Universiteit van Amsterdam
3e jaars bachelor AI project

Multi-agent Quest Learning

auteurs: **Attila Houtkooper** (coll. nr. 0274542)
De Maalkom 7
1191 LP Ouderkerk aan de Amstel

Gideon Maillette de Buy Wenniger (coll. nr. 0234222)
Dinges
Amsterdam

supervisor: **Maarten van Someren** (http://hcs.science.uva.nl/usr/maarten/)

datum: 29 Juni 2005

Samenvatting

Wat we in dit project wilden oplossen was of we een techniek konden ontwikkelen om agenten in verschillende werelden relevante informatie te laten generaliseren zodat ze gebruikt kan worden in nieuwe situaties. Daarnaast was het doel om de agenten te laten communiceren op een dusdanige manier dat de performance beter is dan wanneer de agenten op zichzelf werken.
Hiertoe hebben we een applicatie gebouwd die meerdere spelwerelden tegelijkertijd kan simuleren. Vervolgens hebben we reinforcement learning geimplementeerd die de optimale route door de spelwereld leert te kiezen. Hieruit is gebleken dat het gebruik van generalisaties zorgt voor convergentie waar normaal policy learning dat niet bereikt. Echter, er bleek ook uit dat communicatie weliswaar een voordeel vormt, maar niet voor alle gevallen even veel.

Inhoud

1. Introductie

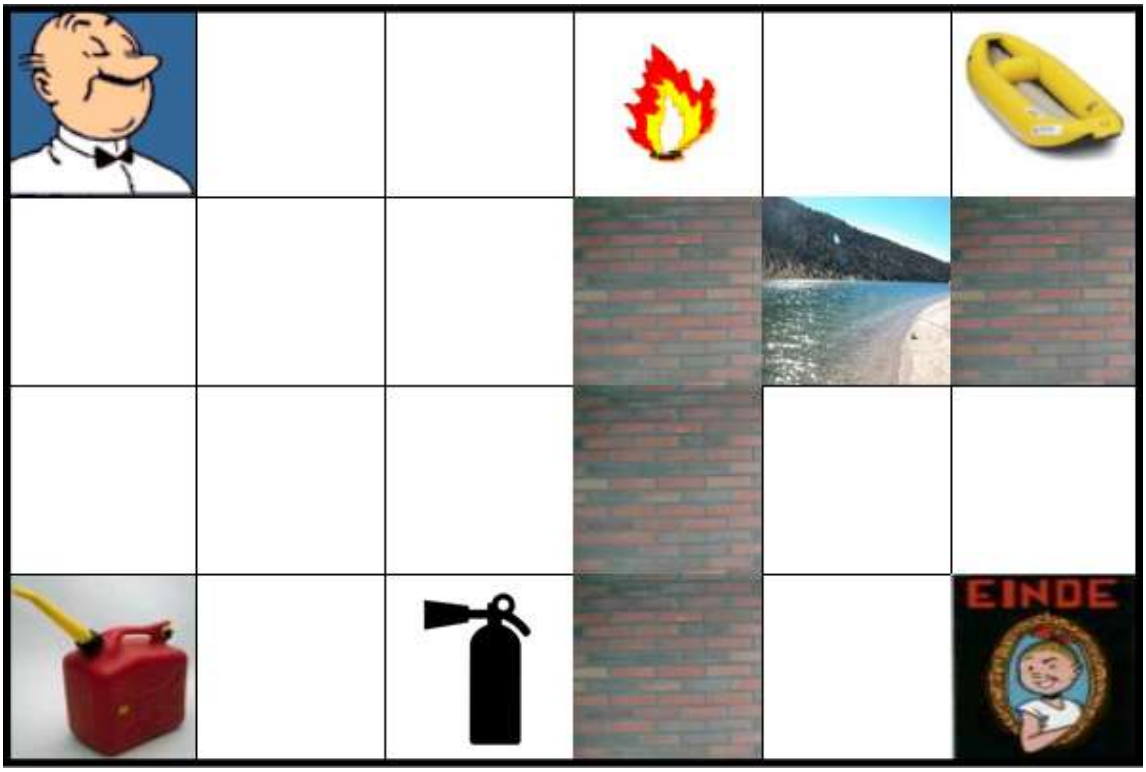
De vraag die we willen beantwoorden is: Heeft het gebruik van generalisaties over analoge werelden en communicatie van die generalisaties tussen agenten een significant effect op de snelheid van het leren en behaalde rewards door de afzonderlijke agenten? Zo ja, onder welke omstandigheden is dit met name het geval?
Om de onderzoeksvraag te beantwoorden hebben we in de eerste plaats een spelwereld moeten implementeren, waarin de agenten de problemen tegenkomt waarover hij moet leren. In de tweede plaats hebben we Reinforcement Learning geïmplementeerd, in de vorm van Q-learning, om te laten zien dat agenten het probleem kunnen leren en als basis voor de generalisatie en communicatie van de volgende stap. Als aanvulling van Reinforcement Learning hebben we Dyna-Q geïmplementeerd om het leren nog te versnellen. Vervolgens hebben we generalisaties geïmplementeerd in de vorm van Obstakel-Actie paren en Item-Actie paren die kort samengevat vastleggen welke items goed kunnen worden gebruikt om bepaalde obstakels op te lossen, en welke items, in dat verband, moeten worden opgepakt. Vervolgens hebben we Multi-Agent learning geïmplementeerd en de communicatie van generalisaties tussen de agenten. Tot slot hebben we een flink aantal simulaties gedraaid met verschillende condities, en grafieken gemaakt van zaken als gemiddeld aantal stappen en gemiddelde reward, om inzicht te krijgen in het effect van het gebruik van de generalisatie en de communicatie en welke factoren de meeste invloed hebben op de grootte van dat effect.

2. Theoretische context

Het onderzoek is geïnspireerd op de traditionele adventure games, waarbij een personage allerlei items moet vinden en moet gebruiken op allerlei obstakels om verder te komen in het Adventure. De motivatie om dit soort werelden te gebruiken voor een multi-agent learn probleem, is dat hierin sterke mate sprake is van modulariteit van problemen en algemene wetmatigheden in de manier waarop bepaalde problemen met behulp van bepaalde items kunnen worden opgelost. Dit zou er in principe toe moeten leiden dat het voor agenten zinnig is om gebruik te maken van door andere agenten verzamelde kennis over obstakels, om het eigen leerproces te versnellen.
Hiertoe hebben wij het plan opgevat om multi-agent reinforcement learning te doen over verschillende maar analoge werelden, waarbij de agenten kennis over de analogie van de werelden gebruiken om algemeen relevante kennis te communiceren met andere agenten. Op die manier worden de agenten in staat gesteld om het probleem sneller te leren en met name in de beginfase sneller een gemiddeld hogere reward moeten halen. Dit is een aanvulling op eerdere onderzoeken, waarbij ook multi-agent reinforcement wordt gedaan (Ming Tan ,"Multi Agent Reinforcement Learning: Independent vs. Cooperative Agents") maar dan op een voor iedere Agent gelijke wereld. Verwant aan Multi Agent Reinforcement Learning met verschillende werelden, zijn de verschillende onderzoeken naar Hierarchical Reinforcement Learning. Het implementeren van Hierarchical Reinforcement Learning op een universele manier is echter een groot probleem op zich en we hebben ons daar in ons onderzoek verder niet expliciet mee bezig gehouden. Onderwerpen zoals planning en zoeken zijn in principe ook relevant bij het oplossen van ons multi-agent leerprobleem. We hebben echter gekozen om zo weinig mogelijk expliciete kennis van de Agent over de wereld te veronderstellen, en om de geformuleerde problemen zoveel mogelijk met Reinforcement Learning technieken op te lossen. In dat kader hebben we wel een zekere vorm van planning geïmplementeerd in de vorm van Dyna-Q (Richard S.Sutton

and Andrew G. Barto, "Reinforcement Learning"), een Reinforcement Learning techniek die een model aanmaakt van de wereld, en dit gebruik om, off-line, ervaring te simuleren en op basis daarvan extra te leren.

2.1 Het spel



Figuur 1: Een voorbeeld van een spelwereld waarin de agent (linksboven) langs de obstakels vuur en rivier naar het einde (rechtsonder) moet zien te komen, gebruikmakend van de juiste gebruiksvoorwerpen.

Het spel bestaat uit een agent die zich door een doolhof moet zien te verplaatsen om bij de uitgang te komen. Onderweg komt hij items en obstakels tegen. De items kunnen hem helpen de obstakels op te lossen, zodat hij verder kan. Naast het feit dat de agent kan bewegen (links, rechts, naar beneden, naar boven), kan hij ook acties uitvoeren op het obstakel. Hij weet echter niet wat de beste actie is op een obstakel wat hij tegenkomt. Daarom heeft hij een lijst met mogelijke acties waaruit hij de beste kiest.

2.2 Reinforcement Learning en Dyna-Q

Met dit onderzoek hebben we gepoogd meer van reinforcement learning te weten te komen, en te onderzoeken wat de bottlenecks zijn bij een implementatie van deze aanpak van learning.
Bij Reinforcement Learning leert de agent een optimale Policy aan de hand van feedback ('reward') die hij krijgt op zijn acties, ofwel direkt ofwel indirekt, in de wereld waarin hij rondloopt. Het betreft een episodische leertaak, waarbij de agent begint op een willekeurig veld,

en eindigt als de agent de uitgang van het doolhof heeft bereikt. Elke zet levert een negatieve reward op, maar het bereiken van het eind een grote positieve reward. Het leren zou kunnen worden versneld door in het model rewards toe te kennen aan het oplossen van losse obstakels, maar het is mooier als de agent dit zelf oplost door in zijn eigen model de uiteindelijke reward te laten herverdelen over de obstakels die hij heeft opgelost om het doel te bereiken.
Q-learning in de platte vorm valt nog uit te breiden met Eligibility Traces en Reinforcement Planning / Dyna-Q; mits we de tijd ervoor hebben. Vooral de laatste methode, waarbij wordt gesimuleerd en geleerd met behulp van een incrementeel model van de wereld, kan het leren mogelijk enorm versnellen.

