Rīgas 64. vidusskola

**Mākslīgā intelekta teorētiskais risinājums**

**attiecībā pret iespējamo realizāciju**

Zinātniski pētnieciskais darbs datorzinātnēs

**Darba autori**:

Rīgas 64. vidusskolas 12. klases skolēni

Kristiāns Magons, Daniels Muļukins

**Darba vadītājs**:

Vadošais arhitekts, SAP SE

Krists Magons

**Darba konsultants**:

Programmēšanas skolotājs, Rīgas 64. vidusskola

Eduards Bukovskis

Rīga, 2025

**Ievads**

Pēdējo gadu laikā ir strauji attīstījušās datortehnoloģijas [1], kas līdzās sekmējušas arī mākslīgā intelekta (turpmāk tekstā – MI) attīstību, jo radās iespējas efektīvāk glabāt daudz datus mākoņos [2, pp. 27-28] [3], un jaudīgās videokartes [4] ļāva ātrāk veikt skaitļošanu un apstrādāt vairāk datus. Šis sniedza iespējas realizēt tādas tehnoloģijas kā TensorFlow, kas vienlaikus paralēli izmanto vairākus procesorus, kas būtiski paātrina matemātiskus aprēķinus [5]. Šādas skaitļošanas platformas brīvi pieejamas, pateicoties daudzkodolu videokaršu attīstībai. Pagaidām MI nav stingri definēts, jo atšķiras izpratnes par to. Vēsturiski ir nonākts pie četrām pamata definīcijām, kā izprast MI jēdzienu – domāšana vai rīkošanās cilvēciski, domāšana vai rīkošanās racionāli [2, p. 2].

MI rīki cilvēkiem jau ir kļuvuši par būtisku palīgu ikdienā, taču prognozes par tā tālāko nākotni ir dažādas. Parasti nepilnības rodas, kad spriedumus mēģina veikt, nevienojoties par MI definīciju, jo, kā jau tika minēts, MI sfēra ir ļoti plaša ar dažādām definīcijām. Sākot no šaha spēles risinājumiem līdz robotam, kas domā kā cilvēks. Tāpēc mēs realizēsim savu racionālo aģentu, lai saprastu MI limitācijas.

**Pētījuma mērķis**:

Izveidot mākslīgā intelekta modeli kā racionāli rīkojošu aģentu, lai izpētītu problēmjautājumu pamatus un gūtu praktisku priekšstatu par mākslīgā intelekta metožu, šajā tvērumā klasiskās meklēšanas uzdevumu limitācijām.

**Darba uzdevumi**:

1. Valodā C++ realizēt zema līmeņa datu struktūras un pamatalgoritmus (saistītais saraksts, prioritāšu rinda, steks, grafs), kā arī iepazīties ar teksta failu apstrādi;
2. Izmantojot realizētās datu struktūras un OOP (Objektorientētā programmēšana) paradigmas, realizēt aģenta un vides implementāciju un darbības simulāciju;
3. Realizēt aģenta stāvokļa pārejas funkciju, izmantojot A\* vai citus meklēšanas algoritmus;
4. Novērtēt aģenta veiktspēju atkarībā no vides iestatījumiem un noskaidrot aģenta potenciālās vajadzības pēc atmiņas daudzuma un izšķirtspējām;
5. Izvērtēt MI potenciālās iespējas nākotnē.

**Hipotēze**:

Jo sarežģītāka vide, jo aģentam, ar teorētiski iespējamu risinājumu, to realizēt kļūs arvien grūtāk līdz aģentam tas praktiski nemaz neizdosies ierobežoto resursu dēļ.

**Darbā izmantotās metodes**:

1. Simulācijas programmas implementācija.
2. Statiskā analīze veiktspējas novērtēšanai.
3. Noteikt algoritmu sarežģītību
4. Dinamiskā testēšana - simulācijas programma tiks vairākkārt palaista. Simulācijas izpildes laikā tiks mērīti iegūtie rezultāti.
5. Empīrisko rezultātu statistiskā analīze (T- tests).

