Algoritme

Dit algoritme beschrijft een manier om vragen uit een ELO te suggereren aan gebruikers afhankelijk van de moeilijkheid van de vragen en het niveau van de gebruiker. Het programma moet hiermee gebruikers binden aan het systeem. Dit doet het door vragen aan te bieden die aansluiten bij het niveau van de gebruiker alsmede een ranglijst bij te houden van gebruikers onderling.

Om de moeilijkheid van vragen te bepalen meten we hoelang het gemiddeld duurt om een correct antwoord te geven op de vraag. Om gokken niet te laten lonen krijgt men ook een aanzienlijke tijdstraf bij ieder verkeerd gegeven antwoord. De score (het niveau) van een gebruiker is de gemiddelde tijd dat het hem (inclusief straftijd) gekost heeft om de vragen die hij heeft gedaan correct te beantwoorden. Het systeem houdt ook een totaal gemiddelde bij. Dit is de gemiddelde tijd dat het álle gebruikers heeft gekost om de vragen die ze hebben gedaan correct te beantwoorden (inclusief straftijd). Als een gebruiker hoger dan gemiddeld scoort (een kortere tijd dan gemiddeld als score heeft), krijgt hij vragen aangeboden waar men evenredig langer over heeft gedaan dan gemiddeld. Andersom krijgt een gebruiker die gemiddeld langer dan gemiddeld over zijn vragen heeft gedaan, vragen aangeboden waar men evenredig korter over heeft gedaan. De tijdscore van aangeboden vragen noemen we de ‘tegengestelde tijd’ en dit wordt als volgt berekend:

Tegengestelde tijd = 2x het totaal gemiddelde – het gemiddelde van de gebruiker

De gebruiker krijgt de vraag aangeboden die het dichtst bij de uitkomst van deze formule ligt. Vragen die hij al eerder heeft gemaakt worden hierbij overgeslagen. Hij mag dus niet zelf kiezen welke volgende vraag hij gaat doen. Als het totaal gemiddelde verandert of de tijd bij vragen verandert, dan krijgt hij mogelijk wel een andere vraag aangeboden door het systeem. De maximale tijd die gescoord kan worden bij een vraag (bijvoorbeeld als men een lange pauze neemt of de vraag helemaal niet beantwoordt) is gelijk aan de score bij het slechtste gokscenario (alles fout gokken alvorens het correcte antwoord te vinden). Je kunt iedere vraag maar één keer starten/maken.

De straftijd bij ieder verkeerd gegeven antwoord wordt berekend aan de hand van de volgende formule:

Straftijd = gemiddelde tijd dat men over de vraag doet / gemiddeld aantal foute antwoorden bij blind gokken

Als men dus blind gokt komt men (gemiddeld) uit op de gemiddelde tijd voor de vraag. Als je je tijd neemt mag je aannemen dat je onder dit gemiddelde gaat scoren (omdat je nadenkt over de vraag). Als je je tijd neemt en alsnog fout antwoordt verdien je de extra tijdstraf van de verlopen tijd. De keuze tussen gokken en nadenken moet op ieder moment om het even zijn (vandaar deze formule). Om te zorgen dat de tijd niet eindeloos doorloopt als een gebruiker geen antwoord geeft op de vraag hebben we het programma zo gemaakt dat de maximale tijd die je bij een vraag scoort gelijk is aan de score die je behaalt in het slechtste gokscenario (dat is als men als laatste pas het juiste antwoord gokt). In de toekomst willen we de tijd dat het duurt om een vraag te lezen ook niet meer meerekenen in de score (zodat men niet gaat gokken als men alleen de antwoorden leest).

Het gemiddeld aantal foute antwoorden bij blind gokken is afhankelijk van het aantal mogelijke antwoorden op de vraag en kan berekend worden met de volgende formule:

Gemiddeld aantal foute antwoorden bij blind gokken = (Aantal mogelijke antwoorden – 1) / 2

Iedere vraag kan dus een willekeurig aantal antwoorden hebben (maar minimaal 2). Bij een vraag met 4 mogelijke antwoorden is de gemiddelde foutenlast (als men gokt) volgens bovenstaande formule 1.5. Dit klopt want men geeft (met een gelijke kans) 0, 1, 2 of 3 verkeerde antwoorden alvorens een goed antwoord te geven. Het gemiddelde hiervan is 1.5. Bij 3 antwoorden is het gemiddelde aantal fouten bij blind gokken 1, bij 5 is dit 2 enz. Dit kan men narekenen met kansrekening.

Als een vraag voor het eerst gemaakt wordt bestaat er nog geen gemiddelde tijd voor die vraag. De gemiddelde tijd van de vraag wordt dan het totaal gemiddelde maar als deze ook nog niet bestaat (geen enkele vraag is nog gemaakt) voert de docent een schatting in van hoelang een gemiddelde vraag moet duren. Het systeem kan ook uitgebreider geïnitialiseerd worden door één leerling alle vragen te laten beantwoorden/maken (zodat ze ten opzichte van elkaar op het juiste niveau komen te staan).

Het systeem suggereert altijd eerst vragen die nog door niemand zijn gemaakt, alvorens de tegengestelde tijd te gaan gebruiken om vragen te suggereren die al wel eerder zijn gemaakt. Dit zodat alle vragen zo snel mogelijk mee gaan doen in het systeem.