Dyna-Q is geinspireerd op het inzicht dat Reinforcement Learning in wezen niet zoveel verschilt van planning, als je planning ziet als leren op basis van een model van de wereld in plaats van op basis van directe interactie met de wereld. Dyna-Q implementeert deze planning in de vorm van Reinforcement Learning over experience gemaakt door een model, door tijdens de gewone Reinforcement Learning de effecten van acties bij te houden en deze aan het model toe te voegen in de vorm van (s,a, -> r,s') combinaties die aangeven welke reward en nieuwe state state-actie paren als gevolg hebben. In het geval van een niet-deterministische wereld wordt dit iets ingewikkelder, er moeten dan kansverdelingen worden bijgehouden over de mogelijke gevolgen van s-a paren in de vorm van r-s' paren en hun kans op voorkomen. Veel verandert dit het concept echter niet. Na iedere Q-learning stap worden de in het model opgeslagen (s,a, -> r,s') overgangscombinaties gebruikt om nieuwe experience te genereren, en deze met de Q-learning regel te gebruiken om de s,a-paren te updaten. In de simpelste implementatie wordt een n-aantal keer een willekeurige overgangscombinatie uit het model getrokken, en wordt de door de combinatie voorspelde nieuwe state en reward gebruikt om de Q-waarde van het s,a-paar van de combinatie te updaten volgens de Q-learning regel.
In combinatie met Q-learnig leidt dit tot het volgende algoritme:

```
Initialize Q(s,a) and Model(s,a) to empty sets
Do forever:
      // Q-learning gedeelte
      (a) s <- current(nonterminal) state
      (b) a  <- e-greedy(s,Q)
      (c) Execute action a; observe state s' and reward r
       // Normale Q-learning stap
      (d) Q(s,a) <- Q(s,a) + a[r + gamma*Max_a'Q(s',a') -Q(s,a)]

      // Dyna-Q gedeelte:
      // Update model, voeg (s,a, -> r,s')  combinatie toe
      (e) Model(s,a) <- s',r

      // Doe N-updates op basis van experience gegenereerd
     // door model
      (f) Repeat N times:
            s <- Randomly previously observed state
            a <- Randomly previously observed action
            s',r' <- Model(s,a)
            Q(s,a) <- Q(s,a) + a[r + gamma*Max_a'Q(s',a')- Q(s,a)]
```

Dyna-Q vormt een toevoeging op het bestaande Reinforcement Learning, maar alleen in zoverre dat het extra leert op basis van bestaande ervaring, het voegt niets toe aan de generalisatie. Dyna-Q kent nog allerlei varianten, waarbij bijvoorbeeld de keuze voor de geüpdate s-a paren niet

willekeurig is maar wordt gekozen op een manier waarbij de paren die de update het meest nodig hebben een grotere kans hebben om te worden geüpdate. Dit kan bijvoorbeeld worden gedaan door bij te houden welke s-a paren het laatst door Q-learning van waarde zijn verandert. Andere varianten behelzen het toekennen van extra subjectieve rewards aan states die lange tijd niet bezocht zijn, om te voorkomen dat het leeralgoritme vast blijft zitten in een locaal minimum wanneer de wereld verandert en een betere oplossing (e.g. kortere route) beschikbaar komt.

2.3 Generalisaties

Het doel is dat de agent generaliseert over locaties waar het obstakels betreft: een actie op een obstakel heeft naar verwachting altijd hetzelfde effect, los van op welke positie het obstakel zich bevind. De agent gebruikt dit gegeven om ervaring te kunnen toepassen wanneer hij hetzelfde obstakel opnieuw tegenkomt, en bij het uitwisselen van kennis met bevriende agenten over goede acties voor bepaalde obstakels. Dit laatste komt aan bod in paragraaf 2.4.
Het is de bedoeling dat de agent ervaring met het oplossen van obstakels generaliseert zodanig dat hij zich een bias op kan leggen voor nog ongeziene instanties van hetzelfde soort obstakel.

De implementatie van de generalisatie van ervaringen omtrend obstakels staat open voor verschillende benaderingen. De plaats waarop generalisatie toegepast wordt kan in het begin, als ervaring met een obstakel zich voordoet, zijn. Alternatief kan ook generalisatie pas toegepast worden als blijkt dat er behoefte aan is; als de agent een obstakel tegenkomt wat hij al eerder gezien heeft. Aan de andere kant, er is wellicht niet altijd verschil tussen, aangezien het vanaf het begin al duidelijk is waarover gegeneraliseerd zal kunnen worden.
Mogelijke aanpakken zijn:

- De agent houdt van alle obstakels die hij oplost Obstakel-Actie (OA) paren bij, waarin vermeld staan: een obstakel, de actie die het obstakel oplost en de gemiddelde totale reward van een succesvol afgerond spel als waarde.

- De agent gaat na, bij het vinden van een obstakel, welke van zijn reeds aangedane states allemaal zo'n obstakel hebben. Vervolgens neemt hij de actie waarvan de Q-waarde gemiddeld het hoogst is. De gemiddelde Q-waardes slaat hij op om in het vervolg er sneller bij te kunnen. Kortom: de generalisatie is de gemiddelde Q-waarde. Een voorwaarde hiervoor is wel dat alle acties op de states generiek, d.w.z. positieonafhankelijk, zijn; dit betekent in onze implementatie dat de move-actie van teruggaan generiek aangeduid moet worden met een actie als 'back'.

- Het bijhouden van een abstract policy; dit linkt de obstakels met de items. Het heeft als effect dat als een state van 'neutralize(vuur)' een hoge waarde krijgt, dit terugpropageert naar achtereenvolgend: 'use(brandblusser)', 'have(brandblusser)', 'find(brandblusser)' en 'pickUp(brandblusser)'.

- DynaQ/Model-based learning uitvoeren na het opvragen van een generalisatie. Hierbij krijgt de state waarin de actie die het obstakel oplost gedaan kan worden een grote waarde, en wordt dit teruggepropageerd. Dit verschilt van de standaard toepassing van Dyna-Q als leren op basis van bestaande ervaring, zoals hierboven beschreven. Het leren wordt ná het toevoegen van de gegeneraliseerde kennis wordt toegepast, waarbij de focus ligt op het zo snel mogelijk bereiken van de state waarbij de optimale actie op het betreffende obstakel

mogelijk is.

- Het toevoegen van subjectieve rewards. Hierbij worden extra immediate rewards gedefinieerd die de agent in de juiste richting 'sturen'.

We hebben ervoor gekozen om de eerste aanpak te implementeren, vanwege de haalbaarheid in het tijdsbestek wat ons gegeven is voor dit project.

Het probleem zit hem in het vinden van algemeen geldende kennis, dat wil zeggen in het vinden van goede generalisaties over het probleem. Dit automatisch doen zou in principe het mooist zijn, maar voor zover ons bekend bestaan er nog geen werkende methodes hiervoor in de context van reinforcement learning.
Daarom hebben we een probleemspecifieke methode gekozen, die gebruik maakt van het feit dat acties op obstakels in alle werelden hetzelfde effect hebben. De agenten gebruiken dit gegeven door wanneer ze een obstakel hebben opgelost te onthouden met welke actie ze dit hebben gedaan. Ze voegen dit toe aan hun kennisbank in de vorm van een OA-paar. In het geval van een actie op een obstakel die juist heel slecht is, bijvoorbeeld gebruik van benzine op een vuur, wordt juist een OA-paar met een hele negatieve score toegevoegd voor de betreffende actie. OA-paren worden door de agent gebruikt bij het kiezen voor acties. Ze overrulen daarbij de normale Q-learning epsilon-greedy actie-keuze procedure. Bovendien geeft de agent, nadat hij een OA-paar heeft proberen toe te passen prioriteit aan het vinden van het item wat nodig is voor het uitvoeren van de actie met de hoogste waarde. Hier is het een goede eerste test gebleken om te hardcoden dat de agent de brandblusser altijd moet oppakken de brandblusser altijd moet gebruiken op het obstakel vuur, en dit vergelijken met het geval waarin de agent dit niet verteld wordt.

2.4 Communicatie

Het is de bedoeling dat agenten de gegeneraliseerde ervaring kunnen uitwisselen. Dit betekent dat OA-paren van agent A toegevoegd worden aan die van agent B. Immers, iedere agent maakt standaard zijn eigen OA-paren aan, onafhankelijk van communicatie met andere agenten.
Hoewel Mult-agent in het meeste onderzoek inhoud dat verschillende agenten acteren binnen dezelfde wereld, hebben wij ervoor gekozen om de agenten in verschillende, maar parallelle, werelden te laten acteren. Deze werelden zijn qua configuratie verschillend (positie van uitgang, obstakels en items), maar qua gedrag analoog. Dit houd in dat het doven van een vuur in de ene wereld hetzelfde werkt als in de andere.
Bij de keuze van de vorm van communicatie zijn er verschillende mogelijkheden:

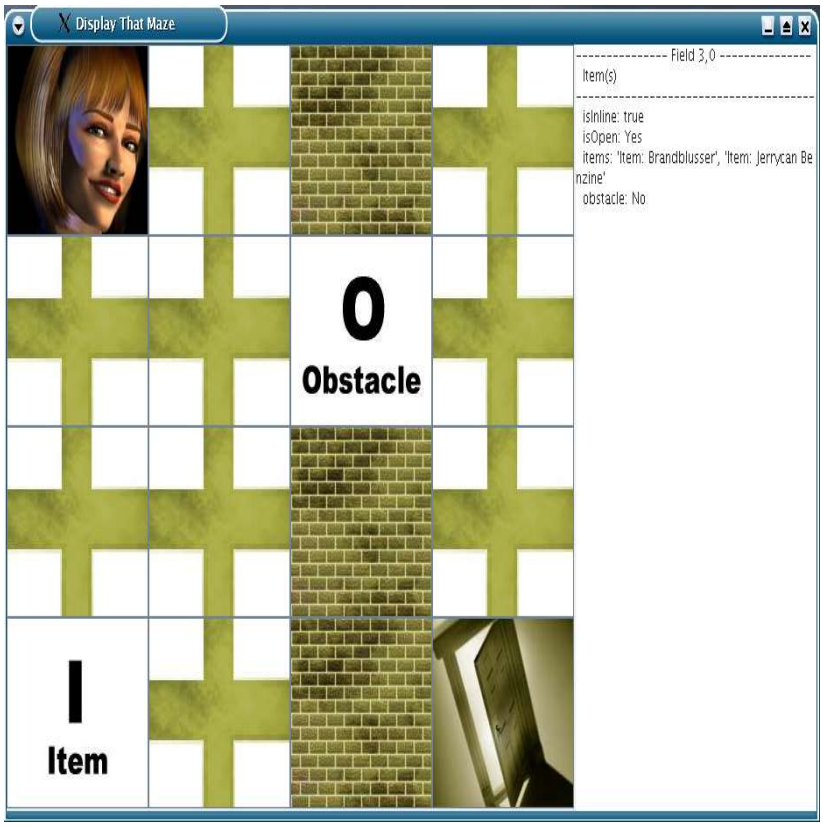
- Iedere keer wanneer een agent bij een obstakel aangekomen is waarvan hij niet zeker weet hoe hij het moet oplossen, vraagt hij advies van zijn mede agenten. Als antwoord krijgt hij relevante generalisaties terug die hij mee laat wegen in zijn actie-keuze.

- De agent communiceert zijn nieuw gevonden generalisaties zodra hij een episode heeft afgerond. Zo blijft de hoeveelheid communicatie beperkt. Deze optie hebben we dan ook gekozen voor onze implementatie.

In het geval van meerdere agenten wordt het interessant om geleerde generalisaties te communiceren met de andere agenten, om de snelheid van het leren te bevorderen. In het algemeen geld dat in het geval van analoge werelden alle kennis over het probleem die algemeen geldig is over de analoge werelden mogelijk communicatie waard is.

3 Implementatie

De agent beweegt zich voor in een adventure-achtige spelwereld, waarin hij als doel heeft om de uitgang te vinden. Daartoe zal hij sommige subquests (of obstakels) moeten oplossen.
Om op een makkelijke manier te controleren of een door ons gemaakte spelwereld ('Maze') wel klopt, hebben we een tool ontwikkeld (zie figuur 3.) waarmee spelwerelden zijn weer te geven, alsmede informatie over de eigenschappen van de velden, weergegeven in de balk aan de rechterkant.



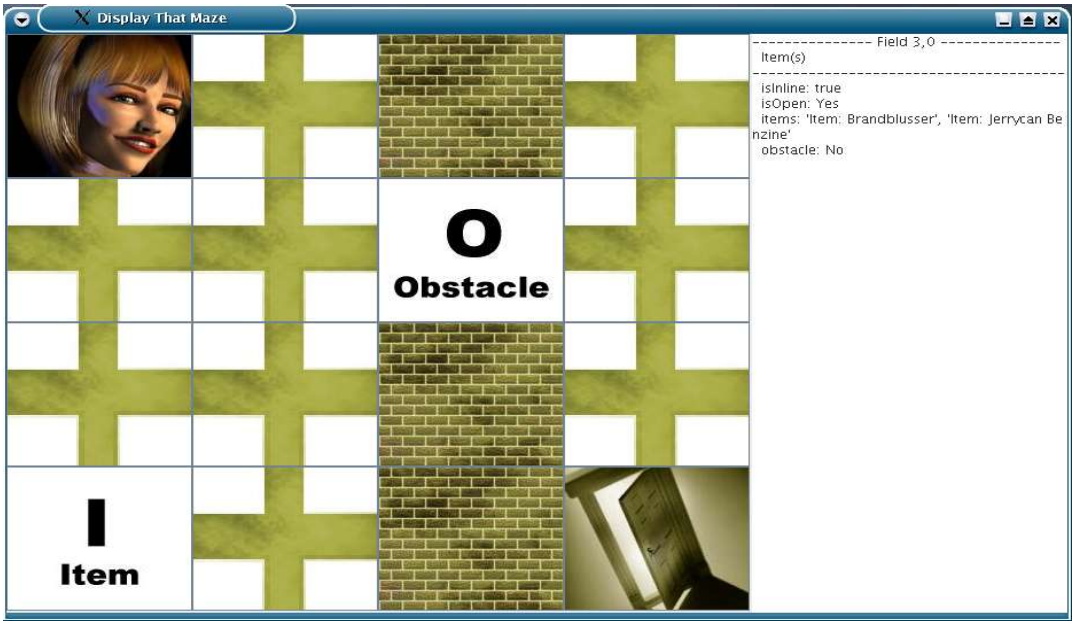
Figuur 3: Een voorbeeld van een spelwereld waarin de agent (linksboven) langs de obstakels vuur en rivier naar het einde (rechtsonder) moet zien te komen, gebruikmakend van de juiste gebruiksvoorwerpen

De spelwereld wordt geimplementeerd in de Game klasse. Dit is in feite de klasse die alle fysieke elementen (de agent en het doolhof) bevat. Het doolhof, de Maze klasse, bevat de velden in de vorm van een 2-dimentionaal array met Field objecten. Deze Field objecten, op hun beurt, bevatten de Items en de Obstakels, maar kunnen ook muur zijn, en dus ontoegankelijk voor de agent.

3.1 De spelwereld

De agent beweegt zich voor in een adventure-achtige spelwereld, waarin hij als doel heeft om de uitgang te vinden. Daartoe zal hij sommige subquests (of obstakels) moeten oplossen.

Om op een makkelijke manier te controleren of een door ons gemaakte spelwereld (‘Maze’) wel klopt, hebben we een applicatie ontwikkeld (zie figuur 3.)waarmee deze zijn weer te geven, alsmede informatie over de eigenschappen van de velden, aan de rechterkant.



Figuur 3: Een voorbeeld van een spelwereld waarin de agent (linksboven) langs de obstakels vuur en rivier naar het einde (rechtsonder) moet zien te komen, gebruikmakend van de juiste gebruiksvoorwerpen

3.1.1 Items en Obstakels

Iedere wereld bevat een agent, een eindpositie en een willekeurig aantal obstakels en items. De agent krijgt heeft de episode succesvol afgerond als hij de eindpositie bereikt, maar gaat dood als hij een verkeerd item op een obstakel gebruikt (zoals een jerrycan benzine op een vuur).

3.1.2 Acties

In iedere state (zie paragraaf 3.3) waarin de agent zich bevind wordt bepaald welke acties mogelijk zijn. Dit wordt gedaan aan de hand van de precodities die voor iedere actie gedefinieerd zijn. In het geval van een muur worden de bewegingsacties beperkt. In het geval dat de agent een obstakel tegen komt, is de enige beweeg actie degene om terug te gaan. Daarnaast kunnen nu de unilaterale acties, en, afhankelijk van of en hoeveel items je in je bezit hebt, contextuele acties uitgevoerd worden.

- Movements
  `up, down, left, right`
  Deze acties hebben invloed op de x en y waardes van de state. Bijvoorbeeld: `up` verplaatst je naar het vakje boven je huidige positie.

- Unilateral actions
  `jump, push, spit`
  Deze acties worden uitgevoerd op een obstakel. Dit kan er toe leiden dat het obstakel opgeheven wordt, maar kan er ook voor zorgen dat er niets gebeurd. In dat geval zul je een andere actie moeten proberen.

- Contextual actions
  `use( Item)`
  In het geval van een obstakel in de huidige state, kunnen de items die je bij je draagt toegepast worden op het obstakel. Als dit geen effect heeft zal je een andere actie moeten proberen, of op zoek moeten gaan naar andere items. Dit is de enige actie die een argument meekrijgt.

De agent beweegt zich voor in een adventure-achtige spelwereld, waarin hij als doel heeft om de uitgang te vinden. Daartoe zal hij sommige subquests (of obstakels) moeten oplossen. Subquests oplossen levert op zich ook al reward op, de agent wordt hiermee gestimuleerd om ervaring op te doen in het oplossen van obstakels, ookal zijn ze niet noodzakelijk om bij de uitgang te komen.

3.2 Agent en AgentState

De agent wordt gerepresenteerd als de huidige state van de agent en de opgebouwde policy. Deze klasse is in essentie alleen een semantische wrapper.
De informatie over de huidige toestand van de agent, zoals de positie, welke items hij bij zich draagt enzijn status, wordt opgeslagen in de AgentState.

3.3 Policy

Het policy van een agent bevat de SA-paren, SATransitions en OA-paren. Daarnaast implementeert de policy voornamelijk de methode chooseAction. Deze methode kiest de actie aan de hand van de lijst met mogelijke acties en de opgebouwde kennis opgeslagen in States en OA-paren. De SATransitions, die het interne model van de wereld vormen, worden alleen gebruikt bij gebruik van de Dyna-Q uitbreiding.

3.3.1 States, SA-paren en SATransitions

De (Agent)State representatie bestaat in de eerste plaats uit de positie van de agent (x- en y-cöordinaten). Indien de huidige positie een obstakel bevat, is het volgende argument het obstakel object. Daarnaast heeft een state de lijst met items, of voorwerpen, die de agent op dat moment met zich mee draagt. Als laatste wordt de lijst gegeven met items, of voorwerpen, die de agent nodig heeft voor het oplossen van obstakels die hij eerder is tegengekomen.
States worden met acties gecombineerd tot SA-paren, die tijdens Q-learning aan de policy worden toegevoegd of worden geüpdate als ze al aanwezig waren. SAParen bevatten een veld dat de Q-waarde van de state-actie combinatie vastlegd
Ten behoeve van Dyna-Q worden er tijdens het leren ook SATransitions toegevoegd aan de policy. SATransitions bevatten de oorspronkelijke state, de actie die genomen werd, de state die het resultaat was en de reward die de agent ervoor ontving.

