ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

თამარ კაპანაძე

ინტელექტუალური ავტომოპასუხის ერთი რეალიზაციის შესახებ

კომპიუტერული მეცნიერება

ნაშრომი შესრულებულია კომპიუტერული მეცნიერების მაგისტრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი: მიხეილ თუთბერიძე, ფიზ.-მათ. მეცნ. კანდიდატი

თბილისი

2017

ანოტაცია

წინამდებარე ნაშრომის ფარგლებში შემუშავებულია პროგრამული უზრუნველყოფა, რომელიც საშუალებას იძლევა, რომ შევქმნათ ინტელექტუალური აგენტი და ეს აგენტი შემდგომში შეძლებს უპასუხოს მომხმარებლის მიერ დასმულ კითხვებს. პროგრამული უზრუნველყოფა შემუშავებულია, როგორც ვებ აპლიკაცია და ამასთან, აპლიკაცია ორიენტირებულია მობილურ მოწყობილობებზე. პროგრამული უზრუნველყოფა შეიძლება გამოყენებულ იქნას სხვადასხვა დაწესებულებების მიერ მათი პროდუქტის უკეთ პოპულარიზაციის მიზნით, ფაქტობრივად აპლიკაციას შეუძლია იყოს ონლაინ კონსულტანტი ნებისმიერი ტიპის კომპანიისთვის.

Annotation

Tamar Kapanadze

On One Implementation of Intelligent Agent

In the scope of the present work the software is developed which allows to create the intelligent agent and answer the questions that the user will ask. Software is developed as web application and is oriented to mobile devices.

The software can be used by any kind of institutions to better promote their product. In fact the application can be an online consultant for any type of company.

სარჩევი

შესავალი 5

ინტელექტუალური ასისტენტის (ჩატბოტის) ისტორია 7

ტიურინგის ტესტი 7

პირველი ინტელექტუალური აგენტები 9

ინტელექტუალური ასისტენტის გამოყენების მაგალითები თანამედროვე სამყაროში .. 10

ამოცანის დასმა 12

ამოცანის გადაწყვეტის ვარიანტები 13

მოდელები 13

გენერირებადი მოდელები 13

ძიებაზე დაფუძნებული მოდელი 14

ამოცანის გადაწყვეტა 16

ბაზის არქიტექტურა 17

პროგრამული უზრუნველყოფის კოდი 18

მომხმარებლის კითხვის შექმნის ლოგიკა 19

ბოტის პასუხის შექმნის ლოგიკა 20

ორ წერტილს შორის კავშირის შექმნის ლოგიკა 21

წაშლის ლოგიკა 23

დიალოგების გრაფის აგების ლოგიკა 24

დიალოგების შენახვის ლოგიკა 26

ჩატბოტის პასუხის ძებნის ევრისტიკა 27

პირდაპირი პასუხის ძებნა 30

ფუძეებზე დაფუძნებული პასუხის ძებნა 30

დასკვნა..... 32

გამოყენებული ლიტერატურა..... 33

შესავალი

ინფორმაციული ტექნოლოგიების განვითარება სულ უფრო და უფრო მეტ შესაძლებლობებს უხსნის ადამიანებს ინოვაციური მიდგომების შემუშავებისათვის. ინფორმაციული ტექნოლოგიების განვითარებას ასევე წარმატებით უწყობს ფეხს კომპიუტერული მეცნიერების განვითარება. კომპიუტერულ მეცნიერებაში და კომპიუტერულ ინჟინერიაში მიღწეულმა წარმატებებმა შესაძლებელი გახადა კომპიუტერის მეშვეობით ისეთი ამოცანების განხორციელება, რასაც სულ რამდენიმე წლის წინ ძნელად თუ წარმოიდგენდა ვინმე.

კომპიუტერული მეცნიერების ერთერთი დარგი, კერძოდ ხელოვნური ინტელექტი (Artificial Intelligence, შემოკლებით AI), მიზნად ისახავს ისეთი მანქანების/პროგრამების შექმნას, რომლებსაც ექნებათ შესაძლებლობა, იყვნენ ისეთივე ჭკვიანი, როგორც ადამიანი და რაც მთავარია, მათაც ისევე შეეძლოთ განვითარება, როგორც ადამიანს, ჰქონდეთ უნარი მიაღწიონ ადამიანის ინტელექტის გაგებას.

შეგვიძლია ვთქვათ, რომ ეს დარგი 21-ე საუკუნეში კომპიუტერული მეცნიერების სხვა დარგებთან შედარებით ბევრად სწრაფად ვითარდება. ეს დიდწილად განპირობებულია იმით, რომ პროგრამისტებმა და ამ სფეროს გამოცდილმა მკვლევარებმა დაინახეს, რომ მისი გამოყენება ბევრ საინტერესო და სასარგებლო საქმეში შეიძლება. სწორედ ამიტომ, დიდი კომპანიები, როგორებიც არიან Facebook, Google, Amazon, Apple, ცდილობენ ამ დარგის სწრაფად განვითარებას და თავის საქმიანობაში მის გამოყენებას.

ხელოვნური ინტელექტი ბევრ ადამიანს ფილმებში ნანახი რობოტებად წარმოუდგენია, რობოტები, რომლებიც იქცევიან და ფიქრობენ ადამიანებივით, სინამდვილეში ამ სფეროს კონკრეტული ვიწრო ჭრილით განხილვა არ არის სწორი. ხელოვნური ინტელექტი არის უფრო მრავლისმომცველი და სასარგებლო, ვიდრე რობოტები, რომლებიც ისე მოქმედებენ, როგორც ადამიანები.

ხელოვნური ინტელექტის ერთ-ერთი ძირითადი მიმართულებაა ინტელექტუალური ასისტენტები (ჩატბოტები). ჩატბოტი არის კომპიუტერული

პროგრამამ რომელთანაც შეგვიძლია ვაწარმოოთ დიალოგი, როგორც მიმოწერის ასევე ვერბალურ დონეზე. ძირითადად ასეთი პროგრამების მთავარი მიზანი არის, რომ კომპიუტერული პროგრამა შეეცადოს მოიქცეს ისე, როგორც იმ კონკრეტულ შემთხვევაში ადამიანი მოიქცეოდა. ჩატბოტები ძირითადად გამოიყენება დიალოგის სისტემებში, მათი პრაქტიკული დანიშნულება ყველაზე ხშირად არის მომხმარებელთან ურთიერთობა და ინფორმაციის ვინმესთვის გაცნობა. ზოგიერთი რთული ჩატბოტი მუშაობს ენის დამუშავების ალგორითმის საშუალებით, მაგრამ ცოტა უფრო მარტივი ჩატბოტების მუშაობის პრინციპი სხვანაირია, ძირითადად მოიცავს მომხმარებლის მიერ შემოსული მონაცემებიდან ძირითადი სიტყვების ამოღების და მისი მონაცემთა ბაზაში პასუხებზე მორგების ალგორითმს.

