alusel (*policy „max“*), siis nüüd korrutati saadud arv omakorda läbi kordajaga, mis andis eelise käikudele üles ja paremale (kordaja 1,2), aga ka vasakule (kordaja 1,1) (joonis 20, valitakse *policy „corner“*). Kordajate väärtused saadi katsetuste tulemusena. Liialt kõrgete kordajate määramine (näiteks vastavalt 2 ja 1,5) andis halvemaid tulemusi kui Monte Carlo puuotsing ilma nurga strateegiat kasutamata.

Joonisel 21 on toodud kõigi nelja eelpool välja toodud meetodi kasutamisel saadud tulemuste võrdlus. Monte Carlo puuotsingu kasutamisel oli valitud Monte Carlo simulatsioonide arvuks 20. Kombineerides Monte Carlo puuotsingut nurga strateegiaga saavutati paremaid tulemusi kui ainult Monte Carlo puuotsinguga või ainult nurga strateegiat kasutades. Suurimaks saavutatud klotsi väärtuseks oli 1024.



Joonis 21. Mängu lõppemisel saadud suurima klotsi väärtus protsentides kõigist valitud meetodit kasutanud mängudest.

4.4.1 Monte Carlo simulatsioonide arvu mõju tulemustele

Välja toodud graafikute puhul oli Monte Carlo simulatsioonide arvuks määratud 20. Selline simulatsioonide arv valiti ühe mängu jaoks kulunud ajast ja saadud tulemustest lähtuvalt, püüdes leida nende vahel optimaalset tasakaalu. Simulatsioonide arvu suurendamisel tulemused mingil määral küll paranesid, kuid oluliselt muutus ka üheks mänguks kulunud aeg. Simulatsioonide arvu mõju tulemustele ehk suurimale mängu lõpuks saadud klotsi väärtusele ning mängule kuluvale keskmisele ajale on välja toodud tabelis 2.

Tabel 2. Monte Carlo simulatsioonide arvu mõju tulemustele ning kulunud ajale.

	32	**64**	**128**	**256**	**512**	**1024**	**Ühe mängu keskmine aeg**
MCTS (20 simulatsiooni)	0,0%	1,0%	18,6%	77,6%	2,9%	0,0%	6,8 sek.
MCTS (40 simulatsiooni)	0,0%	0,0%	12,7%	80,9%	6,4%	0,0%	15,2 sek.
MCTS (20 simulatsiooni) koos nurga strateegiaga	0,5%	0,5%	17,1%	52,9%	28,1%	1,0%	10,5 sek.

Tabelist 2 on näha, et kahekordistades Monte Carlo simulatsioonide arvu enam kui kahekordistus keskmine üheks mänguks kulunud aeg, ent muutus tulemustes pole kuigi suur. Jättes ühe mängukäigu kohta tehtavate simulatsioonide arvuks 20 ning lisades

Monte Carlo puuotsingule nurga strateegia pikenes küll keskmine üheks mänguks kulunud aeg umbes poole võrra, kuid mõju tulemustele on seejuures märkimisväärne. Siinkohal tasub ära märkida, et üldjuhul tähendab parema tulemuseni jõudmine ka suuremat mängu jooksul tehtud käikude arvu, mistõttu võib oletada, et osa mänguks kulunud aja pikenemisest võib kirjutada rohkemate tehtud käikude arvele.

Vaatamata sellele, et Monte Carlo puuotsing läbib tippe valikuliselt, on tegu siiski ajakuluka meetodiga. Kui 10 000 juhusliku mängu mängimiseks kulus kokku aega 27 sekundit, siis ligikaudu sama palju aega kulus kõigest nelja Monte Carlo puuotsingut kasutava mängu mängimiseks 20 simulatsiooniga käigu kohta.

5 Kokkuvõte

Käesoleva lõputöö eesmärgiks oli luua mängu 2048 mängiv tehisintellekti programm, mis kasutab käikude valimisel Monte Carlo puuotsingut ning hinnata saadud tulemusi. Töö teoreetilises osas toodi välja mängu 2048 reeglid ning minimaalne tehtav käikude arv, mis on vajalik mängu võitmiseks. Viimane võimaldas anda hinnangu mängupuu suurusele, mis oli oluliseks teguriks just Monte Carlo puuotsingu algoritmi valimisel.

Mängu 2048 puhul on tegu mänguga, milles sisaldub juhuslikkuse element ning mille võimalike mänguseisude hulk on väga suur. Seetõttu ei ole ajaliselt optimaalne kõigi mingist seisust võimalike lauaseisude hindamine ja läbimängimine. Monte Carlo puuotsingu meetod, mis kasutab juhuslikke simuleeringuid, sobib hästi suure hargnemisteguriga mängude jaoks. Meetod ei pruugi anda parimat tulemust, kuid võimaldab saada optimaalse aja jooksul küllaltki häid tulemusi. Monte Carlo simulatsioonide arvu suurendamisel tulemused paranevad, ent suureneb ka ühe mängu mängimiseks kuluv aeg. Keeruliste mängude puhul tasub kaaluda Monte Carlo puuotsingule heuristiliste meetodite lisamist. Antud töös kombineeriti Monte Carlo puuotsingut nurga strateegiaga.

Kasutades vaid Monte Carlo puuotsingut 20 simulatsiooniga jäi parimaks saadud klotsi väärtuseks 512 ning enim jõuti klotsini väärtusega 256. Paremate tulemuste saamiseks lisati Monte Carlo puuotsingu meetodile nurga strateegia. Nurga strateegia on üheks populaarseimaks mängu 2048 mängimise strateegiaks, milles suurema väärtusega klotse püütakse hoida ühes valitud nurgas. Lisades selle küllaltki lihtsa strateegia Monte Carlo puuotsingule saadi märkimisväärselt paremaid tulemusi ning jõuti ka klotsini väärtusega 1024.

