1 Wat zijn zelflerende computersystemen?

1.1 Inleiding

Elk jaar boekt de mens grootschalige vorderingen op het gebied van computers, zowel hardware als software. Iets waar wij echter nog niet in geslaagd zijn te maken is een ware **Artificial Intelligence**, al lukt het steeds beter de schijn van denken te creren. Voorbeelden zijn de *persoonlijke assistenten* die inmiddels in elke smartphone gentegreerd zijn. *Siri*, *Google Now* en *Cortana* maken gebruik van spraakherkenning om de gebruiker de gevraagde informatie te tonen, maar denken zoals mensen doen ze hierbij niet. Hoe een computersysteem toch beter kan worden in het imiteren van menselijk gedrag en van zijn eigen fouten kan leren onderzoeken we in deze deelvraag.

1.2 Verschillende algoritmes

Computers hebben geen dus bewustzijn. Om deze reden kunnen ze niet zelf bepalen iets te doen. Waar computers wel in uitblinken, is het uitvoeren van taken die ze zijn opgelegd. Vaak komen deze taken in de vorm van code. Via code kan je computers opdrachten geven, bijvoorbeeld laat een scherm zien. De boodschap valt echter niet op deze manier over te brengen, afhankelijk van de taal waarin je programmeert zijn er vaste commando's waar de computer op zal reageren. Naarmate de opdracht die je een computer wil laten uitvoeren complexer wordt, zal ook het gebruik in deze commando's een verandering zien. Hier komen algoritmes in het spel. Een algoritme is een soort stappenplan voor de computer, waarin een complexere handeling in duidelijke opdrachten weergegeven wordt. De volgende definitie geeft een betekenis in de meest algemene zin: een algoritme is een eindige reeks instructies om vanaf een beginpunt een bepaald doel te bereiken.[**?**]

Een toegankelijke vergelijking is koken. Er is een **input** van voedsel waar uiteindelijk een gerecht uit moet komen, de **output**. Voor het tot stand komen van dit gerecht gebruik je misschien een recept. Dit recept is als het ware het algoritme. Uit de gegeven definitie is af te leiden dat het aantal mogelijke algoritmes ontzettend groot is. Niet alleen is het een ruim begrip, ook kan het desbetreffende doel waarschijnlijk op meerdere manieren bereikt worden. In deze verschillende methodes kan de een echter beter zijn dan de andere, bijvoorbeeld door efficiënter te zijn.

Uiteraard zijn er ook vele algoritmes die gebruik maken van toepassingen, zoals een **queue** en een **stack**, die betrekking hebben tot ons onderwerp. Enkele hiervan zullen hier beschreven worden:

1.3 Breadth-first search (BFS)

Dit algoritme, bedacht in de jaren vijftig van de vorige eeuw door E.F. Moore[b], een Amerikaans professor in de wiskunde en computer sciences en een voortrekker in kunstmatig leven, is een zoekalgoritme voor datasets in de vorm van grafieken of 'boom'-structuren. In deze dataset wordt een **node** als oorsprong benoemd, de **root**. Ook wordt een bepaalde uitkomst als doel gesteld. Vervolgens krijgt elke node drie waardes aangewezen:

- De afstand van de huidige node naar de root. Dit is het aantal stappen dat gezet moet worden om bij de root te komen.
- De node die vóór de huidige node kwam, de **predecessor**. Anders gezegd: bij welke node je uitkomt als je een enkele stap terug zet.
- Een **state**. De state houdt bij of een node al gecontroleerd is.

Bij Breadth-first search wordt gebruik gemaakt van een queue. Dit is een lijst waar nodes aan toegevoegd en uitgehaald kunnen worden. Net zoals een daadwerkelijke wachtrij wordt het 'als eerste erin, als eerste eruit' principe toegepast.

1. Maak een lege lijst S voor bezochte nodes.
2. Maak een lege lijst Q met de queue.
3. Benoem n node als root en voeg deze toe aan S.
4. Voeg de root toe aan Q.
5. Terwijl Q niet leeg is:
   (a) Haal de voorste node uit de queue. Dit is de 'current' node.
   (b) Als current het doel is:
      i. Return current.
   (c) Voor elke node die grenst aan* current:
      i. Als deze node nog niet bezocht is en dus niet in S zit:
         A. Voeg de node toe aan S.
         B. Zeg dat de predecessor van de node de current node is.
         C. Haal de node uit de queue.