**1.** **Ieskats mākslīgajā intelektā**

**1.1** **Kas ir mākslīgais intelekts?**

Cilvēki jau sen ir mēģinājuši saprast, – kā cilvēks spēj uztvert, saprast, paredzēt, un ietekmēt pasauli, kas ir daudz plašāka par viņu. MI sfēra tiecas vēl tālāk – tā nav tikai mēģinājums izprast, bet arī atveidot inteliģenci [2, p. 1]. Vadoties pēc Stjuarta Rasela un Pītera Norviga mācību grāmatas “Artificial Intelligence: A Modern Approach”, MI nav vienotas definīcijas. Izmantojot astoņas dažādas MI definīcijas [2, p. 2], autori tās ir apvienojuši četrās galvenās kategorijās:

**Cilvēciska rīcība**: 1950. gadā Alans Tjūrings izgudroja testu, definējot MI, kas ir plaši zināms, kā “Tjūringa tests”. Šis tests iekļauj pratinātāju, kurš aizklāti veic interviju. Ja pratinātājs nevar noteikt, vai atbildes sniedz dators vai cilvēks, tad dators ir nokārtojis testu. [2, pp. 2-3].

Tjūringa tests mūsdienās joprojām ir aktuāls, bet viedokļi par tā atbilstību MI definēšanā ir mainījušies. Tjūringa tests paredz, ka dators spēj simulēt cilvēcisku rīcību, bet ne gluži rīkoties cilvēciski [6]. Proti, mērķis ir, lai MI nonāk līdz konkrētajai rīcībai, to apdomājot tāpat, kā cilvēki, nevis imitējot cilvēciskumu. Pagaidām šis mērķis paliek īpaši grūts, jo tas, kā cilvēki domā un rīkojas, vadoties pēc savām domām, nav līdz galam izprasts [7].

**Cilvēciska domāšana**: Šī MI pieeja cieši saistās ar kognitīvo zinātni, jo, lai konstruētu cilvēciski domājošu MI moduli, jāzina precīzas un pārbaudāmas teorijas par cilvēka prātu. Tiekot pie pietiekami precīzas teorijas par prātu, ir iespējams to realizēt, kā datora programmu. Ja programmas ievade-izvade saskan ar cilvēka rīcību, tad tiek iegūts reāls pierādījums, ka šis programmas mehānisms varētu darboties arī cilvēkā. [2, p. 3]

**Racionāla domāšana**: Racionāla domāšana saista loģikas jomu un izskata jautājumus par pareizu un sistematizētu domāšanu. Šī MI pieeja balstās uz nenoliedzamiem spriešanas procesiem, kur pie pareiziem pieņēmumiem, veicot deducēšanu, tiek iegūti loģiski secinājumi. Piemērs: “Sokrāts ir cilvēks. Visi cilvēki ir mirstīgi. Tātad Sokrāts ir mirstīgs.” [2, p. 4]

**Racionāla rīcība**: Aģents ir kaut kas, kas rīkojas. Protams, visas datora programmas kaut ko dara, bet datora aģentam ir jāspēj vairāk – rīkoties autonomi, uztvert apkārtējo vidi sev apkārt, pastāvēt ilgstošu laiku, adaptēties izmaiņām, veidot un sasniegt mērķus. Racionāls aģents ir tāds, kas rīkojas, lai sasniegtu labāko rezultātu vai labāko sagaidāmo rezultātu. [2, pp. 4-5]

Šajā pētījumā MI pieeja būs putekļsūcēja modelis kā racionāli rīkojošs aģents. Tiks izstrādāts putekļsūcēja aģents, kurš pārvietosies pa vidi, kas sastāv no vairākām istabām, un iztīrīs visas istabas. Tas spēs uztvert, vai istabā, kurā tas atrodas, ir piesārņojums. Tad attiecīgi tas atkritumus sasūks vai pārvietosies uz nākamo istabu, ja piesārņojums netiks uztverts. Kā jau rīkojoties racionāli, putekļsūcēja mērķis ir visefektīvāk nokļūt no sākuma stāvokļa līdz rezultāta stāvoklim, proti, nonākt stāvoklī, kad visas istabas ir tīras.

**1.2** **Izcelsme**

MI ir viena no jaunākajām zinātnes un inženierijas nozarēm [2, p. 1]. Runājot par MI tēmu, vārdam “nozare” ir jāliek īpašs uzsvars, jo nereti MI tiek pārprasts par atsevišķu, patstāvīgu zinātni. MI saista vairākas zinātnes, piemēram, matemātika, psiholoģija, datortehnika un pat filozofija [2, pp. 5-16]. Ievērojamie rezultāti MI jomā gūti vien pēdējo gadu laikā (par MI uzplaukumu lasīt nodaļā 1.3 Uzplaukums), taču tas ir bijis garš process vairāku desmitu gadu garumā. Mēģinājumi veiksmīgi izveidot MI modeli ir bijuši gan ar lielām cerībām, gan vilšanām.