3.3.2 OA-paren

Tijdens de episode wordt bijgehouden wanneer er obstakels worden opgelost, in de zin van dat ze uit de wereld verdwijnen als gevolg van een actie. Steeds als dit gebeurd wordt er een OA-paar (Obstacle-Action paar) aan de lijst met opgeloste obstakels toegevoegd. Als de episode is afgerond, en het einde van de maze levend is bereikt, wordt de lijst van OA-paren die de agent bijhoudt vernieuwd. OA-paren in de lijst van opgeloste obstakels die de agent nog niet in zijn lijst heeft staan worden toegevoegd als nieuw OA-pair, met als waarde de final reward die de agent aan het eind van de episode behaald heeft. In het geval van OA-paren die de agent wel al in zijn lijst heeft staan, wordt de waarde vernieuwd, door de oude waarden te middelen met de nieuwe waarde – de final reward van de laatste episode.
In de laatste betrouwbaardere versie, maken we ook gebruik van Item-Actie(IA)-paren. Deze worden gelijktijdig met de OA-paren aangemaakt en bevatten een hoge waarde voor het oppakken van een item wat tot het oplossen van het obstakel leidt, en een negatieve waarde voor het oppakken van een item wat dat niet doet, of tot de dood van de agent leidt.

Wanneer de agent informatie opvraagt over een obstakel wat hij tegenkomt in de vorm van een OA-paar, dan kijkt hij welke actie het beste resultaat geeft, en welk item hij daarvoor nodig heeft. Als hij dit item niet heeft, gaat hij hiernaar opzoek door het toe te voegen aan zijn ‘wishlist’. Hierbij krijgt het oppakken van het item prioriteit heeft over andere items en worden er zo nodig andere items achtergelaten om plaats te maken voor het wishlist-item in de inventory van de agent.
De OA-paren kunnen op twee verschillende manieren worden gebruikt. De meest logische, en ook allereerst geïmplementeerde manier, is om de waarden te gebruiken bij het kiezen van acties in states. Daarbij wordt de Q-waarde van de verschillende toegestane acties in een AgentState opgezocht, en vergeleken. Vervolgens wordt met een zekere kans 1-ε een van de beste acties/greedy acties (die met de hoogste Q-waarde) gekozen, en met overgebleven kans ε een van de minder goed acties/ exploratory acties . Bij gebruik van OA-paren wordt gekeken of er niet een OA-paar voor de desbetreffende actie aanwezig is. Is dit het geval, dan wordt de waarde van dit OA-paar opgezocht, en wordt als Q-waarde voor de actie het maximum van de gewone Q-waarde en de waarde van het OA-paar genomen. Analoog hieraan wordt in het geval van gebruik van IA-paren (in het geval van een actie op een Item), als er een passend IA-paar aanwezig is voor de gekozen actie, het maximum van de waarde van dit IA-paar en de gewonde Q-waarde voor de actie genomen.
Het precieze gebruik van de OA-paren en IA-paren bij het bepalen van waarden van acties en het uiteindelijk kiezen van acties kan nog worden gevarieerd, maar het globale idee blijft hetzelfde als bij de hierboven beschreven implementatie. Het idee van het gebruik van het OA- en IA-pair, is dat acties die in het verleden voor soortgelijke situaties goed bleken te zijn in de toekomst veel sneller zullen worden gekozen. Onze verwachting was, dat dit de behaalde resultaten tijdens training met Q-learning, en mogelijk ook de snelheid van convergentie, zou moeten kunnen bevorderen.

3.4 TrainQLearn

TrainQLearn bevat de methodes die Q-Learning en Dyna-Q implementeren. Q-learning is geïmplementeerd in de methode trainOneEpisode die Qlearning uitvoerd volgens het bij de

paragraaf over Dyna-Q beschreven standaardalgoritme. Het algoritme bevat een do-while loop die doorgaat met het kiezen van acties en uitvoeren van updates, net zolang totdat het einde van het spel is bereikt, de Agent is doodgegaan of een bepaald maximum redelijk aantal stappen wordt overschreden.

Onze implementatie van Q-learning is cruciaal afhankelijk van de methodes `legalActionsInAgentState(AgentState s)` uit Maze en `chooseAction ( Vector possibleActions, AgentState, agentState, boolean useGeneralizations)` uit OnlinePolicy. De eerste methode geeft een lijst van mogelijke acties in een pepaald state. De tweede methode gebruikt deze lijst van mogelijke acties om op basis van de policy een actie te kiezen (eventueel daarbij geholpen door generalisaties in de vorm van OA-/IAPairs).

TrainQLearning gebruikt vervolgens de methode `applyActionEffects(Action chosenAction, AgentState agentState)` uit Maze om de behaalde reward en de nieuwe AgentState te bepalen. Vervolgens worden deze reward en nieuwe AgentState gebruikt om de Q-learning update stap uit te voeren.

TrainQLearn gebruikt de methode `legalActionsInAgentState` eveneens bij het bepalen van de maximale actiewaarde voor de nieuwe state Max_a(newState,a). Voor alle mogelijke acties wordt de Q-waarde opgezocht in het bijbehorende SAPair in de tabel van de policy, en de maximale waarde wordt als Max_a(newState,a) geretourneerd. Voor mogelijke maar nog niet uitgevoerde acties een waarde van nul verondersteld. Deze laatste veronderstelling betekend dat we voorzichtig moeten zijn bij het gebruik van generalisaties. Als we generalisaties gebruiken die maken dat bepaalde acties nooit worden uitgevoerd in een bepaalde state, dan moeten we deze acties eigenlijk ook uit de set van mogelijke acties voor die state halen. Doen we dit niet, dan krijgen we de onwenselijke situatie dat de waarde van Max_a(AgentState,a) altijd minimaal nul zal blijven, omdat een bepaalde actie wel mogelijk wordt geacht maar nooit wordt uitgevoerd. Dit zou de convergentie van Q-learning potentieel kunnen verstoren.

Terugkomend op het algoritme. Nadat reward, de niewe state newState en de maximale Q-waarde voor de nieuwe state is bepaald, wordt de Q-learning update stap uitgevoerd.

Vervolgens begint de loop opnieuw, met als nieuwe AgentState newState.

TrainQLearn bevat een apparte methode die Dyna-Q implementeert. Aangegeven kan worden hoeveel Dyna-Q stappen er na iedere episode moeten worden uitgevoerd. De werking van Dyna-Q is zoals eerder beschreven. Dyna-Q maakt gebruik van SATransitions die worden toegevoegd aan een lijst tijdens het trainen in de gewone Q-learning loop. In die loop worden ook de OA-paren toegevoegd aan de policy.

3.5 MultiTrain

Bij het implementeren van meerdere agenten hebben we een klasse MultiTrain gemaakt. Het is een multi-agent wrapper die de functionaliteiten van TrainQLearn aanroept en aanpast en biedt ondersteuning voor meerdere pools van gelijksoortige agenten die parallel episodes leren. Tevens communiceert de MultiTrain klasse de nieuw gevonden oplossingen van iedere agent naar de overige agenten.

Bij het implementeren van meerdere agenten hebben we een klasse MultiTrain gemaakt. Het is een multi-agent wrapper die de functionaliteiten van TrainQLearn aanroept en ondersteuning biedt voor meerdere pools van gelijksoortige agenten die parallel episodes doorlopen. Tevens communiceert de MultiTrain klasse de nieuw gevonden oplossingen van iedere agent naar de overige agenten. MultiTrain is ook verantwoordelijk voor het bijhouden en wegschrijven van behaalde resultaten, die worden weggeschreven naar csv-files. In Excel kunnen deze vervolgens worden ingeladen, om er grafieken mee te genereren.