Töö olulisima järeldusena võibki tuua välja, et Monte Carlo puuotsingu tulemuste parandamiseks on võimaluse korral tasuv kombineerida puuotsingu algoritmi heuristiliste teadmistega. Monte Carlo simulatsioonide arvu suurendamine parandab tulemusi, kuid suurendab oluliselt otsinguks kuluvat aega. Heuristilise meetodi lisamisel võib aga märkimisväärselt paremaid tulemusi saada ajas oluliselt kaotamata.

Kasutatud kirjandus

[1] M. Koit ja T. Roosmaa, Tehisintellekt, Tartu: Tartu Ülikooli Kirjastus, 2011.

[2] P. Sweetser ja J. Wiles, „Current AI in games: a review," *Australian Journal of Intelligent Information Processing Systems,* kd. 8, nr 1, lk. 24-42, 2002.

[3] L. Võhandu, „Koduküberneetikutele," *Horisont,* lk. 43, 12 1974.

[4] D. Robilliard, C. Fonlupt ja F. Teytaud, „Monte-Carlo Tree Search for the Game of '7 Wonders'," *European Conference in Artificial Intelligence (ECAI)*, Praha, 2014.

[5] B. Goel, „Mathematical Analysis of 2048, The Game," *Advances in Applied Mathematical Analysis,* kd. 12, nr 1, lk. 1-7, 2017.

[6] S. Langerman ja Y. Uno, „Threes!, Fives, 1024!, and 2048 are Hard," *Theoretical Computer Science,* kd. 748, lk. 17-27, 2018.

[7] H. J. Herik, „Five New Games," *ICGA Journal,* kd. 37, nr 3, lk. 129-130, 2014.

[8] K.-H. Yeh, C.-C. Liang , K.-C. Wu ja I.-C. Wu, „2048-bot Tournament in Taiwan," *ICGA Journal,* kd. 37, nr 3, lk. 186-187, 2014.

[9] T. W. Neller, „Pedagogical Possibilities for the 2048 Puzzle Game," *The Journal of Computing Sciences in Colleges,* kd. 30, nr 3, lk. 38-46, 2015.

[10] D. Eppstein, „Making Change in 2048," *9th International Conference on Fun with Algorithms*, La Maddalena, 2018.

[11] „What is the Largest Tile Possible in 2048," [Võrgumaterjal]. Kättesaadav: https://puzzling.stackexchange.com/questions/48/what-is-the-largest-tile-possible-in-2048. [Kasutatud 05 05 2021].

[12] J. Lees-Miller, „The Mathematics of 2048: Minimum Moves to Win with Markov Chains," 05 08 2017. [Võrgumaterjal]. Kättesaadav: https://jdlm.info/articles/2017/08/05/markov-chain-2048.html. [Kasutatud 05 05 2021].

[13] J. Tepandi, „Intelligentsed süsteemid. Pragmaatiline käsitlus. Versioon 10.01.2021," Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Tarkvarateaduse instituut, Tallinn, 2021.

[14] M. Liu, „General Game-Playing With Monte Carlo Tree Search: A primer on the game-playing algorithm behind AlphaGo," 11 10 2017. [Võrgumaterjal]. Kättesaadav: https://medium.com/@quasimik/monte-carlo-tree-search-applied-to-letterpress-34f41c86e238. [Kasutatud 05 05 2021].

[15] S. Sharma, „Monte Carlo Tree Search: MCTS For Every Data Science Enthusiast," 01 08 2018. [Võrgumaterjal]. Kättesaadav: https://towardsdatascience.com/monte-carlo-tree-search-158a917a8baa. [Kasutatud 05 05 2021].

[16] „What is the Optimal Algorithm for the Game 2048?," [Võrgumaterjal]. Kättesaadav: https://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048/22498940. [Kasutatud 05 05 2021].

[17] G. Chaslot, S. Bakkes, I. Szita ja P. Spronck, „Monte-Carlo Tree Search: A New Framework for Game AI," *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, Stanford, California, 2008.

[18] „Monte Carlo Tree Search for Tic-Tac-Toe Game in Java," 3 10 2020. [Võrgumaterjal]. Kättesaadav: https://www.baeldung.com/java-monte-carlo-tree-search. [Kasutatud 05 05 2021].

[19] P. I. Cowling, E. J. Powley ja D. Whitehouse, „Information Set Monte Carlo Tree Search," *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES,* kd. 4, nr 2, lk. 120-143, 2012.

[20] T. Kollo, Monte Carlo meetodid, Tartu: Tartu Ülikooli Kirjastus, 2004.

[21] E. N. Yarasca ja K. Nguyen, „Comparison of Expectimax and Monte Carlo algorithms in solving the online 2048 game," *PESQUIMAT,* kd. 21, nr 1, lk. 1-10, 2018.

[22] M. Liu, „Implementing Monte Carlo Tree Search in Node.js," 12 07 2018. [Võrgumaterjal]. Kättesaadav: https://medium.com/@quasimik/implementing-monte-carlo-tree-search-in-node-js-5f07595104df. [Kasutatud 05 05 2021].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Laura Katariina Teder

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Mängu 2048 lahendamine Monte Carlo puuotsinguga“, mille juhendaja on Peeter Ellervee
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

17.05.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.