კომპანიებს, რომლებიც მომხმარებლებს სთავაზობენ სერვისებს ან პროდუქტებს, აქვთ სპეციალური განყოფილება, რომელიც უზრუნველყოფს მომხმარებელთან ურთიერთობას. მომხმარებელთან ურთიერთობა ხდება როგორც ელფოსტის, ასევე ტელეფონის საშუალებით. ბოლო დროს აქტუალურია კიდევ ერთი საკომუნიკაციო არხი. ეს არის ონლაინ ჩატის საშუალებით დახმარება. ამ დროს კომპანიაში სპეციალურად გამოყოფილნი არიან თანამშრომლები, რომლებიც სხედან ჩატის მეორე მხარეს და პასუხობენ ნებისმიერ კითხვაზე, რაც შეეხება ამ კომპანიის სერვისებს თუ პროდუქტს.

წინამდებარე ნაშრომში განხილულია ინტელექტუალური აგენტის შექმნის ერთ-ერთი ვარიანტი. ამასთან, აპლიკაციის შექმნის პროცესში გათვალისწინებულია მსგავსი ტიპის აპლიკაციის შექმნის სამეცნიერო–ტექნოლოგიური მიდგომები. ნაშრომზე მუშაობის ფარგლებში შექმნილია შესაბამისი აპლიკაცია და იგი თან ერთვის ნაშრომს.

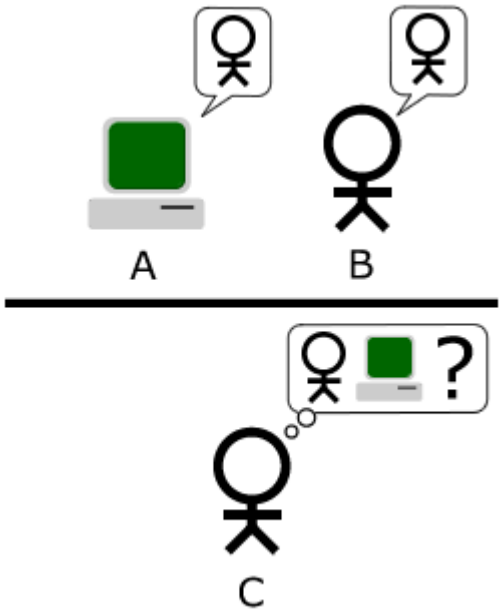
ინტელექტუალური ასისტენტის (ჩატბოტის) ისტორია

ტერმინი “ChatterBot”, ანუ ქართულ ენაზე - ჩატბოტი დაამკვიდრა Michael Mauldin-მა (პირველი ვერბალური ბოტის შექმნელმა) 1994 წელს. ეს ტერმინი შეიქმნა იმისთვის, რომ რაიმე სახელი დაერქმიათ დიალოგური პროგრამებისთვის. დღესდღეობით ჩატბოტი არის ვირტუალური ასისტენტის შემადგენელი ნაწილი და იგი ხშირად გამოიყენება ისეთ ცნობილ პროგრამებში, როგორებიცაა google assistant, facebook messenger და ა.შ.

1950 წელს გამოცემა MIND-ში გამოქვეყნდა ალან ტურინგის ცნობილი სტატია “კომპიუტერული მანქანები და ინტელექტი”, რომელშიც მთავარ კითხვად დასმული იყო, შეუძლიათ თუ არა მანქანებს ფიქრი.

ამ სტატიაში ტიურინგი ხსნის, რომ შეგვიძლია ე.წ. “ტიურინგის ტესტის” ან “იმიტაციის თამაშის” მიხედვით განვსაზღვროთ, შეუძლია თუ არა კომპიუტერს ფიქრი.

ტიურინგის ტესტი



ტესტისთვის საჭიროა გვყავდეს ორი ადამიანი (ერთი ვინც კითხვებს დასვამს A და ერთის მოპასუხე B) და ერთი ჭკვიანი მოწყობილობა C, რომელიც აგრეთვე იქნება მოპასუხე.

ტესტის მიმიდინარეობის პროცესში A იმყოფება B და C-სგან განსხვავებულ ოთახში.

A უსვამს გარკვეულ კითხვებს B და C-ს. ტესტის დასასრულს A-მ უნდა დააფიქსიროს თავის შეხედულება თუ B და C-ს შორის რომელია ადამიანი და რომელი ჭკვიანი მოწყობილობა.

თუ არმოჩნდა, რომ A-მ დაუშვა შეცდომა და ვერ გაარჩია ერთმანეთისგან მოწყობილობა და

ადამიანი, მაშინ ვიტყვით, რომ მიზანი მიღწეულია და ჭკვიან მოწყობილობას შეგვიძლია თავისუფლად ვუწოდოთ ხელოვნური ინტელექტი.

ტიურინგი თავისი ნაშრომით შეეწინააღმდეგა მანამდე არსებულ მოსაზრებას, რომ კომპიუტერულ მოწყობილობებს შეუძლიათ იაზროვნონ მხოლოდ და მხოლოდ მათში თავიდანვე ჩადებული ლოგიკით და რომ მათ არ შეუძლიათ თვითგანვითარება.

დღესდღეობით ჩატბოტების შემქნელი პირებისა თუ კომპანიებისთვის დიდ გამოწვევას წარმოადგენს ლოებნერის პრიზისთვის ბრძოლა.

ლოებნერის კონკურსი წარმოადგენს ტიურინგის ტესტის გამოყენების მაგალითს რეალურ ცხოვრებაში. ამ კონკურსზე ტიურინგის ტესტის მიხედვით ხდება იმის გამოვლენა, არის თუ არა კონკურსზე წარმოდგენილი რომელიმე ჩატბოტი ხელოვნული ინტელექტი.