**Aangrenzend zijn betekent hier 'in directe verbinding staan met'.*

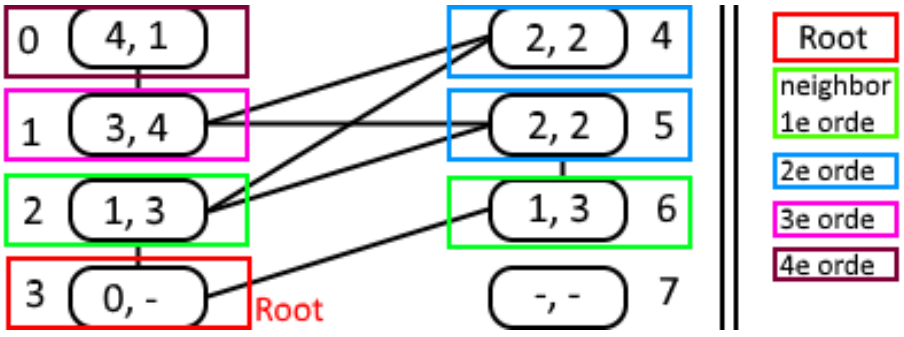

Figure 1: Schematische weergave van een willekeurige dataset. De volgorde waarin de nodes bezocht worden is 3 2 6 4 5 1 0.

Hierboven is een voorbeeld van een simpele dataset weergegeven, genummerd van 0 tot en met 7. Node 3 is de root en 0 het doel. Voor het gemak staan in elke node twee getallen: de afstand tot 3 en het nummer van de predecessor. Met kleuren zijn aangegeven in welke volgorde de nodes bezocht worden. 3 is

de root. 3 wordt toegevoegd aan de lijsten Q en S. Hierdoor is de queue niet leeg. 3 wordt weer uit de queue gehaald en n voor n worden de aangrenzende nodes bekeken. Hierbij worden ze toegevoegd aan de stack. Omdat zowel 2 als 6 het niet het doel is, herhaald het algoritme zich. Nu wordt 2 gecheckt. Het doel is niet gevonden. Daarna 6, ook zonder succes. (Let hierbij op dat node 5 niet nogmaals gecheckt wordt, dit is namelijk als bij node 2 gedaan en is dus al aanwezig in lijst S) Intussen zij 4 en 5 toegevoegd aan de queue, ze zijn immers verbonden met 2. Ook hier wordt het proces herhaald, node 1 zit nu in de queue. Uiteindelijk wordt node 1 gecheckt en wordt het doel, node 0, gevonden.

Met BFS kan je de weg van de root naar het doel achterhalen. Dit is nuttig als je bijvoorbeeld een wegennetwerk hebt en wil weten wat de kortste weg van de ene naar de andere stad is.

1.4 Depth-first search (DFS)

Evenals breadth-first search is depth-first search een algoritme voor het doorlopen van datasets in grafieken of trees. DFS verschilt echter op twee manieren van BFS:

- Depth-first search gebruikt een stack in plaats van een queue. Waar nodes in een BFS systeem in een wachtrij werden geplaats met een 'Als eerst erin, als eerst eruit' principe, handhaaft een DFS systeem een wachtrij meer vergelijkbaar met een stapel papieren. Telkens pak je het bovenste element van de stapel om mee te werken, maar als je iets in de wachtrij stopt, komt dit ook weer bovenop de stapel te liggen. De meest recente toevoeging zal dus als eerste weer eruit gehaald worden.

- Breadth-first search begon bij een root. Vervolgens werd gekeken naar alle neighbors. Als de gewenste uitkomst niet tussen deze neighbors zit, worden de neighbors van deze neighbors gecontroleerd. Dit proces herhaalt zich totdat het doel gevonden is. Depth-first search begint ook bij een root, maar kijkt direct naar een weg tot een node bereikt is die geen neighbors meer heeft. Als het doel dan niet bereikt is wordt een andere weg geprobeerd. Hiervoor wordt gebruik gemaakt van **recursive backtracking**.

In figuur **** is de werking van BFS en DFS weergegeven. Het getal in elke node geeft aan als hoeveelste het bereikt wordt. De kleuren representeren de gebieden die het algoritme per stap doorloopt.

