Självlärande brädspelare

EMRE BERGE ERGENEKON
och ANTON JONSSON

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2010

Självlärande brädspelare

E M R E B E R G E E R G E N E K O N
o c h A N T O N J O N S S O N

Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik
Kungliga Tekniska Högskolan år 2010
Handledare på CSC var Mads Dam
Examinator var Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/
berge_ergenekon_emre_OCH_jonsson_anton_K10043.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Sammanfattning

Belöningsbaserad inlärning eller BBI är en av flera maskininlärningsmetoder. BBI kan användas för att lära en dator att spela spelet tre-i-rad. Den lärande delen kallas för en *agent*, pjäsernas aktuella placering på brädet kallas för *tillstånd* och ett drag kallas för en *handling*. 1992 kom en viktig implementation kallat Q-learning vilket är den algoritm som använts.

Rapporten undersöker hur Q-learning algoritmen fungerar, implementerar den för att lära datorn att spela det enkla brädspelet tre-i-rad samt analyserar hur olika parametrar påverkar inlärningsprocessen.

Rapporten inleds med en kortare sammanfattning av bakgrunden till BBI och Q-learning för att sedan gå vidare till att beskriva underbyggande teori för Q-learning algoritmen. Efter detta kommer avsnitt som beskriver implementation, analys samt slutsatser.

Vår implementation använder tre parametrar, alfa α, gamma γ och epsilon ε. Vi har kommit fram till att för vår implementation av tre-i-rad leder följande värden till stabil inlärning: $\alpha = 0.2$, $\gamma = 0.8$ och $\varepsilon = 0.1$.

Abstract

Reinforcement learning, RL, is one of many machine learning techniques used today. RL can be implemented to teach a computer to play a game of tic-tac-toe. The learning game player is called the *agent*, the appearance of a specific game play is called the *state* and a move is called an *action*. In 1992 an algorithm called Q-learning was created, the same algorithm is used in this document.

This document contains research on how Q-learning works, an implementation of the algorithm and analysis on how different parameters affect the result.

The beginning of this document contains background information on RL and Q-learning. Later on the Q-learning algorithm is described in more detail. The final chapter contains an analysis, conclusion and discussion regarding the results obtained with different parameter values.

Our implementation uses three parameters, alpha α, gamma γ and epsilon ε. Our results lead to the conclusion that the following parameter values lead to stable learning: $\alpha = 0.2$, $\gamma = 0.8$ and $\varepsilon = 0.1$.

Förord

Denna kandidatuppsats har lästs vårterminen 2010 under namnet DD143x vid CSC/KTH. Först skulle vi vilja tacka vår handledare Mads Dam för stöttning och vägledning igenom detta arbete. Ett tack ges också till Johan Boye som hjälpte oss utveckla vårt resonemang och föra arbetet vidare. Vi började med att dela upp arbete så lika som möjligt men riktade senare in oss på lite olika saker. Emre har varit mer inriktad på programmeringen och den java implementation som gjorts för att analysera Q-learing algoritmen och Anton har jobbat mer med rapportens olika delar och dess struktur.

Innehållsförteckning

Beteckningar

Belöningsbaserad inlärning (BBI), eng. Reinforcement learning (RL).

Inledning

När man tänker på den mänskliga inlärningsprocessen är det nog många som menar att vi människor lär oss via interaktion med vår omgivning. Detta är tydligt när man tänker sig små barn som utforskar sin omgivning. Genom direkt respons av smak, lukt eller känsel skaffas information som sedan kan användas för att göra ”bättre” val. Responsen från omgivningen leder till förståelse om orsak och verkan, om konsekvenser av handlingar. Ofta är vi medvetna om hur vår omgivning reagerar på det vi gör och försöker påverka vad som händer genom vårt beteende. Lärande via interaktion med vår omgivning är en fundamental idé för många teorier om intelligens och lärande.

Inom datorsammanhang kallas själva inlärningsprocessen för maskininlärning och är ett stort och som all forskning ständigt växande område. Brädspel är ett typiskt implementationsområde för maskininlärning. Några exempel är det simpla tre-i-rad (eng. Tic-Tac-Toe), schack, Gammon (Gerald Tesauro, 1995) med flera. Många maskininlärningsmetoder har utvecklats, undersökts och diskuterats då forskare arbetat med dessa problem.

Under 1990-talet blev en maskininlärnings metod som kallas belöningsbaserad inlärning mycket populär (Barto, 1998). Senare föreslogs en viktig implementation kallad Q-learning, algoritmen bevisades konvergera vilket gör den klart kraftfull (Watkins & Dayan, 1992). Då Q-learning tillämpas för ett brädspel behövs endast information om sluttillstånden, alltså tillstånden då en av spelarna vunnit, förlorat eller spelat lika.

Denna rapport har för avsikt att beskriva vår analys och implementation av Q-learning för brädspelet tre-i-rad.

Syfte / Problemformulering

Att undersöka hur Q-learning algoritmen fungerar, implementera den för att lära datorn att spela ett enklare brädspel (tre-i-rad, luffarschack eller liknande), samt analysera hur olika parametrar påverkar inlärningsprocessen.

Bakgrund

Avsnittet går kort igenom grundläggande historia bakom uppkomsten av belöningsbaserad inlärning och därmed bakgrunden till Q-learning.

Inledning

Till att börja med bör det nämnas att uppkomsten av belöningsbaserad inlärning (BBI) har varit en lång och till viss det snårig historia. Det har gjorts många publikationer av flera olika forskare relaterat till BBI. Däremot har flertalet inte konsekvent behandlat BBI utan, under en tid runt 1960-talet (Barto, 1998, s. kap: 1.6), även offentliggjort publikationer som per definition egentligen behandlat övervakad inlärning. Att beskriva bakgrunden till BBI är i sig ett mycket omfattande arbete och detta avsnitt kommer därför endas att lätt vägleda läsaren fram till state of the art för Q-learning. En exakt redogörelse för all bakomliggande forskning inom området BBI är utanför det här

arbetets omfattning men en mycket bra och relativt djupgående beskrivning av BBIs historia finns att läsa i kapitel 1.6 i (Barto, 1998).

Uppkomst och Bakgrund

BBIs uppkomst kan huvudsakligen delas in i två delområden som senare kom att sammanfalla till BBI som man definierar den idag. Den ena delen behandlar inlärning via trial-and-error. Den uppkom från början när man studerade djurs inlärningsprocess och har anknytningar till de tidigaste arbetena rörande artificiell intelligens. Den andra delen handlar om hur man med hjälp av värde funktioner (eng. value functions) och dynamisk programmering löser optimal Kontroll (eng. optimal control) problem. Trots att de två delområdena är klart olika finns det undantag som sammanfaller i ett tredje delområde kallat temporal-difference (TD) metoder. Dessa tre delområden sammanföll runt 1980-talet senare hälft till det som idag klassas som BBI.

