Un motor de cerca per a l'aualé

Joan Sala Soler
Grau d'Enginyeria Informàtica

David Isern Alarcón

11 de juny de 2014

Resum

L'aualé és possiblement el joc de mancala més conegut. Essent un joc amb adversari i de suma nul·la que s'ha estudiat profusament en el marc de la intel·ligència artificial, fins a arribar a la solució d'una de les seves variants l'any 2002. El projecte plantejat se centra en l'estudi pràctic d'una variant d'aualé de més difícil solució, *abapa*, a través l'elaboració d'un motor de cerca.

Aquest treball de fi de grau, conseqüentment, explora les possibilitats d'optimització de la cerca en l'espai d'estats d'un joc amb adversari. S'analitzen els algorismes existents i es planteja la seva aplicació a l'aualé abapa, per a finalment proposar-ne una implementació altament optimitzada. L'aplicació obtinguda posseeix la capacitat de disputar partides d'aualé amb una competència almenys equivalent a la dels millors jugadors humans.

En el desenvolupament d'aquesta memòria s'investiguen solucions al problema de la representació de jocs, s'introdueixen els mètodes d'optimització més habituals d'un algorisme de cerca i es descriu un conjunt d'estructures de dades que permeten dotar l'aplicació amb l'habilitat de recordar les decisions preses. Finalment, l'optimització del motor de cerca es clou amb la presentació de mètodes d'aprenentatge automàtic amb aplicació a l'aualé.

L'elaboració d'un motor de cerca no seria completa sense la definició d'una interfície que li permeti comunicar-se amb el món exterior. En darrer terme doncs, la implementació descrita en aquesta memòria s'ultima amb l'adaptació al joc d'un protocol de comunicació existent.

Paraules clau: aualé, mancala, motor de cerca, joc amb adversari, espais d'estats, cerca alfa-beta, avaluació heurística, anàlisi retrògrad, teorema minimax, protocol de comunicació

Índex de continguts

Índex de figures

Capítol 1: Introducció

Amb el nom de *mancala* es coneix una família de jocs de tauler força estudiada en el camp de la intel·ligència artificial. No són pocs els autors que han aplicat amb èxit, a diverses variants de la família de jocs, mètodes i algorismes coneguts per aconseguir implementacions capaces d'un nivell de joc anàleg al dels experts humans.

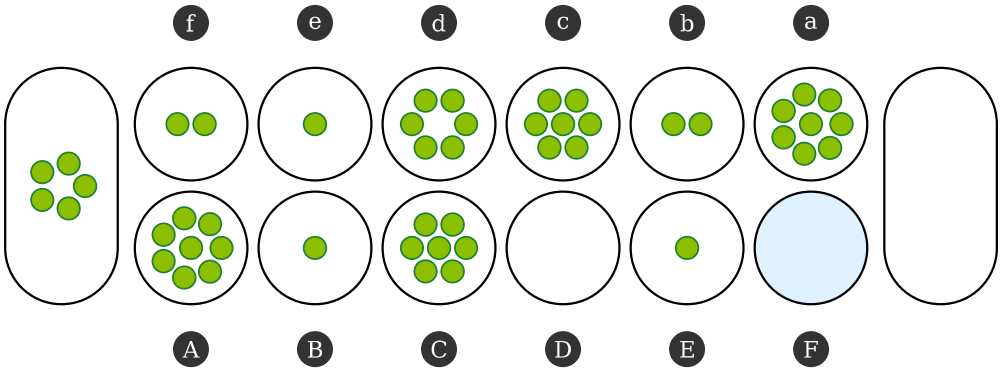
D'entre els jocs de mancala, possiblement el més conegut i també el més estudiat per la intel·ligència artificial és l'aualé —altrament conegut com a *awari*, *oware* o *wari*, entre nombroses denominacions més—. Aquest és precisament el joc al voltant del qual aquest treball es desenvolupa, per mitjà de l'elaboració d'una aplicació amb habilitat per a avaluar l'estat del joc en un instant concret i aproximar el millor moviment que s'hi pot realitzar.

1.1 L'aualé abapa

L'aualé és un joc amb adversari que es juga amb quaranta-vuit fitxes, anomenades llavors, en un tauler rectangular format per dues fileres de sis forats cadascuna. Pertany a la família dels jocs de mancala, altrament dits de compte-i-captura per la peculiaritat del desenvolupament d'una partida, consistent en la distribució de les fitxes del joc pel tauler i la posterior supressió de les mateixes del tauler quan es compleixen un seguit de condicions determinades.

Cal destacar que a diferència d'altres jocs de tauler similars, les fitxes de l'aualé no pertanyen a cap dels jugadors. Cadascun dels dos adversaris controla en canvi una part del tauler de joc —ja sigui la filera inferior de forats o bé la superior— des de la qual els jugadors prenen les llavors i posteriorment les dipositen, una per una, en els altres forats del tauler amb independència de quin sigui el jugador que en té el control. Una descripció detallada del funcionament de l'aualé i el seu reglament es pot consultar a l'*Annex A* d'aquesta memòria.

A la figura 1.1 es pot observar la representació d'un tauler d'aualé durant el transcurs d'una de les partides més curtes que es poden donar en el joc, de només dotze moviments. Normalment una partida d'aualé consistirà tanmateix en prop d'un centenar de moviments.



Anotació de la partida: **1.** E b **2.** D f+2 **3.** B e+3 **4.** F f+2 **5.** C d+5 **6.** A+3 c+13 **3-25**

Figura 1.1: Tauler d'aualé en el transcurs d'una partida

1.2 Justificació

En els darrers anys, alguns dels jocs de mancala més estudiats han estat resolts de forma forta —*kalah* (Irving, Donkers i Uiterwijk, 2000) i *awari* (Romein i Bal, 2002)—, mitjançant la construcció de bases de dades que contenen totes o algunes de les posicions possibles dels jocs juntament amb el seu resultat teòric. També s'han solucionat algunes de les posicions més simples de la variant *bao* (Donkers i Uiterwijk, 2002). En el cas concret d'*awari*, però, la publicació de la solució no va resultar manca de discussió, pel fet que les regles a partir de les quals es va construir la base de dades no són les usades habitualment entre jugadors humans.

En aquest treball de fi de grau, tot i que pot resultar discutible, es distingiran les variants *awari* i *oware abapa* com a jocs diferenciats. El primer, *awari*, segueix un conjunt de normes habituals en el marc de la intel·ligència artificial, mentre que el segon, *abapa*, és la variant més popular entre humans i l'escollida per als campionats d'aualé d'arreu del món. Addicionalment, es mostrarà com aquesta variació de les regles augmenta la complexitat del joc i s'exploraran les possibilitats de solucionar la variant *abapa* amb la tecnologia actual.

Formalment, es coneixen algorismes de cerca que ofereixen una solució genèrica per a la resolució de diferents jocs amb adversari. A la pràctica, però, les limitacions de la tecnologia fan que sovint l'aplicació d'aquests algorismes resulti computacionalment inviable per a molts jocs i calgui aplicar diverses optimitzacions per arribar a solucions heurístiques que permetin simular un nivell d'intel·ligència equivalent a l'humà.

El projecte proposat aquí pretén aprofundir en l'estudi dels algorismes coneguts amb aplicació a la resolució de jocs i en les tècniques d'optimització més utilitzades, per mitjà de la implementació d'un programari que sigui capaç de jugar a la variant d'aualé coneguda habitualment amb el nom *oware abapa*. Aquesta variant resulta d'especial interès, per una banda, pels motius anteriorment exposats i, per altra banda, per què, tot i la popularitat del joc, les seves implementacions són poc comunes.

1.3 Objectius

Aquest treball, consegüentment, se centrarà en l'elaboració d'un motor de cerca amb aplicació al joc d'aualé *abapa*. S'estudiaran les tècniques i algorismes disponibles a la literatura per a l'obtenció d'una aplicació d'intel·ligència artificial que compleixi amb el requisit indispensable de tenir un nivell de joc almenys equivalent al de la majoria d'humans. El programari implementat ha de ser útil tan als jugadors més experts com als inexperts.

El disseny d'una interfície gràfica per al joc no s'inclou en els objectius del projecte, tot i que es proposa la implementació del protocol Universal Chess Interface (UCI), amb les adaptacions oportunes, com a canal de comunicació entre el motor d'intel·ligència artificial i les possibles interfícies gràfiques que es puguin elaborar en un futur. L'UCI és un protocol obert de comunicació entre processos molt estès per al joc dels escacs, la funció del qual és permetre l'ús de motors d'escacs amb independència de la interfície gràfica utilitzada.

El darrer objectiu plantejat per al projecte és que el codi obtingut ha de ser una eina suficientment vàlida per al futur estudi i implementació de motors de cerca amb aplicació als jocs amb adversari de suma nul·la. Amb aquesta finalitat, es proposa la implementació d'una aplicació multiplataforma distribuïda amb llicències de codi obert i el disseny d'interfícies genèriques que facilitin la reutilització de la feina feta.

1.3.1 Abast del projecte

Els objectius del projecte es concreten en els punts següents. Deixant de banda els requisits més específics que forçosament han variat al llarg de l'elaboració del treball.

- Implementar d'un motor d'intel·ligència artificial per al joc *oware abapa*.
- Estudiar els algorismes de cerca disponibles a la literatura i implementar aquells que resultin més adequats.
- Definir una funció d'avaluació heurística optimitzada i específica per al joc. Cal investigar l'aplicació de tècniques de mineria de dades a l'optimització de la funció d'avaluació.

- Estudiar i implementar les optimitzacions del motor de cerca que siguin més idònies. Això inclourà, entre d'altres, l'optimització de l'ordenació de moviments, l'elaboració de taules de transposicions i l'experimentació amb algorismes d'extensió de la cerca.
- Construir una base de dades d'obertures per al motor de cerca.
- Construir una base de dades de posicions finals per al motor, aplicant al joc les tècniques d'anàlisi retrògrad que han permès solucionar la variant *awari*.
- Adaptar al joc i implementar el protocol de comunicació entre processos Universal Chess Interface per al motor obtingut.

1.4 Planificació

L'elaboració d'un motor de cerca per a jocs és una tasca complexa que requereix, en la mateixa mesura, grans dosis de recerca, implementació i experimentació. Un procés que s'haurà de repetir contínuament al llarg del projecte i requerirà sovint revisar les tasques completades. Precisament per això, s'ha cregut convenient fer una planificació inicial el més acurada possible, però tenint en compte que la planificació s'haurà de revisar també de forma constant.

1.4.1 Descomposició del treball

Es mostra a continuació la descomposició inicial del projecte en tasques i la seva planificació temporal. Addicionalment, a l'*Annex B: Planificació temporal* s'hi pot trobar també el diagrama de Gantt corresponent a aquesta planificació inicial.

	Tasca	Comença	Acaba	Feina
1	*Planificació*	Mar 5	Mar 8	4d
2	Entrega de la planificació	Mar 8	Mar 8	
3	*Memòria*	Feb 28	Jun 4	72d
3.1	Bibliografia i glossari	Feb 28	Apr 12	33d
3.2	Annexos	Mar 10	May 22	5d
3.2.1	Reglament del joc	Mar 10	Mar 10	1d
3.2.2	Altres annexos	May 19	May 22	4d
3.3	Capítols	Apr 2	Jun 4	26d
3.3.1	Capítol 1: Introducció	May 30	Jun 4	4d
3.3.2	Capítol 2: Lògica del joc	Apr 2	Apr 7	4d
3.3.3	Capítol 3: Motor de cerca	Apr 17	Apr 28	5d
3.3.4	Capítol 4: Optimització	May 13	May 17	5d
3.3.5	Capítol 5: Llibre d'obertures	May 26	May 29	4d
3.3.6	Capítol 6: Llibre de finals	May 26	May 29	4d

3.4	Valoració econòmica	May 26	May 28	3d
3.5	Conclusions	May 29	May 30	2d
3.6	Resum i agraïments	Jun 2	Jun 2	1d
3.7	Maquetació i revisió	Jun 3	Jun 4	2d
4	*Implementació*	Mar 10	May 23	98d
4.1	Prototipatge	Mar 10	Mar 14	5d
4.2	Definició d'interfícies	Mar 17	Mar 18	2d
4.3	Representació del joc	Mar 19	Mar 25	5d
4.4	Lògica del joc	Mar 26	Apr 1	5d
4.5	Eines d'assaig	Apr 2	Apr 4	3d
4.6	*Motor de cerca*	Apr 2	Apr 16	28d
4.6.1	*Minimax*	Apr 2	Apr 16	18d
4.6.1.1	α-β i Negamax	Apr 2	Apr 7	4d
4.6.1.2	Cerca de variant principal	Apr 8	Apr 14	6d
4.6.1.3	MTD(f)	Apr 8	Apr 16	8d
4.6.2	*Monte-Carlo*	Apr 2	Apr 14	10d
4.6.2.1	UCT	Apr 2	Apr 14	10d
4.7	*Optimització*	Apr 17	May 12	24d
4.7.1	Avaluació heurística	Apr 17	Apr 28	5d
4.7.2	*Ordenació de moviments*	Apr 17	Apr 25	10d
4.7.2.1	History heuristic	Apr 17	Apr 25	4d
4.7.2.2	Killer heuristic	Apr 17	Apr 25	4d
4.7.2.3	Internal Iterative Deepening	Apr 17	Apr 22	2d
4.7.3	*Taula de transposicions*	Apr 29	May 8	7d
4.7.3.1	Funció resum	Apr 29	Apr 30	2d
4.7.3.2	Implementació i proves	May 2	May 8	5d
4.7.4	Iterative deepening	May 9	May 12	2d
4.8	*Aprenentatge*	May 13	May 23	20d
4.8.1	Llibre d'obertures	May 13	May 23	10d
4.8.2	Llibre de finals	May 13	May 23	10d
4.9	Protocol UCI	Apr 17	Apr 25	4d
4.10	Interfície de linia d'ordres	Apr 28	Apr 28	1d
4.11	Revisió de la documentació	Apr 29	Apr 29	1d
5	Primera entrega	Apr 12	Apr 12	
6	Segona entrega	May 17	May 17	
7	Finalització	Jun 4	Jun 4	
8	*Presentació*	Jun 5	Jun 11	5d
8.1	Elaboració de diapositives	Jun 5	Jun 9	3d
8.2	Revisió de la presentació	Jun 10	Jun 10	1d
8.3	Enregistrament	Jun 11	Jun 11	1d
9	Entrega final	Jun 11	Jun 11	
10	Entrega del treball	Jun 16	Jun 16	

1.5 Productes obtinguts

Els resultats d'aquest treball es materialitzen en un seguit de paquets per al llenguatge de programació Java, incloses les eines implementades per a l'optimització del motor de cerca, i dues bases de dades que contenen l'avaluació precalculada de diverses posicions de l'aualé.

Els paquets inclosos a l'entregable són els que s'especifiquen a continuació. El codi font del projecte conté, a més, la documentació Java completa de les classes implementades junt amb una descripció més detallada de la seva funcionalitat.

- **com.joansala.engine:** Interfícies i classes genèriques. Ofereix una interfície de programació per a la implementació de motors de cerca per a jocs amb adversari. Inclou un motor de cerca basat en l'algorisme alfa-beta i completament reutilitzable; un servidor i un client per al protocol UCI; un intèrpret d'ordres del mateix protocol; i la implementació d'una interfície gràfica simple per a la línia d'ordres.
- **com.joansala.oware:** Motor de cerca per a l'aualé. Inclou la implementació de la lògica del joc, de la taula de transposicions per al motor de cerca alfa-beta, d'un llibre d'obertures i d'un llibre de finals de joc usables pel motor. També conté classes que estenen les interfícies de línia d'ordres i el protocol UCI descrits en el paquet `com.joansala.engine`.
- **com.joansala.xo:** Motor de cerca per al tres en ratlla. Es tracta de la implementació d'un motor de cerca simple que serveix de demostració de la utilització del paquet `com.joansala.engine` per a l'elaboració de motors d'intel·ligència artificial per a jocs.
- **com.joansala.tools:** Eines experimentals creades per a l'afinació del motor de cerca de l'aualé. S'inclouen en aquest paquet les proves de rendiment del motor i els algorismes elaborats per a la creació de les bases de dades d'obertures i finals de joc.

Addicionalment al codi font, l'entregable conté dues bases de dades de posicions precalculades per a l'aualé. Cada base de dades està continguda en un únic fitxer, en un format creat específicament per a poder ésser utilitzat de forma òptima pel motor de cerca.

- **Llibre d'obertures**: Una base de dades de més de 160 mil estats del joc junt amb la seva avaluació heurística precalculada. La informació que conté descriu part del graf del joc que resulta de la posició inicial predeterminada de l'aualé.
- **Llibre de finals:** Una base de dades de posicions finals de l'aualé amb informació de tots els estats possibles que es poden donar amb 15 o menys llavors; inclosa la seva avaluació exacta quan ha estat possible calcular-la. És a dir, descriu diversos centenars de milions de nodes terminals de l'aualé abapa i el seu resultat final.

1.6 Descripció dels altres capítols

Aquesta memòria gira al voltant de la implementació d'un motor d'intel·ligència artificial per a l'aualé i la seva optimització pràctica. Els capítols que la conformen descriuen l'elaboració d'aquest motor d'aualé, des de l'elecció dels algorismes fins a l'optimització progressiva dels mateixos. Amb aquesta fi, els primers capítols presenten la recerca efectuada i els fonaments teòrics bàsics, mentre la resta de capítols detallen els mètodes utilitzats per a l'optimització.

- El **Capítol 2: Estat de l'art** descriu els principals mètodes i algorismes disponibles a la literatura que s'han aplicat amb èxit a diversos jocs de la família mancala, per a després discutir la seva possible utilització per a l'aualé abapa.
- El **Capítol 3: Representació** versa sobre el problema de la representació de jocs amb adversari. S'hi expliquen les estructures de dades i tècniques usades més habitualment, i es presenta la implementació escollida per al joc de l'aualé.
- El **Capítol 4: Optimització de la cerca** introdueix les tècniques habituals d'optimització d'un motor de cerca alfa-beta. S'hi descriuen els mètodes utilitzats per a l'afinació de l'aplicació d'aualé i els resultats dels experiments realitzats.
- El **Capítol 5: Cerca amb memòria** tracta sobre les tècniques i estructures de dades han permès dotar el motor d'aualé de la capacitat de recordar. S'hi detalla la combinació de la cerca alfa-beta amb algorismes de cerca en posicions precalculades.
- El **Capítol 6: Aprenentatge automàtic** presenta les tècniques d'aprenentatge computacional que s'han utilitzat per a la implementació del motor d'aualé. Es desenvolupa un model de regressió lineal per a l'afinació de l'avaluació heurística, es descriu l'aplicació al joc d'un algorisme de creació automàtica de llibres d'obertures i es presenta un mètode que combina la cerca amb l'anàlisi retrògrad per a elaborar d'una base de dades de finals.
- El **Capítol 7: Protocol de comunicació** explica el protocol de comunicació entre processos implementat i la seva utilitat pel que respecta a l'optimització de la cerca.

Capítol 2: Estat de l'art

En el camp de la intel·ligència artificial la recerca referent als jocs de mancala es remunta gairebé als inicis de la disciplina. Els esforços d'investigació han portat a la solució d'alguns dels jocs més populars de la família, com són *kalah* i *awari*, i també s'han fet avenços importants en la resolució de *bao*; considerat un dels jocs de mancala més complexos.

