KTH Computer Science
and Communication

Q-learning för fyra i rad

OLLE HASSEL - 880531-0672 OHASSEL@KTH.SE
ADRESS: IGELKOTTSVÄGEN 10A, 19146 SOLLENTUNA
TELEFON: 070-4699098
&
PETTER JANSE - 860106-1537 PJANSE@KTH.SE
ADRESS: KÖRSBÄRSVÄGEN 4B, 11423, STOCKHOLM
TELEFON: 070-9426640

DD143X
Examensarbete inom datalogi, grundnivå
Johan Boye - jboye@kth.se

Abstract

Q-learning is an algorithm for self-learning where the the learner is rewarded for encouraged behaviour and punished for unwanted behaviour. The report investigates how many matches of connect-four for a self-learned player through Q-learning is needed to win on average 90 % of all matches against a random player, a pattern matchin player and against a calculating player. It will also be investigated wherever the Q-learning player can be combined with other algorithms to create an improved player. Within realistic time on a personal computer the Q-learner beats the pattern matching and calculating player. The random player cannot be beaten in reasonable time, and would also require an unreasonable ammount of memory. The improved Q-learning player that combines Q-learning with pattern matching and calculated start values can be created and this one beats all the three players after a very short learning period.

Sammanfattning

Q-learning är ett inlärningsalgoritm där den lärande får belöning vid positiva beteenden och bestraffning vid negativa. Rapporten avser undersöka hur många matcher i fyra i rad som krävs innan en självlärd spelare som använder Q-learning vinner i snitt 90 % av alla matcher mot en slumpande spelare, en mönstermatchande samt en beräknande spelare. Det undersöks även om Q-learning kan kombineras med andra algoritmer för att skapa en bättre spelare. Q-spelaren slår inom rimlig tid de mönstermatchande och beräknande spelarna men får problem att inom rimlig tid och minnesanvänding slå den slumpande spelaren. Den förbättrade Q-learning-spelaren som kombinerar Q-learning med mönstermatchning och beräknade startvärden slår däremot alla tre spelare inom mycket kort inlärningstid.

Innehåll

Kapitel 1

Förord

Detta är en kandidatexamensrapport för kursen DD143X genomförd vårterminen 2011 på KTH, Stockholm. Större delen av arbetet har utförts sittandes ihop. Endast mindre delar har delats upp och utförts på egen hand, men i dessa fall har de gåtts igenom tillsammans vid nästa möte. Vi delade dock upp ansvaret så att Olle hade det yttersta ansvaret för programmeringsdelen medan Petter hade ansvaret för den skriva rapporten.

Kapitel 2

Introduktion

Denna rapport är skriven för projektet "Q-learning för fyra i rad" vilket är ett kandidatexamensarbete för datavetenskap- och kommunikation (CSC) vid KTH. Målet med projektet är att låta en Q-lärande spelare med hjälp av Q-learning lära sig att spela 4 i rad mot icke lärande datorspelare samt en förbättrad version av sig själv.

Termer

- Den Q-lärande spelaren benämns som Q-spelaren.
- En spelare benämns ha slagit en annan spelare om den vinner i snitt 90 % av alla framtida matcher.

Problemformulering Målet är att undersöka hur många spelade matcher det tar för Q-spelaren att bli tillräckligt bra för att slå tre olika icke-lärande datorspelare. Den första är en mönstermatchande spelare, den andra en beräknande spelare och den tredje en slumpande spelare.

Det ska även undersökas om Q-spelaren kan lära sig att slå dessa inom rimlig tid och minnesanvändning, så pass att det på en persondator[1] kan passera denna brytpunkt. Det ska även undersökas om Q-spelaren kan förbättras genom att kombineras med den mönstermatchande och den beräknande spelaren och därmed lära sig slå de tre andra spelarna efter färre spelade matcher än den vanliga Q-spelaren.

Hypotes Hypotesen innan projektet startade var att de mönstermatchande och beräknande datorspelarna kommer att vinna så gott som alla matcher till en början, men att Q-spelaren efter tillräckligt många spelade matcher till slut kommer att ha lärt sig tillräckligt bra för att slå dem.

Frågan blir då om det är möjligt i praktiken att spela tillräckligt många matcher för att Q-spelaren ska lära sig att vinna, eller om spelets komplexitet gör att

[1] I detta fall en Intel Q9550 CPU, 4 GB RAM.

det krävs alltför många matcher. Hypotesen här är att tid och minne kommer att begränsa i sådan grad att den vanliga Q-spelaren aldrig kommer kunna slå någon av spelarna.

När Q-spelaren däremot kombineras med mönstermatchning och startvärdeberäkning kommer komplexiteten att minska. Denna kombinerade spelare förväntas att inom rimlig tid och minnesanvändning kunna lära sig att slå de andra spelarna i sådan grad att det går utföra på en persondator.

2.1 Bakgrund

2.1.1 Maskininlärning

Maskininlärning är ett område inom artificiell intelligens. Det används inom datalogin för att med hjälp av olika algoritmer låta datorer själva lära sig och förbättra sig på en uppgift. En viktig del av maskininlärning är att låta datorn lära sig att känna igen mönster och ta beslut utifrån dessa. Om en uppgift är för komplex kan det uppstå problem eftersom det då kan ta orimligt lång tid att gå igenom alla möjliga indata och utforska alla möjliga utdata[2] .

Det finns flera grenar av maskininlärning, men den gren som kommer att användas i detta projekt är belöningsbaserad inlärning vilket förklaras närmare här nedan.

2.1.2 Belöningsbaserad inlärning

Q-learning är en typ av belöningsbaserad inlärning[3,4] som innebär att du påverkas av miljön omkring dig. Den används ofta inom maskininlärning och datorvetenskap men är från början inspirerad av den pedagogiska psykologin. Ett enkelt exempel är en ko i en hage som går för nära ett el-stängsel, får en stöt, och lär sig att därefter hålla sig undan.

På samma sätt låter man datorprogram få en belöning alternativt en bestraffning då olika händelser uppstår. Programmet upprepar beteende som leder till belöning och undviker det som leder till straff. Långsiktigt innebär detta att programmet lär sig vilka drag det ska göra för att vinna ett parti 4 i rad (vilket belönas) och vilka drag den ska undvika för att förlora (vilket bestraffas)[5].

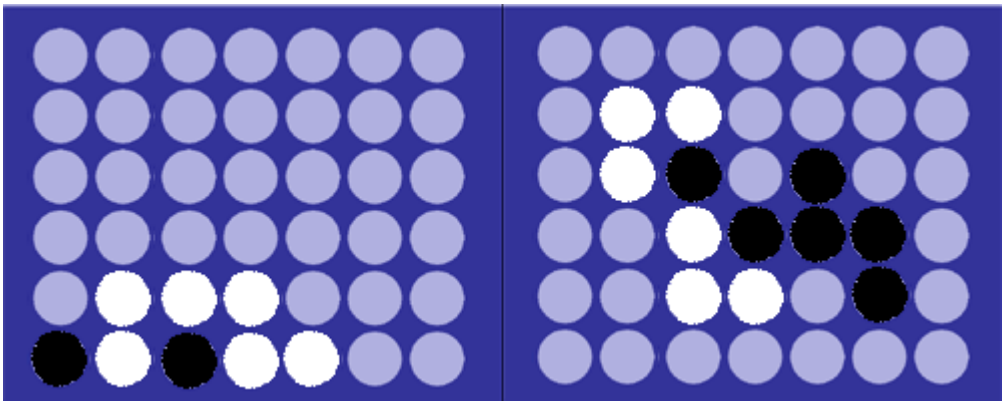
[2]Wikipedia, *Machine Learning*
[3]Wikipedia, *Reinforcement Learning*
[4]Michael Gasser. Introduction to Reinforcement Learning.
[5]Richard S. Sutton & Andrew G. Barto. Reinforcement Learning: an Introduction.

2.2 Teori

2.2.1 Spelet fyra i rad

Spelet fyra i rad spelas mellan två spelare, en som placerar vita brickor och en som placerar svarta. Spelplanen består av ett rutnät av sex rader med längden sju rutor. Spelarna turas om att släppa ner brickor en i taget, som landar ovanpå brickor med samma position i raden, eller, om ingen bricka tidigare lagts på denna position, på botten av brädet. Brickor måste alltså staplas på varandra. Den spelare som först har fyra i rad antingen vertikalt, horisontellt eller diagonalt har vunnit spelet.

Komplexiteten för fyra i rad är teoretiskt mycket stor, men i praktiken mycket mindre. Spelplanen består, som nämnts ovan, av 7x6 rutor. Varje ruta kan vara antingen vit, svart eller tom. Detta innebär att det finns 3^{42} unika bräden vilket är, grovt avrundat, i storhetsordning 10^{20}. Dock är det en mycket stor mängd av dessa bräden som inte är praktiskt möjliga. T.ex. så inkluderar detta omöjliga bräden, exempelvis de i figur 2.1 och bräden som är helt fyllda med vita brickor. Eftersom spelarna turas om att lägga var sin bricka är det givet att det måste finnas lika många brickor av varje färg, alternativt en extra vit, eftersom det är den spelaren som börjar. Även andra omöjliga bräden, t.ex. översta raden är fylld men inte de undre raderna, är inkluderade.



Figur 2.1. Två omöjliga bräden.

En närmare uppskattning har gjorts av Victor Allis, som med hjälp av ett eget skrivet program beräknat den övre gränsen av alla praktiskt möjliga bräden till $7,1 * 10^{13}$ [6].

Samma Victor Allis har även ”slagit spelet” och bevisat att om man är vit och alltså börjar, så kan man forcera vinst. En vit spelare som spelar perfekt vinner alltså 100 % av matcherna.

[6] Victor Allis, *A Knowledge-based Approach of Connect-Four.*

2.2.2 Q-learning

Något som lär sig igenom Q-learning brukar ofta kallas för agent och det är den term som kommer att användas i följande stycke.

Q-learning är en belöningsbaserad inlärningsteknik som går ut på att agenten belönas vid positiva resultat och bestraffas vid negativa resultat. Agenten eftersträvar att få belöning och att undvika bestraffning. Den är inte medveten om vad den gör rätt eller fel, men då den kommer ihåg sina drag arbetar den efter principen att det som tidigare ledde till belöning borde kunna leda till belöning även denna gång, och på samma sätt undviker den att upprepa det som tidigare lett till bestraffning

För att kunna värdesätta olika drags potential att leda till belöning eller bestraffning finns en tabell av tillstånd med tilldelat värde, en så kallad Q-tabell. Agenten har även en lista med de drag som den gjorde under matchen för att kunna uppdatera alla tillstånd med dess nya värde efter ett avslutat parti. Agenten har som mål att vinna. Därför belönas vinst, medan förlorade och oavgjorda partier bestraffas.

Algoritmen När agenten ska göra ett drag, tittar den på alla möjliga drag den kan göra, och slår upp dessa drag och dess värden i Q-tabellen. Om ett av de potentiella tillstånden inte är besökt tidigare används ett startvärde som bestäms av en konstant.

Om inget av dessa drag har ett mycket högre värde än de andra, slumpas istället ett av dragen som ligger nära det bästa värdet. Om flera drag är ungefär lika bra, vill vi nämligen utforska även dessa drag. Det finns en konstant som bestämmer hur mycket större ett värde måste vara än de andra för att detta värde ska bli valt utan slumpning. När Agenten sedan valt ett drag sparas detta längst bak i listan med de drag som gjorts under matchen.

Tilldelning av belöning eller bestraffning Vid avslutat parti tilldelas en belöning eller bestraffning till Agenten. Detta sker genom att ett värde adderas till samtliga valda drag, dvs. de tillstånd som sparats i listan av besökta tillstånd under matchen.

Detta värde bestäms genom en funktion som beror på dragets avstånd från partiets sista drag. Anledningen till detta är att det är större sannolikhet att det var något av de senare dragen som orsakade förlusten eller vinsten och som därmed bör få en högre värdeförändring än tidigare drag. Den funktion vi valt innebär att alla värden multipliceras med konstanten 0,85 upphöjt i antal drag från det sista draget.

Konstanter och variabler:

$$p = poängkonstant,\ som\ är\ positiv\ vid\ belöning,\ och\ negativ\ vid\ bestraffning$$

$$k = konstant\ mellan\ 0\ och\ 1,\ vi\ använder\ 0,85$$

$$n = avståndet\ till\ sista\ draget\ i\ antal\ steg$$

Formel för poängräkning:

$$Nytt\ värde = gammalt\ värde * p * k^n$$

Exempel, agenten har vunnit:

$$Vinnande\ dragets\ värde = gammalt\ värde * p * k^0$$

$$Draget\ innan\ det\ vinnande\ dragets\ värde = gammalt\ värde * p * k^1$$

Q-tabellen behålls mellan partier medan listan med gjorda drag rensas efter matchens slut.

2.2.3 Den mönstermatchande datorspelaren

Den mönstermatchande klienten är en datorspelare som inte tänker alls på framtida drag. Det finns ett antal fördefinierade mönster som klienten kan känna igen på brädet. När det är denna spelares tur söker den igenom alla dessa mönster och väljer ett drag efter en prioritetskö.

Den börjar med att söka igenom brädet för att se om den själv har tre i rad någonstans och lägger i sådana fall den fjärde brickan för att vinna. Om detta mönster inte finns så kollar den om motspelaren har tre i rad och blockerar i sådana fall denna rad. Därefter letar den på samma sätt efter mönster av två brickor och slutligen om inget sådant finns så lägger den en bricka bredvid en annan av sina brickor för att skapa en rad av två.

2.2.4 Den beräknande datorspelaren

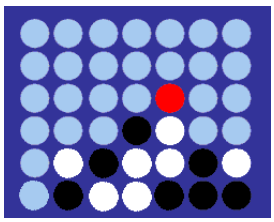
Den beräknande datorspelaren börjar med att undersöka alla möjliga drag som kan göras och sparar dessa koordinater i en lista. Därefter beräknas värdet för alla dessa drag med hjälp av algoritmen som beskrivs nedan. Om värdet för ett av dessa drag är mycket högre än de andra så väljs det draget. Om flera drag ligger runt samma värde så slumpas istället ett av dessa. Det finns en konstant som bestämmer hur stor procentuell skillnad det ska vara mellan de bästa värdena för att det bästa ska väljas direkt.

Algoritmen Algoritmen utgår från det nuvarande brädet samt x- och y-koordinaterna för det drag som ska göras. För det aktuella draget beräknas antalet potentiella vinster som kan inträffa som innehåller den koordinaten. Värdet tillsätts sedan beroende på om dessa potentiella vinstrader redan är partiellt fyllda, där en nästan fylld rad tilldelas ett högre värde än en som är tom.

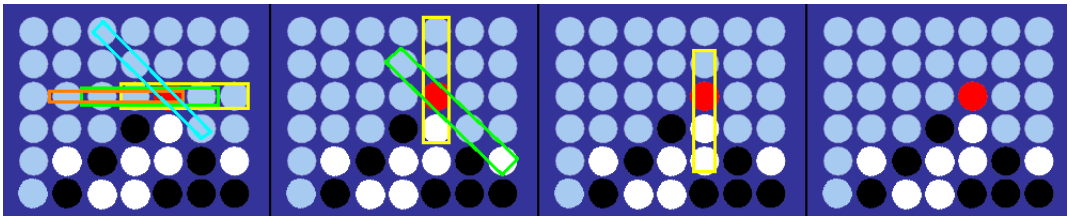
Algoritmen beräknar värdet utifrån båda spelarnas perspektiv, där motståndarens värde minskas marginellt genom att multipliceras med konstanten 0,9. Detta

för att en vinstmöjlighet för agenten ska prioriteras framför att stoppa ett hot från motståndaren med lika många fyllda positioner. Om motståndaren t.ex. kan vinna på ett visst drag är det intressant att lägga sin bricka där för att blockera, förutsatt att agenten inte i stället kan vinna med ett annat drag.

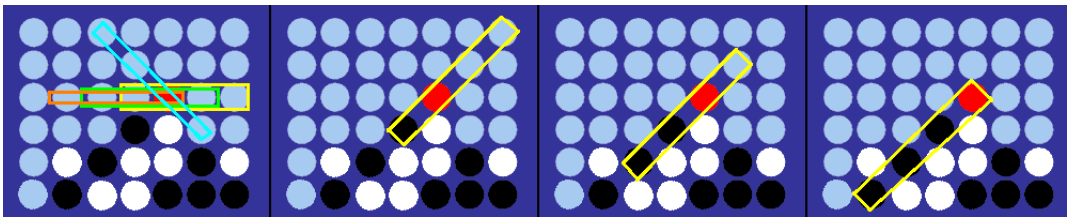
Givet brädet i figur 2.2 ska värdet för den röda brickan beräknas genom att antalet potentiella vinster hittas och poäng tilldelas efter hur ifyllda de är.



Figur 2.2. Ett bräde med ett potentiellt drag med röd bricka.



Figur 2.3. Bräden med vits potentiella vinster markerade i färg. Bild för bild: fyra tomma potentiella vinster. Två potentiella vinster med en bricka. En potentiell vinst med två brickor. Inga potentiella vinster med tre brickor.



Figur 2.4. Bräden med svarts potentiella vinster markerade i färg. Bild för bild: fyra tomma potentiella vinster. En potentiell vinst med en bricka. En potentiell vinst med två brickor. En potentiell vinst med tre brickor.

$$k_0 = tom,\ inga\ brickor\ i\ den\ potentiella\ vinsten$$

$$k_1 = en\ bricka$$

$$k_2 = två\ brickor$$

$$k_3 = tre\ brickor$$

Först beräknas värdet ur vits perspektiv, se figur 2.3.

$$4k_0 + 2k_1 + 1k_2 + 0k_3$$

Sedan beräknas värdet ur svarts perspektiv, se figur 2.4.

$$4k_0 + 1k_1 + 1k_2 + 1k_3$$

Slutgiltiga värdet = värdet ur vits perspektiv + (värdet ur svarts perspektiv)*0,9.

2.2.5 Q-spelaren

Q-spelaren använder sig av Q-learning för att lära sig spela. Den har ingen som helst förkunskap om hur spelet fungerar. Det enda sättet den lär sig på är de belöningar och bestraffningar som ges för drag, beroende på om de leder till vinst eller förlust. Ju fler matcher den får spela desto fler drag poängsätts och uppdateras med korrekt poäng. Den fortsätter att välja det drag som ger störst belöning.

2.2.6 Den kombinerade Q-spelaren

Den kombinerade Q-spelaren fungerar på samma sätt som Q-spelaren förutom att den även använder sig av mönstermatchning och beräknade startvärden.

Många av valen som Q-spelaren ställs inför är triviala med endast ett bra val. Detta förstår inte Q-spelaren från början utan måste testa sig fram innan den börjar lära sig. När t.ex. motståndaren har tre brickor i rad och därmed kan vinna, så förstår inte Q-spelaren detta under sina första matcher.

Den kombinerade Q-spelaren får här hjälp av den mönstermatchande datorspelaren för att stoppa dessa drag. Samtidigt använder den sig fortfarande av Q-learning och kommer i längden att själv märka att detta är ett bra drag och lära sig att känna igen dessa mönster. Detta gör att antalet drag som Q-spelaren ursprungligen måste testa minskar och då komplexiteten minskar så ökar inlärningshastigheten.

Den kombinerade Q-spelaren utnyttjar även den beräknande datorspelaren. Medan den vanliga Q-spelaren använder en konstant för alla startvärden i Q-tabellen så använder den kombinerade Q-spelaren de värden som beräknats av algoritmen som den beräknande datorspelaren använder.

2.3 Implementation

2.3.1 Kärna/klient

Programmet är logiskt uppdelat i två delar, kärna och klient. Kärnan är själva spelet, som kopplar ihop sig med två klienter, vilka implementerar interfacet Client. Client innehåller fyra metoder: win, loss, tie samt makeMove. Metoden makeMove

anropas varje gång som en klient ska göra ett drag och får då en referens av den nuvarande spelplanen som indata.

Kärnan innehåller även inversen av spelplanen, och efter varje drag byts pekarna som skickas vidare på så sätt att båda klienterna tror att de är vita. Detta för att undvika duplicerad kod inuti klienterna.

2.3.2 Datastruktur

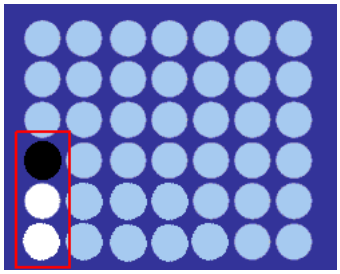
Q-tabellen ska, givet en spelplan, kunna spara ett Q-värde. Det totala antalet tillstånd går att beskrivas på 2^{63} bitar, se förklaring nedan, och därför används en Long som nyckel och en Double som värde i HashMapen.

För att optimera med hjälp av pekarreferenser skapas ett containerobjekt för Q-tabellens värde DoubleContainer och Q-tabellen sparas därmed som en HashMap<Long,DoubleContainer>

Antalet möjliga bräden är, som nämnts tidigare i teoridelen, 3^{42} vilket blir alldeles för stort för att sparas i en variabel av typen long. Eftersom brädet endast kan byggas på höjden så finns det dock ett bättre sätt att spara det på.

Brädet delas in i kolumner som vardera sparas som nio bitar. De tre första bitarna sparar kolumnhöjden och de följande sex bitarna sparar om det är en vit (etta) eller svart (nolla) bricka. Index på dessa bitar avser vilken höjd brickan är placerad på, den första biten avser därmed den nedersta brickan. Av de sex bitarna som sparar brickfärgen används därmed bara så många som höjden på kolumnen är och resterande sätts till 0 och ignoreras.

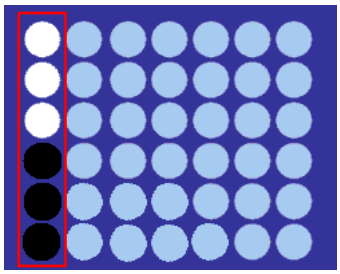
Kolumnerna sparas på följande sätt: *[9 bitar (kolumn)] = [3 bitar (kolumnhöjd)] [6 bitar (brickfärg)]*



Figur 2.5. Kolumn med höjden tre, de två nedersta brickorna är vita och den tredje svart och sparas därmed på följande sätt: *[011] [110000]*

Kolumnerna sätts sedan ihop så att det slutgiltiga brädet sparas på följande sätt: [bit 1-9] [bit 10-18] [bit 19-27] [bit 28-36] [bit 37-45] [bit 46-54] [bit 55-63]

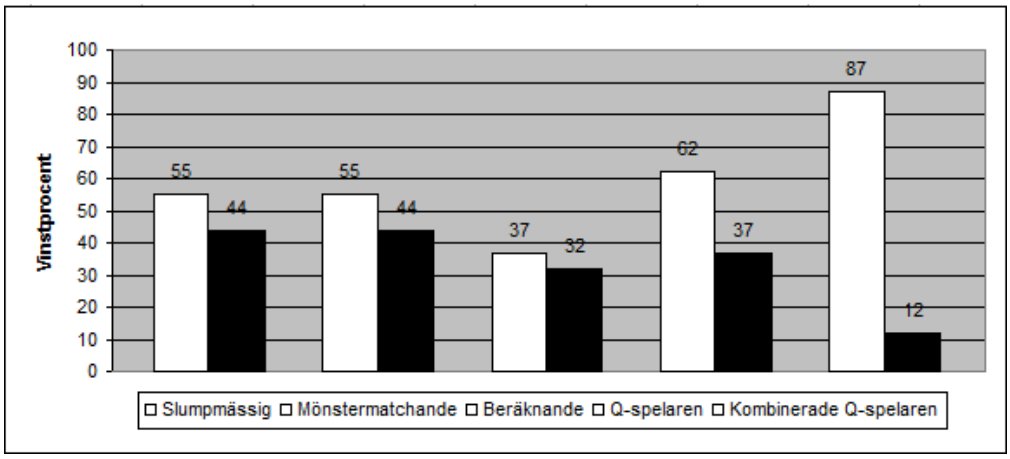
Totalt krävs därmed 63 bitar för att representera ett möjligt bräde.



Figur 2.6. Kolumn med höjden sex, de tre nedersta brickorna är svarta och de tre översta vita och sparas därmed på följande sätt: *[110] [000111]*

Kapitel 3

Resultat



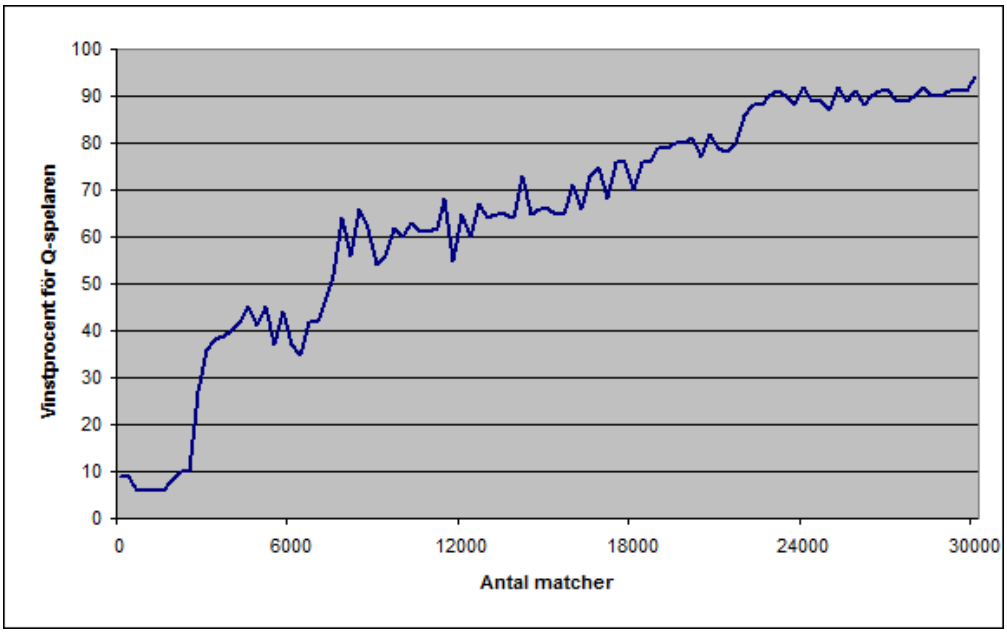
Figur 3.1. Spelet är ojämnt, vit (som börjar) har en fördel. Vinstprocent när identiska spelare tävlar mot varandra, vit avser vinstprocent för spelaren som är vit och börjar, svart avser vinstprocent för spelaren som är svart och som inte börjar.

Som nämnts i teorin så kan en spelare som börjar, och spelar rätt, forcera vinst. Men även för spelare som inte spelar perfekt märks det att det är en fördel att vara vit. Figur 3.1 visar de fem klienterna som har spelat mot kopior sig själva, för att visa vilken fördel var och en har då de spelar vit. Att respektive staplar inte summeras till 100 % beror på att resten utgörs av oavgjorda matcher.

För att inte överskatta Q-spelarens skicklighet, och även för att vara konsekvent då de olika klienternas spelar mot Q-spelaren, har vi valt att alltid låta Q-spelaren spela som svart.

3.1 Q-spelaren mot den mönstermatchande datorspelaren

Q-spelaren börjar som väntat svagt och vinner endast 5 % av matcherna från början. Men redan innan 3.000 matcher spelats ser man en rejäl ökning och strax därefter



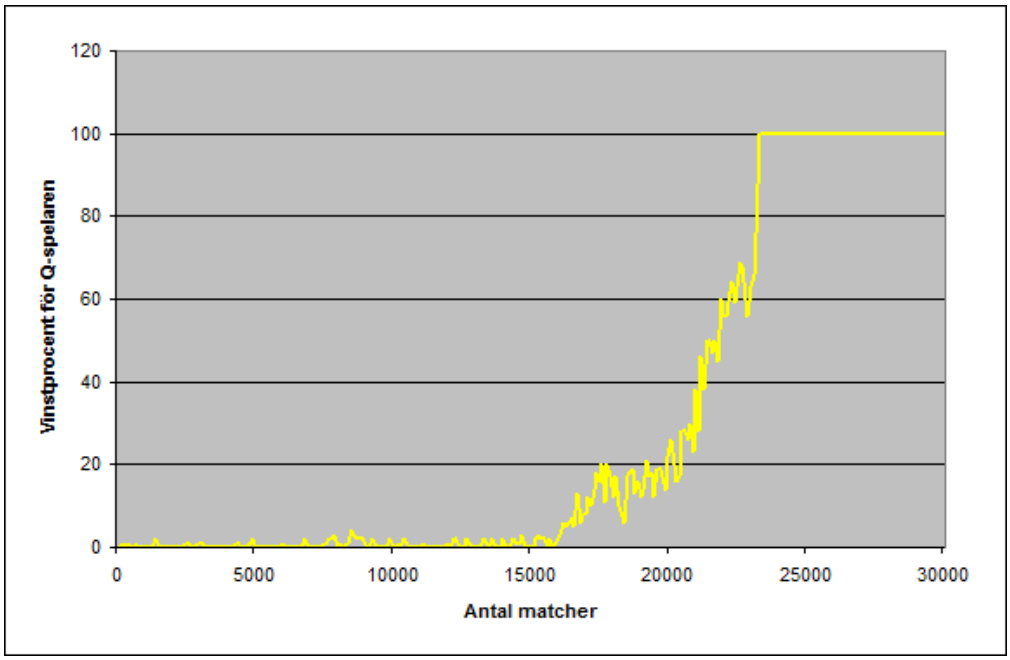
Figur 3.2. Q-spelaren mot mönstermatchande spelaren.

är Q-spelaren uppe i 40 % vinster. Efter 7.000 matcher vinner den över 50 % av matcherna och efter 23.000 matcher så vinner den runt 90 % av matcherna.

3.2 Q-spelaren mot den beräknande datorspelaren

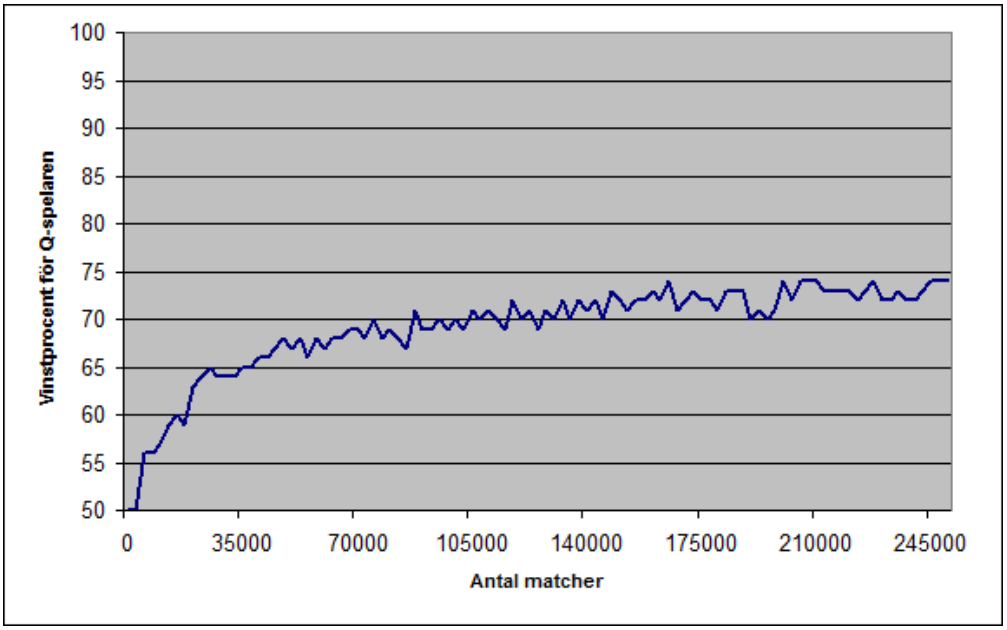
Den beräknande datorspelaren klarar att försvara sig mot Q-spelaren mycket längre innan den börjar förlora. Fortfarande efter 15.000 matcher så vinner datorspelaren i princip alla matcher. Man ser dock en stor skillnad mot den mönstermatchande datorspelaren. Eftersom den beräknande datorspelaren är näst intill deterministisk så kommer den, givet samma bräde, alltid att göra liknande drag. Det betyder att om Q-spelaren till slut lär sig att slå den beräknande spelaren, så kommer den strategin få högre poäng och börja upprepa dessa drag igen. Det slutar med att båda spelarna börjar upprepa samma drag och de går in i ett dödläge där samma parti upprepas oändligt.

Detta innebär att Q-spelaren, i det här fallet efter 24.000 matcher, har listat ut hur den ska slå den beräknande spelaren, och de hamnar i ett dödläge där samma parti upprepas. Därefter vinner Q-spelaren alltså 100 % av matcherna. Detta läge inträffar inte alltid efter 24.000 matcher utan beror helt och hållet på hur snabbt Q-spelaren har turen att börja slå den beräknande spelaren. Läget kan uppstå efter 5.000 matcher eller 50.000 matcher, men någon gång händer det. Dödläget föregås alltid av en ökning som är lik den i grafen.



Figur 3.3. Q-spelaren mot beräknande spelaren.

3.3 Q-spelaren mot en slumpmässig spelare



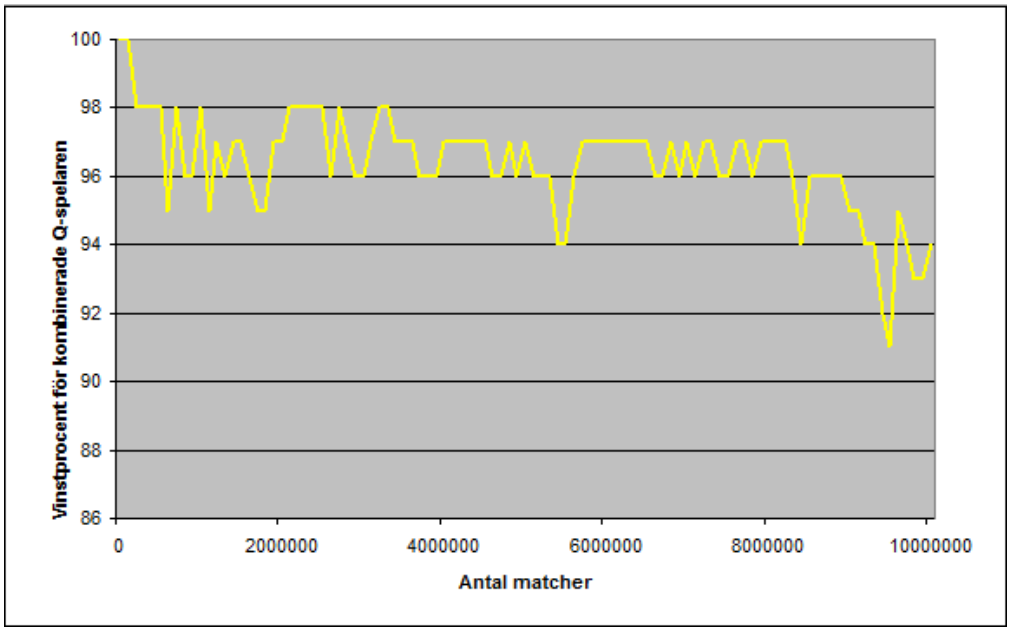
Figur 3.4. Q-spelaren mot slumpmässiga spelaren.

Det visade sig under kodandet av de icke lärande datorspelarna att ju bättre de blev, desto snabbare lärde sig Q-spelaren att slå dem. Detta var fascinerande, så för att utforska det ytterligare skapades även en helt slumpmässig spelare.

Då Q-spelaren möter den slumpmässiga spelaren får den stora problem. Eftersom även den till en början i princip slumpar sina drag, då den inte har någon data att gå på, är det till en början väldigt jämnt. Men fortfarande efter att ha spelat 100.000 matcher så vinner den bara runt 70 % av matcherna. Under 250.000 matcher ser man en tydlig ökning av vinstprocent, men mycket långsamt. Efter 250.000 matcher är den fortfarande inte över 75 % vinster. Detta är inte i närheten av de resultat den får (95 % och 100 %) mot de två ”smartare” datorspelarna efter endast 30.000 matcher.

Båda dessa datorspelare slår i sin tur den slumpmässiga spelaren konsekvent nära 100 % matcherna då de möts.

3.4 Q-spelaren mot den kombinerade Q-spelaren



Figur 3.5. Q-spelaren mot den kombinerade Q-spelaren.

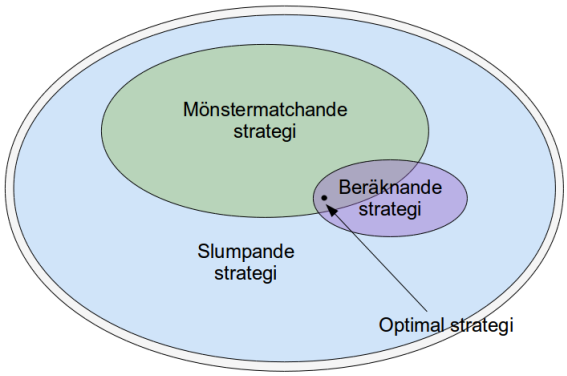
Slutligen testas den kombinerade Q-spelaren mot den vanliga Q-spelaren. Vi ser att den kombinerade Q-spelaren från början vinner i princip 100 % av matcherna. Detta är inte särskilt oväntat eftersom den använder sig av datorspelarnas beräkningar och matchningar, som själva slår Q-spelaren i början. Den vanliga Q-spelaren kör helt slumpmässigt från början medan den kombinerade Q-spelaren använder beräknade startvärden och därför har en stor fördel.

Man kan dock se att allt eftersom båda Q-spelarna lär sig, blir skillnaden något mindre. Den kombinerade Q-spelaren har hela tiden en stor fördel, men efter 1 miljon matcher så har dess vinstprocent gått ner från 100 % till ca 94 %. Båda spelarna blir bättre, men den vanliga Q-spelaren har gått från att vara helt slumpmässig i början till att ha lärt sig att spela någorlunda bra. Den inlärda förbättringen är större för den, så trots att båda har blivit bättre, har skillnaden mellan de två spelarna minskat något.

Kapitel 4

Analys

4.1 Diskussion



Figur 4.1. Mängden av alla möjliga strategier. Mindre mängd medför mer konsekvent strategi. Den optimala strategin är liten, den innehåller endast de bästa dragen.

Som det kort nämndes i resultatdelen ovan märks det tydligt att ju mer specialiserad en spelare är, desto lättare blir det för Q-spelaren att slå den i slutändan. Detta gäller oberoende av hur bra den specialiserade spelaren är. Detta eftersom det blir färre drag den måste lära sig. Majoriteten av alla möjliga drag är dåliga och en smart spelare utför därför aldrig dessa. Den mängd drag som Q-spelaren måste lära sig emot en bra spelare blir därför enormt mycket mindre än de drag den måste gå igenom för att lära sig att slå en slumpmässig spelare, som väljer drag från hela mängden av möjliga drag.

Därmed kan vi förklara vårt resultat via figur 4.1: Slumpspelaren har en stor mängd potentiella drag och kräver därmed att Q-spelaren lär sig många drag. Den mönstermatchande är mer specialiserad, har mindre mängd potentiella drag och därmed kan Q-spelaren lära sig att slå den fortare. Den beräknande är ännu mer specialiserad, och kan därför slås fortast av alla trots att den slår både slumpspelaren och den mönstermatchande spelaren.

Komplexiteten för de n första dragen som Q-spelaren måste lära sig då den möter en slumpande spelare blir i värsta fallet 7^n. Antalet drag mot en spelare som alltid väljer det bästa draget blir endast n.

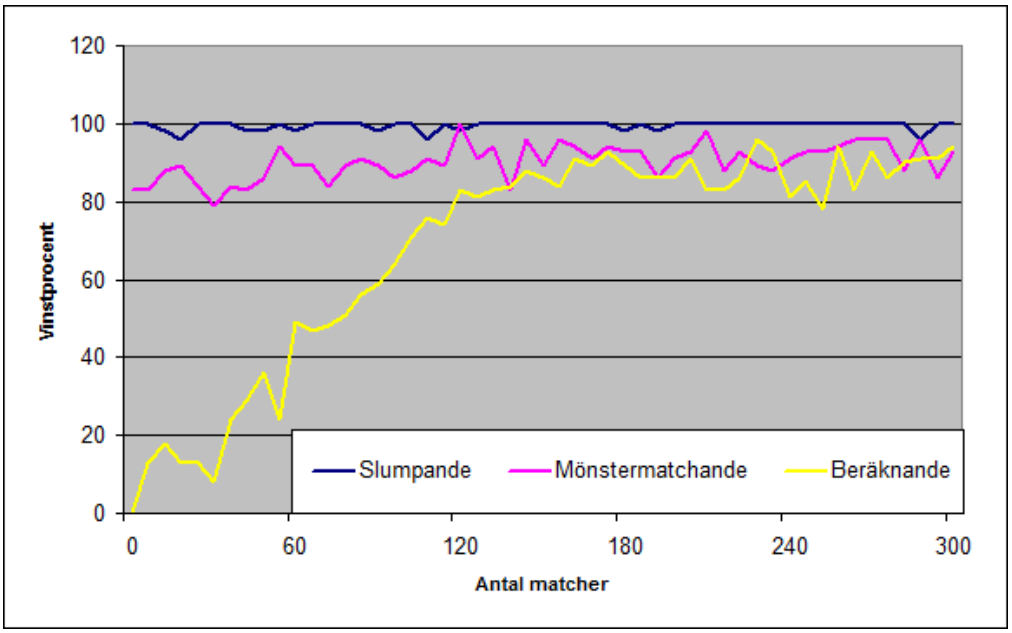
4.1.1 Q-spelaren mot den mönstermatchande datorspelaren

När man analyserar matcherna som spelades mot den mönstermatchande datorspelaren, den som endast planerar sitt eget drag, ser man att Q-spelaren relativt fort lär sig att skapa situationer där den mönstermatchande spelaren lägger en bricka, som leder till att Q-spelaren vinner. Den mönstermatchande tar ingen som helst hänsyn till den andra spelaren och dess fällor, utan bygger bara vidare på det som är bäst just detta drag.

4.1.2 Q-spelaren mot den beräknande datorspelaren

Den beräknande datorspelaren förebygger däremot dessa fällor och klarar sig, som grafen visar, mycket längre. För att slå denna spelare måste mer avancerade fällor byggas vilket tar längre tid att lära sig. När den å andra sidan har lärt sig att bygga fällor som den vinner med, kommer den att upprepa dessa mot den beräknande spelaren, som alltid gör liknande drag, och därför hamnar i en loop där samma eller liknande matcher upprepas.

4.1.3 Den kombinerade Q-spelaren



Figur 4.2. Kombinerad Q-spelare mot slumpande, matchande och beräknande.

Då den kombinerade Q-spelaren möter samma tre motståndare som den vanliga Q-spelaren, ser man tydligt dess storhet. Den slår alla tre datorspelare efter endast 300 matcher. Detta jämfört med den vanliga Q-spelaren som behövde tiotusentals matcher för samma resultat mot den beräknande och den mönstermatchande. För att slå den slumpande spelaren krävdes okörbart många matcher för Q-spelaren. Antalet vunna matcher blev endast 75 % efter 250.000 matcher. Här är den kombinerade Q-spelaren överlägsen, eftersom den vinner i princip 100 % redan från start mot den slumpande spelaren.

Den kombinerade Q-spelaren använder även avsevärt mycket mindre minne än Q-spelaren. Minnesökningen i proportion till antalet spelade matcher spelade är en bråkdel av vad som krävs av Q-spelaren. Detta därför att den kombinerade Q-spelaren ignorerar de flesta dåliga dragen, som därmed aldrig behöver sparas.

4.2 Slutsats

Slutsatser om Q-learning Vi blev förvånade över hur snabbt Q-learning faktiskt lär sig att slå datorspelarna. Som vi nämnde i vår projektspecifikation, var vi rädda att det skulle krävas så pass många matcher, att vi aldrig ens skulle få se Q-spelaren slå vare sig den mönstermatchande eller den beräknande, för att det skulle ta för lång tid eller för att minnet skulle överbelastas. Men som vi kunnat se så tog det inte ens 30.000 matcher och mindre än 10 sekunder innan de båda datorspelarna fick se sig slagna.

Det var även förvånande hur snabbt den kombinerade Q-spelaren vann över datorspelarna. Vi insåg att den skulle lära sig snabbare men så få som 300 matcher var mindre än förväntat antal.

Vi har också insett att en spelare som endast bygger på Q-learning är långt ifrån perfekt. Den får endast rimligt bra resultat om det går att simulera processen via datorer, då det krävs stora mängder av iterationer innan systemet är inlärt. Om det som ska läras exempelvis är ett krock-avvärjningssystem för bilar och det inte går att simulera är det inte praktiskt möjligt att krocka bilar tills dess att systemet fungerar. Däremot fungerar det att istället använda en icke-lärande algoritm som bas, då det krävs förvånansvärt få fall för att få en anmärkningsvärd förbättring.

Framtida undersökningsmöjligheter Något intressant vi funderade på, men som låg utanför vårt eget projekt, är att låta Q-spelaren träna mot en speciell typ av spelare, men att sedan möta en annan. Hur skulle Q-spelaren prestera om den tränade mot en slumpmässig spelare och sedan mötte en mänsklig spelare?. Alternativt att träna mot en av våra “smarta” spelare för att sedan möta en slumpmässig spelare.

Något annat vi skulle velat göra, men som låg utanför vårt kompetensområde, är att testa Q-learning mot en spelare som använder sig av ett neuralt nätverk. Vi ville inledningsvis testa detta, men eftersom vi inte är tillräckligt insatta i neurala

nätverk, ansåg vi att vi inte kunde garantera ett resultat, och att det därmed inte skulle varit ett realistiskt projekt inom den tidsram vi hade.

Slutord Vi testade slutligen att själva möta Q-spelaren efter att ha låtit den träna 50.000.000 matcher. Att vi faktiskt blev slagna tio matcher i rad säger väl förvisso mer om oss än om Q-spelaren, men samtidigt är det utan tvekan fascinerande att bli slagen av sin egen skapelse.

Kapitel 5

Källförteckning

2 Wikipedia, *Machine Learning* (hämtad: 4 mars 2011).
URL: http://en.wikipedia.org/wiki/Machine_learning

3 Wikipedia, *Reinforcement Learning* (hämtad: 2 mars 2011).
URL: http://en.wikipedia.org/wiki/Reinforcement_learning

4 Michael Gasser (hämtad: 2 mars 2011). Introduction to Reinforcement Learning.
URL: http://www.cs.indiana.edu/ gasser/Salsa/rl.html

5 Richard S. Sutton & Andrew G. Barto (hämtad: 3 mars 2011). Reinforcement Learning: an Introduction.

6 Victor Allis, *A Knowledge-based Approach of Connect-Four*. Department of Mathematics and Computer Science, Amsterdam, 1988.
URL : http://www.connectfour.net/Files/connect4.pdf

- Edvin Ekblad & Oskar Werkelin Ahlin. *Implementation av Q-learning för fyra-i-rad*. School of Computer Science and Communication, Stockholm, 2010.
URL: http://www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/rapport/ekblad_edvin_OCH_werkelin_ahlin_oskar_K10055.pdf

Kapitel 6

Bilagor

Källkod

URL: http://code.google.com/p/dd143x-project-janse-hassel/