Trial-and-error

Detta spår började inom psykologin där belöningsteorier är vanliga. Grundidén kan sammanfattas av att om positiv respektive negativ respons (välbehag resp. lidande) är resultatet av en viss, av flera möjliga, handlingar i samma situation ökar respektive minskar sannolikheten för att samma handling utförs nästa gång samma situation uppstår. Barto och Sutton menar att Edward Thorndike förmodligen var den första att framlägga detta och han kallade teorin för ”Law of Effect” (fritt översatt, ”handling och verkan”) (Barto, 1998). Man kan beskriva ”Law of Effect” som ett sätt att kombinera sökning och memorering. Sökning sker genom att för en viss situation välja bland olika möjliga handlingar och memorering sker sedan genom att komma ihåg vilken eller vilka handlingar som fungerade bäst. Handling och konsekvens associeras därmed till den situation där de gav bäst resultat. Att kombinera sökning och memorering till en viss situation är centralt inom BBI. Ordet belöning och belöningsbaserad inlärning användes för första gången i ingenjörsmässig litteratur på 1960-talen.

1960 publicerade Minsky en uppsats som till mångt och mycket diskuterade flertalet relevanta frågeställningar relaterade till BBI, även vad han kallade ”the *credit assignment problem”*. Hur fördelar man belöning över många handlingar som kan ha varit upphov till belöningen? De metoder som förknippas med BBI försöker till stor del ge svaret på denna fråga.

1961 och 1963 beskrev Donald Michie ett simpelt trial-and-error system för hur tre-i-rad spelas, det kallades för MENACE. 1968 beskrev Michie och Chambers en annan tre-i-rad spelare kallad GLEE och även en polbalanserande vagn kallad BOXES. Som det nämndes tidigare producerades det under en tid runt 60- 70-talet flera publikationer som inte direkt behandlade BBI utan snarare övervakad inlärning. Harry Klopf (1972, 1975, 1982) upptäckte detta och lade grunden för uppdelningen av övervakad inlärning och BBI vilket senare vidare preciserades av Barto, Sutton, Brouwer och Anandan i en serie publikationer (Barto, 1998).

Optimal kontroll

Ordet optimal kontroll myntades under sent 1950-tal och behandlar problem där man vill hitta en *metod* som i ett givet *system* optimerar något *kriterium*, en kostnadsfunktion.

För att ge ett kort exempel. En man cyklar längs en slingrig väg, hur ska han cykla för att minimera resetiden? *Systemet* är cykeln och vägen som den färdas på. *Metoden* är helt enkelt hur och när mannen växlar, ökar och minskar sin hastighet, det vill säga bromsar respektive trampar på. *Kriteriet*

som ska minimeras är resetiden. Kontroll problem, även detta, begränsas vanligen av restriktioner. Till exempel cyklistens uthållighet, maxhastighet med mera. Problemets kostnadsfunktion är ett matematiskt uttryck för resetiden som funktion av hastigheten, vägens utformning och systemets starttillstånd. Så är ofta fallet att restriktioner och kostnadsfunktionen är överensstämmande.

Runt mitten av 1950-talen utvecklade Rickard Bellman, med flera, tillvägagångssätt för att lösa optimal kontroll problem. Metoden kan lättsamt beskrivas som en rekursionsekvation med syfte att dela upp och lösa delproblem av ett större optimeringsproblem, ekvationen kallas idag för Bellman ekvationen. Den typ av metod som med hjälp av Bellman ekvationen löser optimal kontroll problem kom att kännas till som dynamisk programmering (Barto, 1998). Bellman introducerade också en diskret stokastisk version av optimal kontroll problemet kallat en Markoviansk beslutsprocess (eng. Markovian decision process, MDP) som 1960 vidareutvecklades av Ron Howard (Barto, 1998). MDPs innefattar en *agent* som interagerar med sin *omgivning* (ett system) för att maximera den ackumulerade *belöningen* som fås med tiden. *Agenten* uppfattar omgivningens *tillstånd* och väljer *handlingar* därefter. Agenten kan approximera en *värde funktion* och använda den för att med tiden konstruera bättre och bättre *policys*. En värde funktion returnerar ett värde för hur ”bra” ett visst tillstånd eller utförandet av en handling är och policys kan ses som regler eller metoder för vilken handling som ska väljas vid specifika tillstånd. Bellman ekvationen, Optimal kontroll, dynamisk programmering, värde funktioner och MDPs är alla viktiga inslag som underbygger teori och algoritmer inom området BBI.

Temporal-difference

Det tredje området TD handlar om metoder som tar hänsyn till det faktum att påföljande beräkningar/förutsägelser ofta är korrelaterade i någon mening. Till exempel sannolikheten för att vinna tre-i-rad. 1959 föreslog och implementerade Arthur Samuel en inlärningsmetod som inkluderade TD idéer som en del i hans kända checkers-spelare. Som tidigare nämnt finns här ett glapp då lite jobb dedikerades trial-and-error och ingenting publicerades då heller beträffande TD. Det var först med Klopf, 1972, som åter satte TD på kartan genom att till viss del sammanföra det med trial-and-error men framförallt kopplade han samman trial-and-error med omfattande psykologiforskning relaterat djurs inlärning. Framförallt Sutton och Barto utvecklade idéerna vidare speciellt länkarna till teorier om djurs inlärning. 1981 var de både Sutton och Barto fullt inlästa på tidigare TD relaterat arbete och utvecklade då en TD-metod som kallas *actor-critic*. Den tidigaste publikationen av en TD-metod publicerades 1977 av Ian Written och kom att kallas ”tabular TD(0)” av Sutton och Barto (Barto, 1998).

Uppkomsten av Q-learning

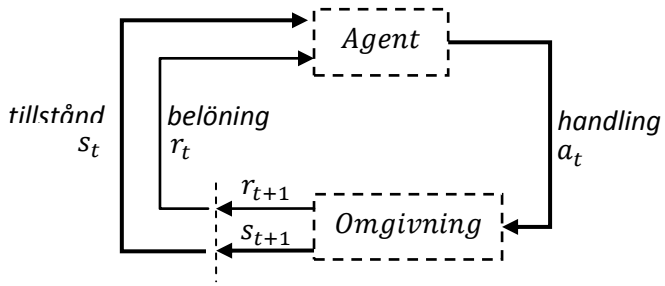
Angående uppkomsten av Q-learning algoritmen, var det just 1989 som Chris Watkins sammanförde de två delområdena temporal-difference och optimal kontroll. Hans arbete utvecklade och integrerade tidigare forskning från alla de tre delområdena. Watkins och Dayan bevisade 1992 Q-learning algoritmens konvergens(Watkins & Dayan, 1992), det vill säga att algoritmen för givna problem hittar en optimal policy. 1998 publicerade Sutton och Barto ”Reinforcement Learning: An Introduction” som ingående beskriver området BBI och är rätt plats att börja på om man finner AI och framförallt BBI intressant.