პირველი ინტელექტუალური აგენტები

ჯოსეფ ვეიზენბაუმში დიდი ინტერესი გამოიწვია ტურინგის ზემოთხსენებულმა სტატიამ და სწორედ ამის საფუძველზე მან 1966 წელს შექმნა პროგრამა ელიზა, რომელიც იმ დროს ადამიანებს უქმნიდა წარმოდგენას, რომ მიმოწერის მეორე მხარეს ადამიანი იმყოფებოდა, სინამდვილეში ეს იყო კომპიუტერული პროგრამა, სადაც ადამიანს კომპიუტერი ესაუბრებოდა. ამ პროგრამის შემქმნელი ამბობდა, რომ ეს არ იყო ხელოვნური ინტელექტი, ვინაიდან მისი ალგორითმი იყო უფრო მარტივი და პრიმიტიული. პროგრამა შემოსული კითხვაში პოულობდა მნიშნველოვან სიტყვებს და შემდეგ უკვე დაწერილი პასუხები გამოქონდა ამ სიტყვებზე დაყრდნობით. მაგალითად თუ პროგრამა მომხმარებლის მიერ დასმულ კითხვაში იპოვიდა სიტყვა “დედა”-ს, მაშინ მას გამოჰქონდა პასუხი: “მომიყევი უფრო მეტი შენი ოჯახის შესახებ” და ა.შ.

მიუხედავად იმისა, რომ ეს პროგრამები არ ყოფილან დაფუძნებული ხელოვნურ ინტელექტზე, ანუ ამ პროგრამებს არ შეეძლოთ, რომ გაეანალიზებინათ შემოსული კითხვა და შემდეგ ენობრივი დამუშავების ალგორითმით შეემუშავებინათ კონკრეტული პასუხი, მომხმარებლების უმეტესობას მაინც სჯეროდა, რომ დიალოგის მეორე მხარეს იყო ადამიანი და არა კომპიუტერი. პროგრამისტებმა მალევე დაინახეს, რომ ამ პროგრამების გამოყენება ბევრ საინტერესო და სასარგებლო საქმეში შეიძლება.

ამ პროგრამების გამოყენების ძირითადი სფერო შეიძლება გამხდარიყო ის სფერო, სადაც შეგვეძლო გვეწინასწარმეტყველა ან განგვესაზღვრა, ძირითადად რა კითხვებს დასვამდა მომხმარებელი, რომ შემდეგ მასზე უკვე წინასწარ შეგვედგინა შესაძლო პასუხები და ეს ყველაფერი მონაცემთა ბაზაში შეგვეყვანა. ანუ ამ პროგრამების გამოყენება შესაძლებელი იქნებოდა კომპანიების მომხმარებლებთან ურთიერთობის განყოფილებაში, რომლებსაც მომხმარებლები ძირითადად ერთი და იმავე კითხვებით მიმართავენ პასუხის გასაგებად ამა თუ იმ პროდუქტსა თუ მომსახურებაზე. კითხვებს შესაძლოა ჰქონდეთ სხვადასხვა ფორმულირება, მაგრამ ძირითადი სიტყვები ყველა კითხვაში განმეორდება.

ინტელექტუალური ასისტენტის გამოყენების მაგალითები თანამედროვე სამყაროში

ჩატბოტს დღესდღეობით მრავალი კომპანია იყენებს მომხმარებებლებთან ურთიერთობის სფეროში და არამარტო იქ. ჩატბოტების შექმნის პლატფორმა სულ ცოტა ხნის წინ დაემატა facebook messenger-ში. ეს საშუალებას იძლევა, რომ მარტივად, ზედმეტი ტექნიკური ცოდნის გარეშე, ფეისბუქის გვერდს მიაბა ჩატბოტი, რომელსაც გაუწერ უკვე წინასწარ კითხვა-პასუხზე მონაცემებს და შემდეგ მსურველს საშუალება ექნება, ელაპარაკოს შენს ჩატბოტს.

როგორც ზემოთ ავღნიშნეთ, ჩატბოტების აქტიურად გამოყენება შეიძლება ასევე ისეთივე დიდი კომპანიების საქმიანობაში, როგორიცაა მაგალითად amazon, ebay და სხვ. დიდი კომპანიის ვებგვერდზე ჩატბოტის ინტეგრაციის თვალსაჩინოდ წარმოსადგენად მოვიყვანოთ რამდენიმე მაგალითი:

<u>კომპანია Zappos-ი</u> ონლაინ მაღაზიაა, რომელიც ყიდის ფეხსაცმელებს. მომხმარებელი, რომელსაც სურს ფეხსაცმლის შეძენა, ჩვეულებრივ შედის მათ საიტზე, პოულობს მისთვის მისაღებ ფეხსაცმელს და შემდგომ ყიდულობს მას.

თუ კომპანია Zappos-ი გადაწყვეტს შექმნას ჩატბოტი, მომხმარებელს, მისთვის სასურველი ფეხსაცმლის შესაძენად, ნაცვლად ზემოთ განსაზღვრული პროცედურისა, საშუალება ექნება უბრალოდ მიწეროს ფეისბუქზე Zappos-ს, თუ როგორი ფეხსაცმელი სურს, ჩატბოტი იმწუთასვე უპასუხებს კლიენტს, თავად შეურჩევს ალტერნატივებს და ჩატშივე უზრუნველყოფს საფასურის გადახდის პროცედურას.

ამგვარად, ნაცვლად ვებ-საიტზე ძებნისა, ჩატბოტის მეშვეობით შესაძლებელი იქნება Zappos-ის პროდუქციის შეძენა ისე, რომ საერთოდ არ ეწვიოთ საიტს, არ დახარჯოთ ზედმეტი დრო და მხოლოდ ბოტთან კომუნიკაციის შედეგად შეიძინოთ თქვენითვის მისაღები პროდუქტი, რომლის მუშაობის პრინციპი ზუსტად ისეთივეა, როგორც გამყიდველის, რომელიც ფეხსაცმლის მაღაზიაში გეხმარებათ სასურველი პროდუქტის არჩევაში.