Handleiding

Voor dit programma hebben we MySQL gebruikt in plaats van SQLite. Dit omdat deze database op het web veel gebruikt wordt en we dit dus wilden leren gebruiken. U moet dus MySQL eerst installeren op uw computer alvorens u het programma kunt gebruiken. Download MySQL hier: <https://dev.mysql.com/downloads/> . Het root wachtwoord wat u moet instellen tijdens de installatie staat bovenaan in de code (in Main.py). Ook bovenaan in de code staat de ingeschatte tijdsduur in seconden voor de eerste vraag die gemaakt wordt. Dit mag u wijzigen maar hij staat standaard ingesteld op 5 minuten (300 seconden). Daarbij staat ook een vraagnummer ingesteld voor de vraag van gemiddelde moeilijkheid die hoort bij de geschatte tijdsduur. Deze vraag is de eerste vraag die door een gebruiker gemaakt gaat worden.

De database tables kunt u bekijken en aanpassen met HeidiSQL. U kunt HeidiSQL downloaden en installeren via <https://www.heidisql.com/download.php>. Hiermee kunt u bijvoorbeeld vragen toevoegen of wijzigen. U kunt hier zowel de input als de output van het programma nagaan. Maak in HeidiSQL eerst een nieuw profiel aan voor MySQL en voer het root wachtwoord in wat bovenaan in de code staat (de andere instellingen hoeven niet veranderd te worden, dus laat staan op MariaDB).

De sample databases zijn bijgevoegd als bijlage in de repository. U moet deze databases eerst met HeidiSQL importeren via File > Run SQL file (daarna eventueel refresh). De SQL code in deze sample files overschrijft bestaande databases met dezelfde naam. Vul vervolgens bovenin de code de naam in van de database in waarmee u wilt verbinden. Dit is de inhoud van de sample databases:

* sample1: Lege database met 4 vragen en 4 gebruikers erin. Het antwoord is de vraag zelf; de vraag geeft dus het juiste antwoord aan. De 4 vragen hebben een variërend aantal mogelijke antwoorden (2, 3, 4 en 5 verschillende antwoorden).

Testrapport

Het volgende is door middel van systeemtesten getest:

* Correct database schema wordt aangemaakt als de database tables nog niet bestaan
* Nieuwe gebruiker wordt toegevoegd bij opgeven nieuwe username, bij opgeven bestaande username worden de gegevens van de bestaande gebruiker gevonden en geladen
* Eerste gebruiker krijgt default values aangeboden bij de eerste vraag die gemaakt wordt (deze geeft men op bovenaan in de code van Main.py)
* Correcte penalty en avg\_num\_of\_penalties worden berekend en vermeld bij elke vraag die gemaakt wordt.
* De penalty is afhankelijk van de total\_average\_time (bij vragen die niet eerder gemaakt zijn) of van de gemiddelde tijdsduur van de vraag (bij vragen die wel eerder gemaakt zijn). In beide gevallen wordt deze tijd gedeeld door de avg\_num\_of\_penalties, die afhanelijk is van het aantal mogelijke antwoorden op de vraag (zie formule hierboven).
* Programma vraagt om het antwoord en de verlopen tijd alvorens te antwoorden. Als een correct antwoord gegeven wordt worden de antwoorden met de verstreken tijd en eventuele penalties verwerkt in de database
* Bij een verkeerd antwoord krijgt de gebruiker de kans om nog een antwoord te geven, de straftijd en verlopen tijd bij een verkeerd antwoord worden onthouden
* Bij het geven van een antwoord wat al eerder gegeven is krijgt de gebruiker nog een kans te antwoorden en wordt de penalty en opgegeven verstreken tijd genegeerd (niet toegepast).
* Als er nog maar één antwoord gegeven kan worden (het correcte antwoord) geeft het programma automatisch dit antwoord met een verlopen tijd van 0 seconden (en zonder penalty)
* Bij het geven van het correcte antwoord stopt het programma en wordt de correcte totale tijd (incl. penalties) toegevoegd aan de database (zowel bij de gebruiker als bij de vraag).
* Als er vragen gemaakt zijn wordt de total\_average\_time hiervan ook gebruikt bij vragen die nog niet eerder gemaakt zijn (om de penalty te berekenen)
* Bij alle vragen wordt de avg\_num\_of\_penalties steeds correct uitgerekend en toegepast aan de hand van het mogelijk aantal antwoorden (zie formule hierboven).
* Als een gebruiker meerdere vragen maakt wordt zijn gemiddelde tijd correct herberekend aan de hand van zijn oude gemiddelde en zijn nieuwe score voor de nieuw gemaakte vraag. Dit gemiddelde wordt dan ingevoerd in de database.
* Als een vraag meermaals gemaakt wordt, dan wordt de correcte gemiddelde tijd voor deze vraag steeds correct uitgerekend (aan de hand van het oude gemiddelde en de nieuwe score) en ingevoerd in de database.
* Niet eerder gemaakte vragen worden altijd eerder aangeboden dan wel eerder gemaakte vragen
* Als alle vragen minstens één keer zijn gemaakt suggereert het systeem de vraag het dichtst bij de opposite\_time van de gebruiker (zie formule hierboven). Vragen die de gebruiker eerder gemaakt heeft worden hierbij genegeerd.
* Als een gebruiker alle vragen heeft gemaakt zegt het systeem dat en sluit het programma
* Tijden in de database worden opgeslagen met één getal achter de komma
* Alle gegeven antwoorden worden correct opgeslagen in de answersgiven tabel
* Bij het geven van een ongeldig antwoord wordt opnieuw naar een antwoord gevraagd en wordt de opgegeven tijd genegeerd
* De maximale tijd die gescoord wordt op een vraag is gelijk aan de score in het slechtste gokscenario (alles verkeerd gokken alvorens het juiste antwoord te vinden)