Ook bij DFS hebben de nodes een state: bezocht of niet bezocht. Ten eerste wordt de root gekozen en wordt deze als bezocht opgeslagen. Zoals te zien wordt er vanaf de root n (willekeurige) neighbor gekozen om te onderzoeken. Elke bezochte neighbor wordt als bezocht genoteerd. De root wordt in de stack geplaats. Als de gekozen neighbor niet het doel is, wordt de meest recentelijk toegevoegde node, de root, gehaald, de eerst bezochte had immers geen aanliggende nodes om te onderzoeken. Deze tweede neighbor, cirkel 2 in de figuur, heeft wel een neighbor. Deze wordt gecontroleerd, evenals diens neighbors. Telkens wordt de huidige node toegevoegd aan de stack.
Als het programma de laatste in een reeks nodes bereikt heeft, wordt de bovenste node uit de stack gepakt en gekeken of daar nog niet bezochte nodes aan

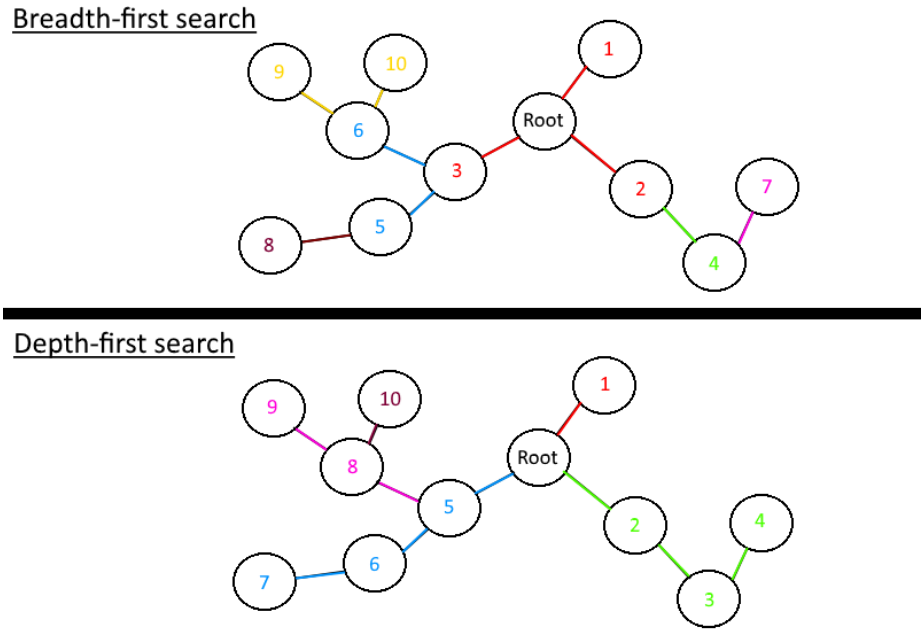

Figure 2: Schematische weergave van een willekeurige dataset.

grenzen. Dit wordt backtracking genoemd. Dit proces wordt herhaald totdat het doel gevonden is of totdat alle nodes geweest zijn.

Als vuistregel kan het volgende gehanteerd worden: depth-first search wordt gebruikt als je weet dat er maar n uitkomst is, breadth-first search als je de kortste weg wil weten.

1.5 Voorbeelden algoritmes

Algoritmes hebben meestal vele toepassingen. Hier zijn enkele voorbeelden van de eerder genoemde algoritmes.

1.5.1 Breadth-first search

In figuur **** is een dataset te zien, bijvoorbeeld een telefoonboom. Elke cirkel representeert een persoon. Zo kan persoon A de personen B en C bellen, maar A bezit geen andere telefoonnummers. Toch zou hij een boodschap naar H kunnen sturen: via C. Stel, persoon A wil nu iets tegen F zeggen. In een kleine dataset als deze is makkelijk met het oog te zien dat de snelste manier hiervoor A C F is en dat A B C E F veel langer is. Bij grotere datasets is dit echter al snel moeilijk met zekerheid te zeggen. Hiervoor kan breadth-first search ingezet worden.