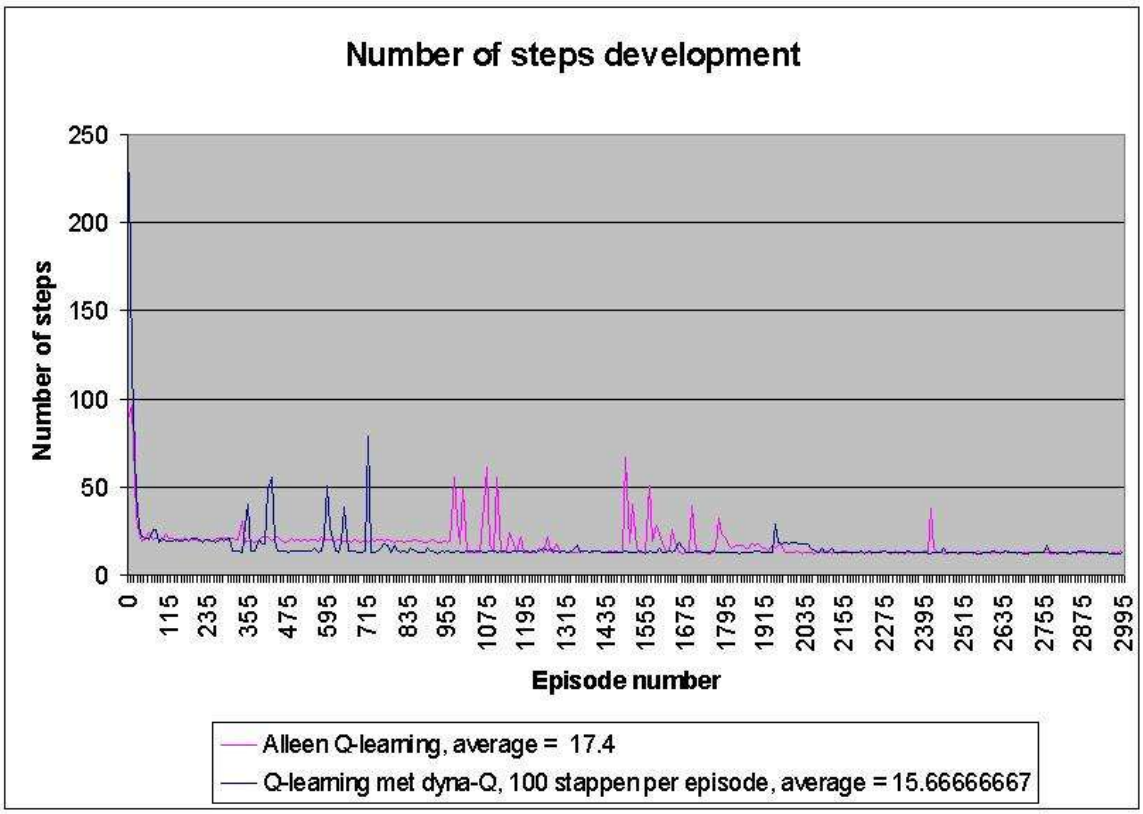
4 Resultaten

De versie waar we aan gewerkt hebben in de laatste paar dagen functioneert nog niet helemaal naar onze tevredenheid, daarom hebben we ook scores genomen van een eerdere versie van ons programma. Hierin wordt de wishlist nog niet gebruikt, aangezien de agent zoveel items met zich mee kan dragen als hij wil, en worden IA-paren gebruikt om de voorkeur voor bepaalde items aan te geven.

4.1.1 Dyna-Q versus Geen Dyna-Q

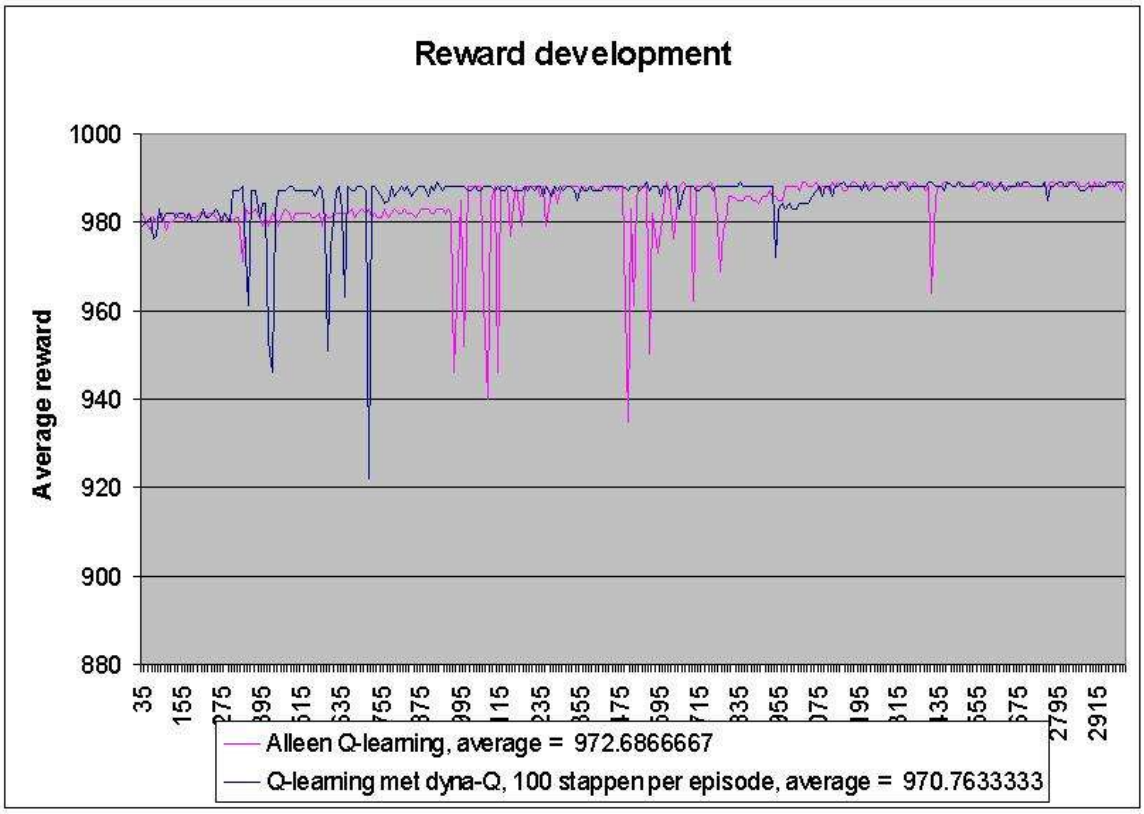
Het gebruik van Dyna-Q bied zoals verwacht inderdaad een aanzienlijk voordeel in de zin van snellere convergentie, en daarmee sneller betere resultaten. Teveel of te vroeg gebruik van Dyna-Q blijkt echter weer slechte resultaten te kunnen opleveren, doordat een nog onoptimale policy mogelijk teveel wordt ingesleten en de optimale policy daardoor later of zelfs niet meer kan worden gevonden

De volgende resultaten laten zowel de voordelen van nadelen zien, als ook de mogelijk nadelige consequenties van een te groot gebruik ervan tijdens het begin van het trainen.



Figuur 4: Hoeveelheid stappen voor iedere episode. Hier komt helder naar voren dat Dyna-Q sneller de optimale policy gevonden heeft. Dit resulteert in een lagere average number of steps.

Dyna-Q wordt gebruikt om na iedere episode extra een aantal cycles offline te leren. Bovenstaande grafiek maakt duidelijk dat de Dyna-Q learning agent eerder de optimale policy gevonden heeft – het gewenste effect van de extra offline learning cycles.



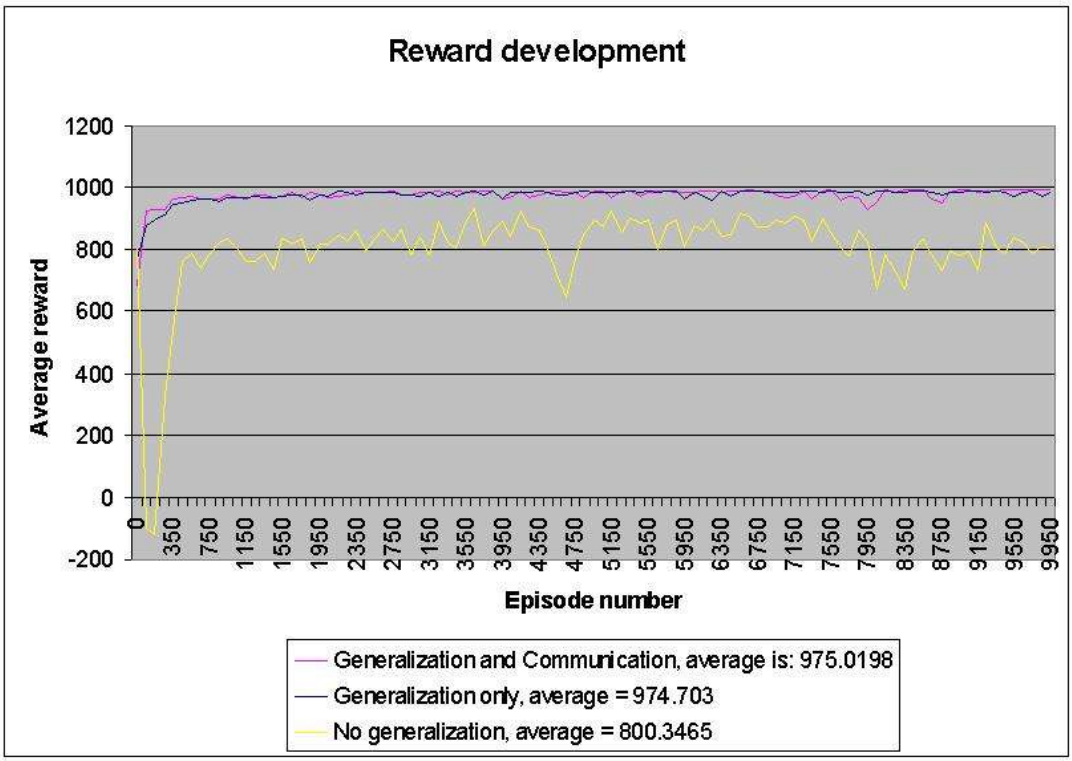
Figuur 5: De ontwikkeling die de reward doormaakt tijdens het leerproces. Horizontaal is het aantal episodes uitgezet, verticaal de totale reward van de episode.

De resultaten van de totale rewards voor Dyna-Q vallen enigszins tegen, met resultaten die, zeker in het begin, minder goed zijn dan de normale Q-learner. Dit is te verklaren doordat hij in het begin te veel specialiseert op Q-waardes die nog niet goed genoeg zijn. Naast deze grondige mishits vind hij sneller de optimale policy dan de normal Q-learner.
Dit probleem zou bijvoorbeeld kunnen worden opgelost met een geavanceerdere versie van Dyna-Q die het aantal Dyna-Q stappen bijvoorbeeld afhankelijk maakt van het aantal SATransitions, zodat in het begin minder updates worden doorgevoerd en naar het einde toe meer.