Q-learning

*Avsnittet går igenom de grundläggande metoderna och idéerna kring BBI eftersom dessa ligger till grund för Q-learning. En snabb uppfattning om algoritmen kan fås genom att läsa underkategorierna "*Grundläggande BBI*" samt "*Q-learning algoritmen*". Resterande underkategorier ger en mer djupgående genomgång av begrepp.*

Grundläggande BBI

BBI är helt enkelt inlärning via interaktion för att uppnå ett mål. Delen som lär sig och fattar beslut kallas för *agenten*. En *agent* interagerar med sin *omgivning*. En komplett beskrivning av omgivningen brukar kallas för en *uppgift*, en instans av BBI problemet. En uppgift kan till exempel vara att spela spelet tre-i-rad. Mer specifikt interagerar *agenten* med *omgivningen* vid varje av ett diskret antal tidssteg, t = 0, 1, 2, 3,… Vid varje tidssteg t, får agenten en representation av den aktuella *omgivningen*, ett *tillstånd*, $s_t \in \boldsymbol{S}$, där $\boldsymbol{S}$ är mängden av alla möjliga *tillstånd*. Vid varje *tillstånd* kan en *handling*, $a_t \in \boldsymbol{A}(s_t)$, utföras där $\boldsymbol{A}(s_t)$ är antalet möjliga *handlingar* i *tillståndet* s_t. Den valda *handlingen* leder, ett tidssteg senare, till att agenten får en numerisk *belöning*, $r_{t+1} \in \boldsymbol{R}$ samt nu befinner sig i ett nytt *tillstånd*, s_{t+1}. Nedanstående figur visar hur *agenten* interagerar med sin *omgivning*.



Figur 1: Interaktionen mellan agent och omgivning.

Vid varje tidssteg implementerar agenten en mappning mellan tillstånd och sannolikheten att utföra, för tillståndet, möjliga handlingar. Mappningen kallas för agentens *policy* och skrivs som $\pi_t(s, a)$ vilket är sannolikheten för att $a_t = a$ om $s_t = s$. Olika BBI metoder specificerar hur agenten ska ändra sin policy medan agentens mål är att maximera den totala belöningen, alltså att maximera den långsiktiga belöningen (Barto, 1998). Mer precist vill man generellt sett maximera den *förväntade belöningen*, R_t, som definieras som någon funktion för returvärdes sekvensen. I sin simplaste form skulle funktionen kunna vara:

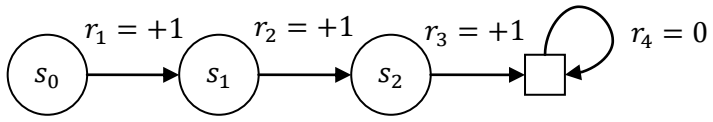
$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T \tag{1}$$

där T är det sista tidssteget. Detta angreppssätt är vanligt när man kan dela upp agent-omgivnings interaktionerna i delsekvenser, även kallat *episoder*, ett exempel är uppdelning i spelomgångar. Ett tillägg är användning av *reducerade belöningar* som gör att agenten väljer handlingar som långsiktigt sett maximerar belöningen. Detta skrivs som

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$$

(2)

där $0 \leq \gamma \leq 1$ och $T \neq \infty$ och gäller för *episodiska uppgifter*, till exempel för brädspel som tre-i-rad där varje spelomgång avslutas inom ändlig tid (Barto, 1998). För att notationen i formel (2) skall fungera måste man införa ett såkallat *absorberande sluttillstånd* – sluttillstånd som leder till sig självt – som returnerar en belöning av värde noll. Bilden nedan visar sluttillståndet som en fyrkant med belöningen noll, där en summering av de första T belöningarna stämmer med definitionen av Formel (2).



Figur 2: Visar att belöningen i sluttillståndet är noll.

Användning av belöningar för att uppnå ett mål är ett av BBIs mest särskiljande drag. Viktigt att påpeka är att belöningen styr vilket mål som ska uppnås och inte hur målet ska uppnås.

Markoviansk beslutsprocess

Om en omgivning på ett kompakt sätt summerar tidigare händelser utan att minska möjligheten att förutsäga framtida handlingar kallas omgivningen för en Markoviansk beslutsprocess (Markovian decision process, MDP). Ett exempel är pjäsernas position vid något tillstånd under en spelomgång i ett brädspel som visar vilka tidigare drag som har gjorts samt möjliga framtida drag. Att man inte vet den exakta sekvensen av gjorda drag påverkar inte vilka framtida handlingar som blir möjliga och brädspel kan därför typiskt beskrivas som MDPs. Det är vanligt med MDPs där antalet tillstånd och handlingar är finit vilket typiskt är fallet för brädspel där varje spelomgång har ett ändligt antal pjäspositioner (tillstånd) och varje tillstånd har ett finit antal möjliga drag (handlingar). MDPs beskriver inte BBIs huvudsakliga syfte, vilket är hur inlärning via belöning skall ske, men de är ändå ett sätt att beskriva vilka modeller/problem som BBI kan appliceras på. (Barto, 1998)

Värde funktioner

Medan belöning indikerar hur bra ett visst tillstånd är i direkt mening säger värde funktionen hur bra tillståndet är i det långa loppet. Löst skrivet är värde funktionen ett värde för hur stor belöningen i framtida tillstånd förvändas bli, eller mer exakt den förväntade belöningen. Belöningar är i en mening primära och värden sekundära då dessa baseras på belöningar. Enda meningen med att uppskatta värde funktioner är att uppnå mera belöning. Belöningarna som agenten förväntas få baseras på vilka handlingar som den väljer. Värde funktioner beror därmed på valda policys. *Värdet* i tillstånd s när man följer policy π skrivs därför som $V^\pi(s)$ och definieras för MDPs av formeln

$$V^\pi(s) = E_\pi\{R_t | s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\}$$

(3)

$E_\pi\{\}$ symboliserar det förväntade *värdet* då agenten följer policy π och t är något tidssteg. V^π kallas för *tillstånd-värde funktionen för policy* π. Tillika definieras värdet av utförandet av handling a vid tillstånd s med policy π som,

$$Q^\pi(s,a) = E_\pi\{R_t | s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right\} \quad (4)$$

där Q^π kallas för *handling-värde funktionen för policy* π.