Un joc es considera com a solucionat quan es pot predir amb exactitud el resultat final de totes les posicions possibles. Això és equivalent a dir que es coneix una estratègia amb la qual un jugador pot moure sense cometre cap error. Com es podria esperar, no són pocs els jocs que s'han solucionat usant algorismes propis de la intel·ligència artificial i la mineria de dades.

Sovint, per a la solució de jocs no trivials s'usen dos tipus d'algorismes diferents, tot i que fonamentats en el mateix principi: la reconstrucció progressiva dels grafs dels jocs. El primer grup d'algorismes, anomenats d'anàlisi retrògrad, basen el seu funcionament en la reconstrucció del graf partint dels nodes terminals. El segon grup el formen els algorismes de cerca i es diferencien dels primers en què la reconstrucció del graf comença en els nodes inicials.

2.1 Anàlisi retrògrad

El procediment d'anàlisi retrògrad es pot resumir en dos passos. Primerament, es generen tots els nodes terminals del graf del joc i s'avalua el seu resultat exacte —per exemple, guanyat, perdut i empatat—. En el segon pas, el graf s'expandeix per mitjà de la generació de tots els nodes que porten a les posicions terminals generades al primer pas i es propaguen els resultats d'aquests darrers als seus pares. El procés continua fins que s'ha expandit tot el graf.

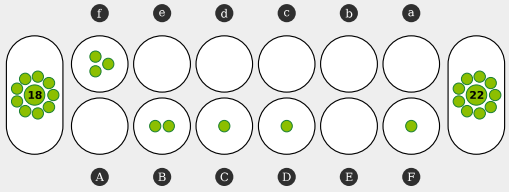
En el cas d'aualé, Romein i Bal (2002) publicaren la solució d'una variant artificial del joc, *awari*, com a part d'un estudi sobre computació distribuïda en paral·lel. L'algorisme d'anàlisi retrògrad distribuït que va possibilitar la solució —una base de dades de totes les posicions possibles i el seu resultat final— fou descrit en detall per Bal i Allis (1995) i també Lincke (2002, pàg. 16-26). Al capítol 6 es presenta aquest mètode i la seva aplicació a l'aualé.

2.1.1 És l'aualé un joc solucionat?

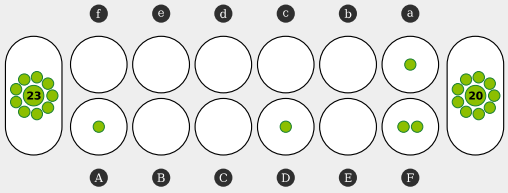
El reglament usat per a la solució presentada tanmateix difereix del que és més habitual entre els jugadors d'aualé, com ja va observar Donkers (2002). Es pot dir que la normativa usual és la del joc anomenat *abapa* —tal com es detalla en l'*Annex A: Reglament del joc*. Aquest reglament, que és el que s'ha seguit en aquest treball, presenta algunes dissimilituds amb el joc solucionat que han resultat força rellevants per a la implementació.

- Abapa restringeix la possibilitat de captura en aquells moviments que podrien deixar l'adversari sense llavors. Awari no presenta aquesta restricció, de forma que una partida pot finalitzar abruptament amb la captura de totes les llavors restants.
- La diferència més important la trobem però en el tractament dels cicles. Mentre que en la variant solucionada quan es produeix un cicle les llavors restants al tauler es reparteixen en parts iguals, en el joc abapa aquest repartiment rarament és equitatiu: per acord mutu, els jugadors capturen les llavors que queden en els seus costats respectius del tauler.

La importància d'aquesta darrera norma radica en l'avaluació que l'algorisme d'anàlisi retrògrad usat fa de les repeticions de posicions. En el cas d'awari implica que el repartiment de llavors no afecta el resultat final si es produeix un cicle, mentre que per a abapa, el resultat depèn del node en el qual es produeix el cicle i, conseqüentment, del camí seguit per arribar-hi.



A. **1.** … f+5 **2.** F a **3.** D b **4.** E c **5.** A d **6.** B e **7.** C f **25-23**



B. **1.** … a **2.** F+2 a **3.** D b **4.** A c **5.** B d **6.** C e **7.** E f **8.** F+2 **24-24**

A partir del segon moviment, les mateixes posicions es repeteixen a totes dues partides. El cicle es produeix però en nodes diferents, amb victòria del jugador sud a A i un resultat d'empat a B.

Figura 2.1: Tractament de cicles a l'aualé abapa

Tot i que existeixen altres diferències menors que afecten el repartiment de llavors en posicions terminals, les dues normatives destacades són les que resulten més rellevants per a la implementació. De les normes en deriva, d'una banda, un espai d'estats efectiu major per a abapa respecte a awari i, d'altra banda, que la solució presentada per aquest darrer no es pugui aplicar fàcilment a la variant estudiada en aquest treball.

2.2 Cerca amb adversari

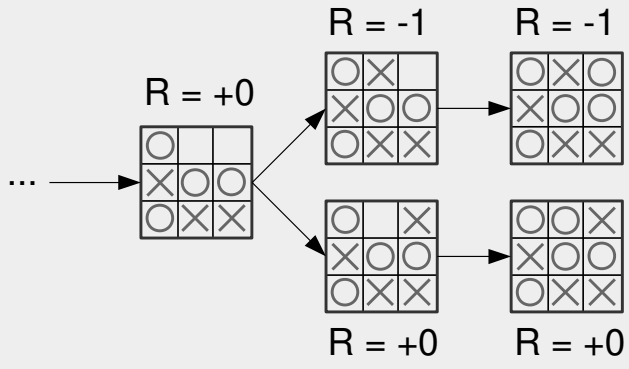
Els algorismes de cerca amb adversari reconstrueixen el graf del joc, molt sovint només de forma parcial, a partir de la posició inicial. El procés consisteix en una expansió progressiva del graf que aquest node arrel defineix, des de l'arrel fins a les fulles, i la posterior propagació de l'avaluació dels nodes terminals cap als seus pares. Aquest procés d'expansió de nodes i propagació de valors continua fins que s'ha pogut avaluar l'arrel de forma exacta.

Una diferència fonamental respecte a l'anàlisi retrògrad, derivada de l'ordre d'expansió dels nodes, consisteix en el fet que habitualment es poden utilitzar els algorismes d'aquest tipus per a la construcció de solucions heurístiques en un temps finit. Donat que per a molts jocs l'elaboració d'una solució completa esdevé intractable, la utilitat principal d'aquests algorismes de cerca rau en la seva aplicació per a la creació de programes amb capacitat de joc.

En l'actualitat, la majoria de programes elaborats per a jugar a jocs amb adversari utilitzen alguna variant optimitzada de l'algorisme de cerca *alfa-beta* (Knuth i Moore, 1975). Els jocs de mancala no en són una excepció. L'algorisme s'ha utilitzat amb èxit en els programes d'aualé més avançats, sovint en combinació amb grans bases de dades de finals. Existeixen però altres algorismes de cerca amb demostrada aplicació a l'aualé.

2.2.1 Teorema Minimax

D'entre els algorismes de cerca amb adversari, els més estudiats són indubtablement aquells que pertanyen a la família *minimax* (Neumann, 1928). Són algorismes basats en el principi de minimització del cost màxim del camí seguit. El teorema minimax manifesta que per a qualsevol joc de suma nul·la existeix una estratègia amb la qual un jugador pot obtenir una recompensa màxima R i el seu adversari pot obtenir com a màxim una recompensa igual a -R.



A la posició inicial d'aquest tres en ratlla, la màxima recompensa que pot obtenir el jugador a qui toca moure és R = +0. Consegüentment, l'estratègia òptima per al jugador és seguir el camí inferior.

Figura 2.2: Arbre de cerca minimax

Això defineix una propagació dels valors terminals de tal manera que el millor moviment per a un jugador esdevé el pitjor moviment per a l'adversari. Per als jugadors, el camí òptim del graf resulta, per tant, en aquell que minimitza la recompensa que el rival pot obtenir.

Dels algorismes derivats directament de *minimax* se'n pot destacar la cerca *alfa-beta*, la cerca de variant principal (Marsland i Campbell, 1982) i el mètode MTD(f) (Bruin, Plaat, Schaeffer i Pijls, 1994). El primer és un mètode de reducció de l'arbre de cerca per mitjà de la poda d'aquelles branques que no resulten rellevants per a l'avaluació d'un estat del joc. Els dos darrers són optimitzacions de la cerca *alfa-beta* consistents en la restricció del rang de valors sobre el qual es realitza la cerca, de forma que es pugui podar un major nombre de branques.

2.2.2 Cerca amb nombres de demostració

Lithidion (Meulen et al., 1990) —el programa d'aualé guanyador de les Olimpíades de Computació en tres ocasions— combinava la cerca *alfa-beta* amb l'algorisme *proof-number search* (Allis, Meulen i Herik, 1994). Es tracta d'un algorisme emprat per a la resolució de jocs capaç de trobar el valor real d'un node a partir d'una aproximació progressiva.

El resultat de l'algorisme quan s'aplica a un estat d'un joc torna un dels valors següents: guanyat, perdut o empatat. Aquesta propietat fa que addicionalment a la seva aplicació per a la resolució de jocs també sigui especialment convenient per a l'avaluació de finals de joc.

2.2.3 Cerca amb nombres de conspiració

La cerca *proof-number* es pot considerar una generalització de l'algorisme *conspiracy-number search* (McAllester, 1988; Schaeffer, 1989). El funcionament d'aquest darrer es basa en l'aproximació del valor d'un node de forma estadística. A cada pas, s'expandeix el node que més probablement pot fer canviar el valor de l'arrel. De forma que l'avaluació consisteix en la restricció progressiva del rang de valors del node. Si bé és un mètode general de cerca, presenta l'inconvenient que requereix mantenir el graf del joc en memòria.

Aquest algorisme és però interessant per la seva aplicació a la creació de bases de dades de posicions inicials. Lincke va utilitzar amb èxit per al seu programa d'aualé Marvin un mètode de creació automàtica de llibres d'obertures que combina diverses estratègies, entre elles la cerca *conspiracy-number*, per tal d'optimitzar l'expansió dels nodes de la base de dades de tal forma que només calgués calcular les posicions més rellevants.

Habitualment se cita aquest algorisme de construcció de llibres d'obertures, conegut com a *drop-out expansion* (Lincke, 2001), com el mètode que va permetre que Marvin sortís victoriós de les cinquenes Olimpíades de Computació celebrades l'any 2000.

2.2.4 Cerca Monte-Carlo

Finalment, si bé la seva possible aplicació a l'aualé no és clara, cal mencionar que un dels algorismes més estudiats en els darrers anys és la cerca Monte-Carlo, basada en un mètode descrit per Abramson (1987). És un algorisme que no requereix una funció d'avaluació heurística, pel que té aplicació a jocs dels quals no es té un coneixement suficient de l'estratègia.

El valor dels nodes s'aproxima amb simulacions, és a dir, jugant de forma aleatòria diverses partides fins al final. Cal destacar que aquest mètode s'ha emprat de forma molt reeixida per al joc de tauler Go. Una implementació basada en aquest algorisme és la que va permetre derrotar per primera a jugadors professionals de Go l'any 2008.

2.3 Plantejament del projecte

Tots els algorismes explicats en aquest capítol es basen en el teorema minimax. La diferència essencial entre els uns i els altres es troba en l'ordre d'expansió dels nodes i en l'avaluació dels mateixos. S'ha vist com l'anàlisi retrògrad permet crear bases de dades on es troben contingudes les posicions d'un joc i la seva avaluació exacta. També s'han vist algorismes de cerca que permeten construir solucions exactes o aproximades per a diferents jocs amb adversari.

En aquest treball es combinaran els algorismes de cerca amb l'anàlisi retrògrad per tal d'obtenir una aplicació que permeti aconseguir solucions heurístiques per a qualsevol posició de l'aualé abapa. Es partirà de l'elaboració d'un algorisme de cerca *alfa-beta* que s'optimitzarà progressivament amb l'ús de mètodes provinents de disciplines tan diverses com són l'estadística, la representació del coneixement o l'aprenentatge computacional entre d'altres.

L'elecció de la cerca *alfa-beta* com a algorisme base respon a necessitats pràctiques. L'estat d'espais de l'aualé es pot estimar en $6 \cdot 10^{12}$ posicions diferents, tot i que serà segurament significativament superior a causa de la dependència d'un estat al camí seguit per arribar-hi. Això fa que una solució exacta resulti inviable amb els recursos disponibles. Per altra banda, una possible avaluació heurística dels estats es pot fer de manera senzilla recomptant el nombre de llavors capturades. D'aquí que *alfa-beta* sembli més idoni que altres mètodes més complexos.

CAPÍTOL 3:

Representació de l'aualé

La representació del coneixement és un dels pilars bàsics de l'elaboració de programes amb capacitat per al joc, juntament amb els algorismes de cerca. Escollir una representació adequada per al problema que es vol resoldre és la tasca que més temps consumeix però és fonamental en la mesura que determinarà l'efectivitat i l'eficiència de l'aplicació.

3.1 Importància de la representació

En la implementació que acompanya aquesta memòria s'hi pot trobar un motor de cerca *alfa-beta* per al joc del tres en ratlla que exemplifica alguns dels problemes que poden esdevenir del fet de no disposar d'una representació idònia i presenta també algunes solucions.

D'entrada, a la implementació s'hi pot observar com resoldre el joc amb un algorisme de cerca no és bona idea: una representació en forma de regles de producció no només eliminaria la necessitat de cerca sinó que exhibiria una millor estratègia. Alhora, però, el motor de cerca mostra també una representació de les posicions del tres en ratlla, basada en l'ús de mapes de bits, que és gairebé òptima des del punt de vista de l'eficiència.

3.1.1 Representació de l'estratègia

Un dels problemes que poden presentar els algorismes de cerca per a jocs és l'excés de coneixement. Com a exemple, el motor de tres en ratlla implementat calcula totes les posicions possibles ràpidament, de forma que s'adona que no pot guanyar si no és que el rival comet algun error. El motor decideix, per tant, que la millor estratègia és perseguir l'empat. És a dir, no és capaç d'aprofitar els errors que l'adversari pugui fer.

Normalment aquest inconvenient es pot minimitzar de dues maneres. El factor de contemplació és un valor usat per a avaluar de forma diferent de les posicions terminals que resulten en un empat. Això permet modelar el motor de joc per tal que presenti més o menys preferència per l'empat, per exemple, si considera que el rival és superior. Un altre mètode fre-

qüent consisteix en l'ús de bases de dades de finals amb una representació que permeti escollir sempre el camí més llarg cap a la derrota o bé el més curt cap a la victòria.

3.1.2 Optimització d'operacions

Normalment, augmentar el nombre de nodes cercats permet aproximar millor quin és el moviment més escaient per una posició determinada. Com a conseqüència, l'altre problema essencial que hauria de resoldre la representació d'un joc és el de l'eficiència.

A l'hora d'escollir una representació cal però trobar un equilibri entre la minimització de les necessitats de memòria —per exemple, per permetre que el motor de cerca recordi un major nombre d'estats del joc— i l'eficiència dels càlculs més habituals que s'hauran de realitzar amb les estructures de dades que s'hagin definit. És evident, per tant, que el joc a implementar guiarà les decisions de representació, però també hi poden influir tant el llenguatge de programació escollit com l'algorisme de cerca que s'utilitzarà.

3.2 Representació de jocs amb adversari

Els algorismes de cerca amb adversari requereixen dues operacions essencials que s'han d'optimitzar al màxim. D'una banda, donat un node del graf dirigit que el joc defineix, ha de ser possible obtenir de forma eficient les seves arestes: és a dir, s'han de poder generar els moviments que es poden fer en un moment concret. D'altra banda, és necessària una operació que donat un node i una aresta permeti obtenir-ne el node fill i, si l'algorisme de cerca no manté el graf en memòria, caldrà també una operació que permeti anar d'un node fill al seu pare.

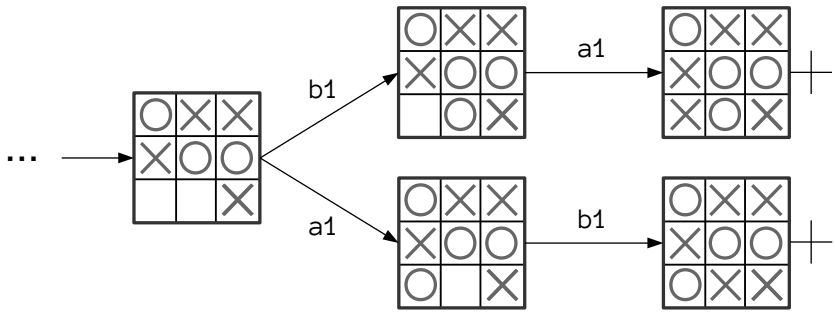


Figura 3.1: Representació del tres en ratlla en forma d'autòmat finit

Com es pot observar, les operacions requerides permeten representar qualsevol joc de tauler amb adversari com una màquina d'estats on les transicions vénen definides per les operacions de fer i desfer moviments. Aquesta és una representació especialment adient per a jocs en els quals un estat estarà determinat per tots els moviments que s'han realitzat anteriorment al tauler, com és el cas concret de l'aualé abapa.

A la implementació que acompanya aquesta memòria, la màquina d'estats es tradueix en una interfície `Game` que descriu quatre mètodes bàsics. El primer, `makeMove`, donat un moviment i una posició, permet establir la posició resultant d'efectuar un moviment al tauler. El segon mètode, `unmakeMove`, implementa l'operació contrària. Finalment, els mètodes `legalMoves` i `nextMove` són els encarregats de la generació de moviments legals per a l'estat actual.

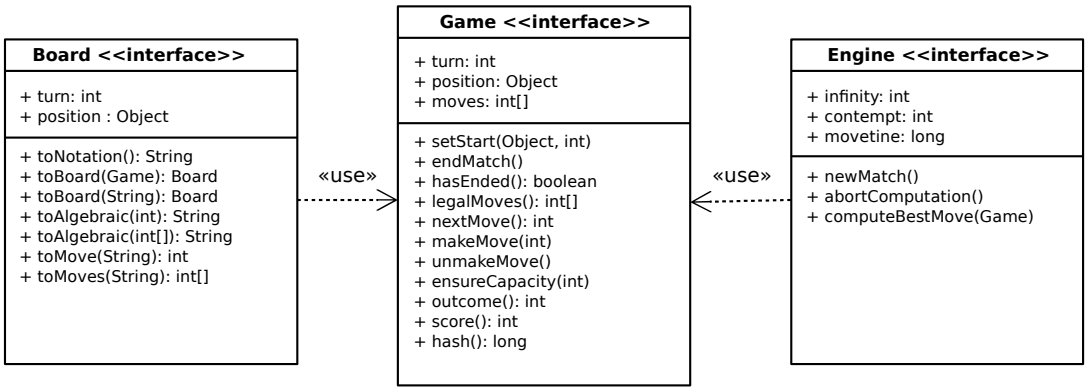


Figura 3.2: Diagrama de classes de la representació d'un joc

La figura 3.1 mostra el diagrama de classes de la representació d'un joc en forma de màquina d'estats. S'hi pot veure com, a més de les operacions descrites, la interfície `Game` també defineix altres operacions necessàries per a establir i consultar l'estat de la màquina. Això permet encapsular la lògica i l'estat del joc en una mateixa interfície, facilitant així l'optimització de la representació i la implementació de motors de cerca completament reutilitzables.