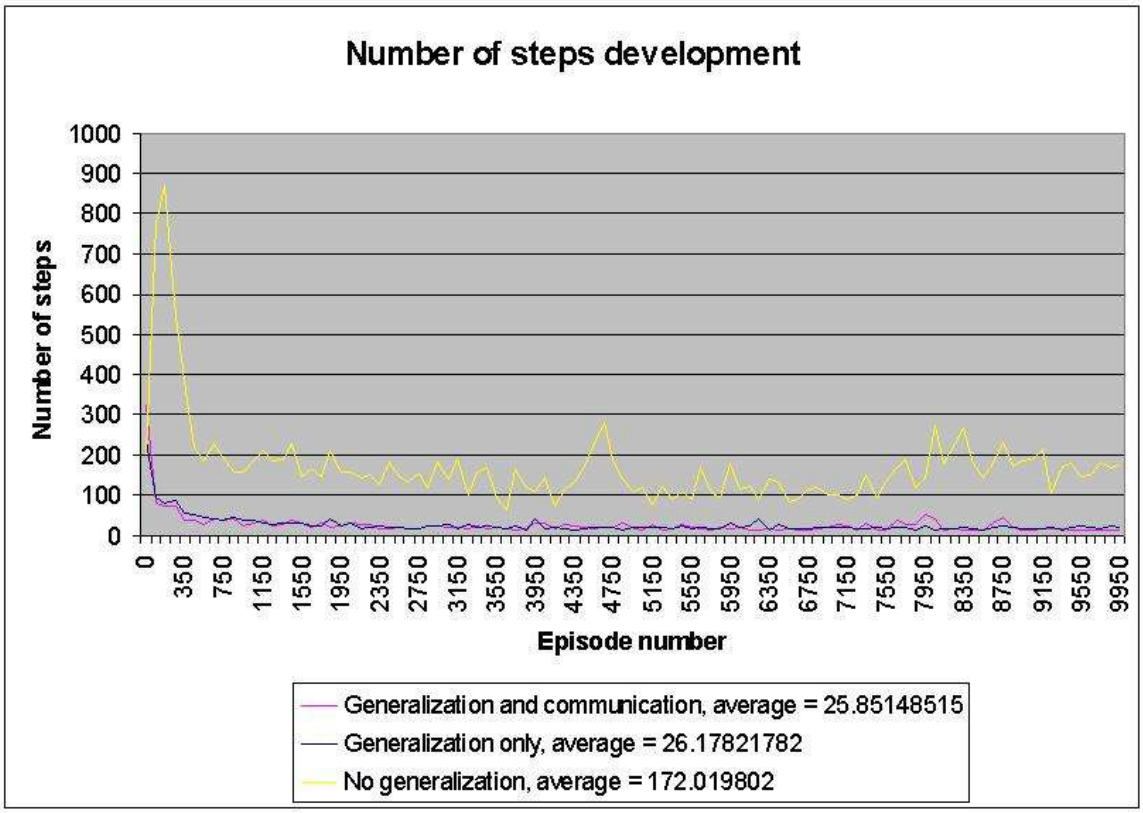
4.1.2 Generalisatie en Communicatie

4.1.2.1 Stabiele versie

Resultaten voor 10.000 episodes, 4 verschillende obstakels, per obstakel 100 verschillende obstakel instanties, en een epsilon afname factor van 0.9996.



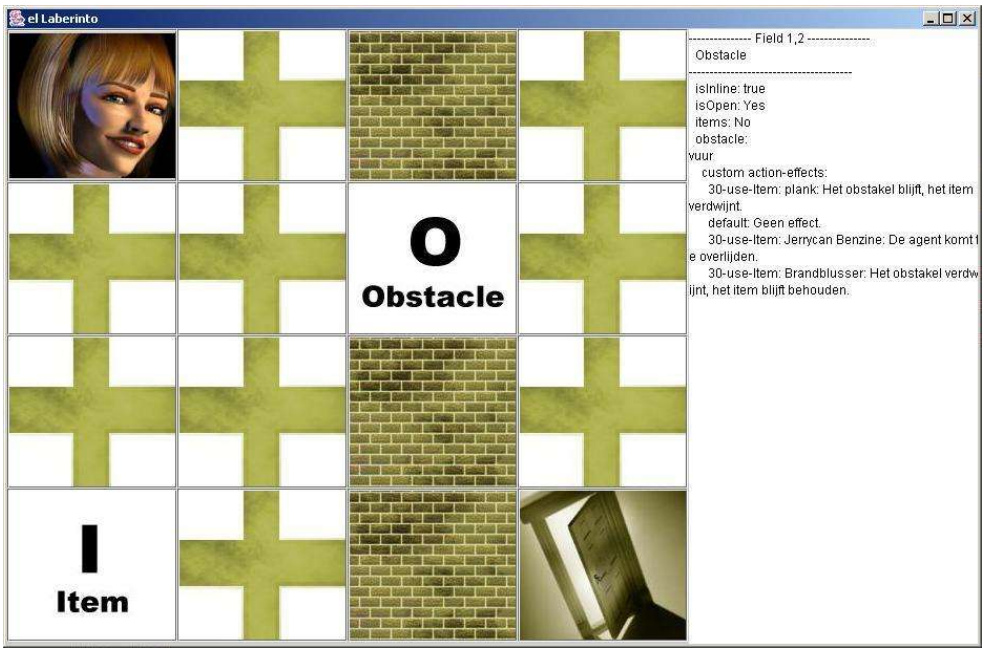
Reward ontwikkeling. Het is duidelijk dat het gebruik van generalisaties ervoor zorgt dat de agent convergeert naar het optimale policy. Echter, het verschil tussen communicatie en alleen generalisatie is verwaarloosbaar klein.

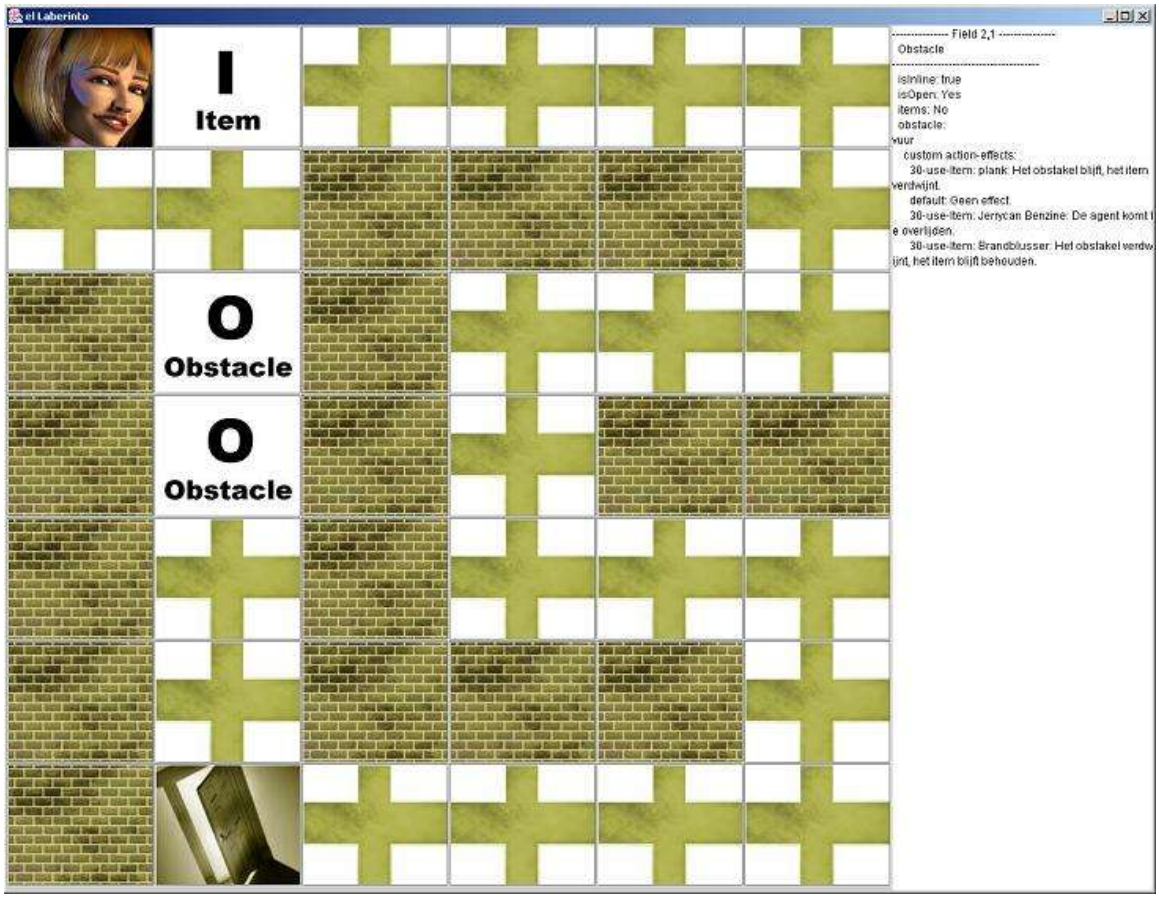


De ontwikkeling van de hoeveelheid stappen per episode laat een soortgelijk beeld zien; de generalisatie maakt veel verschil met de normale Q-learner. Echter de communicerende agenten doen het niet veel beter dan de agenten die alleen generaliseren.