För att lösa en BBI instans gäller det att finna en policy som uppnår mycket belöning över lång tid. För finita MDPs kan man definiera en *optimal policy*. Värde funktioner definierar en ordningsföljt av policy. Att policy π är bättre än policy π' beror på deras förväntade belöningar, med andra ord är $\pi \geq \pi'$ omm $V^\pi(s) \geq V^{\pi'}(s)$ för alla $s \in \boldsymbol{S}$. Om detta gäller för en av alla andra policys är denna en *optimal policy* och skrivs som π^*. Optimala policys delar samma tillstånd-värde funktion som kallas för den *optimala tillstånd-värde funktionen*, V^*, och definieras som

$$V^*(s) = \max_\pi V^\pi(s) \quad (5)$$

för alla $s \in \boldsymbol{S}$. Optimala policys delar också samma *optimala handling-värde funktion*, Q^*, och definieras som

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \quad (6)$$

för alla $s \in \boldsymbol{S}$ och $a \in \boldsymbol{A}(s)$. För tillstånd-handlings paret (s,a) ger Q^* den förväntade belöningen av handling a vid tillstånd s med fortsatt följande av en optimal policy. Därför kan Q^* skrivas som

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \quad (7)$$

Ovanstående enligt (Barto, 1998). Så länge mängden handlingar och tillstånd är relativt små kan värde funktioner och policys approximeras via ovanstående metoder men ju mer mängderna ökar desto mer minnes- och beräkningskrävande blir de. Till slut kommer metoderna inte längre att fungera och man måste då använda andra lösningsmetoder så som *funktions approximationer* vilket inte innefattas i detta arbete.

Q-learning algoritmen

I Q-learnings fall approximerar handling-värde funktionen, Q, direkt Q^* som alltså är den optimala handling-värde funktionen. Det optimala handling-värde funktionen approximeras korrekt oberoende av vilken policy som följs. Detta leder till att analysen av algoritmen drastiskt förenklas samt möjliggjorde tidiga bevis för dess konvergens (Barto, 1998). Allt som krävs för korrekt konvergens är att alla tillstånd-handling par besöks och uppdateras kontinuerligt (Barto, 1998). Q-värdena för tillstånd-handlings paren uppdateras enligt

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma Max_{a'}(Q(s',a')) - Q(s,a)]$$

(8)

där s är aktuellt tillstånd, a är handlingen som gjords i s, s' är resulterande tillståndet av a, a' är en möjlig handling i tillstånd s' och $Max_{a'}(Q(s',a'))$ är maximala Q-värdet för för alla a'. r är belöningen (eller bestraffningen) av att komma till tillståndet s'. Parametern α är inlärningsvärdet och anger hur mycket det gamla Q-värdet påverkas av handling-värde funktionen. Parametern γ bestämmer hur mycket framtida Q-värden påverkar det aktuella Q-värdet. Ett lägre värde innebär att man tar mer hänsyn till nära belöningar än långtgående. α och γ kan båda anta värden mellan 0 till och med 1.

Q-learning algoritmen beskrivs enligt följande pseudokod.

Initiera $Q(s,a)$ matrisen med godtyckliga värden.
För varje inlärnings session:
    Initiera s
    Kör medans aktuelltillstånd inte är ett sluttillstånd:
        Välj en möjlig handling a från s med policy härled ifrån Q.
        Utför handling a och observera r och s'.
        $Q(s,a) = Q(s,a) + \alpha[r + \gamma Max_{a'}(Q(s',a')) - Q(s,a)]$
        $s \leftarrow s'$

Formel 9: Visa uppbyggnaden av Q-learning algoritmen

Tre-i-rad

Metod

Spelplanen

Spelplanen ser ut som följer

X		
X	X	O
X		O

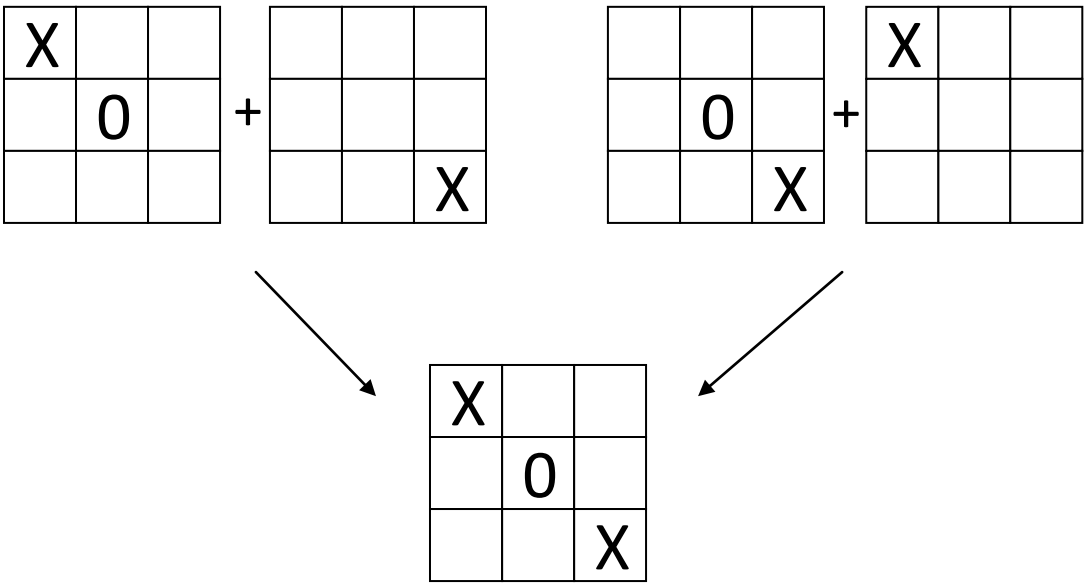
X	X	X
	X	
O	O	

Figur 3: Visar två olika spelomgångar där spelare X har vunnit.

Spelplanen, se Figur 3, innehåller nio rutor som vid olika tillstånd antingen kan vara spelbar, X eller O. Detta ger maximalt $3^9 = 19683$ tillstånd. Alla dessa tillstånd är dock inte möjliga då X alltid börjar och O följer är antalet X alltid lika med eller en mer än antalet O. Dessutom fortsätter man inte en spelomgång när någon spelare vunnit. Genom numeriska beräkningar finner man att det totala antalet tillstånd är 5478 stycken. Detta antal tar hänsyn till eftertillstånd (se nedan) men inte till spelplanens symmetriska egenskaper som speglingar och rotationer (Se Figur 3).

Eftertillstånd

För att på bästa sätt representera tillstånd och handlingar tillämpar vi något som heter eftertillstånd (eng. afterstates). Eftertillstånd används då två olika tillstånd, se bilden nedan, och handlingar kan leda till samma ”eftertillstånd” vilket utan tillämpning leder till redundanta tillstånd-handlings par (Barto, 1998).



Figur 4: Visar två olika tillstånd där en handling leder till samma eftertillstånd.