Pirmie atzītie darbi MI jomā datējami 1943. gadā, kad, izmantojot primitīvu psiholoģiju, apgalvojumu loģiku un Tjūringa skaitļošanas teoriju, tika izgudrots pirmais modelis, kas reprezentēja mākslīgus neironus. 1950. gadā, tika izveidots pirmais neironu tīkla dators. Lai gan tas simulēja vien 40 neironus, tas sastāvēja no 3000 vakuuma caurulēm un tehnoloģijām, kas tika aizgūtas no bumbvedēja B-24 automātiskā pilota mehānisma pārpalikumiem [2, pp. 16-17].

Ietekmīgs veikums 1956. gada vasarā bija Allena Nevela un Herberta Saimona izveidotā spriešanas programma “Loģikas teorētiķis”, kas spēja domāt neskaitliski [2, pp. 17-18]. Vēlāk viņi izveidoja programmu “Vispārējais problēmu risinātājs”, kas bija programma, spējīga atdarināt cilvēcisku domāšanu. Savukārt 1958. gads ir nozīmīgs ar Džona Makartija izveidoto teorētisko aprakstu programmai “Padomu ņēmējs”, kas hipotētiski spēja izmantot vispārējas pasaules zināšanas, lai meklētu problēmu risinājumus, un ko uzskatīja par pirmo pilnīgo MI sistēmu [2, pp. 18-20].

Kad tik salīdzinoši īsā laika posmā tika izveidotas mašīnas, kas spēja domāt, mācīties un veidot – cerības par tuvākajiem nākotnes plāniem bija spožas. Ārpus jaunu metožu mēģinājumiem [2, pp. 22-27], MI joma uz ilgāku laiku stagnēja, jo viss atdūrās pret datoru nespēju veikt grūtākus vai apjomīgākus uzdevumus [2, pp. 20-22].

**1.3** **Uzplaukums**

MI 60 gadu attīstības laikā akcents lielākoties ir bijis algoritmu izvēlei un atbilstībai, taču šī gadsimta laikā lielāka uzmanība tikta vērsta datiem, un to milzīgajai pieejamībai dažādos avotos, interneta tīmekļos. Ar plašo datu pieejamību, MI modulēšana atvieglojas, jo ar mazāk informāciju nepieciešami cilvēciski piemēri un sarežģīti algoritmi, turpretī ar vairāk informāciju var iztikt ar primitīvāku algoritmu. Šī iemesla dēļ pieeja MI veidošanai mainījusies no mehāniskas zināšanu iekodēšanas uz datora pašmācību pie dotiem datiem [2, pp. 27-28].

Lai apstrādātu datus un trenētu MI modeļus, ir nepieciešams veids, kā šos lielos daudzums datu efektīvi glabāt. Lai to realizētu, dati tiek glabāti mākoņos, un to apstrādei tiek izmantoti jaudīgi skaitļotāji vai jaunas pieejas, kā datus apstrādāt.

Jau vairākus gadus, izpildoties Mūra likumam [8], mums ir pieeja jaudīgiem daudzkodolu procesoriem, kuri var ātri un efektīvi veikt daudzus uzdevumus. No otras puses, milzīgu daudzumu datu apstrāde, pat tādiem procesoriem, aizņemtu pārāk daudz laika, lai apstrāde tiktu veikta efektīvi. Tādēļ cilvēki pievērsās grafiskajiem procesoriem jeb videokartēm [4], jo, atšķirībā no procesoriem, tām ir daudz vairāk atsevišķu kodolu, lai veiktu 3D kalkulācijas. Piemēram, grafiskais procesors “GeForce RTX 3070” satur 5888 kodolu, kuri paralēli veic savas funkcijas [9]. Šī grafisko procesoru īpatnība, veikt vienlaikus paralēlus darbus, padara tās par ļoti efektīvu rīku paralēlajā datu apstrādē [10]. Spilgtākais piemērs videokaršu pielietošanā, lai veidotu un apmācītu MI, ir CNN (konvolucionālie neironu tīkli). Tas ir neironu tīkls, kuru visbiežāk izmanto, lai uztvertu un apstrādātu bildes. Lai paātrinātu bilžu apstrādi, bildes un neironu tīklu pārveido, kā divus masīvus, bet apstrādes rezultāts ir šo masīvu reizinājums. Šo masīvu reizinājumu var veikt paralēli ar videokaršu palīdzību, kas būtiski samazina laiku masīvu sareizināšanai [4].