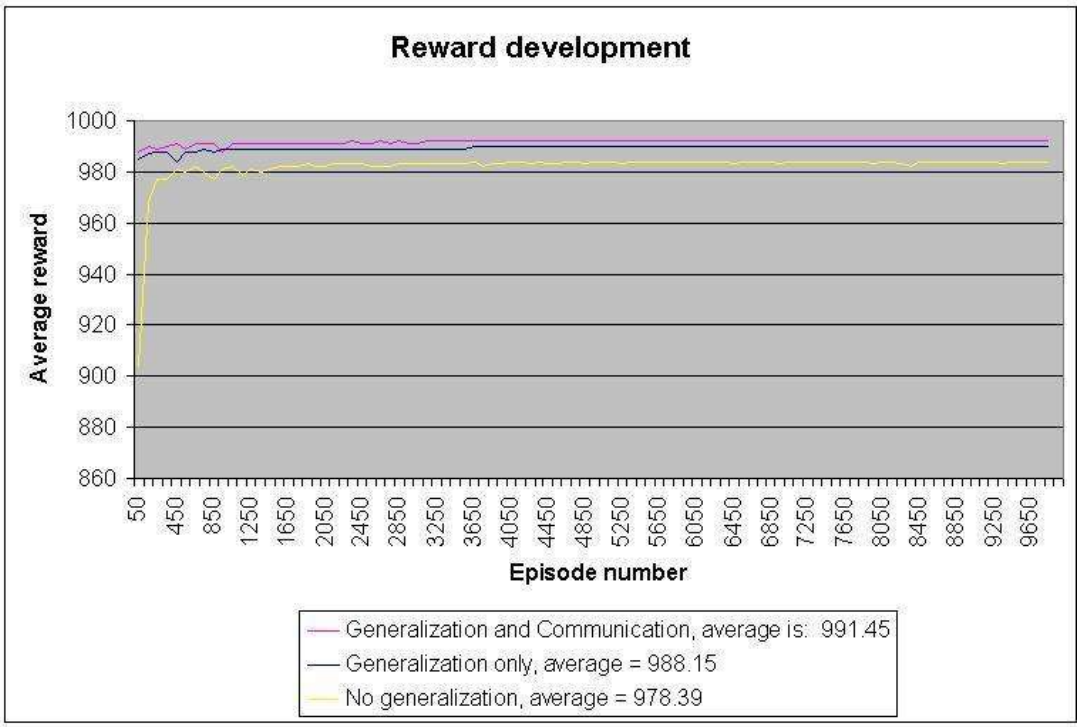
4.1.2.1 Laatste implementatie

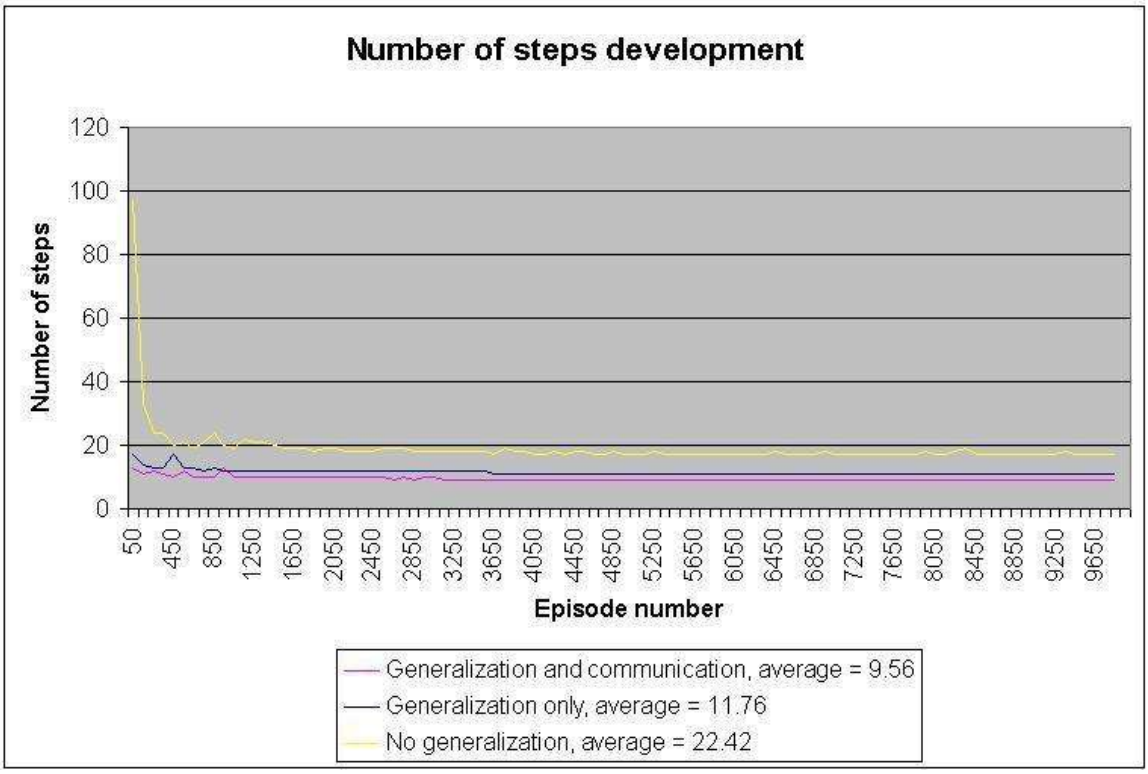
Laatste implementatie; wishlist, geen IA-paren en een beperkte inventory.
Twee verschillende Agents: Agent1 en Agent2. Agent1 heeft simpele maze Maze1 als wereld, met een obstakel: vuur. Agent2 heeft een wereld met een korte weg met twee obstakels (twee vuren), en een langere weg zonder obstakels. , maximaal aantal items in de inventory 1, gebruik wishlist en generalisaties.
Met gebruik van generalisatie leert Agent2 de kortere weg, met het oplossen van de obstakels. Zonder communicatie blijft hij steken in de lange weg zonder obstakels.


el Laberinto
O
Obstacle
I
Item
---------- Field 1,2 ----------
Obstacle
isInline: true
isOpen: Yes
items: No
obstacle:
vuur
custom action-effects:
30-use-Item: plank: Het obstakel blijft, het item verdwijnt.
default: Geen effect.
30-use-Item: Jerrycan Benzine: De agent komt te overlijden.
30-use-Item: Brandblusser: Het obstakel verdwijnt, het item blijft behouden.


el Laberinto
I
Item
O
Obstacle
O
Obstacle
---------- Field 2,1 ----------
Obstacle
isInline: true
isOpen: Yes
items: No
obstacle:
vuur
custom action-effects:
30-use-Item: plank: Het obstakel blijft, het item verdwijnt.
default: Geen effect.
30-use-Item: Jerrycan Benzine: De agent komt te overlijden.
30-use-Item: Brandblusser: Het obstakel verdwijnt, het item blijft behouden.

Figuur 6:

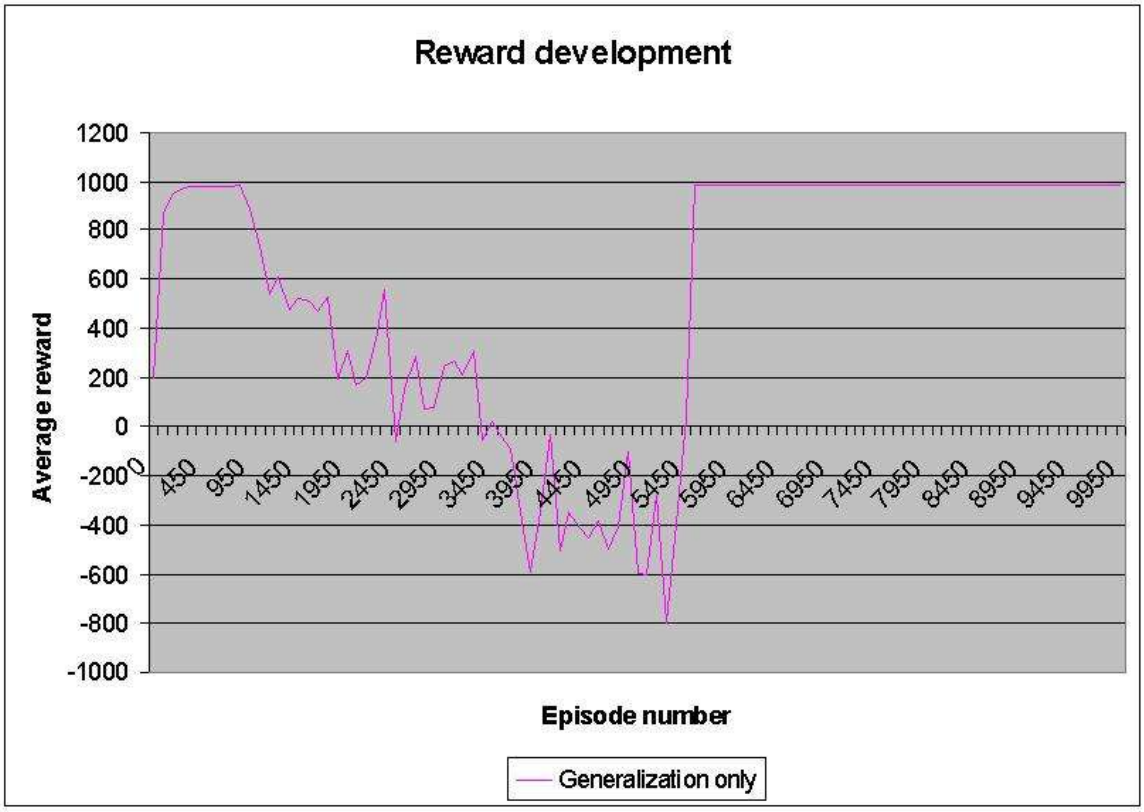


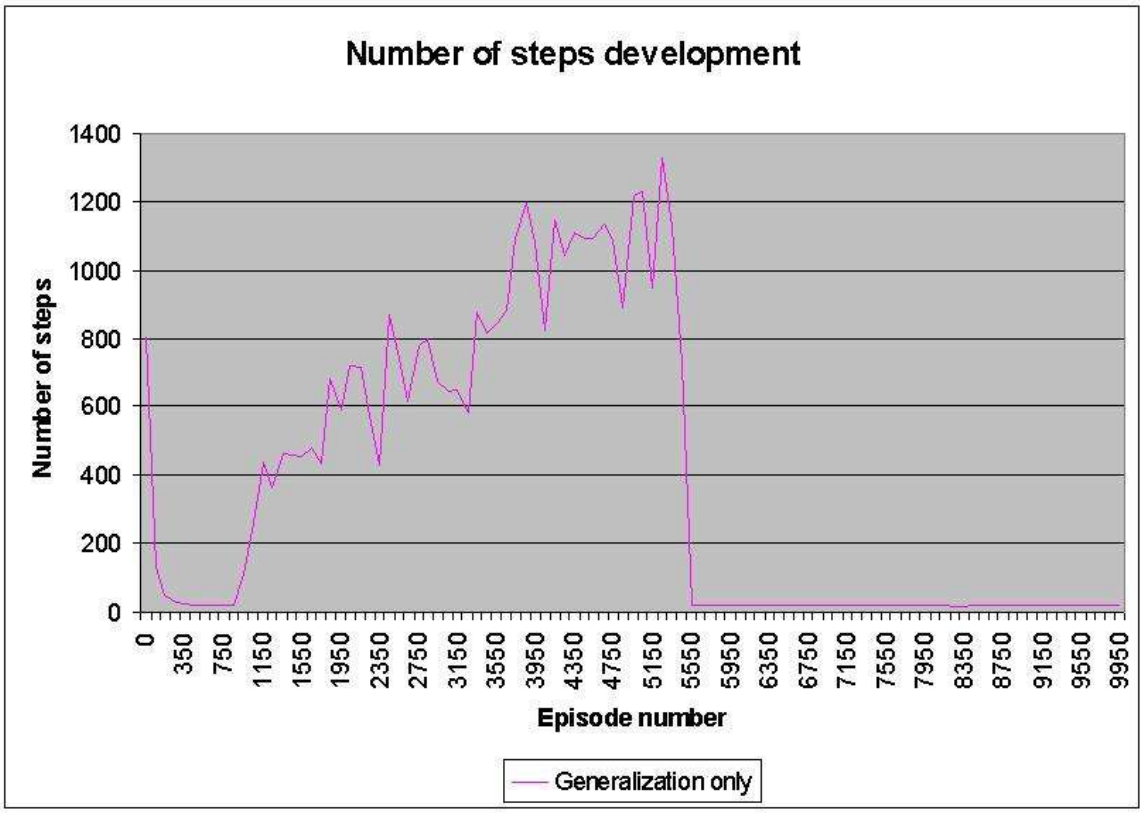


De communicerende Agent presteert het beste, hij leert de kortere weg het snelst en haalt de meeste reward. De niet communicerende Agent leert de kortere weg ook, dit komt doordat hij als

hij een van beide vuren oplost en daarna via de langere weg de maze verlaat, hij toch leert hoe je vuur moet oplossen, en dit vervolgens kan gaan gebruiken om beide vuren op te lossen. De niet generaliserende Agent leert de kortere route nooit, en heeft relatief veel slechtere scores.

Laatste implementatie; wishlist, geen IA-paren en een beperkte inventory.
2 verschillende random obstakels (2 instanties), maximaal aantal items in de inventory 1, gebruik wishlist en generalisaties





De agent leert uiteindelijk optimale policy, die bestaat uit eerst naar het obstakel lopen om te kijken wat het is, en daarna terug te lopen om het juiste item op te halen. Voordat dit gebeurd wordt het resultaat echter eerst steeds slechter. Dit lijkt te komen doordat de Agent in eerste instantie zich steeds verder specialiseerd op het oplossen van een van de twee obstakels, ten koste van het andere obstakel. Pas als de policy omslaat, en de Agent uitvind dat het beter is om altijd eerst naar het obstakel toe te lopen, kan hij de optimale policy leren. Waarom dit zo enorm spontaan gebeurd en de omslag zo groot is, is ons niet geheel duidelijk.

4.2 Analyse en Conclusie

Uit de resultaten komt duidelijk naar voren dat generalisaties een grote toegevoegde waarde vormen voor de performance van de agent. Zeker met een random component tussen iedere episode waarbij het soort obstakel op een plek wordt veranderd is het verschil met agenten zonder generalisaties duidelijk aantoonbaar: de generaliserende agent ontwikkelt een policy die convergeert, terwijl de normale agent niet convergeert, omdat de wereld steeds anders wordt.
Minder eenduidig zijn de resultaten van communicerende agenten. Alhoewel communicatie wel resultaten oplevert die iets beter zijn dan generalisatie an sich, hebben we nog niet goed kunnen aantonen dat communicatie in sommige situaties een uitkomst kan bieden.
Een hypothetische situatie is dat er een agent is die de oplossingen voor obstakels heel snel kan afleiden (omdat hij een heel kleine spelwereld heeft). De informatie van deze agent is van grote waarde voor collega's die in veel grotere spelwerelden rondlopen waarin het oplossen van hetzelfde obstakel veel langer duurt (omdat er meer mogelijke states zijn).

5 Toekomstig onderzoek

In het project wat wij de afgelopen maand gedaan hebben, hebben we meer bedacht dan dat we in de praktijk hebben kunnen brengen. Hier volgt een aantal van deze ideëen en suggesties:

De implementatie van Hierarchical Reinforcement Learning zou betekenen dat er geen expliciet policy meer gedefinieerd hoeft te worden voor obstakels, bepaalde configuraties zouden automatisch als obstakel worden geïdentificeerd en in gegeneraliseerde vorm gecommuniceerd worden naar andere agenten.

Naast de aanpak van de generalisatie zelf, is wat de agent er vervolgens mee kan doen een net zo interessant probleem. De wetenschap dat hij voor het oplossen van een bepaald soort obstakel een bepaald item nodig heeft, suggereert dat een vorm van reasoning/planning hier op zijn plaats zou zijn. Idealiter zou de agent, als hij een bepaald item nodig heeft, eerst controleren of hij dat in zijn inventory heeft, anders wordt zijn nieuwe doel om het te gaan halen. Nu kan het zijn dat de agent het item dat hij nodig heeft al ergens onderweg heeft zien liggen; in dat geval kan hij een pad plannen terug naar het betreffende veld. Als hij niet weet waar hij het item vandaan kan halen, dan zal hij moeten plannen hoe hij het beste ernaar kan zoeken. Daarnaast moet hij, zodra hij het item heeft gevonden, zijn weg weer terug naar het obstakel vinden en de betreffende actie uitvoeren. Een aanpak voor dit planning probleem kan zijn om goal-based Dyna-Q policy optimalisation toe te passen, dat wil zeggen Dyna-Q/Model-based learning uitvoeren na het opvragen van een generalisatie. Hierbij krijgt de state waarin de actie die het obstakel oplost gedaan kan worden een grote waarde, en wordt dit teruggepropageert. Dit verschilt van de standaard toepassing van Dyna-Q als leren op basis van bestaande ervaring, zoals hierboven beschreven. Het leren wordt ná het toevoegen van de gegeneraliseerde kennis wordt toegepast, waarbij de focus ligt op het zo snel mogelijk bereiken van de state waarbij de optimale actie op het betreffende obstakel mogelijk is.

Een andere manier om de OA- en IA-paren te gebruiken, is om de waarden van de OA- en IA-paren ook bij het daadwerkelijke Q-learning zelf te gebruiken, dat wil zeggen bij het updaten van acties. Een eerste manier om dit te doen, is om in het geval van nieuwe toestanden waarvoor een OA-/IA-paar bekend is, deze waarde te gebruiken als default waarde voor de nieuwe toestand die volgt na het uitvoeren van de in het OA-/IA-paar voorgeschreven actie (in plaats van een default waarde van nul). Dit heeft als gevolg dat de toestanden die eindigen in het uitvoeren van een gunstige actie naar verwachting sneller hoge waarden zullen krijgen.
Een simpelere, en wellicht ook zelfs effectievere manier om dit te implementeren, is om subjectieve rewards toe te kennen aan acties die overeenkomen met positieve OA-/IA-paren. Deze subjectieve reward komt dan nog eens bovenop de gewone reward van de actie. Onze verwachtingen voor deze tweede manier van gebruik van OA-/IA-paren waren gemengd. Aan de ene kant zou het het leren kunnen versnellen, aan de andere kant zou het introduceren van oneigenlijke rewards het convergeren naar de optimale policy wel eens kunnen verstoren.

Ondanks de beperkte tijd die we nog hadden, hebben we de laatste paar dagen geprobeerd het probleem interessanter te maken door het aantal items wat de agent mee kan dragen te voorzien van een limiet. Om de agent te helpen de juiste items mee te nemen hebben we de wishlist geïmplementeerd.

Deze nieuwe implementatie lijkt het probleem voor Q-learning weer veel moeilijker te maken, en hoewel de implementatie van de wishlist en het gebruik ervan werkend is, geeft deze geen goede resultaten. Mogelijk is het aantal episodes (100000) nog te klein, maar een waarschijnlijker

probleem is dat de agent het moeilijk kan leren omdat er twee strategieën zijn die onderling sterk concurreren en het probleem moeilijk maken.
De eerste strategie is het meteen oppakken van een item, en als dit het goede item is voor het random obstakel, dan het obstakel hiermee meteen oplossen en doorlopen naar het einde. De andere strategie, waarvan je zou willen dat de agent die leerde, is om eerst naar het obstakel te lopen, te kijken wat het is, dan terug te lopen naar de items, het juiste item op te pakken, terug te lopen naar het obstakel, het obstakel op te lossen, en door te lopen naar het einde van het doolhof. De agent blijkt deze tweede strategie echter niet te leren. In plaats daarvan leert hij een strategie die optimaal is voor een van de random mazes (voor een specifiek obstakel) en voor alle andere obstakels slecht. Als hij geluk heeft dan krijgt hij de juiste maze, en lost hij deze zo in het minimale aantal stappen op. In de meeste gevallen echter zal het obstakel verschillen en zal de agent zijn strategie ontoereikend zijn.
Deze nieuwe versie geeft helaas nog niet erg robuuste resultaten, met convergentie van de policy slechts op een beperkt domein van werelden. Toekomstig onderzoek zou er uit kunnen bestaan deze nieuwe versie als uitgangspunt te nemen en aan te passen zodat hij goed convergeert.

6 Literatuur

"Multi Agent Reinforcement Learning: Independent vs. Cooperative Agents", Tan, 1993
"Reinforcement Learning An Introduction", Sutton & Barto, 1998
"The MAXQ Method for Hierarchical Reinforcement Learning", Dietterich, 1998