Användning av eftertillstånd påverkar inte inlärningsresultatet men påverkar självklart inlärningstiden vilket gör applicering obestridligt. (Barto, 1998)

Val av policy under inlärningsfasen

För att Q-learning algoritmen både ska utforska nya handlingar och utvidga redan funna handlingskedjor använder oss av parametern epsilon, ε, där $0 \leq \varepsilon \leq 1$. ε anger sannolikheten för att slumpvis vald handling skall utföras i ett visst tillstånd. Ett större värde medför att en slumpmässig handling väljs oftare än en optimal handling, alltså leder $\varepsilon = 1$ till att alla handlingar väljs slumpmässigt.

Implementation

Avsnittet beskriver hur tre-i-rad implementerats, både inlärning och möjlighet till testspel mot Q-learning spelaren. Vidare beskrivs tre olika strategibaserade spelare som använder naiv, optimerad respektive slumpvisa val av drag. Sist beskrivs spelplan och hur lagring av Q-värden sker.

Tre-i-rad programmet består av en egen implementation gjord i java (Se bilaga KOD). Programmet är i stora drag indelat i två delar, en inlärningsdel och en spelfas. Under inlärningsprocessen låter man två spelare spela mot varandra ett visst antal spelomgångar, hädanefter en inlärningssession. De två spelarna symboliseras av pjäserna X och O och det är alltid X som gör första draget. Vår implementation har fyra olika typer av spelare där en av dessa implementerar Q-learning. De andra tre använder sig av bestämda taktiker och används för att vid olika inlärningssessioner träna Q-learning spelaren. Under varje inlärningssession samlas data in som sedan används för att analysera inlärningen samt påverkan av alfa, gamma och epsilon parametrarna. Efter en inlärningssession erbjuds möjlighet att spela mot Q-learning spelaren. Vid varje drag ritas spelpositionerna och även Q-

värdet för handlingarna i det aktuella tillståndet ut. Under inlärningsdelen visas även allmän statistik. Analysen är inte baserad på dessa resultat utan på en mer detaljerad datainsamling som sker i bakgrunden.

```
0%
Learning Stat:
Stats of last 0 games.
X wins: 0, 0%
O wins: 0, 0%
Draw: 0, 0%

Optimal Stat:
Stats of last 1000 games.
X wins: 1000, 100%
O wins: 0, 0%
Draw: 0, 0%

…

100% totalTime: 8.765 sec
Learning Stat:
Stats of last 1000 games.
X wins: 992, 99%
O wins: 0, 0%
Draw: 8, 1%

Optimal Stat:
Stats of last 1000 games.
X wins: 0, 0%
O wins: 0, 0%
Draw: 1000, 100%

-----------------------------------
_|_|_
_|_|_
_|_|_
Still playing!
Your turn type the location you want to play.
q|w|e
a|s|d
z|x|c
s
_|_|_
_|X|_
_|_|_
Still playing!
Qvalues for cpt player:
0,00000|-0,36000|0,00000
-0,36000|       |-0,36000
0,00000|-0,36000|0,00000
_|_|_
_|X|_
_|_|O
Still playing!
Your turn type the location you want to play.
q|w|e
a|_|d
z|x|_
```

Antalet avklarade spelomgångar I procent. ”learningstat” sker under inlärningsfasen och anger utfallet av de tusen senaste körningarna. ”optimal stat” pausar inlärningen och kör tusen spel där handlingarna väljs efter bästa aktuella Q-värde.

Q-värdena som datorspelaren grundar sina drag på visas på skärmen. Spelaren väljer den handling med högst Q-värde. Tomma rutor symboliserar handlingar som inte är möjliga.

Tabell 1: En utskrift ifrån tre-i-rad programmet.

Primitiv spelare

Strategin som denna spelare använder är att placera sin pjäs på första lediga plats från position noll till åtta enligt figuren nedan. Inlärnings- och spelningsfas påverkar inte hur handlingar väljs.

0	1	2
3	4	5
6	7	8

Figur 5: Den primitiva spelarens handlingsprioriteringar

Smart spelare

Strategin är att placera pjäsen på den position där flest blockeringar av motspelaren samt flest egna vinstmöjligheter är utförbara. En mer detaljerad beskrivning finns på (EDais :: Tutorials :: Tic tac toe A.I., 2002). En modifikation är att X alltid börjar i mitten vilket gör att Q-learning spelaren i bästa fall spelar lika annars förlorar. För att Q-learning spelaren skall ha en chans att spela lika krävs att dennes första drag är i ett hörn. Om första draget är på en kant leder det alltid till förlust mot den smarta spelaren. Förklaringen till detta är trivial och omfattas inte av detta projekt. Inlärnings- och spelningsfas påverkar inte hur handlingar väljs.

Slumpande spelare

Denna spelare väljer helt slumpmässigt bland alla möjliga lediga positioner för det aktuella brädtillståndet och följer helt enkelt ingen strategi alls. Inlärnings- och spelningsfas påverkar inte hur handlingar väljs.

Q-learning spelare

Av de fyra spelarna är denna den enda som använder sina Q-värden. Vid inlärning kan en handling väljas på två olika sätt som avgörs av parametern ε. Antingen väljs en slumpmässig handling eller så utforskas den av de för tillståndet möjliga handlingarna med maximalt Q-värde. Q-värdena uppdateras därefter. Vid spelningsfasen väljs endas den handlingen med högst Q-värde och ingen inlärning sker.

Representation av spelplanen samt lagring av Q-värden

Spelplanens nio rutor beskrivs av en heltalsvektor av längden nio. Varje index har ett av värdena 0,1 eller 2 som representerar en tom plats, X respektive O. Tillståndet i Figur 6 beskrivs till exempel av vektorn

$s = [1, 1, 2, 2, 1, 1, 0, 2, 0]$.

X	X	O
O	X	X
	O	

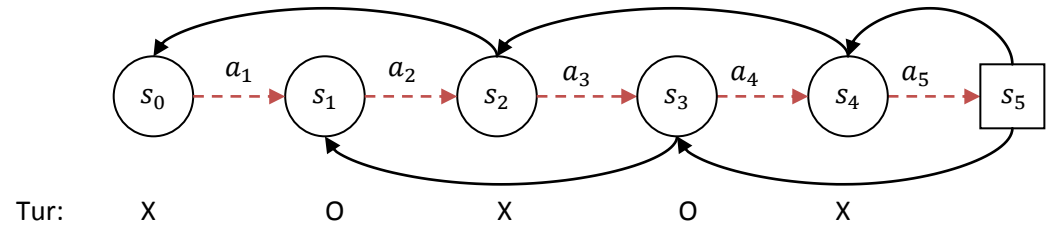
Figur 6: Ett speltillstånd där O har nästa drag

Varje tillstånd kan maximalt ha nio olika handlingar (positioner att placera nästa drag). Det enda tillståndet då alla handlingar är möjliga är starttillståndet, alltså den tomma spelplanen. Varje tillstånds Q-värden lagras i en flyttalsvektor av längden nio. Icke tillgängliga handlingar får det speciella värdet NaN (eng. not a number). Figur 6s Q-värden blir $Q(s) = [NaN, NaN, NaN, NaN, NaN, NaN, -1.0, NaN, 0.0]$ som innebär en förlust för O om nästa drag är i nedre vänstra hörnet (-1.0) och oavgjort spel för högre nedre hörnet (0.0).