კომპანია Domino, რომელიც კვების სფეროში არის ერთ-ერთი წარმატებული პიცების კომპანია. შესაძლებელია, რომ ოპერატორთან დარეკვის და მასთან საუბრის გარეშე თქვენ მარტივად შეძლოთ პიცის შეკვეთა ჩატბოტთან საუბრის შემდეგ. ჩატის საშუალებით შეგიძლიათ შეარჩიოთ თქვენთვის სასურველი პიცა, თქვენთვის შესაბამისი ინგრედიენტებით და შემდეგ ამ ჩათის ფანჯრიდან გაუსვლელად შეძლოთ საფასურის გადახდაც. ჩატბოტი გკითხავთ მისამართს, თუ სად უნდა მოვიდეს თქვენს მიერ შეკვეთილი პიცა და შემდეგ კურიერი სახლში მოგართმევთ მას.

ამინდის ბოტი – შეგიძლიათ მიწეროთ ჩატბოტს ფეისბუქ ჩატში და გაიგოთ, რა ამინდია თქვენთვის საინტერესო ადგილებში.

ახალი ამბების ბოტი – იგი შეგატყობინებთ ნებისმიერი სიახლის შესახებ, რაც თქვენ გაინტერესებთ.

ფინანსების მმართველი ბოტი – იგი დაგეხმარებათ, უფრო ეფექტურად მართოთ თქვენი ფინანსები.

ამოცანის დასმა

როგორც შესავალში უკვე ვთქვით, წინამდებარე კვლევის მიზანს წარმოადგენს ის, რომ შემუშავდეს რიგი ღონისძიებებისა, რათა შეიქმნას პლატფორმა, რომლის საშუალებითაც შესაძლებელი იქნება, ტექნიკური ცოდნის გარეშე შეიქმნას ქართულევანოვანი ჩატბოთი, რომლის საშუალებითაც კომპანიას შეეძლება ჩაანაცვლოს თანამშრომელი, რომელიც უზრუნველყოფს ჩატის საშუალებით მომხმარებელთან ურთიერთობას.

ამ ყველაფერთან დაკავშირებით წინამდებარე ნაშრომში განხილულია ჩატბოტების იდეა და შექმნის ისტორია, მათი ძირითადი მიზანი, პლატფორმის დეტალური განხილვა, რომელზეც ეს პროექტი უნდა შეიქმნას და გამოყენებული ინსტრუმენტების მიმოხილვა.

ამოცანის გადაწყვეტის ვარიანტები

მოდელები

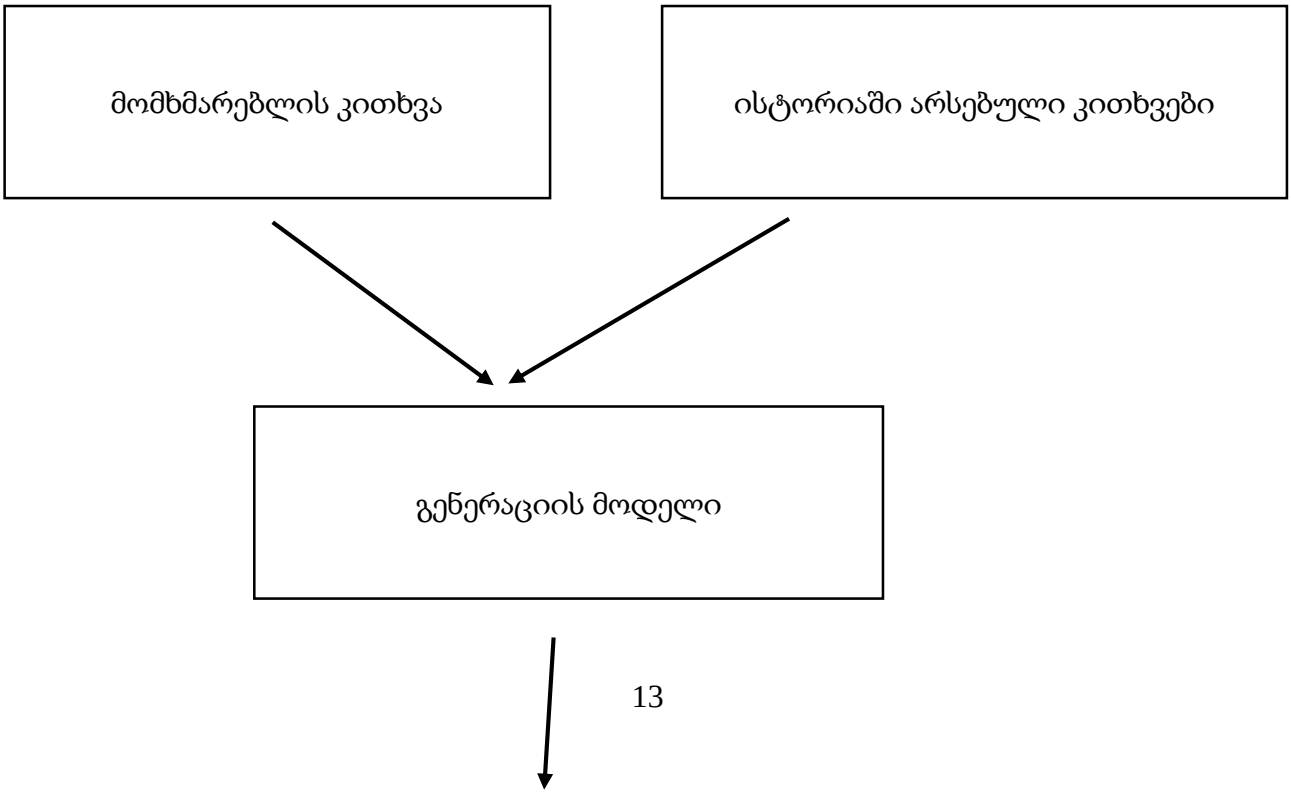
როგორც დენი ბრითმა დაწერა მის სტატიაში - „ინტელექტუალური აგენტების ღრმა სწავლა“ (“Deep learning for chatbots“), ჩატბოტს შეუძლია მკითხველისთვის გასაცემად პასუხი დააგენერიროს მანქანური სწავლების მოდელზე დაყრდნობით, ან გარკვეული ევრისტიკის გამოყენებით აარჩიოს პასუხი არსებული პასუხების ბაზიდან.

გენერირების მოდელი არის გაცილებით რთული, რადგან ის მოითხოვს მილონობით მაგალითს, რომ შეიქმნას დამსწავლელი ჩატბოტი, რომელთან საუბრის შედეგადაც მივიღებთ ხარისხიან დიალოგს და მიუხედავად ჩატარებული ტესტებისა, ჩვენ საბოლოოდ მაინც ვერ ვიქნებით დარწმუნებული, რომ ეს არის ზუსტი მოდელი.