3.3 Propietats de l'aualé abapa

Fins a aquest punt, s'ha comentat la problemàtica de trobar una representació idònia i s'ha descrit una representació genèrica per a jocs amb adversari. En aquest apartat es fa un incís amb l'objectiu d'explicitar aquelles propietats de l'aualé abapa útils per a l'optimització o que han guiat l'elecció d'una representació enfront d'una altra.

- Es pot comptar el nombre de posicions possibles del joc. Això és el nombre de maneres que hi ha de distribuir exactament 48 objectes en 14 caixes diferents, menys el nombre de posicions que no es poden donar en el joc (no es pot capturar mai una única llavor).

$$\binom{48+14-1}{14-1} - 2 \cdot \binom{47+12-1}{12-1} = 6.110.837.699.295 \approx 6 \cdot 10^{12}$$

- Una divisió útil dels moviments del joc és la classificació en *a*) atacants: quan realitzen una captura, *b*) defensius: aquells que sembren menys de 3 llavors, *c*) estratègics: sembren llavors als forats del rival i *d*) la resta de moviments.

- Un jugador no pot capturar mai totes les llavors que es troben en forats de l'adversari. D'aquesta normativa en deriva que en un moviment es puguin capturar com a màxim les llavors de cinc forats diferents.
- Un moviment és legal si després d'efectuar-lo hi ha almenys un forat del rival que conté llavors. Per consegüent, els moviments que realitzen captures o sembren llavors als forats de l'adversari seran sempre legals.
- Es pot saber si un moviment realitzarà una captura observant la distribució de llavors al tauler abans de fer el moviment. Un moviment no capturarà llavors si *a*) la sembra acaba en un forat del rival que contenia més de 2 llavors o *b*) el moviment capturaria totes les llavors dels forats del seu adversari.
- Per a què es produeixi un cicle en el joc és imprescindible que les llavors hagin donat una volta completa al tauler. Per tant, tot cicle estarà format per un mínim de 12 nodes.
- Si una posició es troba en un cicle, la seva avaluació dependrà del node d'entrada al cicle.
- Un cicle no conté mai moviments que capturin llavors. El nombre de llavors distribuïdes als forats dels jugadors és únic per a tots els nodes d'un mateix cicle.
- La distribució de les llavors al tauler pot canviar completament després d'un moviment.

3.4 Representació de posicions

La *posició* d'un joc fa referència a la distribució de les fitxes en un tauler de joc en un moment determinat. Com ja s'ha comentat, la seva representació és crítica per assegurar l'eficiència dels algorismes de cerca. En aquest apartat es descriuen les representacions més habituals per a les posicions d'un joc i la seva possible aplicació a l'aualé.

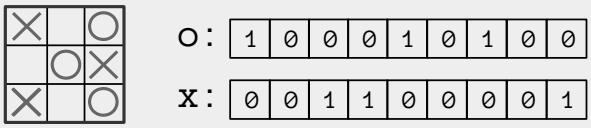
Com es veurà, no és freqüent que existeixi una representació única i òptima per a tots els usos. És per això que habitualment un motor de cerca combinarà estratègicament diferents representacions segons quines siguin les seves necessitats a cada moment. Els requisits de memòria, les possibilitats d'optimització dels càlculs i les necessitats de portabilitat són els factors que influiran en l'elecció d'una representació o una altra.

3.4.1 Mapes de bits o *bitboards*

El mapes de bits són estructures de dades que permeten emmagatzemar bits de forma compacta. Es tracta d'una implementació del concepte matemàtic de conjunt finit de tal forma que elements del conjunt es representen com a bits encapsulats en valors numèrics. Cal notar que no és una representació òptima en termes de memòria utilitzada. Això és degut al fet que

normalment el nombre de bits que s'han d'emmagatzemar no és múltiple de la mida de la paraula utilitzada per encapsular-los.

La seva utilitat en un motor de cerca es troba en les possibilitats que ofereix per a l'optimització de càlculs, ja que permet implementar operacions de forma molt eficient amb simples càlculs a nivell de bits. D'aquesta manera es poden explotar al màxim les capacitats de paral·lelisme que ofereix el maquinari. Tot i així, normalment els mapes de bits s'usen en combinació amb representacions vectorials que són més idònies per als càlculs més complexos.




```
és_legal (m) ← ((o | x) & 0x01 << m)
```

El valor superior representa els elements del tauler que contenen una O, mentre que el valor inferior representa aquells que contenen una X. Aquesta representació permet implementar els mètodes més crítics per al joc utilitzant operacions simples a nivell de bits.

Figura 3.3: Mapa de bits per al tres en ratlla

Malgrat que els mapes de bits són estructures molt eficients, s'han descartat per a la implementació de l'aualé pel fet que el joc presenta propietats que a la pràctica en dificulten l'ús. D'entrada, es pot observar com la disposició de les llavors al tauler canvia excessivament entre posicions, de manera que els càlculs necessaris per actualitzar els mapes de bits van en detriment dels possibles guanys. D'alta banda, les operacions més crítiques per al joc —com per exemple, la classificació de moviments— sovint es poden implementar amb comparacions senzilles, moltes vegades tant o més eficients que els càlculs a nivell de bits.

3.4.2 Representació matricial

Qualsevol joc de tauler admet altrament una representació en forma de matriu o de vector numèric. Aquesta és una de les representacions més simples i una de les que més memòria requereix, però sovint és també la més adient des d'un punt de vista computacional. En una representació d'aquest tipus, una matriu de mida fixa simbolitza el tauler de joc i els valors que prenen els seus elements les fitxes distribuïdes a cada emplaçament del mateix.

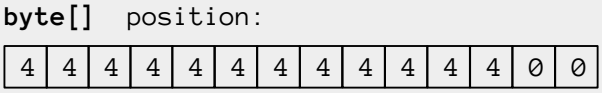


Figura 3.4: Representació vectorial de la posició inicial de l'aualé

La figura 3.4 mostra la representació utilitzada per al motor implementat. L'aualé es pot representar com un vector de catorze elements, on cada índex del vector simbolitza un dels forats del tauler de joc i els valors numèrics emmagatzemats en cadascun dels índexs correspon al nombre de llavors que es troben al forat en un moment concret. Com que el nombre de fitxes del joc és quaranta-vuit, només calen catorze octets per cada posició.

Les propietats del joc fan que aquesta estructura de dades sigui idònia per a la majoria de càlculs. La sembra i la captura de llavors es poden fer en temps constant recorrent de forma seqüencial els elements del vector i, per a implementar la resta d'operacions crítiques, normalment només és necessari comparar si un element del vector conté un valor concret. Per a millorar l'eficiència d'aquestes operacions, la implementació que acompanya aquesta memòria usa, addicionalment, tot un seguit de taules precalculades d'índexs i valors.

3.4.3 Representació numèrica o resum

El resum és una representació força usual en motors de cerca amb memòria. Obtenir una representació numèrica d'una posició consisteix en l'aplicació d'una funció resum a un estat del joc per tal d'aconseguir un nombre enter que l'identifiqui de forma única o gairebé única. La representació resultant exhibeix la propietat de ser molt compacte i, alhora, permet també comparar de manera molt eficient els diferents estats que pot prendre un joc.

Freqüentment se'n fa ús com a índex d'una taula de transposicions, que no és més que una estructura de dades utilitzada pels motors de cerca per tal de recordar certa informació estratègica sobre les posicions analitzades anteriorment. També és una representació útil per a desar informació sobre els estats d'un joc en una base de dades, atès que així es pot minimitzar la memòria necessària per emmagatzemar les posicions.

De funcions resum amb aplicació als jocs amb adversari n'hi ha moltes i molt diverses, sent segurament la més coneguda el *resum Zobrist* (Zobrist, 1970). Aquesta funció permet obtenir identificadors gairebé únics de forma incremental. A partir de l'identificador d'una posició, s'afegeix al mateix —amb unes poques operacions a nivell de bits— la informació dels canvis realitzats al tauler per tal d'obtenir un nou identificador. És doncs una funció molt eficient, d'aquí la seva popularitat, tot i que d'utilitat limitada per a jocs com l'aualé, on la informació d'una posició pot canviar completament després d'efectuar un moviment al tauler.

Per aquest motiu, per a la implementació del joc d'aualé sobre la qual tracta aquest treball la funció escollida ha estat una altra: l'anomenat *resum binomial*. El funcionament d'aquest mètode és simple, consisteix a comptar les combinacions possibles del nombre de bits certs en una representació numèrica per mitjà de l'ús de coeficients binomials. Es tracta d'un algorisme que s'ha utilitzat anteriorment per als jocs de mancala amb demostrada eficàcia, sent en l'actualitat possiblement la representació més popular per a la família de jocs.

Resum binomial d'una representació vectorial.

```
long rankPosition(byte[] position) {
    long rank = 0x00L;
    int n = (int) position[13];

    for (int i = 12; n < 48 && i >= 0; i--) {
        rank += COEFFICIENTS[n][i];
        n += (int) position[i];
    }

    return rank;
}
```

Conversió d'un resum a una representació vectorial.

```
byte[] unrankPosition(long rank) {
    byte[] position = new byte[14];
    byte seeds = 0;
    int i = 12;
    int n = 0;

    while (i >= 0 && n < 48) {
        long value = COEFFICIENTS[n][i];

        if (rank >= value) {
            rank -= value;
            position[i + 1] = seeds;
            seeds = 0;
            i--;
        } else {
            seeds++;
            n++;
        }
    }

    position[i + 1] = (byte) (48 - n + seeds);

    return position;
}
```

Figura 3.5: Funció de resum binomial optimitzada per a l'aualé

Si bé la construcció de resums binomials no és incremental, com en el cas dels resums Zobrist, a la pràctica això no és en detriment de l'eficiència gràcies a dues propietats úniques que fan el mètode molt eficaç: és un resum perfecte i mínim. La primera propietat significa que un resum identifica inconfusiblement una posició; la segona fa referència al nombre de bits necessaris per a la representació. A la pràctica, aquestes propietats fan que se simplifiquin molt els càlculs que posteriorment caldrà fer amb la representació, alhora que possibiliten la reconstrucció de posicions a partir dels seus resums.

3.4.4 Representació alfanumèrica

Una posició i torn de joc es pot descriuré també com una cadena de caràcters alfanumèrics. Aquesta no és una representació ni compacta ni tampoc eficient, nogensmenys, no és habitual en la implementació de motors de cerca. La seva utilitat rau però en les possibilitats de portabilitat que ofereix una representació d'aquest tipus. Tal com es veurà al capítol 7, la representació alfanumèrica es pot utilitzar, per exemple, per a la comunicació entre processos a través del protocol Universal Chess Interface.

3.5 Generació de moviments

La importància de la representació dels moviments d'un joc i, especialment, de la seva ordenació es farà patent en el capítol 4 d'aquesta memòria. En aquest apartat només es comenten breument els avantatges i inconvenients dels dos mètodes principals que es poden utilitzar per a generar les arestes d'un node en un joc amb adversari.

3.5.1 Generació en massa

El mètode de generació en massa consisteix a produir tots els moviments legals que es poden fer en una posició i emmagatzemar-los en un vector de moviments. És un mètode que requereix travessar de forma seqüencial tots els moviments possibles d'una posició per a verificar-ne la legalitat. En conseqüència, es tracta d'una operació que s'executarà en temps lineal O(n).

Presenta però l'avantatge que un cop generats tots els moviments aquests es poden reordenar posteriorment per optimitzar la cerca. Com es veurà al següent capítol, ordenar les arestes de tal manera que el motor de cerca sigui capaç de recórrer primer els millors moviments de cada node redueix sovint el nombre de branques que caldrà inspeccionar.

La generació en massa s'utilitza normalment només en el node arrel d'una cerca. Això es deu al fet que en aquest node inicial el guany obtingut amb una ordenació més acurada és sovint superior al que es pot obtenir en els nodes interiors de l'arbre de cerca. A la interfície `Game` presentada a la figura 3.2 aquesta operació està definida en el mètode `legalMoves`.

3.5.2 Generació per etapes

En la generació per etapes els moviments es produeixen i verifiquen un per un. És habitual que es generin de forma ordenada segons una classificació. Per exemple, per al joc de l'aualé és útil poder generar primer els moviments que capturen llavors abans que la resta. Es tracta d'una operació que es pot fer en temps constant O(1), mitjançant una funció generadora que torni un únic moviment a cada invocació.

Aquesta representació s'utilitza per als nodes interiors d'un arbre de cerca. El principal avantatge que presenta per a un motor de cerca és que, si l'ordenació d'arestes és prou bona, habitualment només serà necessari generar els primers moviments de cada posició. Com que l'operació s'executa en temps constant, a la pràctica això representa un estalvi de càlculs important. La interfície `Game` defineix aquesta funcionalitat amb el mètode `nextMove`.

Capítol 4: Optimització de la cerca

El capítol 2 introduïa els principals algorismes de cerca amb adversari existents, tot posant l'èmfasi en l'aplicació pràctica del teorema minimax com a fonament dels mateixos, per acabar concloent que l'elaboració del motor de cerca sobre el qual tracta aquest treball partiria de la implementació d'un algorisme alfa-beta. En el present capítol es descriuen les tècniques més usuals d'optimització d'un algorisme minimax, incloent-hi la poda alfa-beta, i s'exposen els mètodes emprats per a la implementació del motor d'aualé.

4.1 L'algorisme de cerca alfa-beta

El funcionament habitual d'un algorisme minimax consisteix a recórrer l'espai d'estats d'un joc en forma d'arbre de cerca, on cada node de l'arbre representa un estat del joc i les seves arestes vénen definides pels moviments que s'hi poden fer. En l'algorisme alfa-beta, a més, aquest recorregut es fa amb una cerca en profunditat, de tal manera que se segueix sempre un mateix camí fins al final i només llavors es recorre un camí diferent.

Partint d'un node inicial, se seleccionen de forma recursiva les arestes de l'arbre fins a arribar als nodes terminals. Un cop localitzats, els nodes terminals s'avaluen per després propagar aquests valors als seus pares. Aquesta avaluació es fa sovint de forma numèrica, amb valors grans si el camí seguit porta a la victòria d'un jugador i amb valors petits quan condueix a la seva derrota. L'algorisme finalitza quan s'han avaluat tots els camins.

L'avantatge de fer un recorregut en profunditat per l'arbre resideix en què l'avaluació dels camins es fa de manera progressiva. Això permet anar aproximant, mentre es recorre l'arbre, un interval de valors que contingui l'avaluació que es propagarà finalment a l'arrel.

La poda alfa-beta aprofita aquesta peculiaritat per reduir el nombre de nodes que cal inspeccionar. Durant el recorregut per l'arbre es difonen en aquest les avaluacions mínimes i màximes susceptibles de propagar-se al node arrel i, quan es descobreix que el valor propagat en una branca es troba fora d'aquest rang, s'evita recórrer-ne la resta de nodes.

4.2 Marc de cerca *negamax*

El motor d'aualé elaborat implementa l'algorisme alfa-beta en forma de marc de cerca negamax. Es tracta d'una formulació molt habitual per a un algorisme minimax fonamentada en la igualtat matemàtica $min(a, b) = -max(-a, -b)$. Així, l'avaluació d'un node per a un jugador esdevé la negació del valor que té aquell node per al seu rival i, conseqüentment, la propagació de valors es pot fer sempre prenent el màxim dels valors negats dels fills d'un node.

```
int search(Game game, int alpha, int beta) {
    int score;

    if (game.hasEnded())
        return game.outcome() * game.turn();


    for (int move : game.legalMoves()) {
        game.makeMove(move);
        score = -search(game, -beta, -alpha);
        game.unmakeMove();

        if (score >= beta)
            return beta;

        if (score > alpha)
            alpha = score;
    }

    return alpha;
}
```

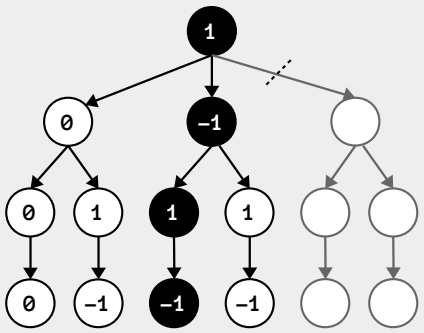


Figura 4.1: Marc de cerca negamax amb poda alfa-beta

Es pot observar a la figura 4.1 que amb una formulació com aquesta la implementació de l'algorisme se simplifica molt. L'únic que cal tenir en compte és que l'avaluació d'un node terminal s'ha de fer sempre des del punt de vista d'un mateix jugador i que és necessari negar aquest valor en el torn de joc del seu rival. Això es pot veure en la quarta línia de codi, en la qual el mètode `outcome` avalua una posició des del punt de vista del jugador que mou primer i el resultat de la funció `turn` alterna entre els valors -1 i 1 per cada invocació de `makeMove`.

4.3 Avaluació heurística

Tal com s'ha descrit fins ara, l'algorisme depèn de l'avaluació dels nodes terminal per a poder descobrir el camí òptim de l'arbre de cerca. Des d'un punt de vista pràctic això resulta però poc adequat per a jocs que com l'aualé tenen un espai d'estats molt gran. Avaluar tots els camins de l'arbre requeriria quantitats enormes de memòria i de temps de computació.

Per superar aquesta limitació és habitual que el motor de cerca no expandeixi l'arbre completament. És a dir, per cada camí de l'arbre es fixa una profunditat de cerca màxima a partir de la qual no es recorreran més nodes. És clar que això ocasiona que sovint no s'arribi als nodes terminals i calgui aproximar l'avaluació del tros de camí que no s'ha inspeccionat.

Quan es posa límit a la cerca en profunditat es fa necessari, per tant, un mètode que permeti predir amb un cert marge d'error quin seria el resultat obtingut en cas de recórrer el camí òptim des d'un estat concret. Amb aquest propòsit s'introdueix a l'algorisme de cerca una funció heurística que aproxima l'avaluació que s'hauria propagat en els nodes no terminals. És a dir, una funció que descriu numèricament quin dels jugadors tindria més possibilitats de guanyar si la partida continués disputant-se des d'un estat concret.

Les propietats de l'aualé permeten crear una funció d'aquest tipus de manera molt senzilla. Com que l'objectiu del joc és capturar més llavors que el rival, es pot dir que és més probable que un jugador guanyi si el nombre de llavors que ha capturat en un moment determinat és clarament superior al nombre de captures que ha realitzat el seu rival.

Les figures 4.2 i 4.3 mostren com efectivament aquesta correspondència existeix per a l'aualé. Al gràfic de l'esquerra s'hi observa l'avaluació d'unes quaranta mil posicions d'aualé, obtingudes per mostreig aleatori amb reposició. L'eix d'abscisses representa la diferència de llavors capturades inicialment en les posicions, mentre que l'eix d'ordenades presenta una aproximació de la diferència de llavors màxima que un jugador podria aconseguir si seguís el camí òptim de l'arbre de cerca des de les mateixes posicions.

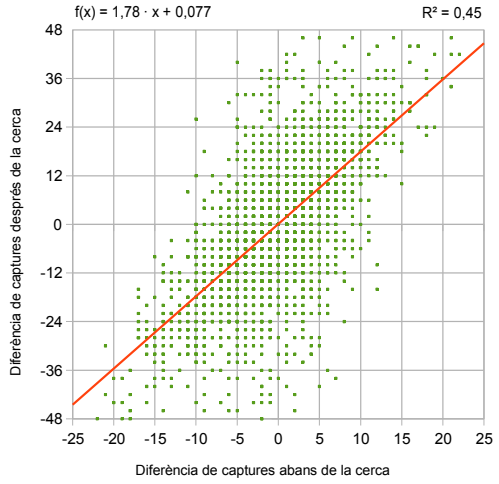


Figura 4.2: Relació lineal entre la diferència de llavors abans i després d'una cerca

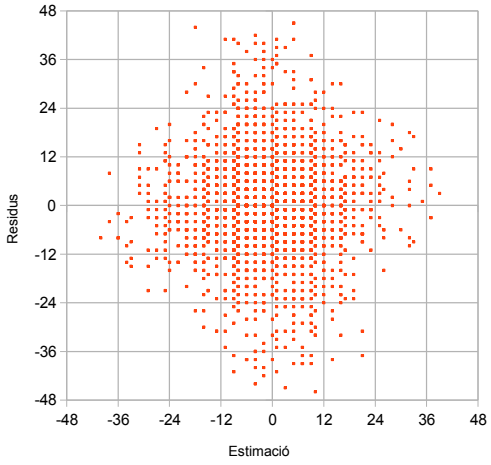


Figura 4.3: Diagrama de residus de la relació lineal

La recta de tendència f(x) defineix clarament una relació lineal moderada que permet predir l'avaluació d'un estat de manera senzilla, tot i que amb un error que pot arribar a ser molt gran. En el capítol 6 es descriu amb més detall el procediment de mostreig utilitzat per a descriure aquesta relació. També s'hi presenta un mètode d'aprenentatge que permet minimitzar el marge d'error de la predicció; tot transformant aquesta relació lineal moderada en una de forta per mitjà de l'ús d'informació estratègica del joc.

4.4 Ordenació de moviments

Al primer apartat d'aquest capítol s'han resumit les peculiaritats de l'algorisme alfa-beta i s'ha descrit breument com l'algorisme aconsegueix reduir el nombre de nodes que cal inspeccionar de l'arbre minimax. Tal com s'ha comentat, aquest mètode consisteix en la restricció progressiva de l'interval de valors que pot prendre el node arrel i en la poda d'aquelles branques que no poden influir en el resultat final.

Aquest mètode de poda es pot entendre més clarament observant l'arbre de la figura 4.1, que representa una cerca en el joc del tres en ratlla. Després d'avaluar la branca central, l'algorisme s'adona que tots els seus camins condueixen indubtablement a la victòria del jugador. Per consegüent, continuar inspeccionant la resta de branques seria superflu. També és fàcil adonar-se que, si l'algorisme hagués avaluat aquesta branca primer, tampoc hauria estat necessari recórrer el subarbre de l'esquerra.

És evident doncs que l'ordre en el qual s'inspeccionen les arestes dels nodes influeix considerablement en el rendiment de l'algorisme. Si fos possible avaluar primer el camí òptim de l'arbre —aquell que recorre els millors moviments— el nombre de nodes que seria necessari inspeccionar esdevindria mínim. A la pràctica això només és viable quan es coneix una estratègia òptima pel joc. Ara bé, com es veurà en els següents apartats, una aproximació a aquest arbre mínim és possible amb una ordenació adequada dels moviments.

4.4.1 Arbre mínim i factor de ramificació

Es coneix com a *arbre mínim de cerca*, d'un algorisme minimax, al conjunt format per tots els nodes que seria imprescindible inspeccionar amb una ordenació d'arestes òptima. Knuth i Moore (1975) van mostrar que és possible quantificar aquest arbre mínim amb una cota inferior del nombre de fulles que qualsevol algorisme minimax necessita avaluar.

La fórmula que permet fer aquesta aproximació relaciona el nombre de fills de cada node amb el diàmetre de l'arbre. Al primer element se l'anomena *factor de ramificació* (b) i el segon coincideix amb la profunditat de cerca màxima preestablerta (d). Cal notar que es pressuposa un arbre de cerca uniforme, pel que a efectes pràctics, com que el factor de ramificació rarament és constant s'haurà d'aproximar.

$$O\left(b^d\right)_{min} = b^{\lfloor d/2 \rfloor} + b^{\lfloor d/2 \rfloor} - 1$$

Fórmula 4.1: Complexitat de l'arbre mínim en un algorisme minimax

Una manera d'estimar aquest factor de ramificació és construir l'arbre de cerca minimax, sense cap mena de poda, per llavors prendre la mitjana del nombre d'arestes dels seus nodes. Per al motor de cerca implementat s'ha partit d'un banc de proves format per vuit partides dis-

putades entre jugadors humans professionals. De manera que l'estimació s'ha fet amb un conjunt de 669 posicions realistes que engloben tant l'obertura com el final de joc.

	Nodes avaluats	Profunditat: d	Complexitat: $O(b^d)$	Ramificació: b
Arbre màxim	12.556.620.384	11	18.769.238,24	4,584
Arbre mínim	7.560.597	11	11.301,34	2,335

Figura 4.4: Aproximació teòrica de l'interval de ramificació

La figura 4.4 mostra els resultats obtinguts i també l'arbre mínim teòric. Els valors per a l'arbre màxim s'han calculat construint l'arbre minimax de cadascuna de les posicions del banc de proves, per finalment estimar la complexitat com el quocient entre el total de fulles avaluades i el nombre de posicions cercades. L'arbre mínim s'ha aproximat llavors a partir del factor de ramificació acotat amb aquest mètode ($b = 4{,}58$).

4.4.2 Aproximació a l'arbre mínim de l'aualé

Fins ara s'ha vist que a cada posició de l'aualé es poden fer 4,6 moviments de mitjana, tot i que per a poder jugar sense cometre cap error només caldria considerar-ne uns 2,3. Amb una ordenació prou acurada dels moviments, el nombre d'arestes inspeccionades per l'algorisme alfa-beta s'hauria d'acostar a aquest factor de ramificació mínim.

L'objectiu final de l'ordenació consisteix a maximitzar el nombre de branques podades per tal d'aconseguir explorar un mateix arbre en menys temps. És important doncs que el cost de l'algorisme d'ordenació sigui menor al de l'expansió de nodes, en cas contrari, no s'obtindria cap guany addicional. Afortunadament, els càlculs d'ordenació que cal fer a cada node es poden reduir amb una funció generadora com la descrita a l'apartat 3.5, aprofitant així la poda de branques per a poder aconseguir una ordenació més precisa.

Tots aquests factors fan que aproximar l'arbre mínim de cerca consisteixi sobretot en un procés de prova i error. Tot i així, habitualment l'elaboració d'un algorisme d'ordenació es basa en la classificació dels moviments del joc i en la posterior prova de diferents combinacions fins a obtenir la més adequada. Per tant, a la pràctica només caldrà efectuar un petit nombre de proves que dependrà sobretot de com d'idònia sigui la classificació.

A l'apartat 3.3 de l'anterior capítol s'ha donat una possible classificació de moviments de l'aualé fonamentada en l'observació empírica. De les categories descrites, els moviments atacants i defensius són els únics que s'han considerat per a la implementació, ja que engloben aquells moviments que poden fer canviar l'avaluació d'un camí de manera radical.

S'han realitzat proves amb els cinc algorismes d'ordenació diferents que es descriuen a continuació. Per a mesurar-ne la bondat, els algorismes s'han provat contra el banc de proves

descrit a l'apartat anterior per després comparar el factor de ramificació obtingut i el nombre de fulles avaluades amb els valors estimats per a l'arbre mínim.

- *Ordenació ingènua:* En aquesta ordenació els moviments es consideren respecte al forat del tauler des del qual se sembren les llavors; d'esquerra a dreta.
- *Ordenació defensiva:* Se cerquen primer aquells moviments que sembren llavors des d'un forat que conté exactament una o dues llavors.
- *Ordenació atacant:* Aquells moviments que capturen llavors s'inspeccionen primer.
- *Ordenació combinada:* Fusiona els dos arranjaments anteriors. S'ordenen primer els moviments atacants, seguits dels defensius i a continuació la resta de moviments.

Els gràfics de les figures 4.5 i 4.6 mostren els resultats obtinguts amb les combinacions més rellevants i una profunditat de cerca prefixada. En honor a l'exhaustivitat, també s'hi representa l'arbre mínim teòric i els resultats obtinguts amb un algorisme ordenació aleatòria. Aquesta ordenació aleatòria simbolitza el pitjor cas possible per a un algorisme alfa-beta.

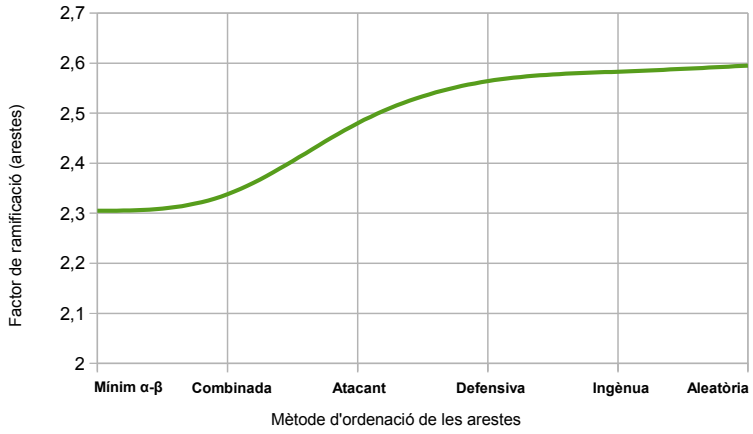


Figura 4.5: Factor de ramificació respecte a l'ordenació ($d = 13$)

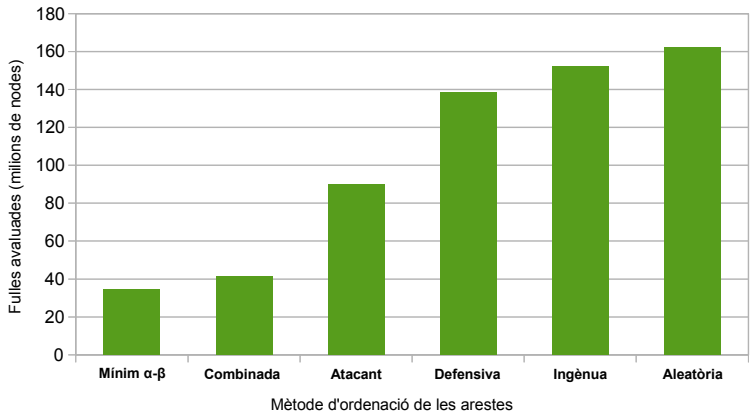


Figura 4.6: Nodes avaluats respecte a l'ordenació ($d = 13$)

Com es pot observar, l'algorisme d'ordenació combinada s'aproxima molt bé a l'arbre mínim. En comparació amb la resta d'algorismes resulta també en l'ordenació òptima quant al temps de còmput, com es pot extreure dels gràfics de la figura 4.7. Aquests gràfics mostren respectivament els nodes per segon expandits per l'algorisme i el temps mitjà necessari per a avaluar una posició en relació a la profunditat de cerca.

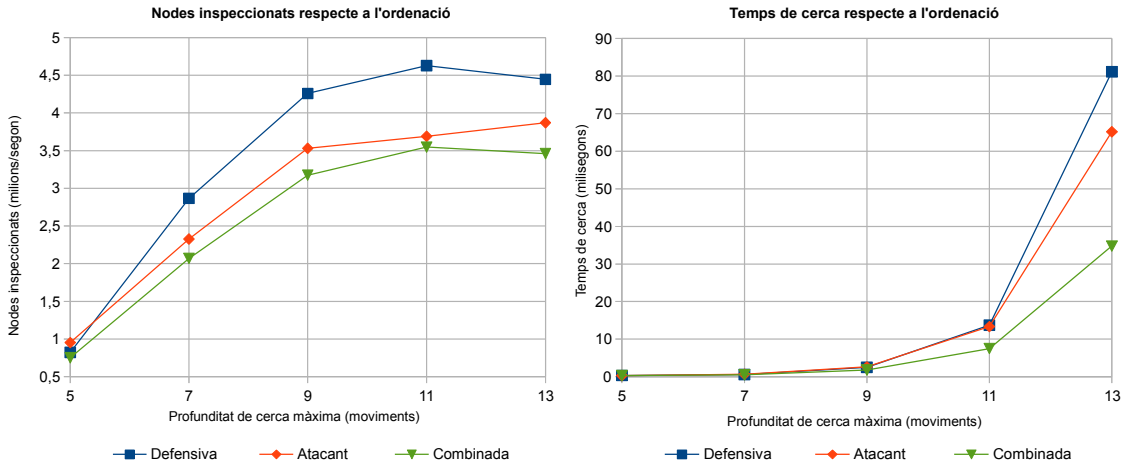


Figura 4.7: Comparació del rendiment de diversos algorismes d'ordenació

No s'ha inclòs als gràfics l'ordenació ingènua perquè és difícil d'implementar de forma òptima, ja que requereix calcular la legalitat de cada moviment individual. Per contra, en la resta d'algorismes aquesta comprovació es fa de forma implícita.

4.5 Gestió del temps

Observant els gràfics de l'anterior figura 4.7 es pot descobrir que el temps requerit per a avaluar l'estat d'un joc creix exponencialment amb la profunditat. És per això que prefixar una profunditat màxima resulta sovint poc apropiat, ja que el temps de resposta variarà considerablement segons la mida de l'arbre a expandir o la capacitat de processament. Normalment serà més interessant, per tant, poder establir un temps màxim de cerca.

L'aprofundiment iteratiu és un procediment usat molt freqüentment en la implementació de motors de cerca per a permetre fixar aquest temps de resposta. Aquest mètode consisteix a realitzar diferents cerques amb profunditats incrementals, de manera que la complexitat de l'arbre augmenta respecte al temps de cerca. S'obtenen així diverses aproximacions ràpides de l'avaluació, de les quals s'escull finalment aquella que s'ha fet amb major profunditat.

La figura 4.5 mostra la implementació d'un algorisme d'aprofundiment iteratiu per a un motor de cerca alfa-beta i un exemple d'un arbre expandit amb aquest procediment. Es pot ob-

servar que l'aprofundiment iteratiu normalment només s'aplica a l'arrel de l'arbre, tot i que es podria utilitzar també per a obtenir una avaluació ràpida de les fulles.

```
int searchRoot(Game game) {
    int depth = MIN_DEPTH;
    int beta = MAX_SCORE;
    int score;

    while (depth <= MAX_DEPTH) {
        for (int move : game.legalMoves()) {
             game.makeMove(move);
             score = negamaxSearch(
               game, -MAX_SCORE, beta, depth);
             game.unmakeMove();

             if (score < beta)
                 beta = score;
        }

        if (depth < MAX_DEPTH) {
            depth += 2;
            beta = MAX_SCORE;
        }
    }

    return -beta;
}
```

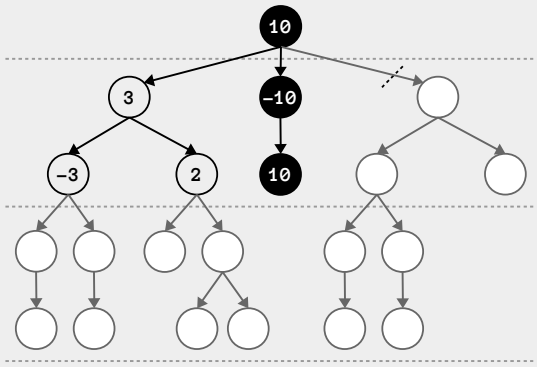


Figura 4.8: Aprofundiment iteratiu d'una cerca alfa-beta

Si bé inspeccionar els mateixos nodes una vegada rere l'altra pot semblar contraproduent, el rendiment de l'algorisme resulta millor que el d'una simple cerca en profunditat. D'entrada, perquè com que l'arbre creix exponencialment, el gruix dels nodes a inspeccionar es concentra a les fulles i, per tant, les primeres iteracions s'executen molt ràpid. A més, com que a la pràctica fer una cerca amb aprofundiment iteratiu és equivalent a fer una cerca en amplitud, l'algorisme és sovint capaç de trobar abans les solucions.

És important notar que tot i que en l'algorisme de la figura la cerca no s'atura fins que s'ha assolit la profunditat màxima, també existeix la possibilitat de forçar-ne l'aturada després d'un temps predeterminat. Quan aquest temps s'exhaureix, simplement es descarta el resultat de la darrera iteració parcial. Prefixar el temps de cerca ofereix l'avantatge addicional que les posicions més simples es podran avaluar més exhaustivament que aquelles més complexes.

CAPÍTOL 5:

Cerca amb memòria

Fins al capítol actual la cerca s'ha descrit únicament en un espai d'estats indefinit, determinat en funció d'expressions matemàtiques més o menys complexes. Aquesta és l'estratègia habitual quan l'espai d'estats d'un joc és molt gran i resulta impossible computar-lo completament. És evident però, que la cerca també es pot plantejar en un espai d'estats precalculat. Per exemple, amb un motor de cerca que obtingui les avaluacions d'una base de dades.

Per a jocs amb estats d'espais grans és fins i tot més habitual la combinació de les dues estratègies descrites, de manera que diversos algorismes de cerca col·laborin amb un objectiu comú. Quan això succeeix es diu que el motor de cerca disposa de memòria. En aquest capítol s'exposen les diferents tècniques i estructures de dades que s'han desenvolupat per a dotar l'aplicació d'aualé d'aquesta capacitat de recordar.

5.1 Disseny del model de memòria

Hi ha diferents solucions possibles per proveir un motor de cerca de memòria. Un recurs freqüent consisteix a calcular porcions de l'estat d'espais i emmagatzemar-les permanentment a l'espai físic de memòria. Això es fa sovint en forma de bases de dades d'obertures i de finals de joc. L'altre mitjà habitual consisteix a desar els resultats parcials obtinguts per un algorisme de cerca a la memòria volàtil per tal de poder-los reutilitzar posteriorment.

El motor de cerca per a l'aualé s'ha plantejat des d'un inici per tenir en compte aquestes possibilitats bàsiques. Al diagrama de classes de la figura 5.1 es descriu el model de memòria implementat. S'hi pot veure que el motor de cerca alfa-beta (`Negamax`) complementa la cerca en un espai d'estats definit per expressions matemàtiques amb la cerca en dos espais d'estats precalculats i residents a la memòria volàtil (`Cache` i `Leaves`). Mentre un tercer algorisme de cerca permet obtenir avaluacions des de la memòria permanent (`Roots`).

Aquesta darrera interfície, `Roots`, defineix la cerca en un llibre d'obertures. És a dir, en una base de dades que representa la porció de l'espai d'estats que s'obtindria de l'expansió, fins a

una profunditat determinada, de l'estat inicial d'un joc. Per la seva banda, la interfície `Leaves` també defineix la cerca en un espai d'estats precalculat, però aquest cop les dades emmagatzemades a la base de dades representen l'altre extrem del graf del joc, aquell format pels estats terminals i els nodes que hi condueixen: el llibre de finals de joc.

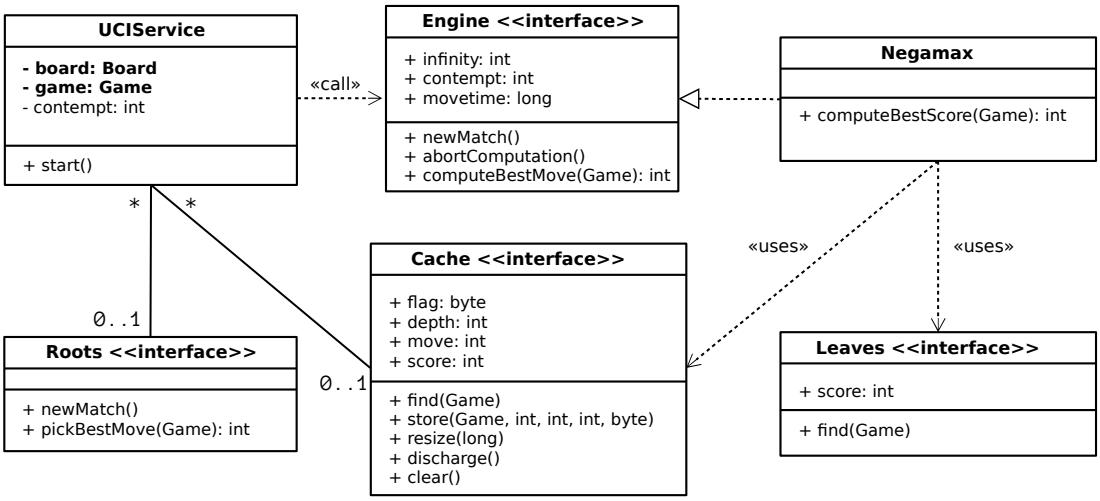


Figura 5.1: Diagrama de classes del model de memòria

El propòsit de la tercera interfície descrita, `Cache`, és omplir el buit entre aquests espais d'estats precomputats. Un algorisme que realitzi la cerca en un espai que no s'hagi precalculat, com pot ser alfa-beta, en pot fer ús per a recordar els càlculs més rellevants que es vagin obtenint durant l'execució. Per consegüent, té la funció d'una memòria cau ubicada a la memòria volàtil i normalment només desa informació de l'espai d'estats corresponent al mig joc.

La utilitat de la classe `UCIService` en tot aquest entramat pot resultar confusa. Es tracta simplement de la implementació del protocol de comunicació, que actua a la pràctica com a coordinador dels diferents algorismes de cerca. Al capítol 7 es descriurà amb més de detall.

5.2 Memòria permanent de l'aualé

Com s'ha assenyalat, la memòria permanent d'un joc està composta per un seguit de bases de dades a on es caracteritzen porcions de l'espai d'estats del mateix. Sovint cada entrada de la base de dades conté la representació d'una posició, que serveix per identificar un estat, i informació sobre l'avaluació exacta o heurística de l'estat que aquesta defineix.

Per al motor d'aulé la memòria permanent s'ha implementat amb dues estructures de dades diferents, dissenyades específicament per al joc i optimitzades respecte a l'ús que se n'ha de fer. Com no podia ser d'altra manera, una representa les obertures i l'altra els finals de joc.

5.2.1 Llibre d'obertures

La base de dades de posicions inicials de l'aualé s'ha implementat en forma de taula ordenada, on cadascuna de les seves entrades ocupa exactament 20 bytes. Els primers vuit octets de cada entrada emmagatzemen la representació d'una posició en forma de resum binomial. La resta d'octets, agrupats de dos en dos, contenen l'avaluació heurística precalculada de cadascun dels fills del node que la posició defineix en el graf del joc.

La funció d'aquesta estructura és descriure els millors moviments que es poden fer en un estat de l'aualé. Determinar el millor moviment possible per a un estat consisteix a cercar a la taula la representació en forma de resum binomial d'una posició i, quan s'ha localitzat, en escollir l'aresta d'avaluació heurística més gran. Cal notar que les avaluacions de les arestes s'hi desen en un ordre prefixat —segons el jugador a qui toca moure, de *A* a *F* i de *a* a *f*— i els moviments il·legals s'avaluen amb un valor molt negatiu.

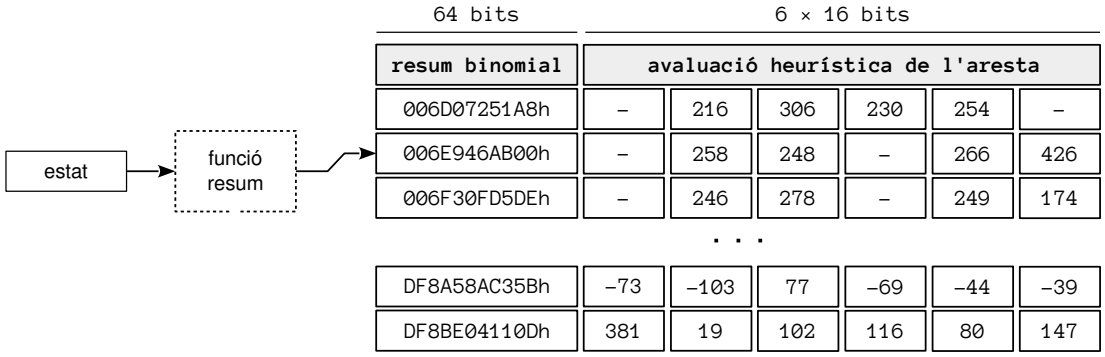


Figura 5.2: Estructura i funcionament de la base de dades d'obertures

La figura 5.2 mostra l'estructura de dades descrita. Es pot observar que el resum binomial hi té dues funcions diferenciades. D'una banda, descriu de manera inequívoca un estat del joc i, d'altra banda, determina l'ordre en el qual s'emmagatzemen els estats a la base de dades. Això darrer permet que es pugui localitzar un estat concret ràpidament i sense massa complicacions amb un algorisme de cerca binària.

Per tant, trobar un element de la taula es pot fer amb una complexitat O($\log n$). Tanmateix, localitzar un estat no és una operació molt crítica, ja que només s'ha de realitzar un cop per cada node arrel; un cop localitzat un estat el motor de cerca pot tornar el resultat immediatament. D'altra banda, una representació en forma de taula presenta els avantatges de reduir les necessitats d'emmagatzematge i de simplificar molt la implementació.

El capítol 6 descriu com s'ha construït un llibre d'obertures que conté unes 33 mil entrades i prop de 162 mil avaluacions. Localitzar-hi un estat requereix com a màxim 11 accessos al disc, tot i que a la pràctica seran molts menys gràcies a la mida reduïda del llibre (673 kB).

5.2.2 Llibre de finals

L'estructura de dades necessària per a la base de dades de finals de l'aualé per força ha de ser molt diferent de la utilitzada per al llibre d'obertures. El motiu és què ambdues responen a funcions també molt diferents. Mentre que el llibre d'obertures s'usa per a realitzar cerques independents en un estat d'espais precalculat, la utilitat principal del llibre de finals és complementar l'avaluació dels nodes fulla en l'algorisme alfa-beta.

Com s'ha vist en els capítols anteriors, un algorisme alfa-beta amb profunditat limitada usa sobretot dues funcions d'avaluació distintes. La funció d'utilitat serveix per avaluar els nodes terminals de l'arbre amb un resultat exacte, mentre que la funció heurística s'usa per a obtenir una aproximació de l'avaluació d'aquells nodes que no són terminals. El problema és que usar una funció d'avaluació heurística significa prendre decisions imperfectes.

El benefici d'un llibre de finals resideix en què possibilita obtenir avaluacions exactes per a nodes no terminals, minimitzant així el nombre de decisions imperfectes. De manera que quan l'algorisme alfa-beta necessita avaluar un node no terminal pot comprovar primer si l'estat es troba a la base de dades de finals i obtenir d'allà la seva avaluació exacta.

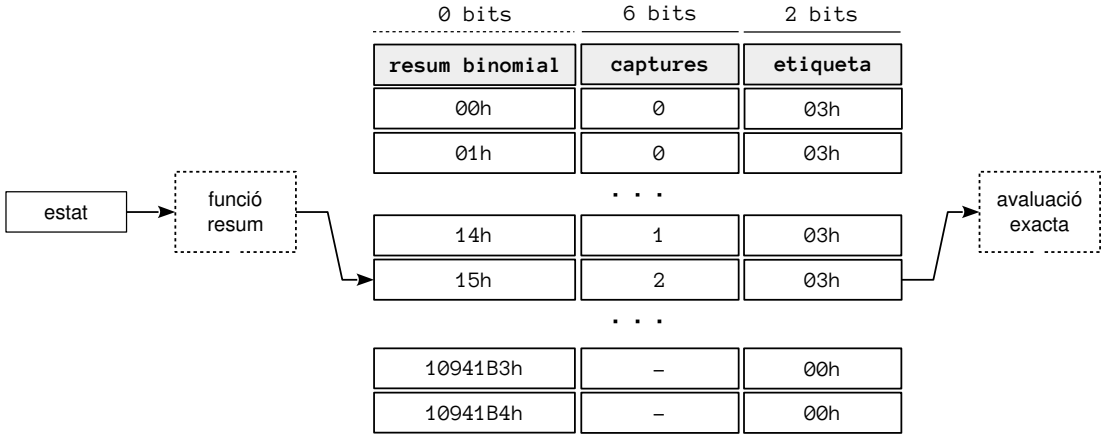


Figura 5.3: Estructura i funcionament de la base de dades de finals

En conseqüència, l'estructura de dades ha de permetre localitzar estats i obtenir la seva informació de manera molt eficient. Això requereix moltes vegades una estructura en forma de taula indexada per fer possible l'execució d'aquestes operacions en temps constant O(1). A més, la base de dades ha de ser molt compacte perquè sovint serà interessant mantenir les dades més rellevants a la memòria volàtil per tal de poder accedir-hi més ràpidament.

L'estructura en forma de taula indexada que s'ha implementat per a l'aualé és la que es pot veure a la figura 5.3. Com s'hi pot observar, cada entrada de la taula ocupa només un byte de memòria, gràcies al fet que l'índex d'un estat ve definit per un resum binomial.

El procediment seguit per a avaluar un estat a partir de la informació de la base de dades es pot resumir amb les següents operacions:

1. S'obté un índex de la taula construint el resum binomial d'un estat.
2. S'accedeix a la ubicació de memòria definida per l'índex. S'aconsegueix així el nombre màxim de llavors que el jugador que mou pot capturar en la posició i el tipus d'avaluació emmagatzemada (pot ser *exacta* o *desconeguda*).
3. Quan el tipus d'avaluació és *exacta*, el nombre màxim de llavors que es podran capturar es suma a les llavors ja captures pel jugador. L'avaluació exacta serà *guanyat* si el total és major a 24, *empat* si el total és igual a 24 i *perdut* en qualsevol altre cas.

Hi ha dues propietats importants a destacar de l'estructura. La primera és que cada entrada de la taula simbolitza no un sinó diversos estats del joc. La segona és que els resums binomials de les posicions són implícits. Romain, Bal (2002) i altres autors utilitzaven estructures similars a l'explicada per construir bases de dades per a l'*awari*. La increïble compacitat del llibre de finals es fonamenta en tres propietats importants de l'aualé.

- Només influeix en el resultat final la distribució de llavors que es troben en joc. El nombre de llavors que ja s'havien capturat en un estat és irrellevant.
- Només cal emmagatzemar els estats en els quals el jugador sud ha de moure. Rotant el tauler 180º es pot obtenir la mateixa posició des del punt de vista del jugador nord.
- Qualsevol posició es pot representar amb un resum binomial que és perfecte i mínim. Això significa a la pràctica que no hi haurà espais buits a la taula i que no cal desar-hi la representació binomial, ja que aquesta es pot inferir de la ubicació d'una entrada.

El fet que la representació sigui tan compacta ha permès mantenir el llibre complet a la memòria volàtil. Només calen 16,5 MB per representar tots els estats amb 15 o menys llavors en joc, és a dir, centenars de milions d'estats diferents. Gràcies a això, aconseguir avaluacions del llibre resulta extraordinariament eficient.

5.3 Memòria volàtil de l'aualé

Al primer apartat d'aquest capítol es descrivia la interfície `Cache` com una espècie de memòria cau dels resultats obtinguts per un algorisme de cerca alfa-beta. Una implementació de la interfície defineix, per tant, un tipus de memòria no permanent, la funció de la qual és recordar les decisions preses anteriorment per tal d'optimitzar millor la cerca.

A l'estructura de dades que fa possible que un algorisme recordi se l'anomena habitualment taula de transposicions. Com en el cas dels llibres d'obertures i de finals, és una estructu-

ra que emmagatzema l'avaluació dels estats del joc, però a diferència els anteriors, també s'hi desa informació dels camins recorreguts per l'algorisme de cerca. L'objectiu perseguit és reduir l'arbre de cerca mínim per dos mitjans: evitant recórrer aquells camins que ja s'havien avaluat anteriorment i millorant l'ordenació de les arestes.

5.3.1 Estructura de dades

La figura 5.4 mostra la taula de transposicions implementada. Les dades que s'hi desen són les usuals en una estructura d'aquest tipus. Un identificador numèric d'un estat; l'avaluació heurística del mateix estat i una etiqueta que descriu el tipus d'avaluació feta; la profunditat de cerca a la qual s'ha avaluat l'estat; el millor moviment que s'ha estimat per a l'estat i, finalment, una estampa per distingir el període de temps en el qual s'ha creat l'entrada.

0 bits	11 bits	7 bits	2 bits	4 bits	2 bits	32 bits	0 bits
índex	**avaluació**	**profunditat**	**etiqueta**	**moviment**	**estampa**	**resum binomial**	
000000h	−60	5	03h	2	02h	AEC39000h	000h
000001h	68	4	01h	–	03h	311DA200h	000h
			...				
002710h	65	10	03h	10	01h	25627401h	710h
002711h	50	5	01h	9	02h	8D008E01h	710h
			...				
3FFFFEh	50	3	02h	2	02h	902259FFh	FFEh
3FFFFFh	–	–	–	–	00h	–	FFEh

Figura 5.4: Estructura i contingut de la taula de transposicions

L'estructura és una taula de dispersió (*hash table*) que conté la informació codificada de manera compacta. Es pot veure que cada entrada emmagatzema informació d'un únic estat i en només 58 bits, el que permet implementar la taula de dispersió en forma de vector de valors de 64 bits. És doncs una estructura que s'ha optimitzat tant en termes de memòria com respecte al temps —totes les operacions crítiques es poden fer en temps constant O(1).

L'optimització quant a la memòria respon sobretot a la necessitat de recordar la informació del màxim nombre d'estats possibles. Com més estats s'emmagatzemin a la taula, més probable és que l'estat que necessita el motor de cerca en un moment donat s'hi pugui trobar. Pel mateix motiu, la taula s'inicialitza amb una mida fixa en engegar el motor de cerca i els estats que s'hi desen es recorden entre les diferents invocacions del motor.

És necessari destacar que només part de la informació que identifica un estat, el resum binomial, es desa a la taula de manera explícita. Els 12 bits més baixos són implícits en l'índex de la taula en el qual es desa la informació. Aquesta és una tècnica usada freqüentment per reduir la mida de les dades, que requereix que la funció d'indexació i la funció de resum binomial siguin la mateixa i, que la mida de la taula sigui una potència de dos.

5.3.2 Funcionament i utilitat

L'algorisme alfa-beta usa la taula de transposicions desant i recuperant la informació dels estats del joc durant la cerca. En la fase de propagació de valors l'algorisme crea entrades a la taula i durant la fase d'expansió n'obté la informació emmagatzemada.

Per a poder implementar aquestes operacions cal però determinar primer la ubicació de la memòria que li correspondria a un estat. Això es fa calculant el resum binomial de l'estat i obtenint llavors l'índex que li correspon com el mòdul del resum per la mida de la taula. Com que la mida és una potència de dos, es pot fer eficientment amb l'operació `resum & (2^n - 1)`.

Les operacions d'emmagatzematge i recuperació de dades es poden implementar llavors amb operacions molt simples que s'executen en O(1). Cal tenir en compte però que, com que la taula és de mida fixa, caldrà alguna estratègia de substitució per a resoldre les col·lisions que es puguin produir. En la darrera secció d'aquest capítol es descriuen aquestes estratègies.

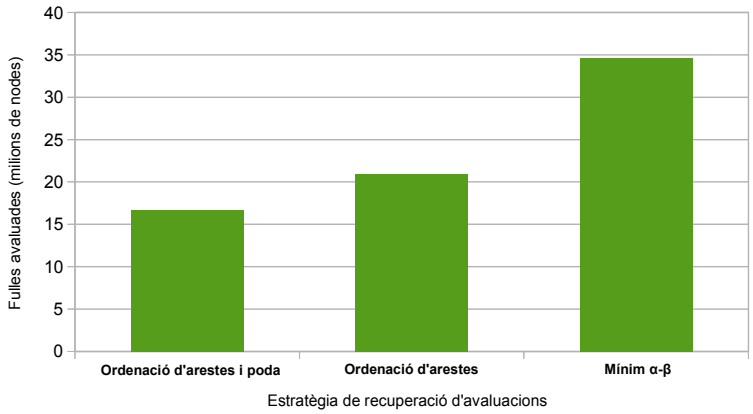


Figura 5.5: Reducció de l'arbre de cerca amb l'ús de memòria volàtil ($d = 13$)

S'ha comentat abans que el resultat buscat amb la taula de transposicions és reduir el nombre de nodes que l'algorisme de cerca ha d'inspeccionar. També s'ha dit que això es feia amb dues estratègies diferents. La figura 5.5 mostra el resultat d'implementar aquestes estratègies en el motor d'aualé. Com es pot veure, la taula de transposicions ha permès reduir els nodes avaluats fins a menys de la meitat dels que s'havien aproximat per a l'arbre mínim.

Això s'ha aconseguit, d'una banda, recuperant de la taula l'avaluació dels camins que ja s'havien inspeccionat per evitar recorre'ls de nou; d'una forma similar a com es feia amb el llibre de finals. D'altra banda, recuperant els millors moviments estimats per a cada estat per tal d'inspeccionar-los abans. Com s'ha vist al capítol 4.4, una ordenació de moviments acurada permet reduir el nombre de branques de l'arbre de cerca que és necessari recórrer.

El benefici més important de la taula de transposicions es troba no obstant això en la ponderació de moviments. Aquest és un mètode molt conegut que s'utilitza per a maximitzar el

temps de càlcul disponible durant el transcurs d'una partida. Consisteix a intentar predir el moviment que farà l'adversari per tal de començar a computar-ne les respostes mentre el rival està pensant. D'aquesta manera, en la següent invocació del motor de cerca la taula de transposicions ja contindrà informació precalculada de la majoria de camins.

5.3.3 Estratègies de substitució

Pel fet que diversos estats del joc es poden assignar a una mateixa adreça de memòria, s'ha evidenciat abans que una taula de dispersió de mida prefixada requereix algun esquema de substitució per a resoldre aquestes les col·lisions.

Que aquest és un tema de recerca important s'evidencia pels múltiples mètodes disponibles a la bibliografia que s'han utilitzat exitosament per a diversos jocs amb adversari. Són tots mètodes però que persegueixen un mateix propòsit: maximitzar el benefici de la taula de transposicions. Aquest benefici serà màxim quan es minimitzi el temps necessari per a realitzar una cerca amb profunditat limitada.

Amb aquesta finalitat, la majoria de motors de cerca moderns utilitzen no un sinó diversos mètodes de substitució. No és l'objectiu d'aquesta secció descriure'ls tots, sinó només aquells implementats per al motor d'aualé. Com a base de l'esquema de substitució s'ha utilitzat el sistema de dos nivells concebut per Condon i Thompson (1983) per al seu computador d'escacs Belle, i s'ha ampliat posteriorment amb algunes millores.

Aquest sistema es fonamenta en dues observacions que poden semblar contradictòries. La primera és que els nodes avaluats més recentment és més probable que es tornin a cercar. La segona, que aquells nodes avaluats amb més profunditat, i que per tant representen camins més llargs, estalviaran més càlculs al motor de cerca. L'una dóna per tant més preferència als nodes més profunds de l'arbre, mentre que l'altra als nodes més interns.

El sistema en dos nivells combina dos esquemes de substitució per mitjà d'una taula en la qual cada estat pot indexar-se en dues posicions de memòria diferents. En la primera posició tenen més prioritat els camins més llargs, de manera que només s'hi desarà un estat quan aquest s'ha avaluat a major profunditat que l'estat que es troba ja a la taula. Quan això no es compleix, l'estat s'emmagatzemarà a la segona posició de memòria.

Aquest esquema de substitució resulta força efectiu, com es pot observar a la figura 5.6, a on els esquemes 3, 4 i 5 representen els sistemes de substitució explicats. L'esquema 3 és el sistema de dos nivells ideat per Condon i Thompson, mentre que els esquemes 4 i 5 representen respectivament un reemplaçament on es dóna més prioritat als nodes avaluats a major profunditat i un reemplaçament on es dóna més prioritat als nodes avaluats recentment.

No està de menys observar que l'esquema 4, tot i tenir una taxa d'encerts molt baixa, re-

presenta un estalvi de temps bastant notable. Els gràfics s'han elaborat mitjançant el banc de proves descrit al capítol 4.4, cercant amb un aprofundiment iteratiu màxim de 17 moviments i sense límit de temps. Per a poder interpretar-los millor, cal indicar que el temps mitjà de cerca sense la taula de transposicions era de 1310 milisegons.

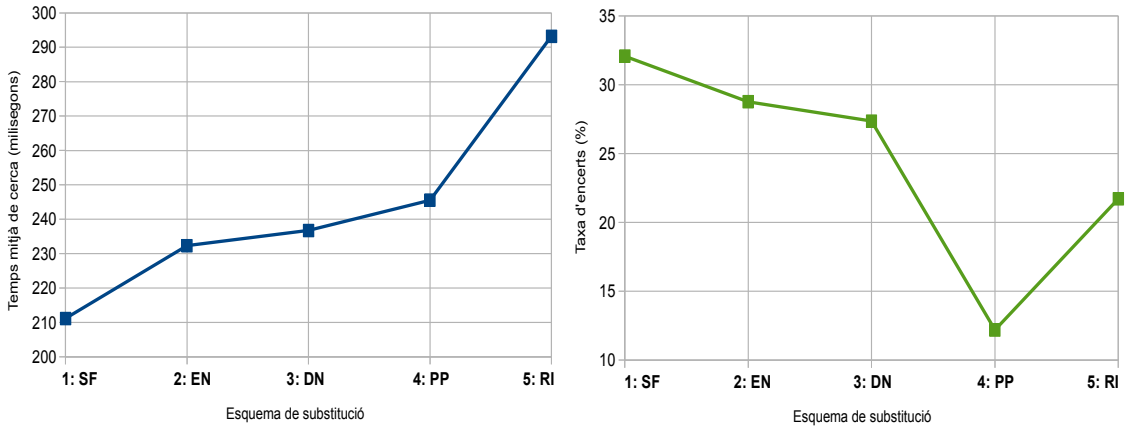


Figura 5.6: Rendiment de la taula de transposicions respecte a l'esquema de substitució

Els esquemes 1 i 2 són una millora de la substitució en dos nivells. El mètode 2 afegeix al sistema de dos nivells un esquema de substitució d'entrades antigues. Cada entrada desada a la taula es marca amb una estampa de temps i, quan es detecta que una entrada és massa antiga, se substitueix incondicionalment. Aquest és un sistema que se sol usar per a millorar la taxa d'encerts de la memòria cau d'un processador.

L'esquema 1, per la seva banda, afegeix a aquest darrer mètode presentat una condició per a evitar emmagatzemar a la taula els nodes de la prefrontera de l'arbre. És a dir, aquells estats que avaluen camins d'una longitud igual o menor a tres. S'ha implementat aquest mètode després d'observar, durant l'elaboració del motor d'aualé, que el temps requerit per a desar aquests estats era major que el necessari per a cercar de nou els nodes.

Capítol 6: Aprenentatge automàtic

Al capítol anterior ja es presentava un mètode d'aprenentatge automàtic amb aplicació als jocs amb adversari. S'hi feia referència a una estructura de dades —la taula de transposicions— que possibilita que un motor de cerca adquireixi coneixement durant l'execució. L'aprenentatge però té molts altres usos en l'elaboració d'aplicacions amb capacitat de joc.

En aquest capítol es presenta un model de regressió lineal aplicat a l'afinació de la funció d'avaluació heurística. El mètode ha permès que el motor d'aualé millori considerablement el seu nivell de joc gràcies a l'adquisició de nou coneixement sobre d'estratègia. Finalment, també es descriuen en aquest capítol els algorismes usats per a l'aprenentatge d'obertures i finals. Essent aquests darrers procediments que basen el seu funcionament en l'anàlisi i memorització dels estats més rellevants d'un joc.

6.1 Afinació de l'avaluació heurística

Tal com s'ha explicat a la secció 4.3, l'avaluació heurística té una funció de predicció en un algorisme de cerca. Ha d'observar els nodes fulla d'un arbre de cerca i determinar de manera més o menys precisa quin seria el resultat obtingut en cas de seguir aquell camí.

L'elaboració de funcions heurístiques es fonamenta sovint en el coneixement humà adquirit al llarg dels anys. De manera que es construeixen expressions matemàtiques que descriuen diverses característiques del joc que se sap que afavoreixen o perjudiquen un jugador. Llavors els pesos d'aquestes característiques en l'avaluació es poden ajustar a partir de mètodes empírics o, per exemple, amb l'ús d'algorismes genètics.

Aquest procediment, basat en algorismes genètics, és el que van utilitzar per exemple Daoud, Kharma, Haidar i Popoola (2004) per afinar l'avaluació d'una aplicació d'awari amb resultats favorables. Es tracta però d'un procediment costós que en les primeres proves no ha donat bons resultats per al motor d'aualé elaborat.

Després d'observar que les propietats de l'aualé fan que sigui possible mesurar numèricament el guany que es pot obtenir d'una posició del joc (vegeu l'apartat 4.3), per a aquest projecte s'ha preferit elaborar un mètode d'afinació fonamentat en l'anàlisi de regressió. Els resultats, tot i que difícils de mesurar, mostren una millora notable de la capacitat de joc. En totes les proves fetes, disputant diverses partides fins al final, la nova funció heurística ha fet possible que el motor de cerca sortís invicte.

6.1.1 Premisses i hipòtesis

Per a què sigui efectiva, una funció d'avaluació ha de complir certes condicions. D'entrada, ha d'estimar la probabilitat d'un jugador de guanyar una partida i, en darrer terme, ha d'explicar com d'aprop de la victòria es troba el jugador. Cal doncs que torni valors positius si un estat és avantatjós i que aquests valors s'aproximin cada cop més al resultat final a mesura que els camins avaluats s'acostin també a la conclusió.

També és important, especialment en un marc de cerca *negamax*, que la funció heurística sigui simètrica. Un estat que s'ha avaluat amb un valor per a un jugador, s'ha d'avaluar amb el mateix nombre i signe invers des del punt de vista del seu rival. És a dir, es parteix de la suposició que les característiques que són favorables per a un jugador són igualment desfavorables per al seu adversari i viceversa.

Les característiques a avaluar han de ser a més mesurables en un temps constant. Cal que la funció heurística sigui simple o en cas contrari el temps consumit en l'avaluació aniria en detriment d'una major profunditat de cerca. Això significa que la funció ha de mesurar només les propietats presents en una posició del joc, sense tenir en compte el resultat exacte d'efectuar moviments en la mateixa.

La darrera hipòtesi i també la més rellevant és que a la pràctica no cal saber el resultat exacte de seguir el camí òptim des d'un estat per tal d'aproximar una funció de regressió lineal apropiada. Per a determinar quines característiques d'una posició són favorables i en quina mesura és suficient amb estimar el guany màxim que es pot obtenir en un futur proper.

6.1.2 Preparació de dades

La primera tasca per a elaborar un model de regressió lineal consisteix en l'obtenció d'una mostra de la població que cal analitzar. Gràcies a la funció de resum binomial descrita a la secció 3.4.3 aquesta és una comesa fàcil per a l'aualé: només cal obtenir un seguit de nombres aleatoris que descriuen inequívocament diverses posicions del joc i llavors eliminar de la mostra aquelles posicions que representen estats terminals.

Amb aquest mètode s'han obtingut dues mostres de la població, la primera per mostreig

aleatori simple amb reposició (41.036 posicions no terminals d'aualé) i la segona per mostreig aleatori proporcional a la grandària (32.591 posicions). Aquesta darrera mostra, que s'ha usat únicament per a la validació dels resultats, respon a l'observació que les posicions amb un nombre menor de llavors en joc és més probable que apareguin en el transcurs d'una partida.

El següent pas ha consistit en la selecció empírica i recompte de les característiques que podrien ser rellevants estratègicament. Aquestes propietats són, per una banda, la diferència entre les llavors capturades per un jugador i les capturades pel seu rival i, per l'altra banda, quants forats existeixen al tauler que contenen un nombre determinat de llavors.

Això darrer pretenen mesurar les possibilitats que té un jugador de capturar llavors i de què el rival no en capturi. Es pot deduir del reglament del joc que un nombre gran de forats amb menys de 3 llavors al terreny d'un jugador farà més probable que el seu adversari pugui capturar llavors en un futur. D'altra banda, com més elevat el nombre de forats des dels quals un jugador pot sembrar al terreny del rival, més gran la probabilitat de capturar llavors.

En la següent fase de preparació de dades s'ha estimat del benefici màxim que es pot aconseguir en una posició. Això s'ha fet amb una cerca alfa-beta de temps limitat (200 milisegons) que tornava com a resultat la diferència màxima de llavors capturades assolible en un futur pròxim. Aquests valors s'han transformat després en aproximacions de l'avaluació exacta en l'interval [-1000, 1000]; multiplicant per una constant quan el valor absolut de captures era inferior a 24 i, ajustant al centre i extrems de l'interval la resta de valors.

Un darrer pas important en la preparació, enfocat a l'obtenció d'una funció de regressió completament simètrica, és duplicar i transformar tots els individus de la mostra per tal que representin les mateixes posicions des del punt de vista del rival. Això s'aconsegueix, d'una banda, negant l'avaluació estimada i la diferència de llavors, i de l'altra banda, capgirant la resta de característiques, de forma que un forat pertanyent al jugador sud que per exemple no contenia llavors ara esdevindrà un forat del nord amb la mateixa propietat.

6.1.3 Regressió lineal simple

A partir de les mostres descrites i un algorisme de regressió lineal simple, s'ha pogut derivar la funció de regressió de la figura 6.1. La taula mostra les característiques seleccionades per l'algorisme i el seu pes en la recta de regressió.

Excepte la primera, que representa la diferència inicial de captures en una posició, les característiques que s'hi observen corresponen al nombre de formats del jugador sud que contenen un nombre de llavors inferior o superior a l'indicat. No es mostren a la taula les propietats anàlogues del jugador nord, que com era d'esperar són les mateixes però de pesos negats.

Aquesta funció de regressió defineix una relació lineal forta respecte al benefici que s'ha-

via aproximat per a les posicions. La correlació de l'estimació és 0,8149 i l'error mitjà 121,75 punts. El valor de l'error mitjà significa que la funció és capaç de predir el nombre de llavors que un jugador podrà capturar amb un marge d'error de només 3 llavors.

Característica	Captures	< 1	< 2	< 3	< 4	< 5	> 7	> 8	> 10	> 11	> 12	> 13
Pes	36,65	-19,09	-18,08	-47,37	-3,33	6,80	-20,55	-4,76	12,13	17,66	20,70	18,92

Figura 6.1: Funció de regressió lineal de l'avaluació heurística

La funció de regressió lineal estima per tant força acuradament com de favorable és una posició per al jugador a qui li toca moure-hi. Aquesta seria ja una bona funció d'avaluació heurística, tot i que és massa complexa i s'ajusta excessivament a la mostra. Per aquest motiu s'ha procedit a simplificar-la, eliminant progressivament de la funció les característiques de menys pes fins a obtenir-ne un conjunt mínim. Només cal tenir en compte que aquest conjunt mínim ha de permetre aconseguir la mateixa correlació que el conjunt original.

Característica	Captures	< 1	< 3	> 12
Pes	36,58	-27,03	-51,94	40,51

Figura 6.2: Funció de regressió lineal simplificada

La taula de la figura 6.2 mosta el conjunt mínim de característiques. La correlació aconseguida amb aquest conjunt és 0,8086 i l'error mitjà de la regressió 124,2605. Continua essent una relació lineal forta i el marge d'error també es correspondria amb 3 llavors.

6.1.4 Anàlisi dels resultats

De la funció obtinguda es poden deduir les estratègies més rellevants del joc. A banda de capturar més llavors que l'adversari, un jugador voldrà minimitzar el nombre dels seus forats que es troben buits o que contenen 1 o 2 llavors, alhora que es maximitza aquest factor en els forats del rival. La darrera característica —el nombre de forats amb més de 12 llavors— es correspon amb una estratègia molt coneguda entre els jugadors d'aualé.

Al joc de l'aualé s'anomena *kroo* a un forat que conté dotze o més llavors. Acumular llavors en grups grans, de forma que es puguin sembrar posteriorment donant una o més voltes al tauler, fa possible que en un únic moviment un jugador pugui capturar sovint un nombre molt elevat de llavors (fins a quinze). Permetre que el rival faci una captura d'aquest tipus és per tant un error estratègic normalment irrecuperable.

A banda d'analitzar les estratègies de l'aualé és interessant també comparar l'ajustament de la funció d'avaluació heurística que s'ha obtingut respecte a la funció més simple que s'ha-

via definit en la secció 4.3. La figura 6.3 mostra els resultats de fer d'aquesta comparació.

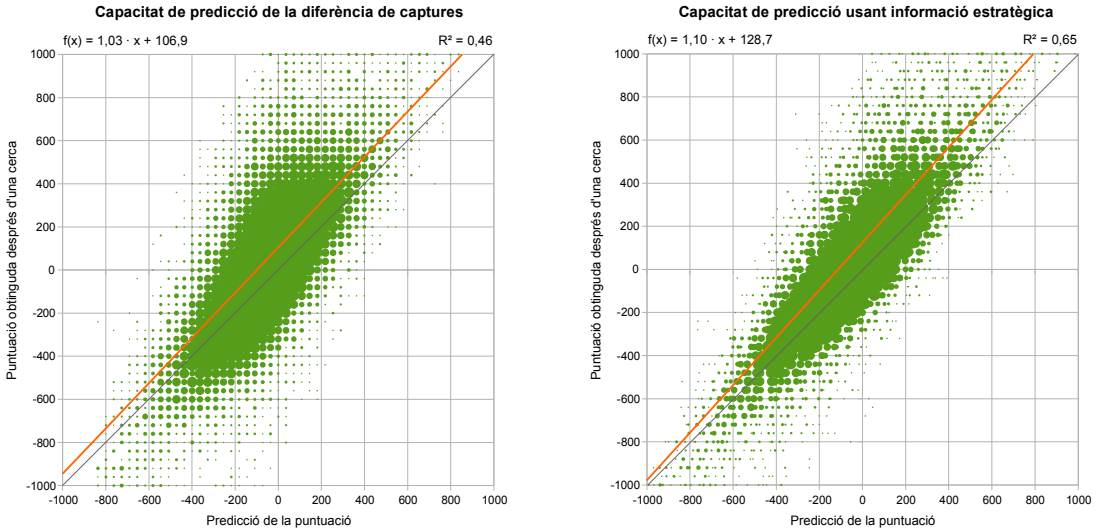


Figura 6.3: Diagrames de dispersió de l'avaluació heurística

El gràfic de l'esquerra posa en relació l'estimació del benefici màxim que es pot obtenir d'una posició —tal com s'ha definit a l'apartat 6.1.2— amb l'avaluació feta a partir d'una funció de regressió que només té en compte les llavors capturades inicialment. El gràfic de la dreta mostra la mateixa relació però respecte a l'avaluació obtinguda amb la funció de la figura 6.2.

Es pot veure que la relació lineal és forta en aquest darrer cas i moderada en el primer. Les observacions es troben menys disperses i s'ajusten millor a la recta de regressió. Cal fixar-se en el fet que la predicció seria perfecta si totes les observacions se situessin sobre la diagonal. Per consegüent, es podria dir que la funció heurística obtinguda és lleugerament pessimista.

6.2 Construcció d'un llibre d'obertures

Un llibre d'obertures ha d'emmagatzemar l'arbre de cerca que s'obté de l'expansió de l'estat inicial fins a certa profunditat. Generalment aquesta profunditat serà variable i comprendrà només la porció dels camins que va de l'estat inicial fins al mig joc.

L'objectiu de l'expansió és doble. D'entrada, es pretén minimitzar els errors que una cerca heurística pot cometre durant l'obertura d'un joc, ja que els estas que compren són sovint els més complexos. Per tant, amb el llibre d'obertures s'intenta desplaçar l'estat de joc a posicions més simples i fàcils de resoldre amb una cerca heurística. El segon objectiu és maximitzar la variabilitat de joc per evitar que el motor de cerca esculli sempre els mateixos camins.

Per a elaborar un llibre d'obertures hi ha dos mètodes possibles. El procediment més habi-

tual aprofita el coneixement humà disponible per mitjà de l'anàlisi d'un corpus de partides anotades del que s'extrauran els estats més rellevants. Sovint aquests estats són els que s'han produït més freqüentment en les partides disputades entre els jugadors humans més experts. El segon mètode, basat en una avaluació heurística completament automatitzada, s'usa quan no existeix un corpus de partides extens per a un joc.

L'aualé és un joc que no s'acostuma a anotar, en conseqüència, no existeix un corpus de partides que es pugui utilitzar. No queda altra alternativa doncs que l'expansió automatitzada. Per a aquesta expansió s'ha utilitzat un mètode ideat per Lincke (2002) —anomenat *drop-out expansion*— per a concentrar els esforços en el càlcul dels nodes més importants.

La construcció automatitzada d'una base de dades d'obertures és un procés lent que requereix molts càlculs. Aquest procediment es pot resumir en un seguit de passos que s'aniran repetint fins que es consideri que la base de dades és suficientment completa o fins que s'hagi aconseguit solucionar el node arrel.

1. S'escull un node del llibre per a la seva expansió.
2. Es generen tots els fills del node i s'afegeixen a la base de dades.
3. Es calcula l'avaluació heurística, per mitjà d'una cerca, de cadascun dels nodes generats.
4. Es propaguen les avaluacions heurístiques fins a l'arrel de l'arbre.

La innovació presentada per Lincke resideix en el mètode de selecció dels estats a expandir. A cada node contingut al llibre d'obertures se li assigna una prioritat d'expansió, que serà més gran com més a prop de l'arrel es trobi el node i més petita com més avantatjós sigui l'estat per al jugador. D'aquesta manera s'aconsegueix que els camins que porten a un avantatge clar per a un jugador s'expandeixin amb menys profunditat que la resta.

Amb aquest mètode s'ha elaborat un llibre d'obertures d'aualé que conté l'avaluació de 162.889 posicions del joc. Aquestes avaluacions s'han calculat amb el motor d'aualé implementat i realitzant cerques en cada estat amb un temps limitat a 10 segons. Per a la construcció del llibre han calgut, doncs, aproximadament 250 hores de càlcul, però els resultats mostren que el nivell de joc de l'aplicació ha millorat considerablement.

6.3 Construcció d'un llibre de finals

Als capítols anteriors s'esmentava una tècnica d'anàlisi retrògrad que s'havia utilitzat per a solucionar completament la variant d'aualé *awari*. També es comentava la problemàtica d'usar aquest mètode per a l'aualé abapa i la impossibilitat de solucionar aquesta variant mitjançant els procediments proposats.

6.3.1 Anàlisi retrògrad de l'aualé

El mètode d'anàlisi retrògrad que va permetre solucionar la variant *awari* es pot descriure amb el següent procediment. Amb una representació d'estats en forma de resum binomial resulta molt eficient respecte al temps de càlcul i també de memòria requerida.

1. Es generen totes les posicions d'aualé que es poden donar des del punt de vista del jugador sud. És important fixar-se que el torn de joc o el nombre de captures efectuades pels jugadors no és rellevant (vegeu la secció 5.2.2).
2. S'assigna una avaluació exacta a tots els estats terminals que s'hagin generat i es marquen com a solucionats. Aquesta avaluació correspon al nombre de llavors que el jugador sud pot capturar en la posició menys les llavors restants, i serà com a màxim n.
3. Per cada node del llibre, es propaguen les avaluacions dels nodes solucionats als seus pares. Aquesta propagació es fa amb el mètode *minimax*.
4. Es marquen com a solucionats els nodes pels quals tots els seus fills s'han solucionat o que l'avaluació d'un dels fills és exactament n (màxima). Es continua des del pas 3 fins que no es pugui marcar com a solucionat cap més node.
5. En aquest punt, tots els nodes que podien rebre una avaluació exacta igual a n ja l'han rebuda. Si n és més gran que zero, se'n decrementa el valor i el procés continua al pas 3.
6. Tots els nodes que no s'han pogut solucionar resulten en empat. Per tant, se'ls assigna la seva avaluació exacta i es marquen com a solucionats.

La problemàtica d'aplicar aquest procediment a l'aualé abapa es troba en el mètode usat per a la detecció de cicles. Aquest mètode, representat finalment en el pas 6, parteix de la suposició que quan un node forma part d'un cicle el repartiment de llavors restant es farà en parts iguals entre els jugadors. Com s'ha vist a la secció 2.1.1 d'aquesta memòria, aquesta suposició no es pot aplicar a la variant d'aualé abapa.

El mètode d'anàlisis retrògrad és vàlid per abapa fins al pas 5, però cal un procediment diferent per avaluar els nodes que formen part d'un cicle. El conjunt de nodes que pertanyen a un cicle és però molt gran el joc. Amb el procediment implementat finalment per a l'aplicació, s'ha estimat en aproximadament el 45% dels nodes en posicions amb 15 o menys llavors en joc i el nombre de nodes creix de forma exponencial per cada llavor que s'afegeix al tauler.

6.3.2 Solució d'estats de la variant abapa

Els problemes descrits han fet a la pràctica impossible la solució d'abapa, tot i que s'han pogut resoldre tots els nodes de 7 o menys llavors combinant l'algorisme d'anàlisis retrògrad explicat amb una cerca alfa-beta molt optimitzada per a resoldre els cicles.

L'algorisme que s'ha dissenyat per a la construcció del llibre de finals es basa en el mètode d'anàlisi retrògrad que s'ha explicat a la secció anterior. Fent però una generació i avaluació progressiva de les posicions. Amb el procediment descrit a continuació.

1. Es generen totes les posicions d'aulé amb fins a n llavors en joc.
2. S'inicialitza el valor $k = 0$.
3. Es solucionen tots els nodes possibles aplicant els passos 2 a 5, de l'algorisme explicat a la secció anterior, només a aquelles posicions amb exactament k llavors en joc.
4. Es procedeix a la detecció de tots els nodes amb exactament k llavors que formen part d'un cicle i no s'han pogut solucionar. Cada node s'etiqueta amb un nombre que identifica el cicle al qual pertany.
5. Per cadascun dels nodes etiquetats es realitza una cerca alfa-beta fins que s'ha pogut obtenir l'avaluació exacta dels nodes.
6. Es marquen com a solucionats tots els nodes dels cicles avaluats i es procedeix a propagar-ne l'avaluació als seus pares.
7. Si el valor de k és menor a n, s'incrementa k i el procediment continua al pas 3.

La detecció de nodes del pas 4 s'ha aconseguit amb una variant molt optimitzada de l'algorisme de detecció de components fortament connectats de Pearce (2005). L'algorisme de detecció implementat és molt ràpid i no requereix memòria addicional. Es fonamenta en l'observació que no es pot produir un cicle en l'aualé si s'ha realitzat una captura, de manera que només cal considerar aquells nodes amb un mateix nombre de llavors en joc.

Aquesta observació és en la qual es basa tot l'algorisme. Representa que en realitat no cal conèixer tot el camí seguit fins a produir-se una repetició per a poder avaluar un node; és suficient amb saber que el moviment que ha portat d'un node pare al seu fill era una captura. Això significa que es pot obtenir una avaluació única per a un node d'un cicle si es considera que aquell node és l'arrel de l'arbre de cerca (vegeu el capítol 2.1.1).

L'inconvenient del mètode implementat és que, per als nodes que es troben en un cicle, només es pot confiar en les avaluacions obtingudes del llibre després de produir-se una captura. Això en dificulta la utilització pràctica, perquè un algorisme de cerca que utilitzi la base de dades podrà conèixer quin és el resultat exacte d'un estat però no sabrà quin és el camí òptim que cal seguir per a poder arribar a aquest resultat.

A més, el mètode implementat és suficientment eficient només quan el nombre de nodes d'un cicle és petit. Malauradament, amb vuit llavors en joc ja es pot detectar un cicle format per més de 75 mil nodes i existeix un cicle de més de 7 milions de nodes que conté únicament posicions en les quals el nombre de llavors en joc és quinze.

CAPÍTOL 7:

Protocol de comunicació

La cerca en un espai d'estats esdevé sovint una tasca intensiva que requereix grans quantitats de còmput. Per aquest motiu i en resposta a la necessitat d'aprofitar al màxim les capacitats dels processadors disponibles és freqüent que un motor de cerca i la seva interfície d'usuari s'executin com a processos diferents en un sistema operatiu multitasca. Aquesta separació de funcionalitats exigeix tanmateix l'existència d'algun sistema de comunicació que faci possible la interacció entre ambdós processos.

7.1 Universal Chess Interface

Aquesta interacció entre una interfície d'usuari i un motor de cerca és la que facilita el protocol Universal Chess Interface. Un protocol ideat per a motors d'escacs però que també es pot utilitzar per a qualsevol altre joc amb només algunes adaptacions mínimes en la representació dels estats del joc i de les seves transicions.

L'UCI és un protocol de comunicació entre processos fonamentat en l'intercanvi asíncron de missatges a través d'un pipeline. La interfície d'usuari comunica a un motor de cerca les ordres que ha d'executar escrivint-les a la sortida estàndard i rep posteriorment les respostes del motor a través de l'entrada estàndard. La diferència fonamental amb altres protocols existents és que la interfície controla completament el motor de cerca —indicant-li els estats a establir i les accions de cerca a efectuar-hi— de manera que es deslliura al motor d'altres tasques secundàries com poden ser la gestió del temps o del llibre d'obertures.

A la figura 6.2 es pot veure com s'estableix la comunicació entre una interfície d'usuari i el motor d'aualé implementat. Una sessió es pot dividir en dues fases essencials, en la primera la interfície d'usuari informa el motor de cerca que la comunicació es farà mitjançant el protocol UCI. Seguidament, el motor respon immediatament identificant-se i comunicant a la interfície d'usuari les opcions de configuració disponibles. En la segona fase la interfície sol·licita al motor que estableixi diferents estats del joc i posteriorment que hi efectuï cerques.

```
> uci
< id name Aalina Abapa 1.0
< id author Joan Sala Soler
< option name Hash type spin default 32 min 1 max 56
< option name OwnBook type check default true
< option name Ponder type check default true
< option name DrawSearch type check default false
< uciok
> isready
< readyok
> position startpos
> go movetime 2000
< info string A book move was chosen
< bestmove F ponder b
> position startpos moves Fb
> go ponder
> stop
< info depth 21 score cp 39 pv CeBcDcBfDaAcCbBcCf
< bestmove C ponder e
> quit
```

Figura 7.1: Exemple de comunicació mitjançant el protocol UCI

A continuació es descriuen breument els missatges més rellevants, que són les ordres `position`, `go` i `stop`, així com les respostes oferides pel motor amb els missatges `info` i `bestmove`. L'especificació completa del protocol es pot obtenir del lloc web dels seus creadors[1].

- L'ordre **`position`** indica al motor l'estat del joc sobre el qual haurà de realitzar les cerques posteriors. Pren com a paràmetres la representació alfanumèrica d'una posició i, opcionalment, dels moviments que s'hi han efectuat. Cal notar aquí que per a l'aualé abapa un estat es pot identificar de forma inequívoca amb la darrera posició en la qual s'ha fet una captura i la llista de moviments que s'hi han realitzat posteriorment.
- Al seu torn, l'ordre **`go`** demana al motor que iniciï la cerca. Aquesta ordre pot incloure com a paràmetres un límit de temps per a la cerca o també una profunditat màxima a recórrer en l'arbre de cerca. Si no s'especifica un límit de temps finit, el motor continuarà cercant el millor moviment fins que rebi l'ordre **`stop`** de la interfície d'usuari.
- Un cop el motor finalitzi una cerca, aquest respondrà sempre amb un missatge **`info`** seguit d'un missatge **`bestmove`**. El primer ofereix informació sobre el resultat de la cerca i normalment inclou la profunditat assolida, l'avaluació heurística de la millor branca en forma de valor numèric i el camí òptim del graf (la variant principal). El segon missatge conté informació sobre els millors moviments trobats.

És interessant observar com el protocol permet fer una gestió integra del motor de cerca i modelar tant l'estil com el nivell de joc de l'aplicació. Com a exemple, el nivell de joc es pot reduir limitant la profunditat màxima de cerca, el temps de computació o la memòria disponible per a la taula de transposicions; o bé augmentar per mitjà de la ponderació de moviments.

1 http://www.shredderchess.com/chess-info/features/uci-universal-chess-interface.html

7.2 Disseny i implementació

Com s'ha vist, el protocol UCI basa el seu funcionament en l'avaluació dels estats individuals que es comuniquen al motor de cerca. No es contempla en el protocol, per tant, la gestió de la lògica del joc; una tasca que es delega a la interfície d'usuari. A la pràctica això significa que es pot usar el protocol també per a jocs diferents als escacs, simplement adaptant representació alfanumèrica de posicions i moviments al joc concret que es vulgui implementar.

L'aplicació elaborada per aquest projecte aprofita aquesta particularitat del protocol per oferir una implementació genèrica del mateix. Això s'ha fet per mitjà de la creació de dos intèrprets del llenguatge definit pel protocol, l'un destinat a la implementació d'interfícies d'usuari i l'altre a la connexió d'un motor de cerca amb les aplicacions client. Se separa així la representació d'estats de les funcions de comunicació requerides pel protocol.

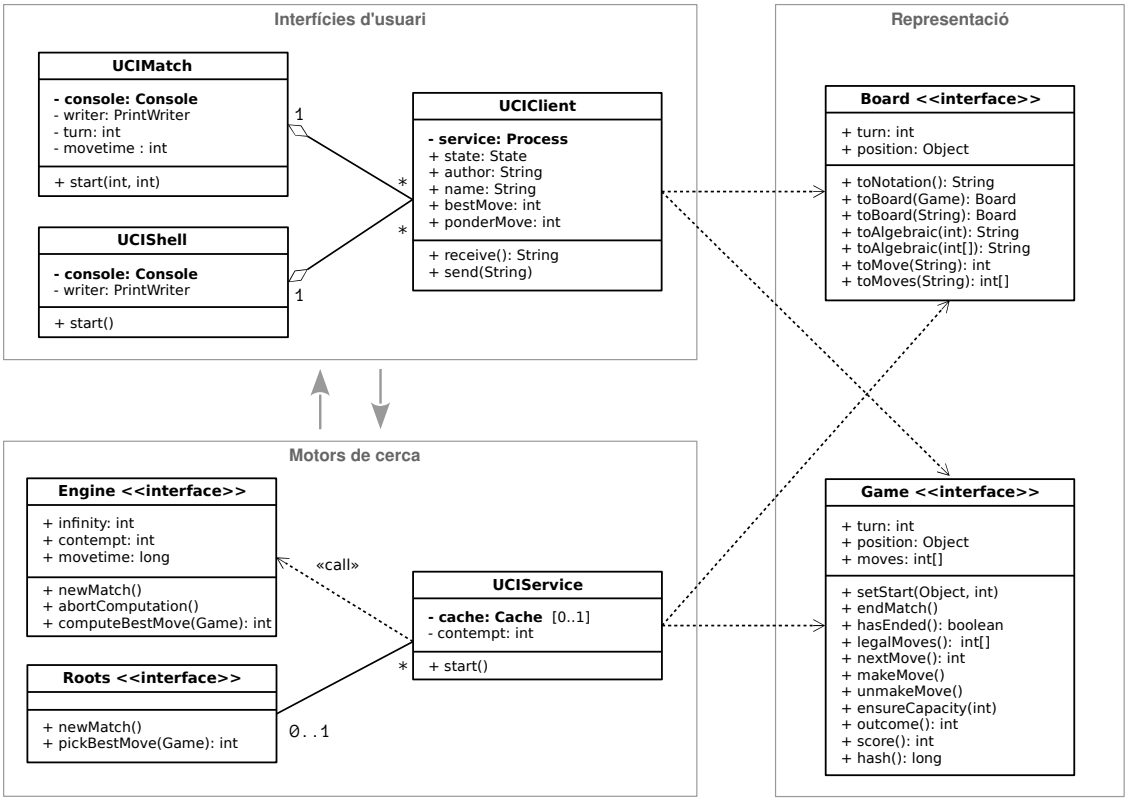


Figura 7.2: Diagrama de classes del protocol de comunicació

La figura 6.2 representa aquesta abstracció, a on instàncies de les classes `UCIClient` i `UCIService` són les que es comunicaran a través dels canals d'entrada i sortida estàndards. A la figura s'hi poden observar també dues interfícies d'usuari simples per a la línia d'ordres, `UCIMatch` i `UCIShell`, que s'han implementat com a eines de depuració del protocol, tot i que també serveixen com a demostració de la utilització pràctica dels intèrprets.

La representació d'estats, que és específica del joc elaborat, es troba definida per la interfície `Board`. Aquesta interfície, que simbolitza l'estat d'un joc, específica els mètodes necessaris per a l'obtenció de representacions alfanumèriques de posicions i moviments, i també per a la conversió d'aquestes cadenes de caràcters a representacions diferents. Com ja s'ha comentat anteriorment, l'estat de qualsevol joc amb adversari es pot descriure de forma inequívoca només amb la informació d'una posició i dels moviments que s'hi han efectuat.

És important recalcar el paper passiu de `UCIService` en la comunicació. La classe simplement implementa un procés servidor que avalua les expressions rebudes des d'un client, transmet aquestes ordres al motor de cerca i, finalment, escriu a la sortida estàndard les respostes obtingudes. Excepcionalment, els missatges de resposta es poden extreure també d'una base de dades d'obertures, tot i que això normalment s'encomana a les aplicacions client.

Capítol 8: Conclusions

Al llarg d'aquesta memòria s'ha descrit el disseny d'una interfície de programació d'aplicacions (API), enfocada a l'elaboració de motors de cerca per a jocs amb adversari, i s'ha documentat la implementació d'alguns algorismes genèrics usats amb la mateixa finalitat. El gruix més important de la memòria s'ha destinat tanmateix a les possibilitats d'ús de les eines dissenyades per a l'elaboració i optimització d'un motor de cerca per a l'aualé.

L'espai d'estats de l'aualé està format per diversos bilions de posicions diferents i els camins que hi condueixen. Escollir quin és el millor moviment que un jugador pot realitzar en un d'aquests estats requereix efectuar una cerca en l'espai d'estats i considerar les implicacions de les decisions preses. Una cerca ingènua examinaria totes les possibilitats, però això no és suficient quan l'espai d'estats és de mides astronòmiques. Es fa necessari doncs reduir el nombre de possibilitats considerades al mínim imprescindible per a prendre una decisió.

En aquest treball s'han examinat diferents tècniques enfocades a l'optimització. Aquestes s'han dividit en mètodes de representació, destinats sobretot a maximitzar el nombre d'estats que un motor de cerca és capaç d'examinar, i en procediments enfocats a minimitzar el nombre de possibilitats que cal considerar. En aquest darrer grup hi trobem mètodes de reducció de l'arbre de cerca i tècniques d'aprenentatge computacional.

S'ha vist que amb les metodologies adequades és possible disminuir el nombre d'estats de l'aualé que és necessari avaluar, des de desenes de milers de milions fins a unes poques desenes de milions en els casos més simples. S'ha fet patent també que això es pot fer en un temps adequat i sense haver de fer sacrificis en el rendiment o l'exactitud dels algorismes.

En última instància s'ha descrit com l'avaluació heurística possibilita obtenir resultats més ràpids amb la presa de decisions imperfectes. L'aualé presenta a més propietats interessants que fan factible predir amb força exactitud el resultat que s'obtindria amb un joc perfecte per part dels dos adversaris. Aquesta observació ha permès elaborar un model de regressió lineal viable per a l'optimització automatitzada d'una funció heurística per al joc.

Durant l'elaboració del motor de cerca també s'ha investigat i analitzat la possibilitat de resoldre completament l'aualé abapa. S'ha vist però que les característiques del mateix fan difícil solucionar fins i tot alguns dels estats més simples. La construcció d'una solució completa requeriria nous algorismes de resolució de cicles en un temps lineal. És complicat preveure però l'aparició d'un algorisme semblant doncs implicaria poder detectar els estats terminals sense haver de recórrer els camins de l'arbre de cerca.

Amb tot, es pot concloure que l'optimització de la cerca és un procés interminable. Cada nou mètode dóna peu a l'aparició de noves tècniques i a la millora de les existents. Al tinter s'ha quedat, entre moltes altres optimitzacions possibles, la implementació de procediments d'extensió de la cerca —com la cerca de quiescència—, la reducció de l'arbre de cerca basada en una inspecció més acurada del camí òptim —per exemple, la cerca de variant principal o el mètode MTD(f)— i la memorització permanent d'estats avaluats durant la cerca.

Glossari

abapa ak *m Oware Abapa*. En llengua àkan, literalment *la pedra bona*. Variant tradicional d'aualé considerada la més difícil i l'escollida per als campionats internacionals.

algorisme de cerca *m* Grup d'instruccions computacionals enfocades a la localització d'un element específic dintre d'un conjunt, ja sigui en una base de dades o en l'espai de cerca definit per un procediment matemàtic.

anàlisi retrògrad *m* Referencia als mètodes de computació utilitzats per a resoldre una posició d'un joc mitjançant la reconstrucció del graf que la posició defineix a partir dels seus nodes terminals i l'avaluació dels mateixos.

aualé *m* Variant dels jocs de mancala que es juga amb quaranta-vuit llavors en un tauler de dotze forats distribuïts en dues files de sis forats cadascuna. També conegut com a *awari*, *oware* o *wari* entre altres noms.

awari en *m* Nom amb el qual es denomina habitualment la variant d'aualé estudiada en el camp de la intel·ligència artificial.

cicle *m* Camí d'un graf que connecta un node amb si mateix passant per un o més vèrtexs.

espai d'estats *m* En referència a un joc, el conjunt de configuracions que el joc pot presentar.

final de joc *m* Darrera fase d'una partida i conjunt de moviments amb els quals acaba.

graf *m* Representació abstracta d'un conjunt d'elements on alguns d'ells es troben connectats amb enllaços. Als elements del graf se'ls anomena vèrtexs o nodes, i als enllaços arestes.

llibre de finals *m* Base de dades d'aquelles posicions d'un joc de tauler que es poden produir habitualment en el transcurs del final de joc. Vegeu *final de joc*.

llibre d'obertures *m* Base de dades de posicions d'un joc de tauler que representa una porció del graf del joc definida per la posició amb la qual el joc comença. Vegeu *obertura*.

mancala *m* Família de jocs de tauler que es juguen sobre una superfície formada per una o diverses rengleres de forats en els quals cal distribuir les fitxes del joc, normalment llavors. El terme mancala no descriu cap variant de joc concreta.

mig joc *m* Segona fase d'una partida d'un joc. Habitualment fa referència al conjunt de moviments i posicions que es produeixen entre l'obertura i el final de joc.

moviment *m* Transició d'estat en un autòmat finit. En un joc de tauler, desplaçament de les fitxes del joc durant el torn d'un jugador de forma que la posició del tauler varïi.

obertura *f* Primera fase d'un joc i conjunt de moviments amb els quals comença una partida.

oware en *m* Vegeu *aualé*.

posició *f* Distribució de les fitxes en el tauler d'un joc en un moment determinat.

suma nul·la *f* En la teoria de jocs aquella situació en la qual la suma dels beneficis de tots els jugadors equival a la suma de les seves pèrdues, de forma que el resultat total és equilibrat.

transposició *f* En un joc de tauler, camí en el graf del joc que porta a una mateixa posició que un altre camí diferent del graf.

Bibliografia

Abramson, B. (1987). *The expected-outcome model of two-player games*. Informe tècnic. Universitat de Columbia. [Data de consulta: 20 de març de 2014]. <http://academiccommons.columbia.edu/catalog/ac:142326>

Allis, L.V.; Meulen, M.; Herik, H.J. (1994). "Proof-number search". *Artificial Intelligence*. Articles (vol. 66, núm. 1). Elsevier Science Publishers Ltd.

Allis, L.V. (1994). *Searching for solutions in games and artificial intelligence*. Tesi doctoral presentada a la Universitat de Maastricht. Wageningen: Ponsen & Looijen. [Data de consulta: 25 de març de 2014] <http://arno.unimaas.nl/show.cgi?did=10501> ISBN 90-9007488-0

Bal, H.; Allis, L.V. (1995). "Parallel Retrograde Analysis on a Distributed System". A: *Proceedings of the IEEE/ACM Conference on Supercomputing* (pàg. 73). Conferència. ACM.

Breuker, D.M.; Uiterwijk, J.; Herik, H.J. (1994). "Replacement schemes for transposition tables". *ICCA Journal*. Articles (vol. 17, núm. 4). International Computer Chess Association.

Daoud, M.; Kharma, N.; Haidar, A.; Popoola, J. (2004). "Ayo, the Awari player, or how better representation trumps deeper search". A: *Proceedings of the 2004 Congress on Evolutionary Computation* (pàg. 1001-1006). Conferència. IEEE.

Davis, J.E.; Kendall, G. (2002). "An investigation, using co-evolution, to evolve an Awari player". A: *Proceedings of the 2002 Congress on Evolutionary Computation* (pàg. 1408-1413). Conferència. IEEE.

Donkers, J. (2002). "Comments on the Awari solution". *ICGA Journal*. Articles (vol. 25, núm. 3). International Computer Chess Association.

Donkers, J; Uiterwijk, J. (2002) "Programming Bao". A: *Seventh Computer Olympiad: Computer-Games Workshop Proceedings* (pàg. 2-3). Informe tècnic. Universitat de Maastricht. [Data de consulta: 30 d'abril de 2014]. <http://www.fdg.unimaas.nl/educ/donkers/pdf/olympiad02.pdf>

Donninger, C.; Lorenz, U. (2006). "Innovative opening-book handling". A: *Advances in Computer Games* (pàg. 1-10). Springer Berlin Heidelberg. ISBN: 978-3-540-48887-3

Edelkamp, S; Sulewski, D.; Yücel, C. (2010). *Perfect Hashing for State Space Exploration on the GPU*. A: *ICAPS*. Informe tècnic. Universitat de Bremen. [Data de consulta: 16 de març de 2014]. <http://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/download/1439/1529> ISSN 1613-3773

Fürnkranz, J. (2007) "Recent advances in machine learning and game playing". *ICGA Journal*. Articles (vol. 26, núm. 2). International Computer Chess Association. ISSN 1389-6911

Irving, G.; Donkers, J.; Uiterwijk, J. (2000). "Solving Kalah". *ICGA Journal*. Articles (vol. 23, núm. 3). International Computer Chess Association. [Data de consulta: 3 de març de 2014]. <http://www.fdg.unimaas.nl/educ/donkers/pdf/kalah.pdf> ISSN 1389-6911

Kishimoto, A; Müller, M. (2004). "A general solution to the graph history interaction problem". A: *Proceedings of the 19th national conference on Artifical intelligence* (California: 2004). AAAI Press.

Lincke, T.R. (2002). *Exploring the computational limits of large exhaustive search problems*. Tesi doctoral presentada al grau de Doctor en Ciències Tècniques de l'Institut Federal Suís de Tecnologia. Zürich. [Data de consulta: 3 de març de 2014]. <http://dx.doi.org/10.3929/ethz-a-004442444>

Lister, L; Schaeffer J. (1994). "An analysis of the conspiracy numbers algorithm". *Computers & Mathematics with Applications*. Articles (vol. 17, núm 1). Elsevier Science Publishers Ltd.

Lorenz, U.; Rottmann, V.; Feldmann, R.; Mysliwietz, P. (1995) "Controlled conspiracy number search". *ICCA Journal*. Articles (vol. 18, núm. 3). International Computer Chess Association. ISSN 1389-6911

McAllester, D.A. (1988). "Conspiracy numbers for min-max search". *Artificial Intelligence*. Articles (vol. 35, núm. 3). Elsevier Science Publishers Ltd.

Meulen, M; Allis L.V.; Herik, H.J. (1990). *Lithidion: an Awari-playing program*. Informe tècnic. Universitat de Limburg.

Pearce, D.J. (2005). *An improved algorithm for finding the strongly connected components of a directed graph*. Informe tècnic. Universitat de Victòria.

Romein, J.W.; Bal, H.E. (2002). "Awari is Solved". *ICGA Journal*. Articles (vol. 25, núm. 3). International Computer Chess Association. [Data de consulta: 3 de març de 2014]. <https://ilk.uvt.nl/icga/journal/contents/awari.pdf> ISSN 1389-6911

Romein, J.W.; Bal, H.E. (2003). "Solving the Game of Awari Using Parallel Retrograde Analysis". *IEEE Computer*. Articles (vol. 36, núm. 10). IEEE. ISSN 0018-9162

Tarjan, R. (1972). "Depth-First Search and Linear Graph Algorithms". *SIAM Journal on Computing*. Articles (vol. 1, núm. 2). SIAM.

Turing, A.M. (1953) "Digital Computers Applied to Games". *Faster than Thought: A Symposium on Digital Computing Machines* (pàg. 286-310). Londres: Sir Isaac Pitman & Sons. [Data de consulta: 20 d'abril de 2014]. <http://www.turingarchive.org/browse.php/B/7>

Winands, M.H.; Uiterwijk, J.W.; Herik, H.J. (2001). "Combining proof-number search with alpha-beta search". A: *BNAIC Dutch-Belgian Artificial Intelligence Conference*. Conferència. Universitat d'Amsterdam. [Data de consulta: 30 d'abril de 2014]. <http://staff.science.uva.nl/~maarten/BNAIC-2001.pdf>

Annex A: Reglament del joc

Existeixen diverses variants de l'aualé, cadascuna amb un conjunt de normes pròpies. En aquest apèndix es descriu només la variant coneguda com a *abapa*, que és la variant implementada en el motor d'intel·ligència artificial sobre el qual aquest treball versa.

1.1 Equipament i disposició inicial

Per a jugar a l'aualé és necessari un tauler i quaranta-vuit fitxes, anomenades llavors. Habitualment el tauler estarà compost per dues fileres de sis forats cadascuna, situades l'una al davant de l'altra. Dos forats més grans a cada banda del tauler serveixen per a desar-hi les llavors que els jugadors van capturant durant la partida. Es diu que la filera inferior pertany al jugador que mou primer, anomenat *sud*, i la filera superior al segon jugador o *nord*.

En la posició inicial, cadascun dels forats, excepte els dos més grans, conté exactament quatre llavors. En aquesta posició el jugador sud farà el seu primer moviment, seguit d'un moviment del jugador nord i així alternativament fins que la partida s'acabi.

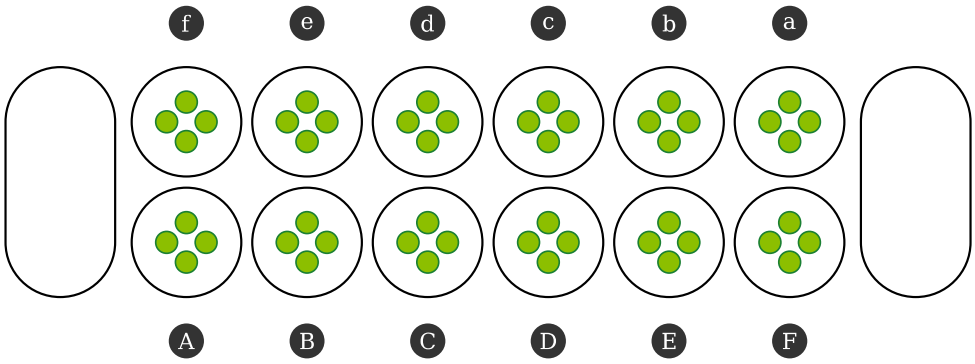


Figura A.1. Tauler del joc en la posició inicial

1.2 Objectiu del joc

La finalitat del joc consisteix a recollir el màxim de llavors possible. Per a fer-ho els jugadors fan moviments en torns alternatius fins que algun d'ells ha recollit més de 24 llavors. Guanya la partida aquell jugador que en finalitzar el joc ha recollit més llavors que l'adversari. També es pot donar el cas que en acabar ambdós jugadors hagin recollit el mateix nombre de llavors. En aquest supòsit cap dels jugadors guanya la partida i es diu que ha acabat en empat.

Cada moviment del joc es fa en tres fases: la recol·lecció, la sembra i la captura. En la sembra el jugador distribueix les llavors recollides al llarg del tauler i en la captura el jugador arreplega quan li és possible les llavors que es troben als forats de l'adversari.

1.3 Recol·lecció

En la primera fase d'un moviment el jugador a qui toca moure escull un dels forats que li pertanyen i n'agafa totes les llavors, deixant aquell forat buit. Posteriorment, aquestes llavors es repartiran pel tauler en la fase de sembra.

El jugador pot recollir les llavors de qualsevol dels forats que li pertanyen si aquest conté una o més llavors, només amb l'excepció que després de fer un moviment el seu adversari ha de poder jugar. No seran legals aquells moviments que deixin els forats del rival sense llavors.

1.4 Sembra

Durant la sembra, el jugador reparteix pel tauler, en sentit antihorari, les llavors recollides en la primera fase; deixant una llavor a cada forat propi i del jugador contrari fins que les ha distribuït totes. El jugador no sembrarà mai llavors en els forats de captura.

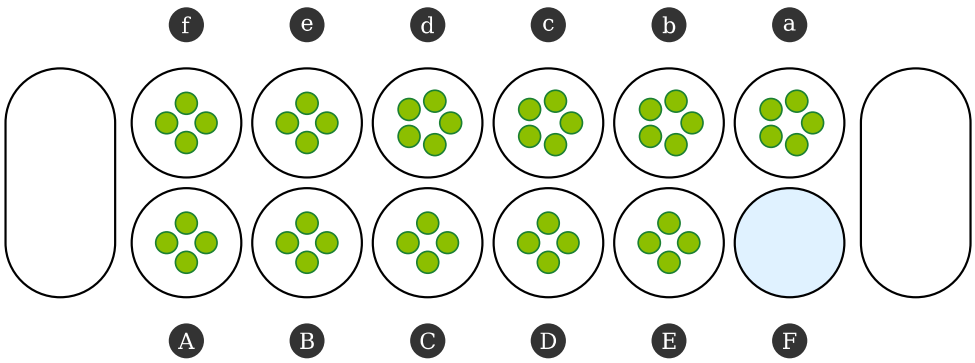


Figura A.2. Posició després de la sembra des del forat F per part del jugador sud

En finalitzar la sembra el forat del qual el jugador ha recollit les llavors estarà buit. Es pot donar el cas que el jugador sembri dotze o més llavors. Quan això passi el jugador les sembrarà donant una o més voltes al tauler, deixant a cada volta una llavor a cadascun dels forats però exceptuant sempre el forat del qual s'han recollit.

1.5 Captura

Si en finalitzar la sembra l'última llavor s'ha deixat en un dels forats que pertanyen a l'adversari i després de deixar-hi la llavor aquest forat conté exactament dues o tres llavors, el jugador les capturarà. Agafant-ne totes les llavors i dipositant-les al seu forat de captures.

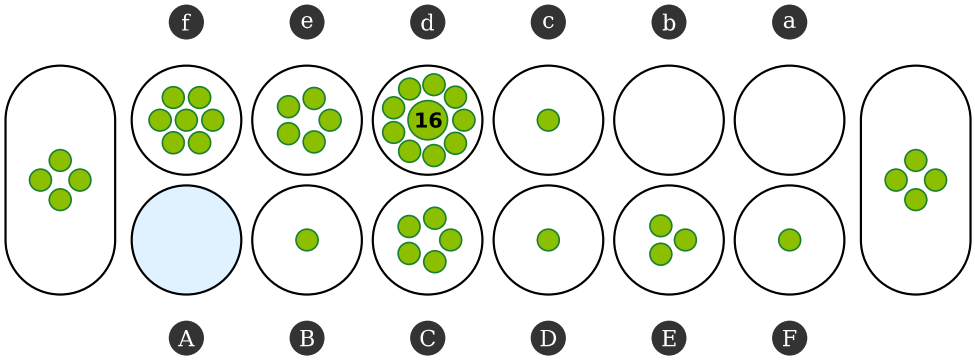


Figura A.3. Posició en la qual cada jugador ha capturat quatre llavors

Si després de capturar les llavors d'un forat resulta que el forat immediatament a la seva dreta també conté dues o tres llavors, el jugador també capturarà les llavors d'aquell forat. La captura continuarà successivament fins que el jugador ja no pugui capturar més llavors. Els jugadors només poden capturar llavors en forats del seu adversari, mai dels seus forats propis.

Un jugador no pot capturar mai totes les llavors de l'adversari. Si el jugador fa un moviment que després de les captures deixaria tots els forats de l'adversari buits, aquell jugador farà la sembra normalment però no capturarà cap llavor.

1.6 Finalització

Normalment el joc s'acaba quan un dels jugadors ha capturat més de 24 llavors o quan tots dos n'han capturat 24. També pot succeir que en el torn d'un jugador aquest no pugui fer cap moviment legal, en aquest supòsit, cada jugador capturarà les llavors que es trobin en els forats del seu costat del tauler i el joc s'acaba.

Una situació especial és quan el joc entra en un cicle, de forma que les mateixes posicions i moviments s'anirien repetint indefinidament. Si això succeeix, els jugadors poden acordar finalitzar la partida i cada jugador capturarà les llavors que es trobin al seu costat del tauler.

1.7 Notació utilitzada

No existeix cap notació estàndard per a l'aualé ni tampoc un corpus de partides anotades més o menys extens. En aquest treball les partides es descriuen usant una notació algebraica simple: els moviments del jugador sud s'anoten amb les lletres majúscules A-B-C-D-E-F i els moviments del jugador nord amb les lletres minúscules a-b-c-d-e-f.

Cadascuna de les lletres representa el forat des del qual el jugador ha recollit les llavors, començant pel de l'esquerra del jugador. Sovint la notació d'un moviment anirà seguida d'un nombre enter que representa el nombre de llavors que el jugador ha capturat en el moviment.

1.7.1 Exemple de partida anotada

1. F f **2.** E e **3.** A a **4.** E e **5.** C c+2 **6.** B+2 b+2 **7.** F+2 c **8.** D a **9.** B c **10.** D b **11.** A c **12.** B e+2 **13.** B f **14.** F+4 d+3 **15.** A a **16.** B f **17.** F+3 e+2 **18.** E+2 a **19.** D b **20.** C e+2 **21.** B c **22.** C b **23.** F+2 b **24.** B f+2 **25.** B e **26.** E b **27.** A d+2 **28.** C a **29.** E d **30.** A f+2 **31.** A e **32.** A c **33.** B d **34.** F+2 a **35.** D a **36.** E c **37.** C b **38.** D f **39.** A e+3 **40.** A f **41.** A d **42.** A e **43.** E c **44.** B d **45.** D e **46.** C f **47.** E a **48.** D b **49.** C c **50.** F+2 b **51.** E a **52.** D c **53.** B e **54.** A d **55.** C b **56.** E e **57.** B c **58.** D d **59.** C e **60.** D f **61.** E a **62.** C b **63.** F+2 b **64.** D c **65.** E a **66.** B d **67.** C e **68.** A b **69.** D c **70.** B d **71.** C e **72.** D f **73.** E a **74.** F+2 a **75.** B b **76.** C c **77.** D d **78.** A e **79.** B f **26-22**

Annex B: Planificació temporal

El següent diagrama de Gantt mostra la planificació temporal inicial del projecte.