En Q-värdesvektor behövs för varje tillstånd och lagras i en niodimensionell vektor, Q, där varje dimension har storleken tre. N:e dimensionens index fås av n:e värdet i tillståndsvektorn. Till exempel uppslagning $Q[1][1][2][2][1][1][0][2][0]$ returnerar $Q(s)$, observera att s är samma som tidigare nämnt tillstånd.

Uppdatering av Q-värden

Uppdatering av Q-värden sker alltid efter spelomgång. Handlingarna, dragen, som spelarna gör under en spelomgång sparas i en logg som sedan gås igenom baklänges för att uppdatera Q-värdena för respektive spelare. Som tidigare nämnt behövs det tre värden för att uppdatera ett tillstånds Q-värde.



Figur 7: Visar hur loggen används för att uppdatera Q-värdena för spelare X respektive O.

Enligt figuren ovan har X vunnit på tre drag medan O gjort två drag. Q-värdena uppdateras i följande ordning:

1. Q-värde för X: $s_4(a_5) \rightarrow s_5$
2. Q-värde för O: $s_3(a_4) \rightarrow s_5$
3. Q-värde för X: $s_2(a_3) \rightarrow s_4$
4. Q-värde för O: $s_1(a_2) \rightarrow s_3$
5. Q-värde för X: $s_0(a_1) \rightarrow s_2$

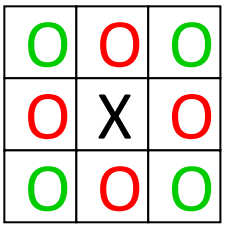
De nya Q-värdena beräknas enligt algoritmen beskriven i avsnitt ”Q-learning algoritmen”. Belöningen för vinnst har satts till 1, bestraffning vid förlust till -1 och vid oavgjort till 0.

Analys

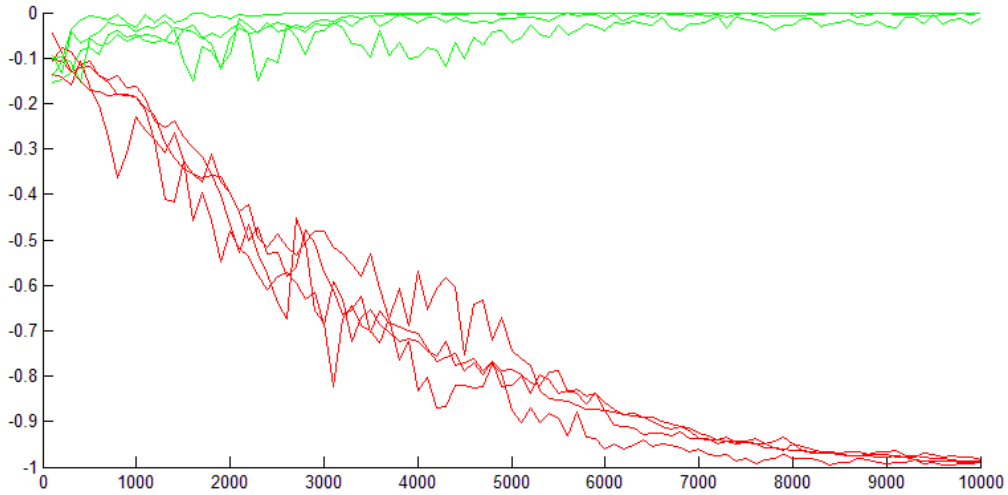
Detta avsnitt är uppdelat i fyra underkategorier en som argumenterar för att inlärningen är korrekt och tre som analyserar hur enskilda parametrarnas påverkar inlärningsförmågan.

Inlärning

För att visa att implementationen fungerar har vi valt att träna O mot den Smarta spelaren som spelar X. Då X börjar och alltid sätter sin pjäs i mitten har O endast möjlighet att spela lika eller förlora. Parametrarna har satts till följande värden: $\alpha = 0.2$, $\gamma = 1$ och $\varepsilon = 1$. X-axeln är antalet inlärningssessioner och Y-axeln anger stoleken på Q-värdet. I det aktuella tillståndet i figuren nedan har X gjort sin första placering i mittenpositionen och det är nu O:s tur. De fyra gröna linjerna representerar Q-värdet för respektive hörn och de fyra röda Q-värdet för respektive kant. Se Figur 8 och Figur 9.



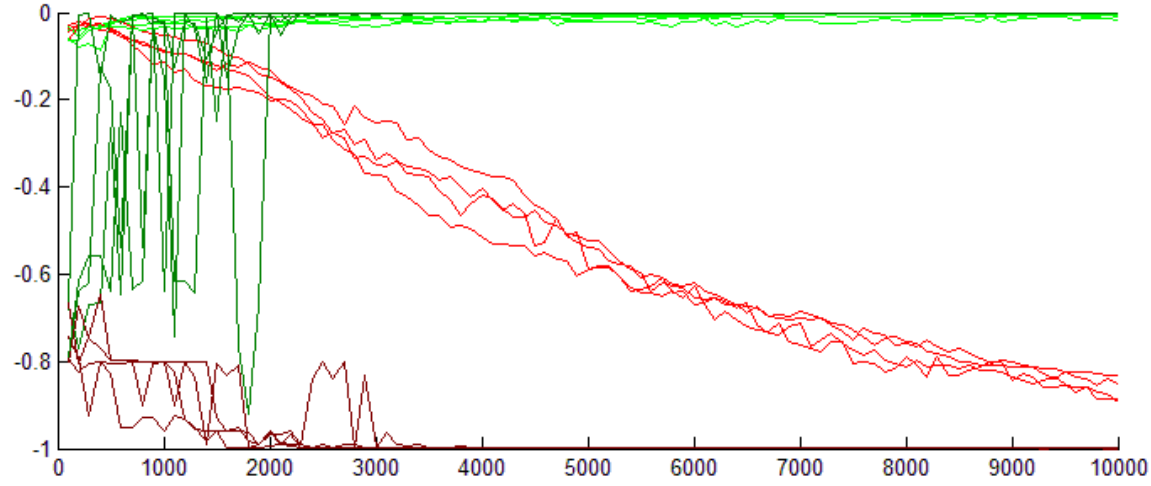
Figur 8: De möjliga dragen för O



Figur 9: $\alpha = 0.6$, $\gamma = 1$ och $\varepsilon = 1$. Grönt representerar hörnen som konvergerar mot 0 och rött kanterna som konvergerar mot -1.