გენერირებადი მოდელები

გენერირებადი მოდელები რეალურად წარმოადგენს ჩატბოტების მომავალს, რადგან სწორედ ამ გზით შექმნილი ჩატბოტები არიან უფრო ჭკვიანები და სწორედ მათ აქვთ თვითგანვითარების უნარი. ძირითადად მსგავსი ტიპის ჩატბოტებს ქმნიან ლაბორატორიებში.

ამგვარი ტიპის ჩეტბოტის შექმნის მოდელი გამოიყურება შემდეგნაირად.



ამ მიდგომის განსახორციელებლად საჭიროა იმ ენის კარგი ანალიზი, რომელ ენაზეც უნდა ისაუბროს ჩატბოტმა.

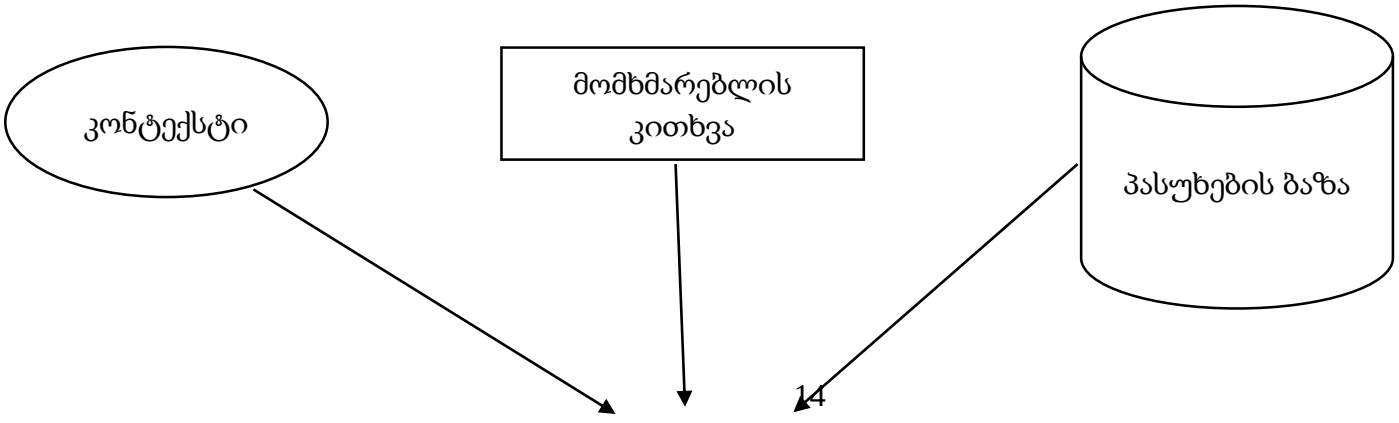
ძიებაზე დაფუძნებული მოდელი

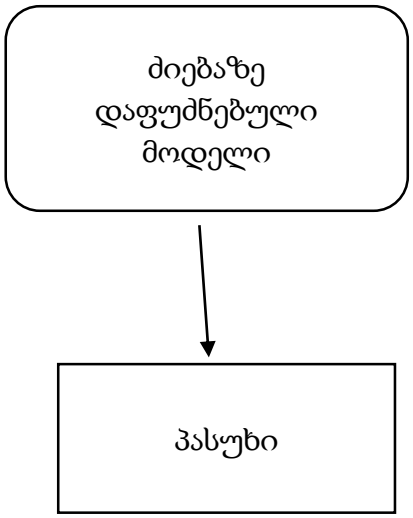
ამ მოდელის ტიპის ჩატბოტების აგება გაცილებით მარტივია, ვიდრე ზემოთ განხილული მოდელის მიხედით. აგრეთვე ამ მოდელით აგებული ჩატბოტის უპირატესობად ითვლება, რომ მათ შეუძლიათ უფრო ზუსტი პასუხების გაცემა. შეიძლება აქაც ვერ მივიღოთ 100%-ანი სიზუსტე, თუმცა გარანტირებულად ვიცით, რა შესაძლო პასუხებიდან გენერირდება შედეგი და რომ გაცემული პასუხი არ იქნება შეუფერებელი და იქნება გრამატიკულად გამართული.

სწორედ ეს მოდელია დღესდღეობით ყველაზე პრაქტიკული და ხშირად გამოყენებადი.

არსებობს ბევრი ალგორითმი და აპი (API), რომელთა გამოყენებითაც დეველოპერები ქმნიან ჩატბოტებს.

ძიებაზე დაფუძნებული მოდელი დიაგრამულად ასე გამოიყურება.





ჩატბოტი იყენებს მომხმარებლის კითხვას და კონტექსტს, რათა უკვე არსებული შესაძლო პასუხების სიიდან აარჩიოს ყველაზე შესაფერისი პასუხი. კონტექსტი შეიძლება ითვალისწინებდეს დიალოგის ხეს, რაც გულისხმობს, რომ კონკრეტული პასუხი შეიძლება გაეცეს კითხვას, თუ ეს კითხვა დიალოგების ხეში იკავებს კონკრეტულ ადგილს.

თუ ბოტი არ იყენებს კონტექსტს, მაშინ კითხვებზე პასუხების გაცემა უკვე ხდება დიალოგის ხის გამორიცხვით და ყოველ კითხვას გაეცემა შესაფერისი პასუხი.

ამოცანის გადაწყვეტა

პროგრამული უზრუნველყოფა შესრულდა C# პლატფორმის გამოყენებით, ხოლო ინფორმაციის სტრუქტურიზებულად შესანახად გამოყენებულია რელაციური მონაცემთა ბაზა, კერძოდ - Microsoft SQL Server-ის მონაცემთა ბაზა.