1.5.2 Depth-first search

Depth-first search kan gebruikt worden voor zowel het maken als oplossen van doolhoven. In figuur **** is een doolhof te zien dat gemaakt is met behulp

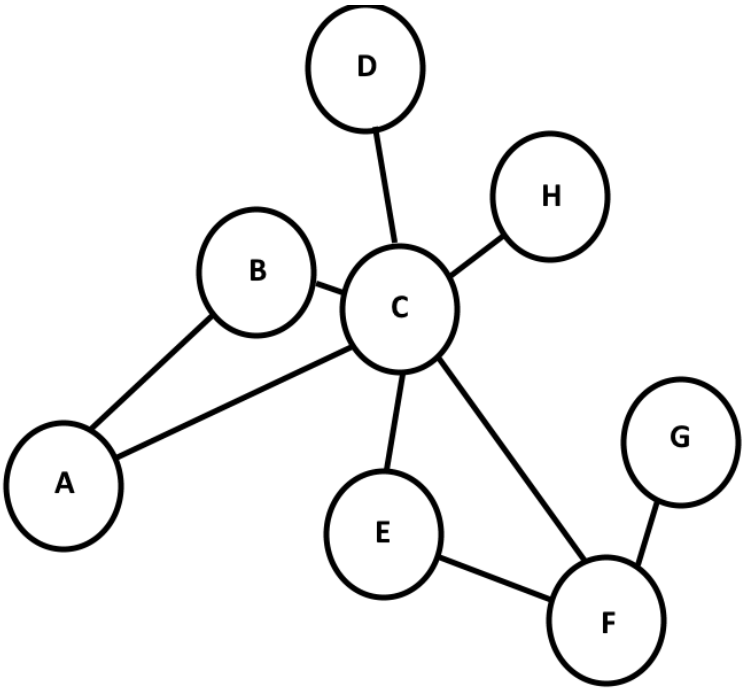

Figure 3: Schematische weergave van een willekeurige dataset.

van DFS. (****programma gemaakt in kader van deze deelvraag****). Het algoritme in de vorm van een stappenplan is als volgt:

1. Maak de start cel current en markeer deze als bezocht.
2. Terwijl er nog niet bezochte cellen aanwezig zijn:
   (a) Als current neighbors heeft die nog niet bezocht zijn:
      i. Kies willekeurig een van de neighbors
      ii. Voeg current toe aan de stack
      iii. Verwijder de muur tussen de huidige cel en de gekozen cel
      iv. Benoem de gekozen cel als current en zet de state op bezocht
   (b) Anders, als de stack niet leeg is:
      i. Haal de laatst toegevoegde cel uit de stack en verwijder deze hieruit
      ii. Maak deze cel current

1.6 Zelflerend?

Breadth-first search en Depth-first search zijn beide algoritmes met vele toepassingen. Toch kunnen beide algoritmes niet als zelflerend worden beschouwd, ze verbeteren hun manier van zoeken namelijk niet. Hoe zit een zelflerend systeem dan wel in elkaar? Hoe kan een algoritme zichzelf verbeteren?