Alfa, α

För att kunna analysera alfa låter vi $\gamma = 1$ och $\varepsilon = 1$. Figuren nedan befinner sig i samma tillstånd som i inlärningsexemplet ovan, alltså då X redan gjort sitt första drag i mitten och O nu ska göra sitt drag. Två olika inlärningssessioner på 10000 spelomgångar har gjorts. I figuren nedan symboliserar grönt Q-värdena för hörnen och rött Q-värdena i kanterna. De ljusare färgerna symboliserar den ena sessionen med $\alpha = 0.1$ och de mörkare färgerna då $\alpha = 0.8$.



Figur 10: $\gamma = 1$ och $\varepsilon = 1$. Två olika sessioner med $\alpha = 0.1$ (ljusa färger) och $\alpha = 0.8$ (mörka färger).

Vi har valt dessa värden för att observera hur höga respektive låga alfavärden påverkar konvergensen.

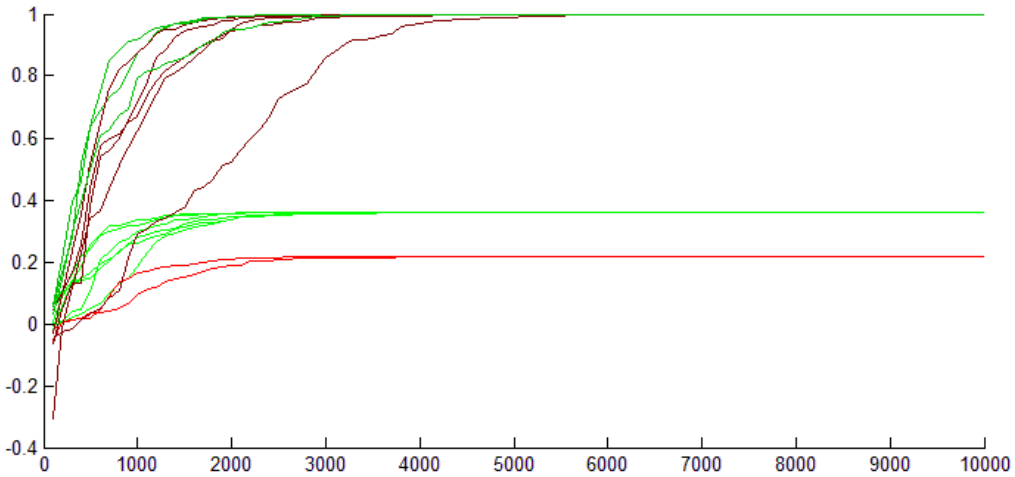
X	O	O
O	O	O
O	O	O

Figur 11: Gammas påverkan

Gamma, γ

Vid detta exempel har vi valt att använda oss av den primitiva spelaren. Detta är för att denna spelares taktik leder till att vid första tillståndet O får spela (se Figur 11) leder alla positioner, utom två, till vinst på lika många drag. Om första draget är på en av de två röda (mörka) platserna krävs det i bästa fall ytterligare tre drag för att vinna. Jämförelsevis kvävs det endas två ytterligare drag om man börjar på en grön

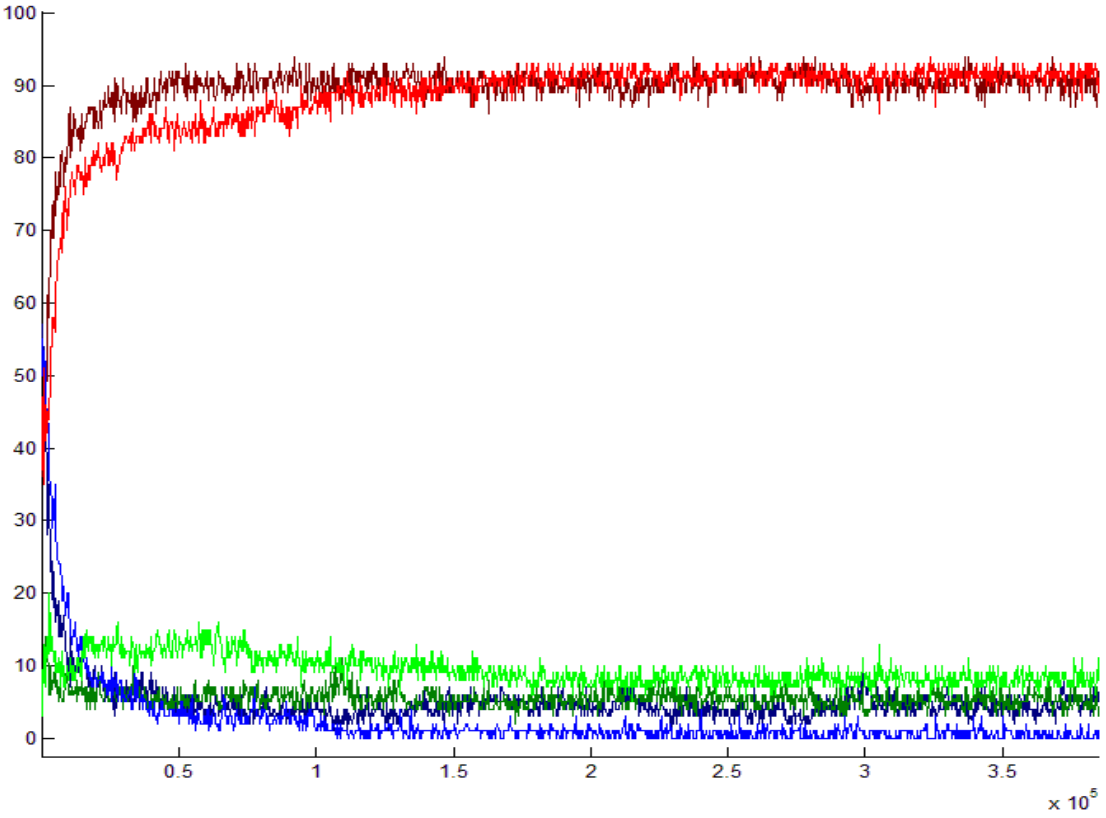
(ljus) plats. De ljusa linjerna, i Figur 12, representerar $\gamma = 0.6$ och de mörkare $\gamma = 1$. Parametrarna $\varepsilon = 1$, $\alpha = 0.2$ och de två sessionerna visas enligt Figur 11 och Figur 12.



Figur 12: $\alpha = 0.2$ och $\varepsilon = 1$. Två olika sessioner med $\gamma = 0.6$ (ljusa färger) och $\gamma = 1$ (mörka färger).