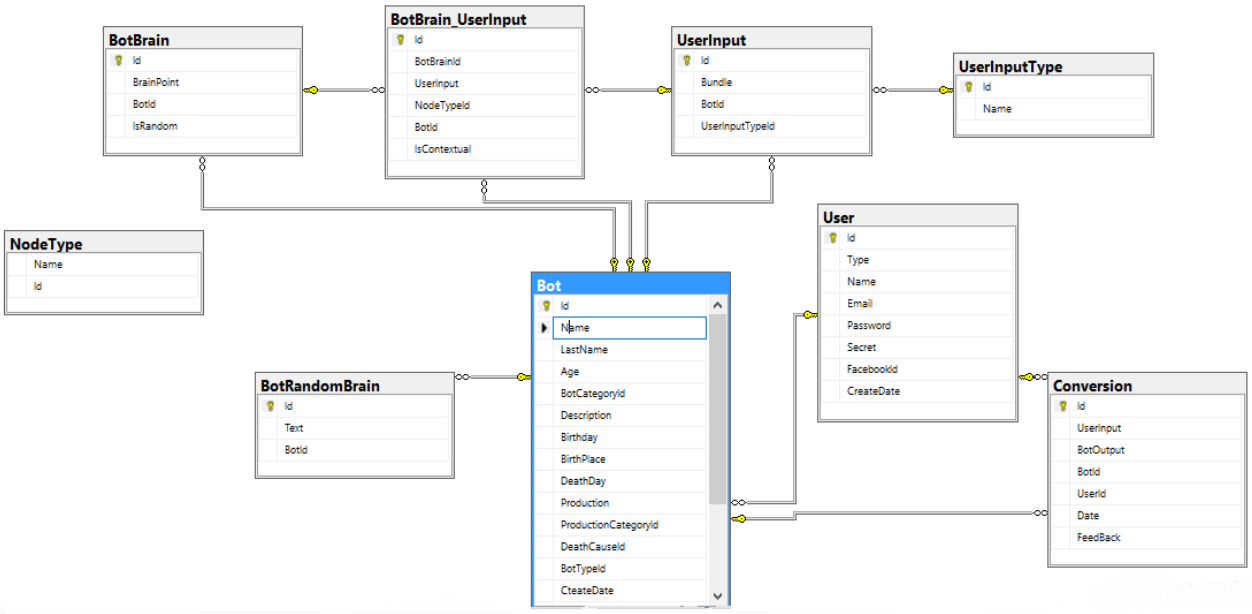
მოცემული ჩატბოტის შექმნის პროცესში გამოყენებულია რამდენიმე ალგორითმი, რაც გულისხმობს შემდეგს: ჩატბოტის „ტვინის“ შექმნის დროს შეგვიძლია მომხმარებლის სავარაუდო კითხვას მივანიჭოთ ორი სტატუსიდან ერთერთი. კითხვა შეიძლება იყოს პირდაპირი ან ფუძეზე დაყრდნობილი:

1. პირდაპირი სახის კითხვა გულისხმობს, რომ არჩეული პასუხი გაეცემა კითხვას მხოლოდ და მხოლოდ იმ შემთხვევაში, თუ კითხვის დამსმელი შეიყვანს კითხვას ზუსტად იმ სახით რა სახითაც ის ბაზაშია ჩაწერილი. მაგ: თუ ბაზაში წერია, რომ მომხმარებლის პირდაპირ კითხვას - „როგორ ხარ?“ - შეესაბამება ჩატბოტის პასუხი: „კარგად, თავად როგორ გიკითხოთ?“, მაშინ თუ მკითხველი დასვამს კითხვას „როგორ ხართ“ ის პასუხად ვერ მიიღებს „კარგად, თავად როგორ გიკითხოთ?“, რადგან ეს პასუხი გათვლილია, რომ უნდა გაეცეს მხოლოდ და მხოლოდ პირდაპირ კითხვას - „როგორ ხარ?“ - და არა მის რომელიმე მოდიფიკაციას.
2. ფუძე სიტყვის შემთხვევა გულისხმობს, რომ ჩატბოტის „ტვინში“ მომხმარებლის სავარაუდო კითხვა შეყვანილია არა პირდაპირ, არამედ დანაწევრებულად, კითხვის შემადგენელ ფუძე ნაწილებად. მაგ: თუ გვსურს, რომ გავაერთიანოთ ყველა ის სავარაუდო კითხვა, რომელზეც ჩატბოტის პასუხი იქნება: „მე 5 წლის ვარ“, მაშინ მომხმარებლის სავარაუდო კითხის ფუძე ვარიანტებში უნდა ჩამოვწეროთ: “ასაკ, წლის, რამდენი წლის“. ამის შედეგად ,როცა მკითხველი დასვამს კითხვას „რამდენი წლის ხარ“, თუ ბაზაში არ არსებობს ამ სახის პირდაპირი კითხვ,ა ალგორითმი შემოსულ კითხვას შეამოწმებს ბაზაში არსებული ფუძე ვერსიებისთვის და თუ ეს კითხვა შეიცავს ფუძეს მაშინ პასუხად დაგვიბრუნდება ამ ფუძის შესაბამისი პასუხი.

3. გარდა ფუძე და პირდაპირი კითხვებისა, ჩვენ საშუალება გვაქვს, თავიდან ავირიდოთ ის მომენტი, როცა რომელიმე კითხვაზე ალგორითმი ვერ იპოვის ვერც ფუძე და ვერც პირდაპირ შესაბამისობას, ამისთვის ბაზაში ვწერთ ე.წ. რენდომ პასუხებს, რომელთაგანაც ლოგიკური პასუხის არარსებობის შემთხვევაში მოხდება შედეგის არჩევა.
4. ჩეტბოტის მოცემული გადაწყვეტა გვაძლევს ერთ დამატებით ფუნქციას, კერძოდ, შეგვიძლია კონკრეტულ პასუხებს მივუთითოთ, რომ ისინი შედეგად დაბრუნდება მხოლოდ და მხოლოდ კონკრეტული კონტექსტური დიალოგური ხის არსებობის შემთხვევაში.

ბაზის არქიტექტურა

ბაზის არქიტექტურას აქვს შემდეგნაირი სახე:



პროგრამული უზრუნველყოფის კოდი

გვაქვს ორი ძირითადი კლასი Node და Edge, რომლებიც შესაბამისად აღწერენ წვეროებს და წიბოებს.