Sakarā ar lielo uzplaukumu MI jomā, lielie valodu modeļi (turpmāk tekstā – LVM) pēdējos gados ir kļuvuši par visātrāk popularizētām tehnoloģijām pasaulē. Piemēram, viens no LVM pārstāvjiem ChatGPT tiek iekļauts vispopulārāko vietņu sarakstā pasaulē [11]. LVM ir transformatoru modeļi – neironu arhitektūrā, – kuriem ir iedoti tik daudz dati, ka tie spēj modelēt vai atbildēt ar loģiskiem, cilvēciskiem tekstiem, kā arī saprast kontekstu [12] [13]. Šīs spējas bija iemesls LVM straujajai popularizācijai, jo LVM ir viegli pielietot un tie ir noderīgs rīks, lai ātri kaut ko uzzinātu, apkopotu garus tekstus vai pat izveidot jaunus tekstus. Protams, LVM nāk ar savām limitācijām, jo šīs tehnoloģijas tiek trenētas, balstoties uz datiem, kuri var gadīties nepatiesi, nepilnīgi vai nepietiekami daudz. Rezultātā LVM tehnoloģiju sniegtās atbildes var būt maldīgas [13].

Šo tehnoloģiju attīstības rezultātā, it īpaši pēdējos gados, ir visai strauji attīstījies MI, kuru ikdienā izmantojam, piemēram, rakstot *ChatGPT*, veidojot bildes ar *MidJourney* vai tulkojot tekstu ar *DeepL*.

**2. Algoritmu sarežģītība**

**2.1** **Kas ir lielā O notācija?**

Lielā O notācija datorzinātnēs ir veids, kā pierakstīt algoritma sarežģītību, proti, cik daudz algoritma soļu skaits vai izmantotā atmiņa pieaug, palielinoties ievaddatu skaitam. Lielo O notāciju pieraksta kā *O(n)*, kas cēlās no vācu matemātiķa “Paul Bachmann” [11] pieraksta. *O(n)* apraksta funkcijas uzvedību, kad arguments tiecas uz noteiktu vērtību vai bezgalību, kur O – nozīmēja augšanas kārtību, bet n – argumenta lielumu [12].

Tostarp datorzinātnēs n definē, nevis, kā argumenta lielumu, bet kā argumentu skaitu vai ievaddatu skaitu [13]. Piemēram, runājot par algoritmu ātrumu, *O(n)* nozīmēs, ka ar n ievaddatu skaitu algoritms izpildīs n soļus – algoritms izpildās lineāri, bet algoritms ar sarežģītību *O(1)* izpildīsies konstanti, veicot 1 soli. Šī algoritmu aprakstīšanas metode ļauj viegli un saprotami salīdzināt algoritmus, jo, pat nezinot neko par algoritmiem, varēsim secināt, ka algoritms ar sarežģītību *O(n)* būs ātrāks par algoritmu *O()*, jo n < .

Svarīgi ir piebilst, ka turpmāk aprakstot, salīdzinot vai pētot algoritmu sarežģītību, īsināsim to šādi:

*O(n + 1)* pārveidosim par *O(n)* tā, ka viens papildus solis algoritma izpildes laiku ietekmēs ļoti minimāli. Tātad īsināsim nelielus konstanta soļus vai atmiņas skaitu.

**2.2 Lielās O notācijas nozīme šajā pētījumā**

MI veidošana, apmācīšana un pētīšana ir saistīta ar lielām un sarežģītām datu kopām, un pareiza algoritma un datu struktūru izvēle var būtiski mainīt risinājumu ātrumu un precizitāti. Kā jau tika minēts 2.1 nodaļā, lielā O notācija ir ērts rīks, kas ļauj viegli salīdzināt algoritmus un novērtēt tā grūtību. Tas ir ļoti svarīgi šī projekta izpildījumā, jo ar lielo O notāciju tiks konkrēti aprakstīti pamatalgoritmu un datu struktūru ātrumi un prasības pēc atmiņas. Tas arī ļaus noskaidrot aģenta potenciālās vajadzības pēc atmiņas daudzuma un izšķirtspējām. Zinot šo informāciju, būs iespējams veikt pamatotus spriedumus par MI iespējām nākotnē.

Apskatot lielo O notāciju citos dzīves piemēros, kā šahā, vēl joprojām ar mūsdienu tehnoloģijām nav iespējams realizēt aģentu, kurš spētu izveidot un apstrādāt visu stāvokļu kopu šai spēlei. Tai ir vismaz stāvokļu, kas ir vairāk par atomu skaitu novērojamajā visumā [2, p. 47]. Vēlviens spilgts piemērs būtu RSA šifra, ko daudzas bankas izmanto, kā šifrēšanas veidu, kurā tiek reizināti divi milzīgi pirmskaitļi. RSA atšifrēšana aizņems pārāk ilgu laiku. Piemēram, 200 ciparu lielu reizinājuma atšifrēšana aizņems aptuveni 3,8 ⋅ gadus [17].