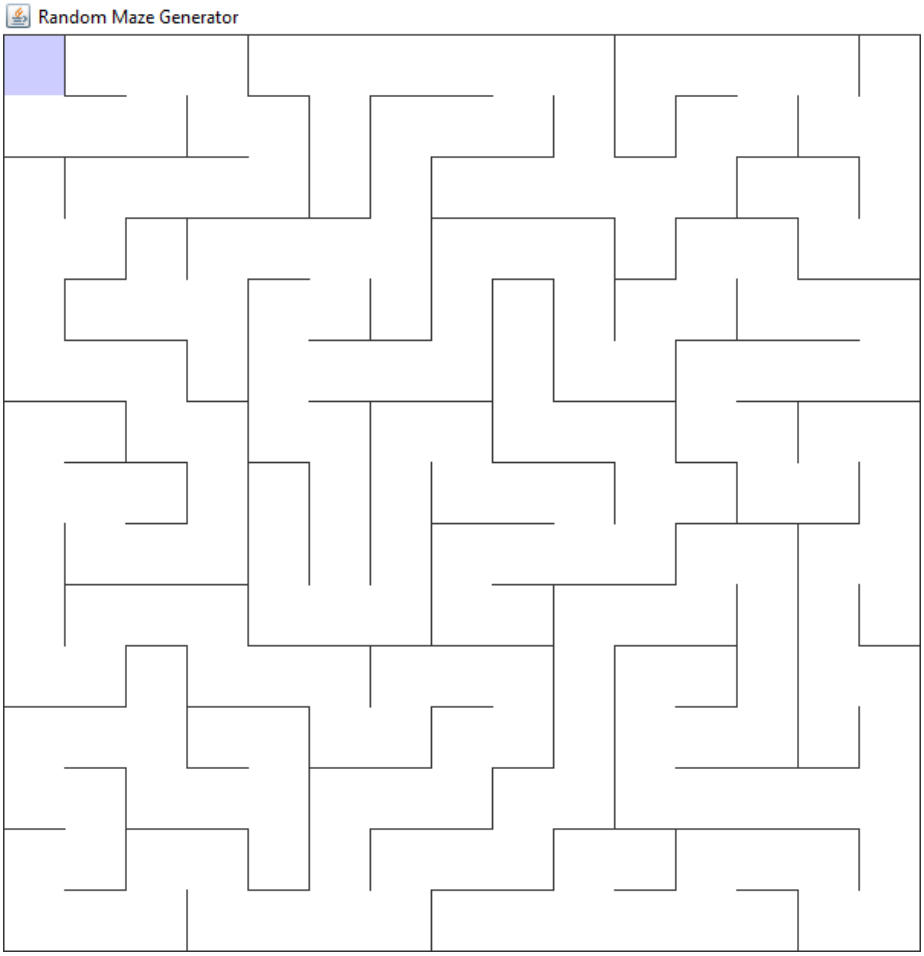

Figure 4: Een voorbeeld van een automatisch gegenereerd doolhof, gebruik makend van DFS.

1.7 Machine learning

Een zelflerend systeem is een algoritme gebaseerd op machine learning. Machine learning werd door Arthur Samuel, een pionier op dit gebied, gedefinieerd als: *A field of study that gives computers the ability to learn without being explicitly programmed.* [**?**] In tegenstelling tot de eerder genoemde algoritmes is een zelflerend systeem in staat zichzelf te verbeteren. Hierdoor kan het taken uitvoeren waarbij reguliere algoritmes tekort schieten. Welke taken dit betreft, zullen we in de derde deelvraag behandelen.

In figuur 4 is een schematische weergave van een zelflerend systeem afgebeeld. Bepaalde input data gaat het systeem in en bepaalde output data komt het systeem uit. De input en output data bestaat uit één of meerdere getallen. Als de input simpelweg een reeks getallen betreft, zal dit direct als input gebruikt kunnen worden. In het geval dat de input uit een ander datatype bestaat, zoals een plaatje, zal dit omgezet moeten worden in een reeks getallen voordat het in een zelflerend systeem gebruikt kan worden. Het algoritme zal deze getallen

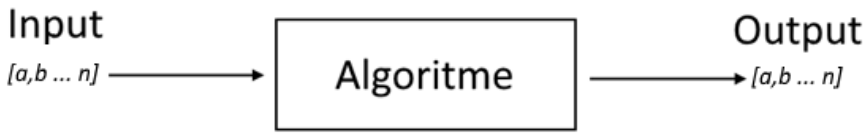

Figure 5: Schematische weergave van een zelflerend systeem

bewerken tot de gewenste output. Deze output wordt eveneens in getallen gegeven. Waar nodig zullen deze getallen dus weer moeten worden omgezet tot het gewenste datatype.

Er zijn veel verschillende algoritmes die gebruikt kunnen worden voor een zelflerend systeem. Elk algoritme heeft voor- en nadelen en is geschikt voor andere doeleinden. Een aantal van deze algoritmes zullen we in de tweede deelvraag behandelen.

1.8 Training

Een zelflerend systeem begint in de meeste gevallen zonder enige kennis van de data. Om de gewenste output te kunnen produceren is het dus nodig om het systeem eerst input data te geven zodat het kan leren. Dit proces wordt het **trainen** genoemd. Voor het trainen van een zelflerend systeem is training data nodig. Deze data moet gelijk of gelijkwaardig zijn aan de *echte* data. De training data kan in veel verschillende vormen voorkomen en de manier van trainen is afhankelijk van de vorm van de (training) data. In figuur 5 is te zien dat het trainen los staat van het algoritme. Dit verschil zullen we in de volgende deelvraag wat duidelijker maken. Er zijn drie prominente manieren waarop een zelflerend systeem getraind kan worden: **supervised**, **unsupervised** en **reinforcement learning**.

