**Verslag programeren voor wiskunde**

Inhoud

[1.1 Background/question 2](#_Toc157734730)

[1.2 algorithm 2](#_Toc157734731)

**Yunus Yavuzyigitoglu, Nard Boeding**

# Background/question

Het SET kaartspel is een spel waar het om draait om het vermogen om patronen/relaties van kaarten te herkennen en analyseren. Bij het spel worden er kaarten uitgedeeld met 4 specifieke eigenschappen: kleur, symbool, aantal en opvulling. Spelers van het spel moeten sets maken van drie kaarten waarbij elke eigenschap van de kaarten van de drie kaarten helemaal gelijk zijn, of helemaal verschillend. Dus bijvoorbeeld als symbool, aantal en opvulling van alle kaarten hetzelfde zijn en de kleur van de kaarten niet dan is het nog steeds een set, dus we kijken specifiek naar alle eigenschappen of ze hetzelfde zijn of helemaal verschillend.

De achtergrond van de opdracht is dat je het spel na op python en zorg ervoor dat de computer hetzelfde vaardigheden als een mens heeft bijvoorbeeld dat de computer logisch kan nadenken, dat de computer logisch kan redeneren en snel kan denken. Om dit te doen zijn er sub vragen die ervoor zorgt dat de computer hetzelfde vaardigheden als een mens heeft in deze opdracht.

De kernvraag van deze opdracht is: maak een spel op python waarbij de computer de rol van een tegenspeler kan overnemen in het SET spel. Dit betekent dat de computer in staat is om snel en accuraat kan bepalen of een combinatie van 3 kaarten een set of niet zonder dat het enige regels van het spel breekt. Om dit te kunnen doen moet de computer de juiste algoritme hebben en logica van de regels van het spel. Verder moet de computer alle mogelijke sets kunnen identificeren binnen een verzameling van 12 kaarten en dit kan je doen kaarteigenschappen te vergelijken.

De ‘main challenge’ van deze opdracht is dat je bij het ontwerpen van een spel het meest efficiënte algoritme maakt dat in staat is om mogelijke kaartcombinaties te vinden die geldig zijn en daar lig ook gelijk de moeilijkheid van de opdracht. Dit kan je doen door het meeste efficiënte code te gebruiken. En als bonus kan je de spel er ook creatief eruit laten zien zodat het aantrekkelijker is. Het gebruik van datastructuren is essentieel voor deze opdracht zodat je informatie over kaarten kan opslaan en/of opvragen.

De sub opdrachten zijn heel belangrijk om de kernvraag op te lossen, zoals we net zeiden willen we dat de computer een rol als mens kan spelen. Om dat te kunnen moet de robot dus aan specifieke eigenschappen voldoen zoals: logisch nadenken, logisch redeneren, snel na denken en de spel kan spelen als een normaal mens, en dit kan je doen door de computer een algoritme te geven om te zien of een bepaalde set geldig is of niet geldig is. Dit moet je heel goed aangeven, want de als de computer niet weet of iets een set is of niet zal het niet aan de regels kunnen houden, waardoor het niet als een mens zou kunnen denken. De tweede ding waaraan de algoritme aan moet voldoen is dat het alle mogelijke sets kan vinden binnen een verzameling van 12 kaarten en om dit te doen moet je ervoor zorgen dat de computer een algoritme heeft die alle mogelijke sets kan vinden om geldige sets te vinden wat later weer essentieel is om een set te vinden van het spel zelf.

# Algorithm

Binnen in de code worden kaarten met elkaar vergleken in de stukjes: def vergrlijk\_nummer, def verglijk\_symboool, def verglijk\_kleur, def verglijk\_shading. Dit doen we om te bepalen of drie kaarten een geldige set vormen doordat ze allemaal hetzelfde eigenschappen (per eigenschap) hebben of omdat ze qua eigenschap (per eigenschap) niet op elkaar lijken. Als dit zo is dan wordt returned the functie de waarde true, wat belangrijk is voor wat er daarna gebeurt. In def verglijk worden de 4 functies bekeken of ze de waarde true returnen en als ze dat doen dan betekent het dat er een geldige set is gevorm en als een van de 4 functies een waarde False returned dan zal er niet een geldige set ontstaan, waardoor def verglijk ook false zal zijn. Vervolgens maken we een class genaamd class Kaart. Dit is een heel essentieel deel van onze code, want class Kaart definieert een structuur voor de eigenschappen die kaarten vertegenwoordigd in het spel. Wij hebben gekozen om onze class Kaart zo te coderen als onze code van nu vanwege de volgende redenen: het is volledig, de class definieert een volledig en zelfstandig object dat alle eigenschappen van bevat. Het is duidelijk qua leesbaarheid en onderhoud en dat komt door de \_\_str\_\_ en \_\_init\_\_. Het is flexiebel en je kan het makkelijk uitbreiden als je dat wilt zonder data structuren te verstoren en in de class zelf hebben we gekozen om self = self (met dingen daar voor) te kiezen, want je kan het in een line zetten in plaats van 4 waardoor we wat lijnen besparen en het niet al te lang maken. Vervolgens hebben we dictionary gemaakt genaamd kaart\_eigenschappen. Hierin hebben we alle namen van de fotos ingezet en ze gekoppeld aan bepaalde waardes bijvoorbeeld 0,1,2 we hebben na de dictionary gelijk een paar dingen aangepast, want er waren kleine fouten gemaakt. We hebben voor deze stijl van een dictionary gekozen, want we vonden dat dit het meest logisch is en de andere manier moeilijk te begrijpen is. Omdat we kleine fouten hebben gemaakt in de kaart\_eigenschappen hebben we een funcite gemaakt genaamd en\_nu\_goed\_om. Hier gebruiken we 2 lijsten genaamd kaartvolgorde en altvolg. Het doel van kaarvolgorde is dat het wordt gebruikt om de kaarten in hun oorspronkelijk vorm op te slaan, zoals gedefineerd is in class Kaart en altvolg is een alternatieve representatie die wordt gemaakt door visuele/tekstuele weergave van de kaarteigenschappen die verschilt van oorspronkelijk eigenschappen. Dit zouden we normaal gesproken niet toevoegen maar doordat we een kleine fout hadden gemaakt en door wouden gaan hebben we dit bedacht. Hierna hebben we een nieuwe algoritme gemaakt genaamd elkeSET en bij deze algoritme hebben we de time complexity O(n^3) gekozen waarvoor n is kaarten. Het doel van elkeSET is dat we hiermee alle SETs uit 12 willekeurige kaarten vinden. De reden van waarom wij O(n^3) hebben gekozen is dat het algoritme werkt voor n != 12, maar de time complexity is O(n^3), het is namelijk een 3-dimensionaal driehoeksgetal. En vergleken met de andere time complexities is dit hebben we voor O(n^3) gekozen want ook al is het wat langzamer dan bijvoorbeeld O(n^2) het past nog steeds beter in de aard van het probleem, de grootte van de dataset en omdat wij een volledigheid belangrijk vinden (de volledigheid van O(n^3) is voor deze opdracht beter dan de volledigheid van O(n^2) en of andere). Vervolgens hebben we de algoritme kiesSET gemaakt waarbij we de computer een van de sets laat kiezen. Deze algoritme roept elkeSET aan om mogelijke sets te vinden van de kaarten die de computer heeft gekregen. Als er geen sets zijn gevonden dan doet het niet zoveel, maar als er meerdere sets zijn dan kiest de computer een van de sets, we hebben hierbij random gebruikt omdat dit het makkelijkste manier is om de computer een van de hoeveelheid sets te laten kiezen die het bevat.

# manual