გვაქვს კლასი Network, რომელიც წარმოადგენს ყველა წვეროს და წიბოს ერთიანობას.

```
class Network
{
    public List<Node> Nodes { get; set; }
    public List<Edge> Edges { get; set; }
}

class Node
{
    public string id { get; set; }
    public string label { get; set; }
    public string group { get; set; }
    public string title { get; set; }
}

class Edge
{
    public long id { get; set; }
    public string from { get; set; }
    public string to { get; set; }
    public string color { get; set; }
}
```

```
static class ColorHelper
{
    static string red = "rgb(255, 99, 71)";
    static string blue = "rgb(0, 102, 204)";

    public static string IsContextual(bool flag)
    {
        return flag ? red : blue;
    }
}
```

მომხმარებლის კითხვის შექმნის ლოგიკა

```
public ActionResult CreateUserPoint(int id, string bundle, int userInputTypeId)
    {
        bundle = GalaRegexHelper.LeaveOneSpaceAndTrim(bundle);

        var userInput = _db.UserInputs.FirstOrDefault(item => item.BotId == id && item.Bundle == bundle && item.UserInputTypeId == userInputTypeId);
        if (userInput != null)
        {
            return null;
        }

        userInput = new UserInput() { BotId = id, Bundle = bundle, UserInputTypeId = userInputTypeId };
        _db.UserInputs.Add(userInput);
```

```
            _db.SaveChanges();

            return Json(new Node()
            {
                id = "user-" + userInput.Id,
                title = userInput.Bundle,
                label = userInput.Bundle.Length > 10 ? userInput.Bundle.Substring(0, 10) :
userInput.Bundle,
                group = userInputTypeId == 1 ? "base" : "direct"
            },
                JsonRequestBehavior.AllowGet);
        }
```

ბოტის პასუხის შექმნის ლოგიკა

```
    public ActionResult CreateBotPoint(int id, string point, bool isRandom)
        {
            point = GalaRegexHelper.LeaveOneSpaceAndTrim(point);

            var brainPoint = _db.BotBrains.FirstOrDefault(item => item.BotId == id &&
item.BrainPoint == point);

            if (brainPoint != null)
            {
                return null;
            }
```

```
                brainPoint = new BotBrain() { BotId = id, BrainPoint = point, IsRandom =
isRandom };
                _db.BotBrains.Add(brainPoint);
                _db.SaveChanges();

                return Json(new Node()
                {
                    id = "bot-" + brainPoint.Id,
                    title = brainPoint.BrainPoint,
                    label = brainPoint.BrainPoint.Length > 10 ?
brainPoint.BrainPoint.Substring(0, 10) : brainPoint.BrainPoint,
                    group = brainPoint.IsRandom ? "botRandom" : "bot"
                }, JsonRequestBehavior.AllowGet);
            }
```

ორ წერტილს შორის კავშირის შექმნის ლოგიკა

```
        public ActionResult Connect(int id, string first, string second, bool isContextual = false)
            {
                int userPoint = 0, botPoint = 0, nodeType = 0;
                if (first.Contains("user"))
                {
                    userPoint = int.Parse(first.Split('-')[1]);
                    botPoint = int.Parse(second.Split('-')[1]);
                    nodeType = (int)NodeTypes.InsideNode;

                }
```

```
                else
                {
                    botPoint = int.Parse(first.Split('-')[1]);
                    userPoint = int.Parse(second.Split('-')[1]);
                    nodeType = (int)NodeTypes.OutsideNode;
                }

                var node = _db.BotBrain_UserInput.FirstOrDefault(item => item.BotId == id &&
item.BotBrainId == botPoint && item.UserInput == userPoint);
                if (node == null)
                {
                    node = new BotBrain_UserInput()
                    {
                        BotId = id,
                        BotBrainId = botPoint,
                        UserInput = userPoint,
                        NodeTypeId = nodeType,
                        IsContextual = isContextual
                    };

                    _db.BotBrain_UserInput.Add(node);
                    _db.SaveChanges();
                }

                if (nodeType == 1)
                {
                    return Json(new Edge()
                    {
```

```
                id = node.Id,
                from = "user-" + userPoint,
                to = "bot-" + botPoint,
                color = ColorHelper.IsContextual(node.IsContextual)
            }, JsonRequestBehavior.AllowGet);
        }

        return Json(new Edge()
        {
            id = node.Id,
            from = "bot-" + botPoint,
            to = "user-" + userPoint,
            color = ColorHelper.IsContextual(node.IsContextual)
        }, JsonRequestBehavior.AllowGet); ;
    }
```

წაშლის ლოგიკა

```
public ActionResult DeleteStuff(int id, string[] nodes, string[] edges)
    {
        if (edges != null)
        {
            var dbEdges = _db.BotBrain_UserInput
            .Where(item => item.BotId == id && edges.Contains(item.Id.ToString())).ToList();
            _db.BotBrain_UserInput.RemoveRange(dbEdges);
            _db.SaveChanges();
```

```
            }

            if (nodes != null)
            {
                string index = nodes[0].Split('-')[1];
                if (nodes[0].Contains("user"))
                {
                    var node = _db.UserInputs.FirstOrDefault(item => item.BotId == id &&
item.Id.ToString() == index);
                    _db.UserInputs.Remove(node);
                }
                else
                {
                    var node = _db.BotBrains.FirstOrDefault(item => item.BotId == id &&
item.Id.ToString() == index);
                    _db.BotBrains.Remove(node);
                }
                _db.SaveChanges();
            }

            return null;
        }
```

დიალოგების გრაფის აგების ლოგიკა

```
    public ActionResult GetNetwork(int id = 0)
```

```
{
    var botBrainPoints = _db.BotBrains.Where(item => item.BotId == id)
        .Select(item => new Node() {
            id = "bot-" + item.Id,
            title = item.BrainPoint,
            label = item.BrainPoint.Length > 10 ? item.BrainPoint.Substring(0, 10) :
item.BrainPoint,
            group = item.IsRandom ? "botRandom" : "bot" }).ToList<Node>();

    var userInputs = _db.UserInputs.Where(item => item.BotId == id)
        .Select(item => new Node() {
            id = "user-" + item.Id,
            title = item.Bundle,
            label = item.Bundle.Length > 10 ? item.Bundle.Substring(0, 10) :
item.Bundle,
            group = item.UserInputTypeId == 1 ? "base" : "direct" }).ToList<Node>();

    var nodes = new List<Node>();
    nodes.AddRange(botBrainPoints);
    nodes.AddRange(userInputs);

    var links = _db.BotBrain_UserInput.Where(item => item.BotId == id).ToList();
    var edges = new List<Edge>();

    foreach (var item in links)
    {
        if (item.NodeTypeId == 1)
        {
```

```
            edges.Add(new Edge()
            {
                id = item.Id,
                from = "user-" + item.UserInput1.Id, to = "bot-" + item.BotBrain.Id,
                color = ColorHelper.IsContextual(item.IsContextual)
            });
        }
        else if (item.NodeTypeId == 2)
        {
            edges.Add(new Edge() {
                id = item.Id,
                to = "user-" + item.UserInput1.Id,
                from = "bot-" + item.BotBrain.Id,
                color = ColorHelper.IsContextual(item.IsContextual)
            });
        }
    }

    var model = new Network() { Edges = edges, Nodes = nodes };

    return Json(model, JsonRequestBehavior.AllowGet);
```