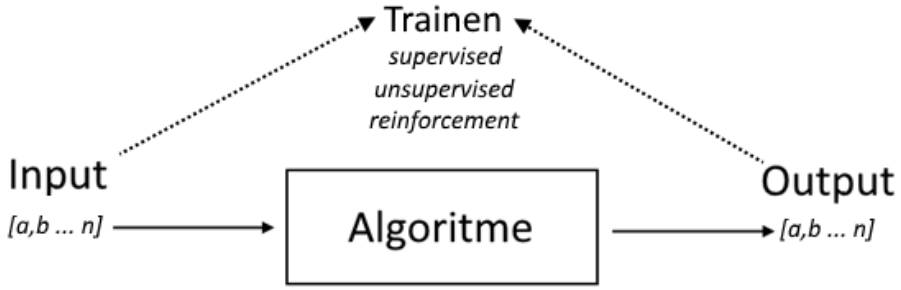

Figure 6: Schematische weergave van een zelflerend systeem

1.8.1 Supervised Learning

In het geval van supervised learning heb je te maken met **labeled** training data. Anders gezegd: van een bepaalde input is de gewenste output al bekend. Een

klassiek voorbeeld van een labeled dataset is een dataset van huisprijzen en huiseigenschappen (zie figuur 1)

Huisprijs (output)	Huiseigenschappen (input)		
	Woonoppervlakte	Perceeloppervlakte	Aantal Kamers
519.000	124 m	311 m	4
569.000	133 m	309 m	5
569.500	170 m	310 m	6

Table 1: Labeled dataset Bron: http://www.funda.nl/koop/huizen/

Bij de training dataset van figuur 1 is de gegeven input de huiseigenschappen en de gewenste output de huisprijs. Het systeem wordt met deze dataset getraind. Hierdoor leert het een output te produceren die steeds dichter bij de gewenste output ligt. Als er een verband bestaat tussen de huiseigenschappen en de huisprijs, wat waarschijnlijk het geval is, zal het zelflerende systeem na genoeg trainen in staat zijn zelf bij nieuwe huiseigenschappen een huisprijs te voorspellen. [**?**]

1.8.2 Unsupervised Learning

Unsupervised learning kan gebruikt worden bij een **unlabeled** dataset ofwel, een dataset waarbij de data niet geclassificeerd is en er geen gewenste output bekend is. Als je een dataset hebt van heel veel niet-geordende foto's is het niet mogelijk om dit te classificeren. Als een deel van de dataset gelabeld wordt, zal met behulp van supervised learning de rest van de dataset geclassificeerd kunnen worden. Dit is echter in veel gevallen niet mogelijk, bijvoorbeeld doordat de dataset enorm groot is of er zodanig veel verschillende groepen bestaan dat het menselijk niet mogelijk is ook maar een deel te labelen. Ook kan het zo zijn dat men niet weet of er een verband aanwezig is. Kortom: unsupervised learning wordt gebruikt voor het classificeren van data, zonder dat er groepen vooraf gedefinieerd zijn. Met behulp van deze vorm van training zullen in een grote dataset verbanden kunnen worden ontdekt, die men misschien niet zonder hulp had kunnen achterhalen.[**?**]

1.8.3 Reinforcement Learning

Reinforcement learning is een zeer specifieke soort van leren. Er is bij deze vorm van learning geen dataset met input data, maar is er een bepaalde **context**. In deze context bevindt zich een **agent**. Een agent is een object dat bepaalde opdrachten kan uitvoeren. De context is een wereld waarin deze agent zich bevindt. Door de agent bij bepaalde acties pluspunten of minpunten te geven kun je bepaald gedrag bevorderen.

In figuur 6 is het spel Pac-Man te zien. Op dit spel zou reinforcement learning toegepast kunnen worden. De agent is hierbij pacman, dit is namelijk een object dat bepaalde opdrachten kan uitvoeren, zoals: beweeg naar links. De context is hierbij het level, ofwel: de positie van de muren (de blauwe obstakels), de posities van de ghosts (de gekleurde vijanden), de posities van de pac-dots (de kleine stipjes) en de posities van de power-pellets (de grotere stipjes). [4] Het eten van de pac-dots is positief, het geraakt worden door de ghosts is negatief.

Figure 7: Pacman

Door reinforcement learing toe te passen op het spel zal de agent steeds beter worden in het spelen van het spel.

1.9 Conclusie

Zelflerende computersystemen zijn algoritmes gebaseerd op machine learning. Een zelflerend systeem verschilt van reguliere algoritmes zoals breadth-first search en depth-first search doordat ze in staat zijn zichzelf te verbeteren.