**3.** **A\* meklēšanas algoritms**

**3.1 Kas ir a\* meklēšanas algoritms?**

A\* ir informēts meklēšanas algoritms, kas strādā ar grafu, kurā katram ceļam ir zināma prioritāte [18]. Piemērs, kā a\* algoritms atrod īsāko ceļu (skatīt 1. attēlu):

Tā kā a\* algoritms zina attālumu no virsotnēm līdz galapunktam, lai sasniegtu mērķi ātrāk, a\* algoritms izvēlēsies virsotnes, kuras ir tuvākas galapunktam.



(1. attēls: grafa attēlojums, kā ceļi starp Latvijas pilsētām)

A\* algoritma ātruma sarežģītība ir eksponenciāla O(), kur b ir sazarojuma faktors – cik sazarojumu ir katrai virsotnei, bet d ir īsākais ceļa garums līdz galapunktam. Tostarp atmiņas sarežģītība ir visu apskatīto virsotņu skaits, kas sliktākajā gadījumā būs O(|V|), kur V ir virsotņu skaits [18].

**3.2** **Kur izmanto a\* meklēšanas algoritmu?**

A\* algoritmu izmanto dažādās jomās, visbiežāk GPS navigācijā, kur programmai var tikt iedots sākumpunkts un galapunkts, līdz kuram jānonāk pēc iespējas ātrāk. Navigācijas programma, ņemot vērā ceļa apstākļus un sastrēgumus, atgriež lietotājam optimālāko maršrutu. Šis algoritms arī saistās, piemēram, ar piegāžu ķēžu pārvaldības jomu, kur efektīva maršruta plānošana ir svarīga, lai ietaupītu laiku, benzīnu un resursus. Arī spēļu veidošanā var tikts izmantots a\* algoritms, piemēram, mašīnu sacīkstēs, lai nodrošinātu, ka citas mašīnas spēlē tiek līdzi [19].

**3.3** **A\* meklēšanas algoritma nozīme šajā pētījumā**

A\* meklēšanas algoritms, kā viens no efektīvākajiem klasiskās meklēšanas algoritmiem [2, pp. 95-99], šajā pētījumā tiks izmantots, lai gūtu priekšstatu par racionāli rīkojoša aģenta limitācijām.

Lai realizētu meklēšanu, būs nepieciešama vide un racionāli rīkojošs aģents, kas šajā pētījumā būs putekļsūcēja modelis, kas atradīsies vidē ar tīrām vai netīrām istabām, kur istabas viena ar otru ir savienotas. Lai reprezentētu šo vidi, tiks izmantots stāvokļu grafs un a\* meklēšanas algoritms, lai aģents varētu veikt meklēšanu un pārvietoties starp grafa virsotnēm.

**4.** **Aģenta apraksts**

MI pieeja šī projekta ietvaros ir putekļsūcēja modelis, kurš ir definēts, kā racionāli rīkojošs aģents. Tas nozīmē, ka tam piemīt spēja rīkoties jeb veikt kādu darbību, piemēram, sūkt istabu, ja tā ir piegružota, vai pāriet uz citu istabu, ja esošajā istabā gružu nav. Attiecīgi tam piemīt arī uztvere, ar kā palīdzību putekļsūcējs var noteikt, vai esošajā istabā ir vai nav gružu. Tostarp raksturīgākā īpašība šim aģentam – izvērtēt situāciju un rīkoties pēc iespējas efektīvāk, lai sasniegtu mērķa rezultātu. Proti, spēt dotā situācijā pieņemt pareizākos lēmumus, lai neveiktu liekas darbības un visas istabas tiktu iztīrītas ar pēc iespējas mazāk patērētiem resursiem.

Šajā projektā putekļsūcēja aģenta izveide ir realizēta, izmantojot objektorientēto programmēšanu (OOP) un C++ programmēšanas valodu. Aģentam esam, kā patstāvīgam klases datu tipam, ar pozīcijas un tīrīšanas atribūtiem, kā arī ar tīrīšanas un pāreju metodēm.

Aģenta implementācija GitHub [šeit](https://github.com/Galaxy2268/VacuumCleanerAgent/blob/main/Agent/VacuumCleaner.cpp), aģenta darbību starp istabām skatīt 2. attēlā