Epsilon, ε

I det här fallet har vi valt att låta O spela mot den slumpande spelaren. X-axeln representerar antalet spelomgångar. Efter var trehundrade spelomgång pausas inlärningen och tusen spel körs med dåvarande optimala handlingar. Y-axeln representerar den procentuella fördelningen av vinst (rött), förlust (blått) samt oavgjort (grönt). I grafen visas två inlärnings sessioner som båda följer samma färgschema. De mörkare färgerna har körts med $\varepsilon = 1$ och de ljusare med $\varepsilon = 0.1$. $\alpha = 0.2$ och $\gamma = 0.8$ under de båda spel sessionerna.



Figur 13: Hur epsilon påverkar inlärningsprocessen med $\alpha = 0.2, \gamma = 0.8$ och $\varepsilon = 1$ (mörk färg) respektive $\varepsilon = 0.1$ (ljus färg).

Resultat och diskussion

I detta avsnitt presenterar vi våra resultat för hur parametervärdena påverkar spelarens inlärning samt vilka värden som vi anser som bra för en tre-i-rad-spelare.

Alfa, α

Det framgår av Figur 10 att α påverkar Q-värdesförändringen radikalt. Högre värden leder till att Q-värdena snabbare får sina slutvärden men samtidigt fluktuerar. Detta medför att en sällsynt handling påverkar ett Q-värde mer direkt än när α är lägre. För att på ett mer stabilt sätt få acceptabla Q-värden anser vi att α bör hållas lågt runt $\alpha = 0.2$. Därmed krävs det flera körningar med samma utfall för att ett Q-värde noterbart skall påverkas. Detta leder i sin tur till att händelser med väldigt låg sannolikhet inte märkbart påverkar spelarens besluttagning som därmed blir mer strategisk.

Gamma, γ

I Figur 12 framgår det hur Q-värdet påverkas av två olika γ-värden. När $\gamma = 1$ spelar det ingen roll hur lång tid det tar för att komma till ett belönande tillstånd. Spelaren skulle lika gärna kunna välja en längre följd av drag framför ett med färre antal. När man sänker γ-värdet proportionerar man Q-värdena för de olika handlingarna, vid ett visst tillstånd, efter hur nära de är ett belöningsgivande tillstånd. Detta ses tydligt i Figur 12 där de gröna handlingarna har högre Q-värde än de röda när $\gamma = 0.6$. Eftersom vi vill att vår spelare ska vinna på snabbaste möjliga sätt föreslår vi att $\gamma = 0.8$. Detta medför att handlingar som snabbare leder till ett belönande tillstånd får högre Q-värden än de

som tar längre tid. Samtidigt är det inte allt för stor skillnad på Q-värderna för de handlingar som med bara få stegs skillnad leder till ett belönande tillstånd.

Epsilon, ε

$\varepsilon = 1.0$ innebär att spelaren alltid väljer slumpmässiga drag vid inlärning. Därmed är sannolikheten för att utföra ett tillstånds olika handlingar lika stor. Detta medför att optimala handlingar, som egentligen skulle kunna leda till hög belöning, lika ofta kan leda till tillstånd med dåliga Q-värden och därmed tolkas som en sämre handling. Detta kan man observera i Figur 13. Vinstprocenten för de olika ε-värdena konvergerar mot samma värde men förlustprocenten är högre för spelaren med $\varepsilon = 1$ än för spelaren med $\varepsilon = 0.1$. Detta är ett resultat av att bra handlingar inte exploateras tillräckligt mycket. Därför är det i detta fall bättre att ha ett lågt ε-värde då detta gör att optimala handlingar exploateras till sin fulla potential.

Referenser

Barto, R. S. (1998). *Reinforcement learning: An Introduction.* Cambridge, Massachusetts, London, England: The MIT Press.

EDais :: Tutorials :: Tic tac toe A.I. (februari 2002). Hämtat från Edais.org: http://edais.mvps.org/Tutorials/TTTAI/index.html den 28 april 2010

Gerald Tesauro. (1995). *Temporal difference learning and TD-Gammon.* Hämtat från research.ibm.com: http://www.research.ibm.com/massive/tdl.html den 8 mars 2010

Teknomo, K. (2005). *Q-Learning by Examples.* Hämtat från http://people.revoledu.com/kardi/tutorial/ReinforcementLearning/index.html

Watkins, C., & Dayan, P. (1992). Q-learning. Machine Learning.

Bilagor

Kod

Beskriver de java klasser som skapats för att utföra projektet. Varje underkategori delar in klasserna i olika paket (eng. packages) för att lättare gruppera klasser med liknande funktioner.

ticTacToe

Game: Denna klass representerar ett spel och innehåller metoder för att uppdatera spelplanen, skriva ut samt ge information om vilket tillstånd spelet befinner sig i.
Learner: Denna klass innehåller metoder för att köra en eller flera inlärningsomgångar med givna X och O spelare. Klassen innehåller även metoder för att köra optimala spelomgångar av de tränade spelarna.
Main: Innehåller main-metoden för att köra en inlärningsomgång som följs av en spelomgång mot den nyinlärda spelaren.
PlayingUI: Innehåller metoder för att låta en användare spela mot en given datorspelare.
Test: Innehåller main-metoden samt hjälpmetoder för att samla statistik under inlärningsprocessen.

ticTacToe.stateStat

BetterHashSet: En implementation av en HashSet.
GameState: Innehåller hjälpmetoder för klassen StateStats och används för att representera ett unikt speltillstånd.
StateStats: Innehåller main-metoden för att ta reda på antalet speltillstånd. Eftertillstånd tas hänsyn till men inte speglingar eller vridningar av spelplanen.
RandomPlayResults: Innehåller main-metoden för att ta reda sannolikheten av XWin, OWin och Draw när två spelare spelar mot varandra med endast slumpade drag.

ticTacToe.statistics

CsvMaker: Innehåller metoder för att lagra information i csvFiler.
LoopProcessWatch: Innehåller hjälpmetoder för att skriva ut hur lång tid det är kvar av en loop. Används under inlärningen för att ge feedback till användaren vid långa körtider.
Statistics: Innehåller metoder för att spara ett spels sluttillstånd och ge genomsnittsvärden för de n senaste körningarna.

ticTacToe.players

DumbPlayer: En spelare som alltid börjar i övre vänstra hörnet och gör ett drag på första lediga plats från vänster till höger, uppifrån och neråt.
RandomPlayer: En spelare som slumpar varje drag.
SmartPlayer: En spelare som alltid som följer en bestämd algoritm för att alltid göra det mest optimala draget. Om SmartPlayer spelar som X börjar den dessutom alltid i mitten.
Player: Implementerar Q-learning och är den enda spelaren som lär sig samt gör drag därefter.