დიალოგების შენახვის ლოგიკა

```
string SaveConversion(string input, string forConversion)
    {
        TimeZone zone = TimeZone.CurrentTimeZone;
```

```
            DateTime local = zone.ToLocalTime(DateTime.Now);

            var con = new Conversion()
            {
                UserInput = input,
                BotOutput = forConversion,
                BotId = this.botId,
                UserId = User.Id,
                Date = local
            };

            _db.Conversions.Add(con);
            _db.SaveChanges();
            string hash = string.Format("{0}-{1}", con.Id, con.UserId);

            return hash;
        }
```

ჩატბოტის პასუხის ძებნის ევრისტიკა

```
    public JsonResult Calculate(string input)
        {
            Brain result = null;
            if (IsBotContextSaved())
            {
                result = FindExactMatches(GetSavedList(), input,
(int)NodeTypes.InsideNode);
```

```
                }

                if(result == null)
                {
                    result = FindExactMatches(_db.BotBrain_UserInput.
                        Where(item => item.BotId == botId &&
                            (!item.IsContextual || item.UserInput1.BotBrain_UserInput.Count(e =>
e.NodeTypeId == (int)NodeTypes.OutsideNode) == 0))
                            .ToList<Brain>(), input, (int)NodeTypes.InsideNode);
                }

                // თუ რეზალთი არის ცარიელი მერე შეიძლება რენდომებში ძრომიალი
                if (result == null)
                {
                    var botRandoms = _db.BotBrain_UserInput.Where(item => item.BotId ==
this.botId && item.BotBrain.IsRandom);
                    result = botRandoms
                        .OrderBy(item => item.Id)
                        .Skip(RandomIndex(botRandoms.Count()))
                        .FirstOrDefault();
                }

                this.GetSavedBotContext = result;

                string answer = Messages.AnswerNotFound, forConversion = answer;
                if (result == null)
                {
```

```
                var botRandoms = _db.BotRandomBrains.Where(item => item.BotId ==
this.botId);
                var randomAnswer = botRandoms
                    .OrderBy(item => item.Id)
                    .Skip(RandomIndex(botRandoms.Count()))
                    .FirstOrDefault();
                if (randomAnswer != null)
                {
                    answer = randomAnswer.Text;
                    forConversion = Messages.AnswerNotFound + " - " + randomAnswer.Text;
                }
            }
            else
            {
                answer = result.BotBrain.BrainPoint;
                forConversion = answer;
            }

            string hash = SaveConversion(input, forConversion);

            return new JsonResult()
            {
                Data = new { brainPoint = input, extra = answer, s =
MD5HashHelper.GetMd5Hash(hash) },
                JsonRequestBehavior = JsonRequestBehavior.AllowGet
            };
        }
```

პირდაპირი პასუხის ძებნა

```
Brain FindExactMatches(List<Brain> list, string input, int nodeTypeId)
    {
        var exactMatches = list.Where(item => item.UserInput1.UserInputTypeId == (int)UserInputType.Direct && item.UserInput1.Bundle == input && item.NodeTypeId == nodeTypeId).ToList();
        var exactMatche = exactMatches.OrderBy(item => item.Id).Skip(RandomIndex(exactMatches.Count())).FirstOrDefault();
        return exactMatche == null ? FindWordBaseMatches(list, input, nodeTypeId) : exactMatche;
    }
```

ფუძეებზე დაფუძნებული პასუხის ძებნა

```
Brain FindWordBaseMatches(List<Brain> list, string input, int nodeTypeId)
    {
        var wordBaseMatches = list
            .Where(item => item.UserInput1.UserInputTypeId == (int)UserInputType.Base && item.NodeTypeId == nodeTypeId)
            .AsEnumerable()
            .Select(item => new BotOutput
            {
                Node = item,
                Point = item.UserInput1.Bundle.Split(',').Count(r => input.Contains(r))
            })
```

```
                .Where(item => item.Point != 0)
                .OrderByDescending(o => o.Point)
                .ToList();

            foreach (var item in wordBaseMatches)
            {
                System.Diagnostics.Debug.WriteLine(input + " " + item.Point + " " +
item.Node.BotBrain.BrainPoint);
            }

            var size = wordBaseMatches.Count(item => item.Point ==
wordBaseMatches.Max(r => r.Point));
            var wordBaseMatche =
wordBaseMatches.Skip(RandomIndex(size)).FirstOrDefault();
            return (size == 0 && wordBaseMatche  == null) ? null : wordBaseMatche.Node;
        }
```

დასკვნა

ამრიგად, წინამდებარე ნაშრომის ფარგლებში შექმნილია ინტელექტუალური აგენტი, რომელსაც აქვს ვებ აპლიკაციის სახე. ამ სისტემის გამოყენებით კომპანიებს შეეძლებათ შექმანან თავიანთი ონლაინ კონსულტანტი, რომელსაც ექნება საკმარისი ცოდნა იმისთვის, რომ კომპანიის პროდუქტის მომხმარებლებს, ან უბრალოდ დაინტერესებულ პირებს გასცეს კითხვებზე პასუხი. პროექტს მიცემული აქვს საბოლოო სახე, რაც შესაძლებელს ხდის მის გამოყენებას კომერციული მიზნებისთვის, მას შემდეგ რაც რომელიმე კონკრეტული დაინტერესებული კომპანია მოახდენს ჩატბოტის „ტვინის“ აწყობას კონკრეტული პროდუქტის ან ბიზნეს სფეროს შესაბამისად.

გამოყენებული ლიტერატურა

- https://www.csee.umbc.edu/courses/471/papers/turing.pdf
- https://en.wikipedia.org/wiki/Computing_Machinery_and_Intelligence
- http://www.loebner.net/Prizef/loebner-prize.html
- http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction