Uppsats nr: BCS-2015-06

Urvalsbaserad evalueringsfunktion

Karin Svensson

Fakulteten för datavetenskaper
Blekinge Tekniska Högskola
SE–371 79 Karlskrona, Sweden

Denna uppsats är en del av examinationen i kursen DV1478 Kandidatarbete i datavetenskap.

Kontaktinformation:
Författare:
Karin Svensson
E-mail: kasp11@student.bth.se

Handledare:
Hans Tap
Institutionen för kreativa teknologier

Fakulteten för datavetenskaper : www.bth.se
Blekinge Tekniska Högskola Tel : +46 455 38 50 00
SE–371 79 Karlskrona, Sweden Fax : +46 455 38 50 57

Abstract

Kontext. Evalueringsfunktioner är en viktig del inom artificiell intelligens. En evalueringsfunktion utvärderar ett spelläge och är svår och tidskrävande att utveckla. Inför denna uppsats utvecklades en ny teknik som kan användas på en evalueringsfunktion för att skapa en urvalsbaserad evalueringsfunktion.
Mål och Objektiv. I denna uppsats utvärderas en ny teknik för att förbättra evalueringsfunktioner för Kalaha. Tekniken bygger på att utvärdera spelläget både som det är och genom att sampla möjliga framtida drag. Teknikens framgångar mäts i antal vinster mot andra evalueringsfunktioner.
Metod. Detta arbete bygger på en implementationsmetod där kvantitativ data samlas in för analys. Spelet och de artificiella intelligenserna utvecklades i C++ med hjälp av Microsoft Visual Studio 12.
Resultat. Utfallet från matcher mellan evalueringsfunktioner och urvalsbaserade evalueringsfunktioner sammanställdes till tabeller.
Slutsats. Den urvalsbaserade tekniken hade framgångar i matcherna och anses därför vara en lyckad förbättring av evalueringsfunktionerna som användes.

Nyckelord: Evalueringsfunktion, Artificiell Intelligens, Kalaha

Abstract

Context. Evaluation functions are an important part of artificial intelligence. An evaluation funtion estimates utility of a game state and is difficult and time consuming to develop. A new technique to improve evaluation function was developed in order to write this thesis.
Objectives. This thesis investigates a new technique of improving evaluation functions for the game Kalaha. The technique evaluate a game state by using an evaluation function on the existing game state and on samples of future game states. The success of the sample based evaluation function is measured in number of wins.
Methods. This thesis uses an implementation method where quantitative data is collected for analysis. The game and the artificial intelligences where developed in C++ using Microsoft Visual Studio 12.
Results. Different evaluation function played against sample based evaluation functions and the out coming was collected and presented in tables.
Conclusion. The sample based evaluation function had a higher rate of winnings and were therefore a successful improvement of the evaluation functions.

Key Words: Evaluation function, Artificial Intelligence, Kalaha

Contents

Chapter 1

Inledning

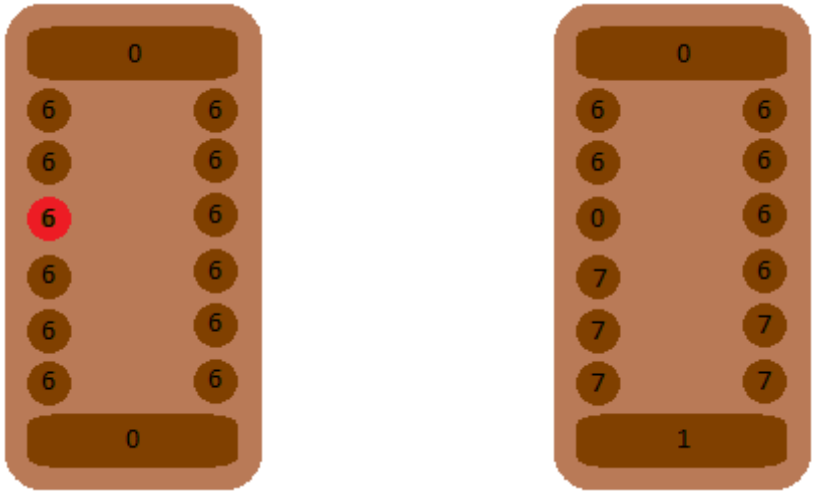
I denna uppsats utvärderas en teknik för evalueringsfunktioner. Tekniken är ett försök till att generellt förbättra andra evalueringsfunktioner genom att sampla värden från framtida drag och har utvecklats specifikt för detta arbete. Under rubriken Urvalsbaserad evalueringsfunktion förklaras tekniken närmre.

1.1 Kalaha

Spelet Kalaha uppfanns 1940 av William Julius Champion och har gått under många andra namn, såsom Kalah och Mancala[5]. Kalaha spelas av två personer på en träbräda med ett antal små hål på varje långsida och 1 större hål på varje kortsida. I början av spelet har spelarna lika många stenar i varje litet hål och inga i de två större hålen. Man kan beteckna Kalaha genom (m, n) där m är antalet små hål per spelare och n är antalet stenar i varje hål i startläget.

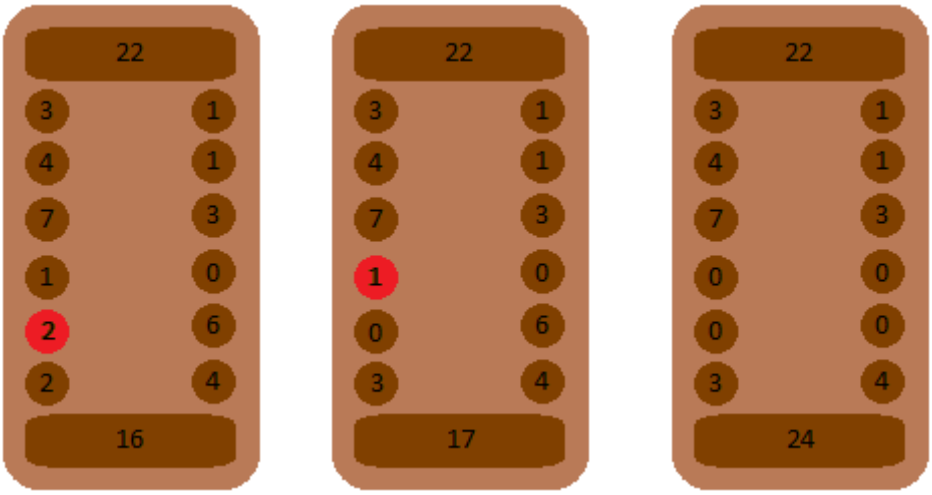
Kalaha finns i många olika versioner och många har egna husregler[5]. De regler som denna uppsats använder är Algas spelregler[9]. Varje spelare har 6 hål och ett näste. Startläget är att spelarna har 6 stenar i alla sina hål och inga i nästena, sammanlagt 72 stenar. En spelares drag är att tömma ett av hålen och sedan lägga en sten i varje efterkommande hål och i sitt eget näste tills stenarna tar slut. Denna rörelse sker motsols. Exempel på ett första drag ses i figur 1.1.

Om ett drags sista sten landar i spelarens näste får denna ett extra drag. Om den sista stenen landar i ett tomt hål på spelarens egna sida töms detta hål, och hålet på motsatt sida, i spelarens näste. Detta kallas att 'fånga' en skål. När en spelare inte längre har några möjliga drag avslutas spelet. Spelarna lägger då de resterande stenarna från sina hål i sitt eget näste. I figur 1.1-1.3 visas exempel på ett par olika drag. De rödmarkerade hålen är draget som kommer att utföras. Alla drag är gjorda av vänster spelare.



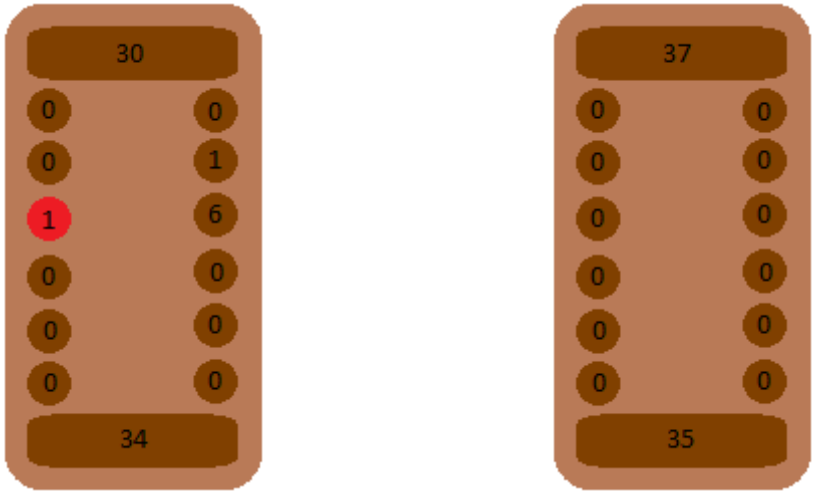
Startläge Efter ett drag

Figure 1.1: Exempel på ett drag



Extradrag Fångad skål Efter dragen

Figure 1.2: Extradrag och drag som fångar en motståndarskål



Sista draget Slutbräde

Figure 1.3: Exempel på slut

1.2 Tidigare forskning

Kalaha är ett klassiskt spel inom artificiell intelligens. Det hade sin stortid på 60-talet, men efter att det löstes 2000 fick det nyhetsvärde[4].

Spelet kan beskrivas som ett deterministiskt, turordningsnollsummespel med fullständig information, för två spelare, precis som schack[10]. Deterministiskt innebär att det är fullt förutsägbart, nollsummespel innebär att en spelare vinner på att den andra spelaren förlorar och fullständig information innebär att all information är känd, det finns inga dolda faktorer.

1.2.1 Lösa Kalaha

Kalaha är löst för alla kombinationer upp till och med (6, 6). "In a Kalaha context, one could interpret it as constructing an AI which never loses when being the starting player, no matter if the other player is an AI or a human."[7] Detta citat säger att Kalaha är löst när man som förstaspelare kan bestämma resultatet i matchen. Man skulle kunna kalla det för den ultimata AIn för spelet. Under rubriken Sökalgoritmer och evalueringsfunktioner förklaras det varför detta inte alltid är det självklara valet för en AI i ett spel.

Alla kombinationer upp till och med (6, 5) löstes år 2000 [8]. För att göra detta användes bland annat en minimaxalgoritm med iterativ fördjupning. Denna optimiserades med många olika tekniker såsom en slutspelsdatabas. Evalueringsfunktionen som användes var samma funktion som i denna uppsats kallas enkel evalueringsfunktion. Det tog ytterligare 11 år före Kalaha(6, 6) löstes[1].

Denna uppsats menar inte att lösa Kalaha igen. Istället inriktas arbetet och analysen på en mindre krävande AI med syftet att spela mot människor. När en människa spelar mot en AI måste man balansera AIn så att den är utmanande, men inte för svår. Detta skapar 'flow' och är eftertraktat när man spelar[2].

1.2.2 Sökalgortimer och Evalueringsfunktioner

För att göra en AI till brädspel används nästan alltid en sökalgoritm och en evalueringsfunktion. Evalueringsfunktionen värderar utkomsten av olika drag och sökalgoritmen bestämmer det bästa draget. Den vanligaste sökalgoritmen till detta är minimax[7].

Minimax bygger upp ett träd för alla möjliga drag, evaluerar löven och bestämmer sedan bästa draget genom att gå nerifrån och upp och anta att den aktuella spelaren alltid väljer det högst värdesatta draget för honom och motspelaren väljer det minst värdesatta draget. I ett nollsummespel innebär detta att spelaren alltid väljer det högst värdesatta draget för sig själv, eftersom vad som är bra för den ena är dåligt för den andra. Figur 1.4 visar ett minimaxträd där endast två drag kan göras varje tur.

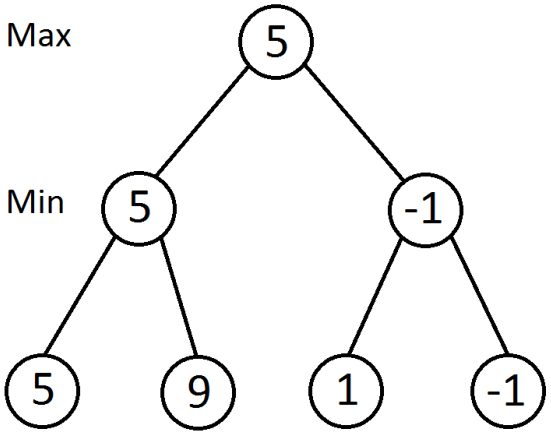


Figure 1.4: Ett minimaxträd

När man söker igenom hela spelträdet efter nästa drag kan man alltid röra sig mot en vinst. Om man inte har möjligheten att söka igenom hela trädet behöver man istället estimera vinstchansen efter ett drag. Att uppskatta detta är evalueringsfunktionens uppgift. Värdet evalueringsfunktionen ger är enbart en uppskattning av vinstchansen, inte helt säkert det bästa draget eftersom enbart slutnoderna kan visa det.

För att en evalueringsfunktion ska funka behöver denna uppfylla vissa krav. Dels ska den vara snabb, eftersom den annars hade kunnat ersättas med andra alternativ, dels måste den vara starkt sammankopplad till vinstchansen och dels

måste den evaluera utfallen i rätt ordning. Det sista betyder att en vinst i evalueringsfunktionen alltid måste vara högre värderad än oavgjort som måste vara högre värderat än en förlust[10]. Evalueringsfunktioner ser olika ut för olika spel och att utveckla en funktion som funkar för just det spelets AI är ett av de stora problemen inom artificiell intelligens[10][3].

I denna uppsats används ordet faktor för att beskriva hur en evalueringsfunktion är uppbyggd. När termen används betyder det att det är en av de saker som poängsätts. Till exempel kan antal stenar i ett näste vara en faktor eller att det finns 13 stenar i ett hål.

Eftersom Kalaha är deterministiskt och har perfekt information kan man teoretiskt sett alltid bygga ett komplett spelträd och göra de bästa dragen utifrån slutresultatet. I praktiken är detta inte alltid möjligt, framför allt för att trädet tar extremt mycket minne och det tar tid att bygga.

Kalaha(6, 6) löses aldrig på under 30 turer[8], varje tur har 6 möjliga drag (mindre om det finns tomma hål) och för varje drag måste man spara minst en integer, 4 bytes, för att kunna använda trädet för en sökning. Detta skulle innebära $6^{30} * 4$ bytes i minne, alltså cirka 900 miljarder terabytes. Dagens datorer klarar inte detta och därför måste man utveckla AI på andra sätt. Ett sätt att göra detta är att avbryta minimaxsökningen efter en viss tid och evaluera spelläget därifrån. Som exempel brukar en vanlig minimaxalgoritm klara 10 drag ner i Kalahaträdet på 30 sekunder om det sker med iterativ fördjupning, alltså tur för tur[7].

1.3 Mål och objektiv

Målet med uppsatsen är att förbättra en viss evalueringsfunktion genom att utveckla en evalueringsfunktion som är baserad på ett urval av möjliga framtida drag. För att göra detta kommer uppsatsen att bygga på en implementationsmetod där kvantitativ data samlas in för analys.

Först kommer den nya urvalsbaserade evalueringsfunktionen att utvecklas. Den kommer sedan att spela mot tre olika evalueringsfunktioner. För att framhäva evalueringsfunktionerna kommer ingen ytterligare algoritm, såsom minimax, att användas.

Spelet som de olika funktionerna kommer att testas i är Kalaha $(6, 6)$. Efter detta kommer resultatet från matcherna att sammanställas för att bestämma vilka evalueringsfunktioner som vann flest gånger och över vilka andra evalueringsfunktioner de vann.

Till slut kommer sammanställningen att analyseras och slutsatser kommer att dras för att besvara forskningsfrågan.

1.4 Forskningsfråga

Är en urvalsbaserad evalueringsfunktion med slumpfaktor en förbättring av en viss evalueringsfunktion i hänsyn till antal vinster i Kalaha$(6, 6)$?

1.5 Evalueringsfunktioner

I detta arbete används tre evalueringsfunktioner, som även utökas med en ny teknik som här kallas en urvalsbaserad evalueringsfunktion. Tillsammans blir detta sex olika evalueringsfunktioner. Längre ner kommer en mer ingående förklaring av de tre evalueringsfunktionerna och den urvalsbaserade evalueringsfunktionen.

De tre evalueringsfunktionerna är valda av olika anledningar. Den första, enkel evalueringsfunktion, valdes för att den är vältestad och simpel. Meningen med denna är att den ska ge ett säkert underlag för att kunna bestämma att den urvalsbaserade funktionen faktiskt är en förbättring i antalet vinster.

Den andra, självmedveten evalueringsfunktion, och den tredje, motståndarmedveten evalueringsfunktion, valdes för att den självmedvetna visade sig vara mer framgångsrik än den enkla i ett tidigare test[7] och den motståndarmedvetna är en variant av den självmedvetna och borde vara bäst av de tre. Meningen med dessa är framför allt att ha något bättre än enkel evalueringsfunktion för att jämföra urvalsbaserad enkel mot. De kommer också att användas som bas för den urvalsbaserade funktionen, för att försäkra att eventuella förbättringar vid urvalsbaserad enkel inte är en engångsföreteelse.

1.5.1 Enkel evalueringsfunktion

Enkel evalueringsfunktion är en funktion baserad på enbart en faktor, poäng i nästena. Den returnerar spelarens näste subtraherat med motspelarens näste. Tabell 1.1 visar hur poängen räknas.

Denna funktion är den mest testade av alla evalueringsfunktioner som används i denna uppsats. Den användes i lösningarna som gjordes 2000[8] och den har bevisats vara den viktigaste faktorn för att vinna i jämförelse med flera andra faktorer[6].

Det bästa med denna evalueringsfunktion är att den är vältestad och att den är simpel. Nackdelen är att den inte värderar resten av spelplanen. Man riskerar att göra ett drag som ger spelaren 1 poäng, men ger motspelaren möjligheten att ta 10 poäng.

Faktor	**Poäng**
1 sten i nästet	1
1 sten i motståndarnästet	-1

Table 1.1: Poängsättning för enkel evalueringsfunktion

1.5.2 Självmedveten evalueringsfunktion

Den självmedvetna evalueringsfunktionen är baserad på spelarens egna sida av brädet. Den ger poäng för varje poäng i nästet och för bra lägen strategiskt sett, såsom 13 stenar i ett hål och tomma hål. Tabell 1.2 visar hur poängen räknas.

Denna funktion har tidigare haft framgångar mot den enkla evalueringsfunktionen[7], men har inte testats lika ingående. Framgångarna borde bero på att den utöver samma faktor som den enkla funktionen är baserad på också värderar sin sida av spelplanen.

Nackdelarna med denna funktion är att den enbart ser på sin sida av spelplanen, inte motspelarens, och att den inte räknar med risken att motspelarens drag ändrar spelplanen. Det finns alltid en risk att motspelaren lägger en kula i ett av spelarens tomma hål eller ändrar räkningen från 13 till 14 stenar i ett hål, vilket neutraliserar de positiva aspekterna.

Faktor	**Poäng**
1 sten i nästet	1
Hål med 13 stenar	1
Hål med antal stenar som leder till ett extra drag	0.5
Tomma hål	0.25

Table 1.2: Poängsättning för självmedveten evalueringsfunktion

1.5.3 Motståndarmedveten evalueringsfunktion

Den motståndarmedvetna evalueringsfunktionen är baserad på hela spelbrädet. Den fungerar liknande den spelarmedvetna, men för båda sidor av planen och då med negativa värden för motspelarens strategiskt bra lägen. Tabell 1.3 visar hur poängen räknas.

Eftersom denna funktion inte har vissa av nackdelarna som den spelarmedvetna funktionen har bör den vara den bästa av de tre evalueringsfunktionerna, även om detta inte är bevisat då den inte tidigare har blivit testad. Fördelen med denna funktionen är att den värderar hela spelplanen, men den har fortfarande nackdelen att den inte räknar med att vissa drag av motspelaren eventuellt skulle

kunna ändra spelarens läge.

Faktor	**Poäng**
1 sten i nästet	1
Hål med 13 stenar	1
Hål med antal stenar som leder till ett extra drag	0.5
Tomma hål	0.25
1 sten i motståndarnästet	-1
Motståndarhål med 13 stenar	-1
Motståndarhål med antal stenar som leder till ett extra drag	-0.5
Tomma motståndarhål	-0.25

Table 1.3: Poängsättning för motståndarmedveten evalueringsfunktion

1.5.4 Urvalsbaserad evalueringfunktion

Den urvalsbaserade evalueringsfunktionen är en ny teknik för att förbättra alla evalueringsfunktioner. Eftersom den är uppfunnen och testad för detta arbete finns det ingen tidigare forskning på funktionen.

I denna uppsats kommer de urvalsbaserade evalueringsfunktionerna att vara baserad på alla tre ovan nämnda funktioner, en efter en. Det funktionen gör är att sampla senare spellägen och använda dessa som en indikation om det strategiska läget.

Mer ingående kommer den urvalsbaserade evalueringsfunktionen att basera 70% av det slutgiltiga värdet för spelplanen på orginalfunktionen. De övriga 30% kommer att beräknas genom att evalueringsfunktionen, för varje möjligt drag, slumpar motståndarens nästa drag, sedan slumpar sitt eget nästkommande drag därifrån. I detta läget kommer den sedan att evaluera draget med orginalfunktionen. Efter att ha gjort detta 3 gånger räknar den ut medelvärdet på spellägena och detta är alltså spellägets återstående 30%.

70% och 30% valdes efter att ha testat alla kombinationer av tiotal. Denna kombination var då bäst.

Fördelar med denna teknik skulle vara att det är ett implementationsmässigt snabbt sätt att förbättra en AI till en relativt billig cpu-kostnad. Att utveckla en evalueringsfunktion kan också vara en subjektiv uppgift eftersom programmeraren på någon nivå måste bestämma faktorerna. Denna teknik kan eliminera vissa delar av detta genom att maximera de självklara eller testade faktorerna.

Nackdelen är att den är slumpbaserad och därför kan ge vilseledande värden.

Chapter 2

Metod

2.1 Implementation

Det Kalahaprogram som denna uppsats är byggd på är implementerat i C++. Det använder konsollfönstret för output och spelarna hårdkodas i spelet. Spelarna kan vara människor eller olika AI som har programmerats i projektet. Spelarna behöver inte använda samma spelsätt utan en människa kan spela mot en AI, två olika AI mot varandra och så vidare.

Varje AI ligger som tre separata funktioner. En funktion som poängsätter spelbrädet, en funktion som beräknar utkomsten av extradrag och en central funktion som väljer nästa drag. De två sistnämnda ser likadana ut för alla AI förutom att de anropar sin egna funktion för poängsättning och beräkning av extradrag.

För att beskriva skillnaden mellan de vanliga evalueringsfunktionerna och den urvalsbaserade visas koden för den motspelarmedvetna evalueringsfunktionen och den urvalsbaserade evalueringsfunktionen nedan.

Den motspelarmedvetna evalueringsfunktionen tar in spelbrädet, int holes[], och antal genomförda drag, int level, som parametrar. Den avgör sedan vilken spelares tur det är med hjälp av antalet genomförda drag. Låt oss i detta fall säga att det är första spelarens tur, alltså if(level%2 == 1) är sann. Då kommer funktionen att för alla spelarens hål, 0-5, addera poäng för bra lägen, och för motspelarens hål, 7-12, subtrahera poäng för bra lägen. Detta tillsammans med skillnaden mellan nästena blir spellägets poäng som returneras.

```
float OpponentAwarePoints(int holes[], int level)
{
    float result=0;
    if(level%2 == 1){
        for(int i=0; i <6; i++){
            if(holes[i] == 13)
                result++;
            if(holes[i] == 6-i)
                result+=0.5;
```

```
            if(holes[i] == 0)
                result+=0.25;
        }
        for(int i=7; i<13; i++){
            if(holes[i] == 13)
                result−;
            if(holes[i] == 13-i)
                result-=0.5;
            if(holes[i] == 0)
                result-=0.25;
        }
        result+=(holes[6]-holes[13]);
    }
    else{
        result+=(holes[13]-holes[6]);
        for(int i=7; i<13; i++){
            if(holes[i] == 13)
                result++;
            if(holes[i] == 13-i)
                result+=0.5;
            if(holes[i] == 0)
                result+=0.25;
        }
        for(int i=0; i<6; i++){
            if(holes[i] == 13)
                result−;
            if(holes[i] == 6-i)
                result-=0.5;
            if(holes[i] == 0)
                result-=0.25;
        }
    }
    return result;
}
```

Den urvalsbaserade funktionen tar in spelbrädet, int holes[], och nivån, int level, som parametrar. Den avgör sedan vilket spelares tur det är med hjälp av nivån. Låt oss i detta fall säga att det är spelare 1, alltså if(level%2 == 1) är sann. Tre gånger om, för att få tre prov, kommer funktionen att slumpa ett möjligt drag för motspelaren, slumpa ett möjligt drag för spelaren och värdesätta spelläget med, i det här fallet, den motståndarmedvetna evalueringsfunktionen. Den kommer sedan att evaluera inputsbrädet med samma evalueringsfunktion, multiplicera med 7 och lägga till medelvärdet av proven multiplicerat med 3. Detta är

returneringsvärdet.

```
float SBPointsDuo(int holes[], int level, int startPosition)
{
    float result;
    float meanValue=0;
    int argHoles[14];
    int nextMove=0;
    int availableMoves=0;
    int availableHoles[6];
    if( level%2 == 1 ){
        result = KBDuoPoints(holes, level, startPosition);
        for(int i=0; i<3; i++){
            for(int j=0; j<14; j++)
                argHoles[j]=holes[j];
            if(PossibleMove(argHoles)){
                availableMoves=0;
                for(int k=7; k<13; k++){
                    if(argHoles[k] != 0){
                        availableHoles[availableMoves]=k;
                        availableMoves++;
                    }
                }
                nextMove=availableHoles[rand()%availableMoves];
                while(Move(nextMove, argHoles, level+1) && PossibleMove(argHoles)){
                    availableMoves=0;
                    for(int k=7; k<13; k++){
                        if(argHoles[k] != 0){
                            availableHoles[availableMoves]=k;
                            availableMoves++;
                        }
                    }
                    nextMove=availableHoles[rand()%availableMoves];
                }
                if(PossibleMove(argHoles)){
                    availableMoves=0;
                    for(int k=0; k<6; k++){
                        if(argHoles[k] != 0){
                        availableHoles[availableMoves]=k;
                        availableMoves++;
                    }
                }
                nextMove=availableHoles[rand()%availableMoves];
```

```
                while(Move(nextMove, argHoles, level) && PossibleMove(argHoles)){
                    availableMoves=0;
                    for(int k=0; k<6; k++){
                        if(argHoles[k] != 0){
                            availableHoles[availableMoves]=k;
                            availableMoves++;
                        }
                        }
                        nextMove=availableHoles[rand()%availableMoves];
                    }
                }
            }
            meanValue+=KBDuoPoints(argHoles, level, startPosition);
        }
        meanValue/=3;
        result*=7;
        result+=meanValue*3;
    }
    else{
        result=KBDuoPoints(holes, level, startPosition);
        for(int i=0; i<3; i++){
            for(int j=0; j<14; j++)
                argHoles[j]=holes[j];
            if(PossibleMove(argHoles)){
                availableMoves=0;
                for(int k=0; k<6; k++){
                    if(argHoles[k] != 0){
                        availableHoles[availableMoves]=k;
                        availableMoves++;
                    }
                }
                nextMove=availableHoles[rand()%availableMoves];
                while(Move(nextMove, argHoles, level+1) && PossibleMove(argHoles)){
                    availableMoves=0;
                    for(int k=0; k<6; k++){
                        if(argHoles[k] != 0){
                            availableHoles[availableMoves]=k;
                            availableMoves++;
                        }
                    }
                    nextMove=availableHoles[rand()%availableMoves];
                }
                if(PossibleMove(argHoles)){
```

```
                availableMoves=0;
                for(int k=7; k<13; k++){
                    if(argHoles[k] != 0){
                        availableHoles[availableMoves]=k;
                        availableMoves++;
                    }
                }
                nextMove=availableHoles[rand()%availableMoves];
                while(Move(nextMove, argHoles, level) && PossibleMove(argHoles)){
                    availableMoves=0;
                    for(int k=7; k<13; k++){
                        if(argHoles[k] != 0){
                            availableHoles[availableMoves]=k;
                            availableMoves++;
                        }
                    }
                    nextMove=availableHoles[rand()%availableMoves];
                }
            }
        }
        meanValue+=KBDuoPoints(argHoles, level, startPosition);
    }
    meanValue/=3;
    result*=7;
    result+=meanValue*3;
  }
  return result;
}
```

2.2 Mätning

Data samlades genom att alla sex evalueringsfunktioner, enkel, självmedveten, motståndarmedveten, urvalsbaserad enkel, urvalsbaserad självmedveten och urvalsbaserad motståndarmedveten, spelade mot alla sex evalueringsfunktioner. Detta innebär $(6 * 6 = 36)$ kombinationer. Alla kombinationer spelades 10 000 gånger.

Matcherna kunde avslutas i vinst, oavgjort eller förlust.

Efter att alla matcherna i en kombination spelats beräknades en vinstprocent för förstaspelaren. Detta räknades ut genom funktionen *vinstprocent = (antalet vunna matcher + antalet oavgjorda matcher / 2) / antalet matcher.* Om

förstaspelaren vann 7500 matcher, spelade oavgjort i 200 matcher och förlorade 2300 matcher fick denne en vinstprocent på $(7500 + (200/2))/10000 = 76\%$.

Denna funktion är utarbetad för att räkna med de oavgjorda matcherna. Hälften av de oavgjorda matcherna räknas nu som vinster och hälften som förluster.

Chapter 3

Resultat

För att presentera resultaten används olika förkortningar som förklaras i tabell 3.1.

Förkortning	**Evalueringsfunktion**
Enkel	Enkel
Själv	Självmedveten
Mot	Motståndarmedveten
U Enkel	Urvalsbaserad Enkel
U Själv	Urvalsbaserad Självmedveten
U Mot	Urvalsbaserad Motståndarmedveten

Table 3.1: Förklaring av förkortningar

Grunden till alla resultat är vinstprocenten för de olika funktionerna. Tabell 3.2 visar vinstprocenten avrundat till två decimaler. Kolumnerna är förstaspelaren och raderna är andraspelaren. Siffrorna är kopplade till kolumnen. I matchen mellan Enkel och Mot var alltså Enkel förstaspelare och hade en vinstprocent på 62,07%.

	Enkel	**Själv**	**Mot**	**U Enkel**	**U Själv**	**U Mot**
Enkel	74,93	70,98	84,29	85,55	81,17	85,24
Själv	75,72	71,81	81,66	85,59	82,44	84,56
Mot	61,97	57,6	68,85	73,92	70,03	73,49
U Enkel	59,14	51,82	62,83	70,77	64,43	68,81
U Själv	61,97	57,93	68,65	73,78	68,19	71,78
U Mot	54,84	51,47	65,78	68,89	62,2	68,31

Table 3.2: Vinstprocent

Baserat på vinnarprocenten är tabell 3.3 en sammanställning på mest vinnande till minst vinnande spelare, först som förstaspelare och sedan som andraspelare. Det är beräknat genom ett medelvärde av funktionens samtliga matcher som berörd spelare. Enkel får alltså $(75,01+75,85+62,07+57,34+61,69+57,35)/6 = 64,885$ som jämförningsvärde som förstaspelare.

Förstaspelare	**Andraspelare**
U Enkel	U Mot
U Mot	U Enkel
Mot	U Själv
U Själv	Mot
Enkel	Själv
Själv	Enkel

Table 3.3: Högst till lägst baserat på medelvärdet av vinstprocent

Tabell 3.4 sätter evalueringsfunktionerna med tillhörande urvalsbaserade version mot varandra.

	Orginal	**Urvalsbaserad**
Enkel Förstaspelare	64,76	76,42
Enkel Andraspelare	19,64	37,03
Själv Förstaspelare	60,27	71,41
Själv Andraspelare	19,70	32,95
Mot Förstaspelare	72,01	75,37
Mot Andraspelare	32,36	38,09

Table 3.4: Orginalversion mot urvalsbaserad funktion mätt i medelvärdet av vinstprocent

Chapter 4

Analys och diskussion

Under denna rubrik kommer resultatet att analyseras och det kommer även att föras en generell diskussion om arbetet. Analysen är baserad på tabellerna 3.2-3.4.

4.1 Den bästa evalueringsfunktionen

Den bästa evalueringsfunktionen, baserat på antal vunna matcher, är den enkla evalueringsfunktionen med den urvalsbaserade tekniken som förstaspelare och den motståndarmedvetna evalueringsfunktionen med den urvalsbaserade tekniken som andraspelare.

Efter den enkla urvalsbaserade funktionen, som förstaspelare, kom båda motståndarmedvetna funktionerna. Man skulle därför kunna anta att den motståndarmedvetna funktionen är bättre än den enkla funktionen, men att kombinationen enkel och urvalsbaserad ger större förbättring än kombinationen motståndarmedveten och urvalsbaserad.

Eftersom den självmedvetna funktionen förlorade mest som förstaspelare och mycket som andra spelare och den var den urvalsbaserade funktionen med lägst värden kan man anta att den var den sämsta av de tre orginalfunktionerna.

4.2 Urvalsbaserade evalueringsfunktionen

För att anta att den urvalsbaserade tekniken är en förbättring på evalueringsfunktionerna måste man framför allt jämföra orginalfunktionerna med de urvalsbaserade funktionerna.

Tabell 3.4 visar tydligt att de urvalsbaserade funktionerna har högre antal vinster än orginalfunktionerna. I tabell 3.2 kan man också läsa av att de urvalsbaserade funktionerna har högre vinstantal.

Om man ser på kombinationer mellan urvalsbaserade och icke urvalsbaserade funktioner har de urvalsbaserade nästan alltid högre siffror mot en orginalfunktion än vad orginalfunktionen har mot den urvalsbaserade. Detta betyder att till exempel enkel mot urvalsbaserad självmedveten har lägre vinstprocent än vad urvalsbaserad självmedveten har mot enkel.

Utöver detta kan man också läsa av att de urvalsbaserade har första, andra och fjärde plats i tabell 3.3, medan orginalen har tredje, femte och sjätte plats.

4.3 Oförväntade resultat och förbättringar

Vid planeringen av evalueringsfunktionerna förutspåddes det att den motståndarmedvetna skulle vara bäst, sedan den självmedvetna och sist den enkla. Detta visar inte resultaten.

En förklaring skulle kunna vara att de har testats tillsammans med en minimaxalgoritm och därmed har skillnaden av antal stenar i nästena varit mer markant. Om det värdet är större tar resten av spelbrädet mindre del av poängen för spelläget i den självmedvetna och motståndarmedvetna. Detta kan innebära att den självmedvetna funktionen valde drag som gav ett bra spelläge framför drag som gav mycket poäng.

Precis som beskrivet i inledningen finns risken att motspelaren förstör spelarens planer med sitt drag och därför kan den självmedvetna bli sämre än den enkla evalueringsfunktionen.

Enligt teorin bakom den urvalsbaserade tekniken borde rangordningen på de urvalsbaserade funktionerna vara samma som orginalfunktionerna. Detta hände inte då den motståndarmedvetna var bätte än den enkla i orginalversionen, men tvärt om i den urvalsbaserade varianten i fallet som förstaspelare.

Anledningen till detta kan vara att två led framåt är mer optimalt för faktorn, skillnad i nästena, än vad det är för de övriga faktorerna i den motståndarmedvetna funktionen.

Chapter 5

Slutsats

De urvalsbaserade evalueringsfunktionerna var bättre än orginalversionerna i hänsyn till antalet vinster. Det var en markant skillnad i vinstantal vid alla tre funktioner.

Chapter 6

Framtida arbete

Den urvalsbaserade teknik som användes i detta arbete var en lyckad förbättring i antalet vinster i Kalaha, men för att säkert kunna säga att det är en generell förbättring av evalueringsfunktioner bör den också testat på andra spel.

Det vore också intressant att se hur väl de urvalsbaserade evalueringsfunktionerna fungerar tillsammans med en minimaxalgoritm, som är den vanligaste algoritmen att grunda en AI på.

Eftersom den urvalsbaserade tekniken minst behöver en evalueringsfunktion enbart baserad på en faktor för att fungera, bör generella metoder att utforma dessa utforskas.

Man bör också jämföra exekveringstiden för denna funktion jämfört med andra lösningar, såsom mer avancerade evalueringsfunktioner och sökalgoritmer.

Till viss del finns det forskning på implementationstid och inlärningsperiod för alternativ, såsom lärande AI, och att jämföra denna forskning med implementationstiden och resultatet av den urvalsbaserade tekniken vore också en fortsättning på detta arbete.

Chapter 7

Referenser

[1] A. Carstensen. *Solving (6, 6)-Kalaha*. 2011. URL: http://kalaha.krus.dk/ (visited on 06/05/2015).

[2] J. Chen. "Flow in Games (and Everything else)". In: *Commun. ACM* 50.4 (2007), pp. 31–34.

[3] J. E. Clune. "Heuristic Evaluation Functions for General Game Playing". In: *Künstliche Intelligenz* 25.1 (2011), pp. 73–74.

[4] J. Donkers, J. Uiterwijk, and A. de Voogt. "Mancala games". In: *Mathematics and Artificial Intelligence* (2001), pp. 17–21.

[5] R. Gering. *Kalah*. 2003. URL: http://mancala.wikia.com/wiki/Kalah (visited on 06/05/2015).

[6] C. Gifford et al. "Searching & Game Playing: An Artificial Intelligence Approach to Mancala". In: (2008).

[7] M. Östergren Göransson. "Minimax Based Kalaha AI". Bachelor Thesis. Blekinge Institute of Technology, 2013.

[8] G. Irving, J. Donkers, and J. Uiterwijk. *Solving Kalah*. Maastricht, 2000.

[9] *Kalaha*. 2008. URL: http://www.algaspel.se/~/media/Alga/Files/Rules/flersprakiga/38018720_Kalaha.ashx (visited on 06/05/2015).

[10] Stuart Jonathan Russell and P. Norvig. *Artificial intelligence a modern approach*. Harlow: Pearson Education Limited, 2014.