Universitat Autònoma de Barcelona

escola d'enginyeria

5663: GORGON: UNA IA PER A TSUMEGOS EN EL JOC DEL GO

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Josep Floriach Ventosinos
i dirigit per
Ramon Grau Sala
Bellaterra, 1 de setembre de 2014

Agraïments

En primer lloc voldria agrair al meu tutor, Ramon Grau, per depositar la confiança en el meu projecte. Sense ell no hagués estat possible.

D'igual manera, tampoc hagués estat possible sense l'ajut que hem va oferir Roger Llopart Pla a l' inici del curs. Si ja sabia que era una gran persona, en aquell moment ho va acabar de confirmar.

Finalment voldria agrair a totes les persones que durant un moment o altre, han demostrar interés en el meu projecte. Gràcies a elles he pogut sortir dels mal moments en els que no sabia com tirar endavant. Gràcies a Albert Oriol, a Aythami Santana, Gabriel Diaz, Abi, Elena García i totes les demés persones que segur que hem deixo.

Índex

Índex de figures

Capítol 1

Introduccio

1.1 Qué és el Go?

El Go és un joc d' estratègia per a dos jugadors, que es va originar a Xina fa més de 2500 anys. La majoria de joc competitiu es du a terme a Xina, Korea i Japò, però en les últimes dècades ha guanyat certa popularitat a occident. Avui en dia hi ha més de 40 milions de jugadors per tot el món.

Normalment es juga en un taulell de 19x19 interseccions i es caracteritza per tindre només dues senzilles regles (veure annex 1: regles del Go). No obstant, conté un component tàctic i estratègic extremadament alt. Alguns estudis matemàtics, han demostrat que el nombre de partides diferents que es poden jugar, és d' aproximadament 10^{800}, superant (i per molt) el nombre d' àtoms en l' univers observable actualment, que es calcula que està al voltant de de 10^{80}.

Tota aquesta complexitat ha despertat l' interès del Go en algunes branques de ciències de la computació. Una d' elles, i la més important, es l' intel·ligència artificial. I no és per menys, perquè a diferència de l' escacs [1], a dia d' avui encara no s' ha pogut crear cap programa que sigui capaç de guanyar a un jugador professional, en una partida justa. Tan es així, que s' organitzen competicions anuals fent lluitar als programes entre sí, per veure quin és el més fort.

Recentment, alguns programes, utilitzant supercomputadors, han aconseguit guanyar a alguns professionals de categoria 9é dan [2], en circumstàncies avantatjoses (posant handicaps al jugador), o en taulells de menor tamany. L' any 2008, el programa MonGo va aconseguir guanyar a un 5é dan en un taulell de 9x9.

[1] Al 1997, el Deep Blue es va convertir en la primera IA en guanyar al campió mundial Kasparov en el joc de l' escacs

[2] La categoria "dan"és otorgada a jugadors professionals, i només aquells amb unes capacitats excepcionals són escollits per el títol de 9é

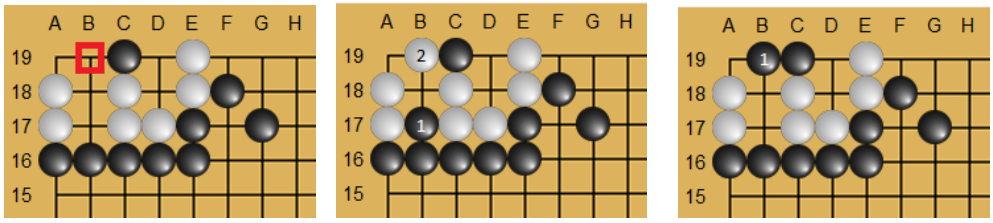
A la data de creació d' aquest document, la fita més important aconseguida, va ser al juny de 2013, quan el programa Zen va guanyar a un 9é dan, en un taulell de 19x19, i amb un handicap de només 3 pedres.

1.2 Tsumegos

Tan en l'aprenentatge del Go, com en l' entrenament per millorar les capacitats d' un jugador, entren en joc els Tsumegos. Similars als típics problemes d' escacs que solen venir en qualsevol diari, els Tsumegos són problemes adaptats al Go. Donada una distribució de peces inicials, el jugador ha d' intentar, o sobreviure en un espai potencialment capturat per l' enemic, o capturar a les peces enemigues causant la mort del grup sencer. Una característica dels Tsumegos és que sempre juguen les negres en primer lloc.

Aquests objectius normalment són assolits mitjançant una seqüència crítica de moviments. És a dir, si el punt crític no és ocupat en el moment adequat, el problema acabarà en derrota. I notis la subtilesa de "acabarà", perquè un jugador pot haver perdut ja un territori, i no obstant seguir jugant en ell. Però totes les pedres que col·loqui a partir del moment que l'ha perdut, són peces virtualment mortes que acabaran sent capturades per l' enemic.

Vegem un exemple:



(a) Estat inicial amb el punt crític marcat

(b) Moviment de les blanques al punt crític

(c) Moviment de les negres al punt crític

Figura 1.1: Exemple d' un Tsumego

A la figura (a) tenim la distribució inicial de peces i el punt crític marcat en vermell. La figura(b) representa una seqüència de jugades en la qual les negres no han jugat aquest punt crític, i ho han fet les blanques. La figura (c) és el cas contrari. Les negres han sabut veure el punt crític.

Les conseqüències de la jugada de la figura B són les següents:

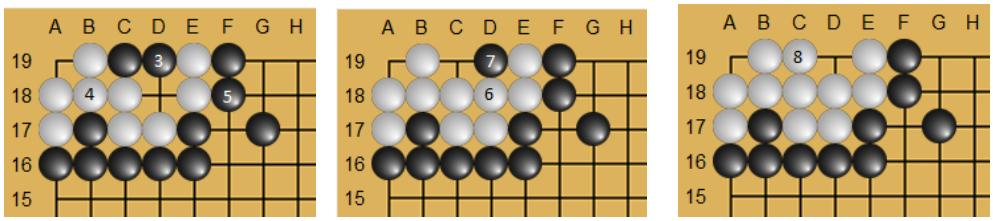


Figura 1.2: Conseqüències de la jugada a B17

Aquesta podria ser la continuació de la seqüència iniciada a la figura 1.B. Podem veure com s' ha arribat a un punt en que les negres no han pogut capturar a les blanques, i a més han sacrificat tres pedres innecessàriament. A partir d' ara, les negres no poden jugar ni a A19 ni a D19. Trencaria la regla de les llibertats. Per tant, les blanques estan permanentment vives en aquesta zona fins a final de partida. No se les pot capturar de cap manera.

Que hagués passat si les negres haguessin jugat al punt crític B19 des d' un inici tal com hem vist a la figura C?

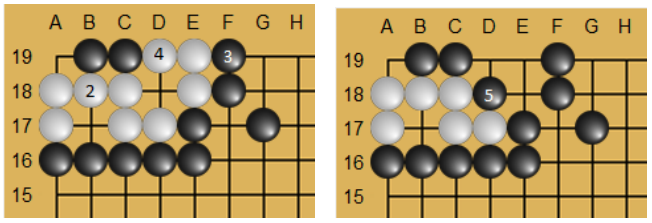


Figura 1.3: Conseqüències de la jugada a B19

En aquest cas, les negres han pogut capturar un grup de blanques impedint que, en un futur s' esdevingui la situació final anterior, en la que les blanques estaven permanent vives.

Per fer un anàlisis complert d'aquest senzill problema, podem deduir fàcilment el perquè el punt B19 és el punt crític. Si ens fixem a la distribució inicial, el punt crític és el que permet a les blanques fer una formació de dos ulls (veure annex 3: ulls en el joc del Go). Els dos ulls és un dels patrons més utilitzats en el Go degut a la seva potencialitat de viure permanentment.

En la seqüència de moviments de la figura 1.B el problema sencer ha quedat sentenciat en el moment en que la blanca ha jugat a B19, formant els dos ulls. A partir d'aquest moment totes les pedres negres, jugades en el territori format per els dos ulls, han estat sacrificis. En canvi, a la figura 1.C, les negres han tallat la possibilitat de que les blanques fessin dos ulls a la zona, permetent capturar-les. I si a més ens fixem en l' estat del taulell a l' última imatge, podem veure com el conjunt de blanques que encara està sobre el taulell, està rodejat completament per les negres, i sentenciat a mort.

En qualsevol dels dos casos es pot comprovar que les primeres jugades són determinants per la resolució del problema. Aquesta premissa és molt comú en els Tsumegos. En aquest senzill exemple és relativament fàcil veure el punt crític i l' estratègia a seguir. Però en exemples més complicats, els punts crítics poden ser més d' un, es poden jugar en diversos ordres, i inclús és necessari el sacrifici de peces pròpies per tal d' aconseguir que l' enemic no sobrevisqui a la zona.

1.3 Estat de l' art: Tsumegos i software en l'actualitat

És clar que els Tsumegos ajuden a potenciar les habilitats dels jugadors. A més, en una de les 3 grans fases d' una partida complerta de Go (veure Annex A: regles del Go), el taulell està ple de batalles locals semblants als Tsumegos. És per això que avui en dia, qualsevol aplicació virtual de Go té un mòdul de Tsumegos.

Una de les característiques d' aquestes aplicacions és que té dos tipus d' usuaris. Els que creen els Tsumegos, i els que els intenten resoldre. A partir d'ara anomenats "creadors" i "jugadors" respectivament. Els creadors defineixen tots els possibles camins de jugades del problema, incloent les que són incorrectes. Això es porta a terme mitjançant la definició d'arbres de jugades com els que es mostren a continuació.

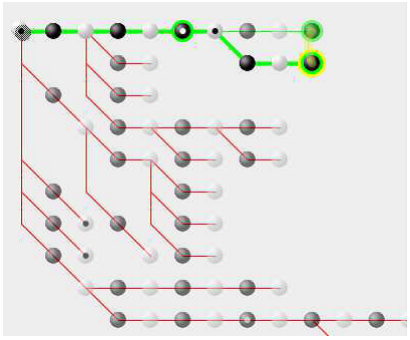


Figura 1.4: Simulacio d' IA mitjançant arbres de jugades

El jugador va fent les seves jugades (amb les negres) i va avançant per l' arbre definit per el creador, juntament amb les respostes de les blanques. Si segueix un dels camins verds, arribarà a una solució correcte. Si segueix un dels camins vermells, arribarà a un estat en el que ha perdut el problema. És un sistema fàcil d' implementar en el que no hi ha cap component d' intel·ligència artificial pròpiament dit, que dificulti el procés.

Ara bé, qué passa si el jugador no segueix cap dels camins definits per el creador? L' aplicació arribarà a un punt en que no serà capaç de respondre i s' aturarà tot el sistema de resposta per part de la màquina, informant al jugador que el sistema no coneix la jugada escollida, i per tant no sap com continuar.

Aquesta filosofia té els seus pros i contres. Els seus avantatges són que és de fàcil implementació, i la possibilitat de posar comentaris a cadascuna de les jugades, per informar al jugador del perquè no hauria de jugar en aquell punt. Però el fet de que siguin els propis jugadors els que defineixen els problemes té dues fortes implicacions.

En primer lloc, la impossibilitat de tindre en compte absolutament totes les jugades. Recordem l' explosió combinatòria del Go. En una zona de joc de 5x5, per exemple, les possibilitats són extremadament elevades ja. Això implica que el creador ha d' acotar la definició del problema a les jugades més rellevants.

I d'aquest últim fet deriva el segon problema. Normalment els creadors són persones amb una experiència en el joc bastant elevada. Això implica que moltes vegades no tenen en compte les jugades més nefastes. A més, la majoria de creadors acaben la definició del problema en la jugada crítica.

Quines repercussions té això? Posem d' exemple a un jugador novell. Per a un jugador novell, totes les jugades són possibles jugades. No té perquè saber que hi ha un punt que ni es contempla jugar de lo dolent que és. Això, sumat al fet que el problema acaba quan posa la peça crítica, i no quan realment es veuen les conseqüències directes d' haver-ho fet, dificulta molt l' aprenentatge sobre el joc.

1.4 Objectius del projecte

L' objectiu del projecte és aprofundir en el camp de l' intel·ligència artificial simbòlica i subsimbòlica, per tal d' agafar confiança amb aquests tipus de problemes, i poder treballar-hi a un nivell més professional.Amb aquest propòsit, s' implementarà una IA que sigui capaç de jugar a Tsumegos en el Go, contra una persona d' un nivell principiant-mitjà. S' estudiaran a fons diverses metodologies i algorismes per tal d' aconseguir aquest objectiu.

Aquesta intel·ligència serà capaç de jugar contra el jugador sense cap coneixement previ, fent que el jugador pugui veure les repercussions de les seves jugades fins al final, i facilitant el procés d' aprenentatge. Com que l' aplicació està destinada a jugadors novells, no serà necessari que l' IA sigui capaç de resoldre problemes extremadament complexos. Es limitarà a Tsumegos de nivell principiant-mitjà.

És evident, però, que aquest sistema també tindrà les seves limitacions. En primer lloc, els Tsumegos que pugui resoldre l' IA, tindran unes característiques determinades, que es veuran en pàgines posteriors. En segon lloc, un dels avantatges que tenia el sistema d' arbres de jugades, es perdrà. I es que per a informar al jugador de perquè esta jugant bé o malament, podria suposar un mòdul d' IA a part, igual de complexe que el del propi joc.

Finalment, i per dotar de mes funcionalitat a l' aplicació, també permetrà jugar a dos jugadors en una partida complerta de Go, seguint les normes pròpies del joc.

1.5 Viabilitat del projecte

1.5.1 Viabilitat tècnica

Aquí és on podríem tindre més problemes. L' implementació d' una IA, com ja s' ha comentat, es una tasca molt complexa i laboriosa. I més en el cas d' aquest projecte, que té la dificultad afegida de que es tracta d' un problema no resolt completament. No obstant, degut a la limitació auto imposada de fer-la per a principiants, és factible com a projecte de final de carrera.

1.5.2 Viabilitat legal

El projecte en principi no suposa cap problemàtica en el marc legal, més enllà de la disposició de llibreries i bases de dades de tercers. Però com que preten ser un projecte sense ànim de lucre i de lliure distribució, no s' haurien de tindre majors problemes. En qualsevol cas, tot el software de tercers que s' utilitzi serà completament lliure.

1.5.3 Viabilitat econòmica

El cost de desenvolupament estarà associat bàsicament a les hores dedicades. No obstant, es podrien tindre costos derivats de la compra de documents o llibres tècnics dels que no es poden disposar de forma gratuita. S' intentarà en la mida del possible que siguin molt baixos o inclús nuls.

D' altre banda també hi haurà petits costos derivats de l' impressió de documents, memòria, previ, etc.

1.6 Fases del projecte

El projecte es dividirà en les següents fases:

- Implementació d'una interfície gràfica.
- Creació de la llibreria Gorgon. Incorporarà els següents subsistemes:
  - Control i suport de jugadors: controlar el torn, pedres capturades, punts d' influència sobre el taulell..
  - Control sobre el taulell. Col·locació de pedres, enmagatzament de dades, etc etc.

- Suport per a arxius SGF.
- Sistema d' intel·ligència artificial.

El desenvolupament de la interfície gràfica està en fases relativament avançades. No obstant, degut a la seva interacció amb la llibreria Gorgon, no es podrà acabar fins que la llibreria estigui completament acabada.

La llibreria Gorgon és la part del projecte en la que més temps es dedicarà. I molt especialment al subsistema d' intel·ligència artificial. El enmgatzament de les dades del taulell també suposarà un punt crític en el projecte i possiblement també implicarà un percentatge relativament elevat del temps de projecte.

1.7 Planificació temporal

Interfície gràfica: no estarà completament acabada fins que estigui complerta la llibreria Gorgon. No obstant, a mida que es vagi desenvolupant la llibreria, s' aniran fent probes d' integració a l' interfície. Així que un cop acabada, es trigarà unes 2 setmanes més per a acabar d' implementar les característiques alienes a la llibreria. Com per exemple els menús d' opcions o preferències.

Control de jugadors: serà relativament fàcil. L' única funció d' aquest mòdul serà mantindre informació dels jugadors i algunes funcions bàsiques sobre ells. Es calcula que amb unes 2 setmanes estarà acabat.

Arxius SGF: aquest mòdul també hauria de ser relativament fàcil d' implementar. Tenint en compte que ja hi ha un estàndard per a aquests tipus d'arxius (Smart Game Format), l' implementació no portarà molts problemes. S' estima una duració d' unes 2 setmanes també.

Control del taulell: en aquest cas la cosa és una mica més complicada. S' han de tindre en compte alguns problemes de rendiment, quines estructures de dades guardar, i a més fer-ho de manera incremental per distribuir la càrrega de còmput a l' hora de calcular un moviment de l' IA. També s' implementarà un submódul de debugging per tal de poder visualitzar les

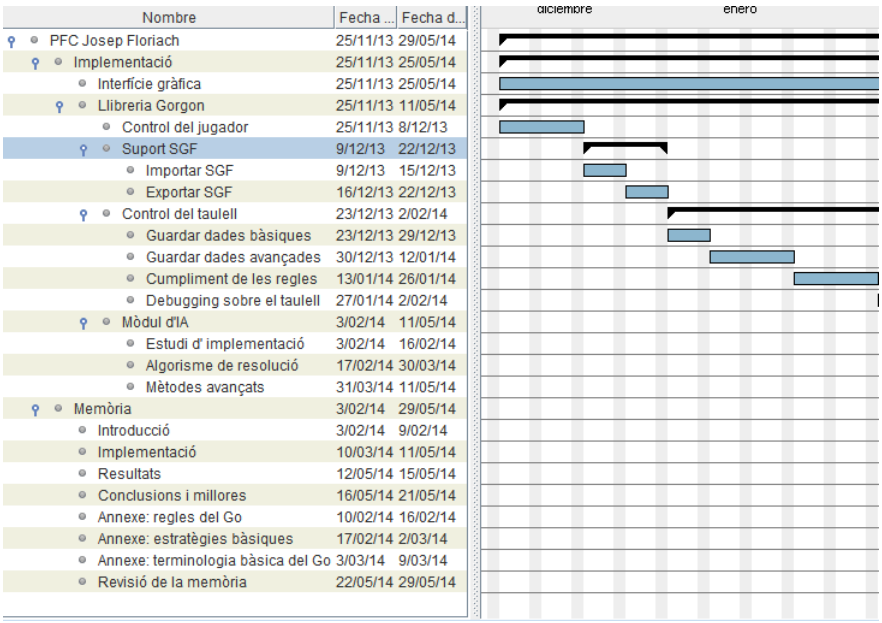
diferents representacions de l'estat del taulell. Sent relativament optimistes, unes 6 setmanes seran suficients.

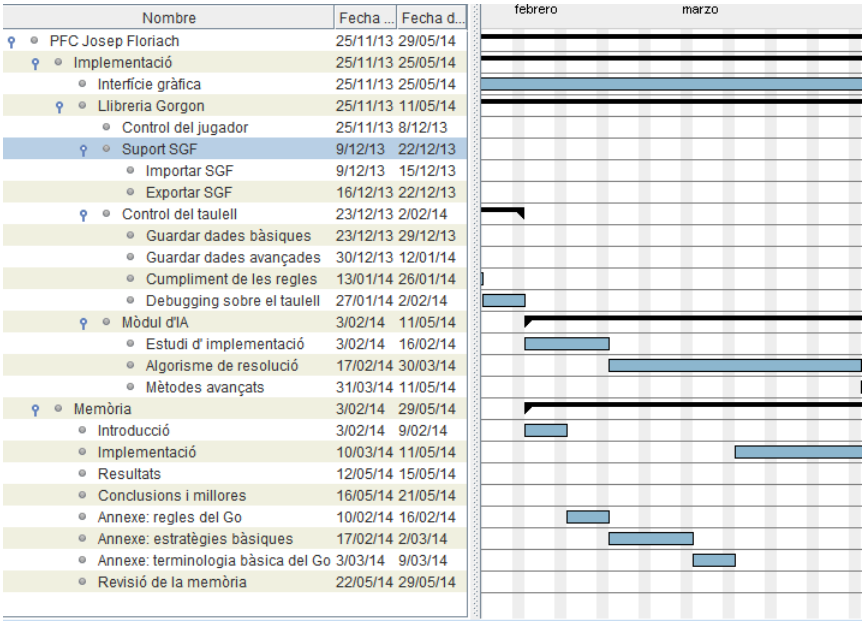
Mòdul d' IA: és el punt més crític del projecte, en quan a duració. En aquest mòdul s' utilitzarà tota l' informació guardada en el mòdul de control de jugadors i d' informació del taulell per tal de poder programar una IA que pugui jugar a Tsumegos contra el jugador. A part s' hauran de tindre en compte diferents tècniques i mètodes per tal de portar a terme aquesta tasca. S' estima una duració total d' unes 14 setmanes.

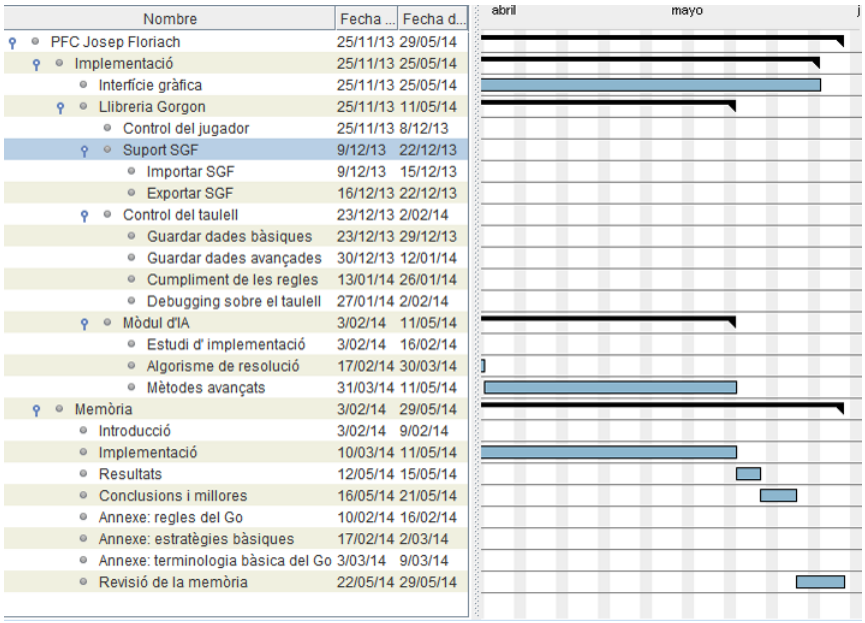
Redacció de la memòria: la memòria, al igual que l' interfície gràfica, s' anirà desenvolupant a mida que avanci el projecte. Es començarà a redactar un cop acabat el mòdul de control del taulell. Un cop acabada l' implementació del projecte, es deixaran uns dies més de marge, per acabar de posar conclusions, revisar-la, etc etc.

1.8 Diagrama de Gantt

Tenint en compte les anteriors estimacions, la càrrega de temps del projecte quedarà distribuïsda de la següent manera:







Cadascuna de les fases anteriorment mencionades té incoporporat un petit període de testeig. No obstant, al finalitzar el projecte (junt amb la memòria) es reservaran uns dies més per completar el testeig general de l' aplicació.

Capítol 2

Implementació

2.1 Introducció

L' implementació del projecte es va dividir en dues parts. El desenvolupament de l' interfície d' usuari GoGUI, i el de la llibreria Gorgon, que al mateix temps tenia diversos mòduls a atacar. Aquest document es centrarà sobretot en el desenvolupament de Gorgon, que és el que va suposar la major part del repte del projecte.

2.2 Tsumegos en Gorgon

Abans de començar amb el desenvolupament pròpiament dit, va ser necessari definir un concepte clau per tal de poder desenvolupar l' aplicació: Què és un Tsumego?

En el cas de Gorgon, i per simplificar les coses, es va decidir que un Tsumego havia de ser un problema localitzat, l' objectiu del qual havia de ser viure o impedir viure, i que no permetia al jugador defensor, fugir cap a àrees buides del taulell. Aquest últim punt es la gran diferència entre un Tsumego i una partida real. En una partida real un grup de pedres podria intentar fugir cap a l' exterior per tal de minimitzar els danys o aconseguir un benefici major en un futur amb conjunció amb la resta del taulell. En un Tsumego hi ha una solució que permet al grup sencer viure. I per tant, com que no hi ha més pedres implicades que les del propi Tsumego, la solució correcte i única per al defensor, és que el grup sencer visqui a l' interior.

2.3 Llenguatges i eines desenvolupament

La primera gran decisió que es va haver de prendre va ser quin llenguatge de programació utilitzar per implementar Gorgon i l' interfície GoGUI. Es tenien coneixements avançats i experiència laboral en Java, i coneixements

mitjans-avançats de C++. El sentit comú deia que el millor era utilitzar Java. Però es va optar per C++ per dos motius:

- En primer lloc, era una oportunitat perfecte per aprendre aquest llenguatge força demandat en el món real.
- En segon lloc, es va haver de pensar desde un punt de vista de mes tècnic. Java s' executa sobre màquina virtual. C++ compila el codi directament a llenguatge màquina. Tenint en compte el tipus de projecte que era, on destacaria la necessitat de velocitat de còmput, C++ era la opció més lògica.

Un cop, escollit el llenguatge, es va decidir fer us de l' eina de desenvolupament Qt. El motiu d' això era que Qt disposa de bastantes llibreries que facilitarien l' implementació de l' interfície gràfica GoGUI. Per comoditat a l' hora d' integrar la llibreria, es va decidir fer us de Qt també per desenvolupar Gorgon.

2.4 GoGUI

2.4.1 Introducció

GoGUI és la interfície gràfica que es va implementar. Es va desenvolupar abans de Gorgon simplement per tindre una manera visualment atractiva de fer les proves necessàries per Gorgon. Es va implementar de la següent manera:

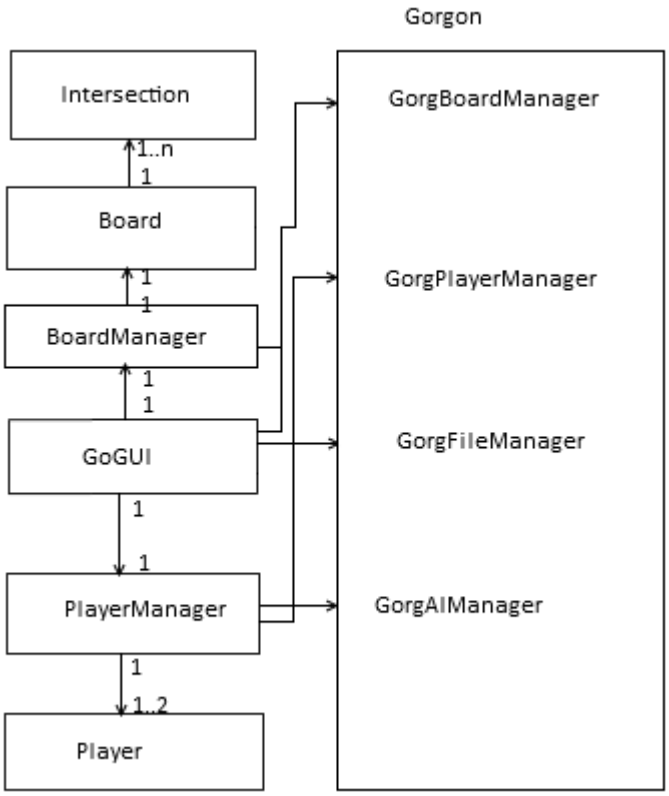


Figura 2.1: Arquitectura de GoGUI

L' avantatge de separar GoGUI de Gorgon és que GoGUI no havia de fer cap càlcul referent al joc en si. L' única cosa que feia GoGUI és actualitzar el taulell gràfic i l' informació necessària per al jugador. Però aquesta informació era Gorgon qui la calculava. Comprovar que es complien les regles del joc, calcular un moviment de la IA, mantindré la representació lògica del taulell... Tot això són tasques que feia Gorgon.

D' aquesta manera, si es volia fer una interfície gràfica completament diferent, visualment més atractiva, i amb un bon disseny gràfic, no havíem de fer cap càlcul referent al joc en si. En resum, des d' un principi es va tindre clar la necessitat de l' abstracció entre visualització del joc i dades del joc.

2.4.2 El taulell desde un punt de vista gràfic

La construcció del taulell es va implementar de manera que si en algun moment es volia canviar el tamany del taulell o fer un zoom, fos relativament fàcil. Així que es va pensar com la unió de les imatges del que eren les pròpies interseccions. Si volíem un taulell de 19x19, simplement teníem que unir 361 imatges representant les interseccions. A més, aquest plantejament facilitava molt el fet de posar una pedra a una intersecció. Simplement teníem una imatge de pedra blanca i de pedra negre, i al fer clic en una intersecció es superposava aquesta imatge. Qt, amb les seves llibreries gràfiques, va facilitar enormement aquesta fase del projecte.

Les imatges utilitzades van ser les següents:

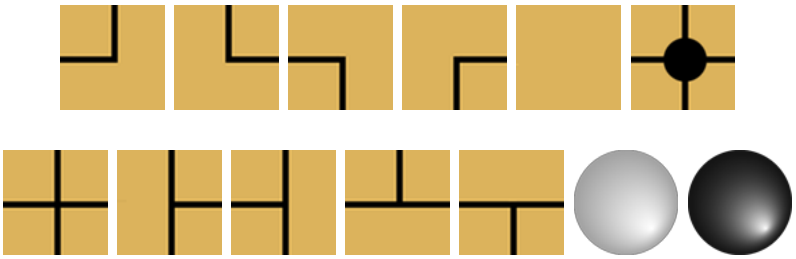


Figura 2.2: Interseccions del taulell

El taulell de 19x19, amb totes les interseccions ja ajuntades, tenia el següent aspecte:

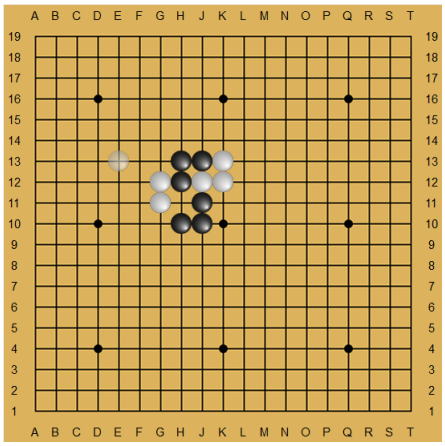


Figura 2.3: Taulell complert

Una característica important, és el fet de ressaltar l' intersecció on és el punter del ratolí. Es pot veure a la intersecció E13, com es veu una pedra blanca transparent (en la captura de pantalla, el punter està en aquella posició). Això es va fer per a que el jugador tingués una visualització immediata d' on estava a punt de jugar.

Una altre característica important d' un joc de taulell és la numeració de les posicions. Per convenció, en un taulell de Go s' etiqueta l' eix de les Y amb un número, començant des de la part inferior, i l' eix de les X amb una lletra d' esquerra a dreta. Es pot notar com la lletra I llatina no hi és. Això és un estàndard, i deriva del fet de que en els alfabets orientals aquesta lletra no existeix.

2.4.3 Ampliant el taulell de joc

GoGUI es va pensar com una aplicació destinada a resoldre Tsumegos. Ja fos contra un jugador, o contra la IA de Gorgon. Un Tsumego és un problema localitzat en una zona relativament petita del taulell. Així que no tenia sentit que el jugador veies tot el taulell quan la zona de joc real estava a una cantonada, per exemple.

Per aquest motiu, es va dotar a GoGUI de la funcionalitat de definir la zona de joc on es volia jugar el Tsumego, i només presentar aquesta zona, amb un tamany més adequat. Es van definir 3 tamanys de zones: taulell complert, zona de joc mitjana, i zona de joc petita. Com més petita era la zona de joc, més gran era la visualització d' aquesta. A l' hora, per al tamany de zona mitjana es van definir 4 àrees de joc, i 6 per al tamany de zona petita.

A continuació es poden veure algunes de les combinacions possibles de zona mitjana, i zona petita.:

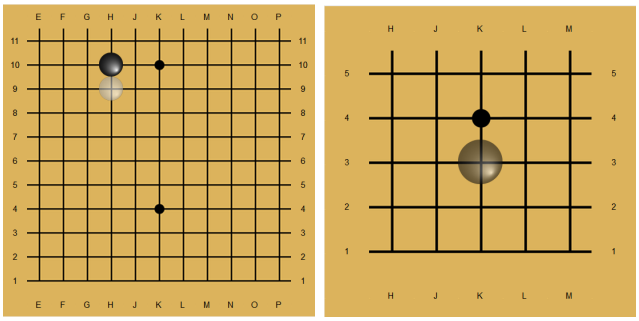


Figura 2.4: Taulell mitjà i taulell petit

Aquesta funcionalitat va fer necessari el fet de mantindre tres resolucions per a les imatges mostrades anteriorment. Una resolució baixa per a les interseccions a taulell complert, i una resolució més alta per a les que es jugaven a zona petita.

2.4.4 Interfície gràfica

Finalment, es mostra la interfície que es va dissenyar per a GoGUI.

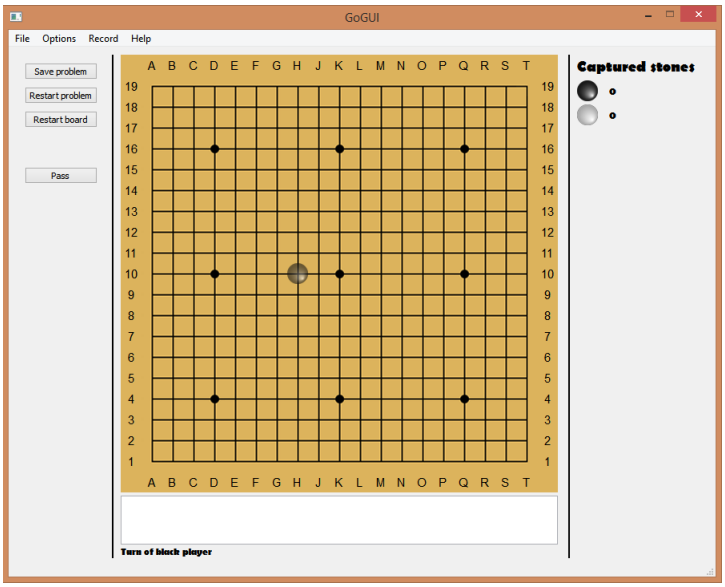


Figura 2.5: Interfície complerta

Com es pot veure, és una interfície molt senzilla, però ampliable a noves funcionalitats. A la secció esquerre estan localitzats tots els botons que

poden ser necessaris en una partida o Tsumego. A la secció dreta està tota l' informació a nivell de jugadors. A la secció central està el taulell de joc, amb una secció a la part inferior per posar comentaris a cada jugada (funcionalitat no implementada). A la part inferior es mostra informació de cada torn. Aquí es mostra desde el torn del jugador, fins a informació relativa al trencament de les regles. A més, aquests tipus de missatges es mostren en vermell per a que es vegin de forma ràpida. Finalment, a la part superior es pot veure el menú amb les funcionalitats bàsiques. El menú de creació de partides i Tsumegos, el menú per configurar les opcions, el de gravar partida (funcionalitat sense implementar), i el menú d' ajuda on es té previst posar ajut per a l' ús de l' aplicació, configuració de la IA, i regles del Go.

2.5 Gorgon

2.5.1 Introducció a Gorgon

L' arquitectura de Gorgon presenta el següent aspecte.

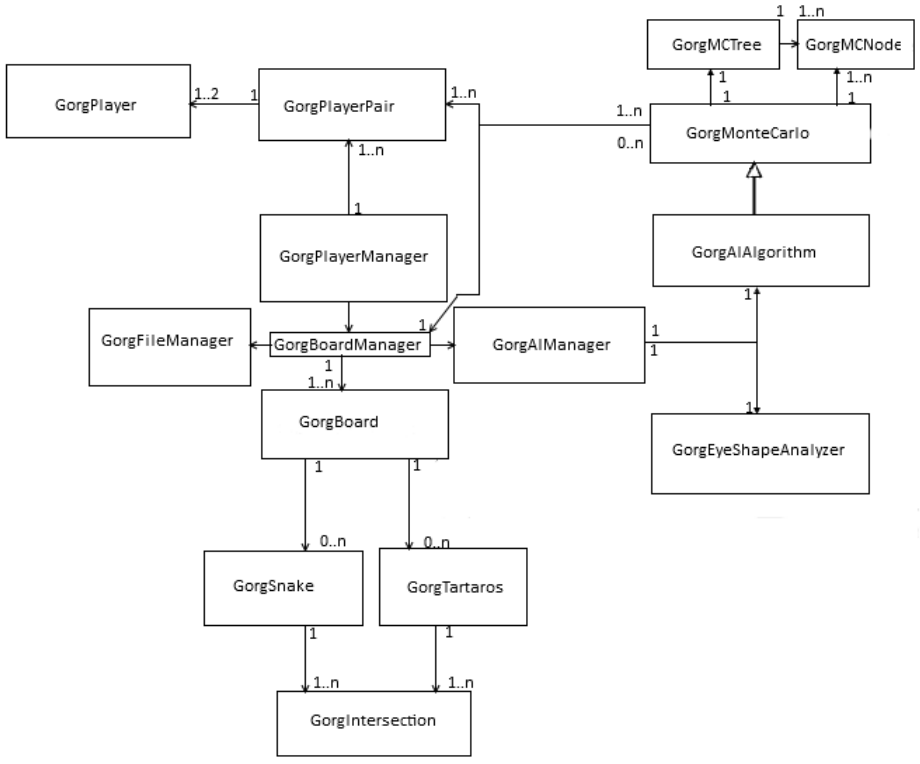


Figura 2.6: Arquitectura de Gorgon

Gorgon es va dissenyar en vistes de ser una llibreria ampliable. És per això, i com es pot veure al diagrama, que permet mantindre la representació de diversos taulells de joc. Encara que GoGUI no va ser dissenyada amb aquesta funcionalitat, Gorgon permetria fer una interfície gràfica en la que es portessin a terme diferents partides simultàniament, i contra diferents jugadors.

A més, el mòdul de IA es va dissenyar de manera que fos possible implementar diferents algorismes. En aquest projecte s' ha implementat només un, que es veurà en capítols posteriors. Però si es volgués, es podrien implementar més algorismes, simplement heretant de la classe AIAlgorithm, i implementant el mètode abstracte playMove.

Aquesta secció s' ha dividit en dos grans parts. La que defineix l' implementació de una partida de Gorgon, i la que defineix l' implementació de l' intel·ligència artificial.

2.5.2 Primeres versions de Gorgon

Les primeres versions de la llibreria Gorgon, es basaven en el recorregut de tot el taulell per a cada pedra que es jugava. En aquest punt, el coll d' ampolla l' ocasionava el càlcul de la regla de les llibertats, que era el procés que més trigava. Per calcular si una pedra complia o no, la regla de les llibertats, Gorgon comprovava el veïnatge de cada intersecció. Si tenia veïns, comprovava els veïns d' aquests veïns. Així successivament, fins a arribar a una intersecció sense veïns explorats, i llavors tornar enrere per extendre tots els veïnatges corresponents . En el pitjor dels casos podia arribar a recórrer tot el taulell moltes vegades. És a dir, en casos molt extrems (gairebé impossibles de donar-se en una partida real), es podien arribar a fer fins a $(19*19)^2$ iteracions.

Amb l' ordinador amb els que es van fer les proves, això es traduïa a una mitjana d'uns 0.2 segons, en fases avançades d' una partida real, on hi han moltes connexions entre les pedres. A ulls d' un usuari, 0.2 segons pot semblar un temps raonable, però com es veurà més endavant, el temps de còmput d' una pedra és un punt vital en el càlcul d'una jugada per part de l' inte·ligència artificial.

S'entrarà més en detall en capítols posteriors, però la IA es basa en el fet de fer simulacions aleatòries de jugades, i agafar la que millor resultats

doni. Això implica que com més simulacions fem, millor resultats obtindrem.

Suposem necessàries 500 simulacions amb una mitjana de 2 jugades per simulació, per calcular un moviment. És una situació hipotètica, ja que realment necessitarem moltes més simulacions. Però per el que es vol demostrar en aquest punt, 500 seran suficients. 500 simulacions a 2 jugades per simulació impliquen la necessitat de fer 1000 jugades per a calcular un moviment de la IA. A 0.2 segons per jugada, es necessitarien 200 segons per calcular una nova jugada. I això sense tindre en compte els càlculs propis de l'algorisme. A la pràctica serien més de 3 minuts per calcular cada moviment de la màquina. És clar que per a la majoria de jugadors seria desesperant.

Així doncs, 0.2 segons per jugada era un temps de còmput impensable per el que es pretenia fer en aquesta aplicació. S' havia de trobar alguna manera alternativa de fer el càlcul d' afegir una nova pedra al taulell, que fos més barata computacionalment.

Per tal d' aconseguir això va ser necessària la definició d' una estructura motl concreta. En el següent capítol s' entrarà més en detall sobre aquesta estructura, què representa, i com es va utilitzar en l' implementació de les regles del Go.

2.5.3 Serps

L' estructura a tractar es va anomenar serp. Les serps representaven la connexió entre un conjunt de pedres del mateix color. Es poden veure alguns exemples a la figura 3.A.

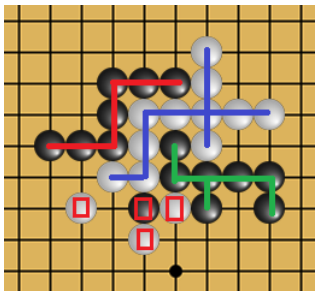


Figura 2.7: Exemple de serps al taulell

A la figura 3.A hi ha 7 Serps. Notis que les pedres marcades amb rec-

tangles vermells no estan connectades a cap altre pedra del seu color, però formen una Serp per sí mateixes. Així doncs, qualsevol pedra del taulell sempre està associada a una Serp.

Aquestes estructures guardaven diferents dades, però les més importants eren el nombre de llibertats de les que disposa el conjunt de pedres, i el conjunt de pedres associades a la serp. Aquesta informació s' anava actualitzant incrementalment a cada jugada. És a dir, per a cada jugada es mirava a quines serps afectarien el moviment, i s' actualitzaven en cas que sigues una jugada legal.

2.5.4 Regla de les llibertats

Un cop definides les serps, vegem com es van utilitzar per a l' implementació de la regla de les llibertats. Es veuran tres punts clau: confirmar que una jugada no trenca la regla de les llibertats i que per tant és una jugada legal, capturar pedres enemigues, i finalment com actualitzar les serps donada una jugada legal.

En primer lloc, vegem l' algorisme que es va idear per tal de comprovar el compliment de la regla de les llibertats. Donada una jugada a una intersecció del taulell:

Si no hi ha cap serp aliada adjacent, i l' intersecció no té cap llibertat local:
Jugada il·legal. Sense llibertats. És suïcidi.
Sino, si l' intersecció té llibertats locals:
Jugada legal. La intersecció té com a mínim una llibertat.
Sino, s' han de mirar les llibertats de les serps adjacents aliades:
Per a cada serp aliada adjacent:
Si té alguna llibertat:
Jugada legal. La nova pedra tindrà llibertats.
Si s'arriba aquí, la nova pedra no tindrà cap llibertat. Jugada il·legal.

L' algorisme per determinar quines pedres s' han capturat es va idear de la següent manera. Donada una jugada legal en una determinada posició:

Per cada serp enemiga adjacent:
Si el nombre de llibertats és igual a 1:

Aquesta serp ha estat capturada. S' ha tancat la única llibertat que tenia.

És fàcil notar que el fet de que una serp hagi estat capturada implica que han estat capturades totes les interseccions d' aquella serp. En aquest cas es comença a veure més clar el potencial de les serps. Com que és una estructura incremental, podem comprovar si un conjunt de pedres ha estat capturat des d' un nivell d' abstracció més elevat, fent una única comparació per cada grup de pedres.

Seguidament es mostra el pseudocodi de l' algorisme que es va pensar per tal d' actualitzar les serps afectades, degut a una nou moviment.

Donada una jugada legal en una determinada posició:

Mirem quin és el nombre de serps adjacents aliades, i actuem de la següent manera segons convingui:

0 serps:

Aquesta jugada crea una nova serp que no existia encara. Creem una nova serp, la qual tindrà només una pedra (la pròpia jugada), i el conjunt de llibertats locals pròpies d'aquesta pedra.

1 serp:

Aquesta jugada ha fet créixer una serp que ja existia.
Afegim la nova pedra al conjunt de pedres de la serp, i afegim al conjunt de llibertats les llibertats locals de la nova pedra.
Seguidament, hem de treure la llibertat que acabem de tancar per la col·locació de la nova pedra

2/3/4 serps:

Aquesta nova pedra causarà l' unió de serps que abans eren independents.
Escollim la serp més gran com la serp que acollirà a les altres.
Afegim la nova pedra a la serp escollida, afegim les llibertats locals, i treiem la llibertat que acabem de tancar.
Per cada serp adjacent:
Afegim totes les seves interseccions a la serp escollida.
Eliminem la llibertat tancada per la nova pedra col·locada
Afegim totes les llibertats restants a la serp escollida.
Eliminem totes les dades de la serp antiga.

Si hi ha serps adjacents enemigues, els hi borrem la llibertat tancada per la nova pedra.

Si hem capturat algun grup de pedres enemigues, afegim les interseccions d'aquestes pedres enemigues com a llibertats de les serps aliades adjacents.
Finalment, eliminem les serps sense dades, ja que ara formen part d' una altre serp més gran.

A continuacio es mostren alguns exemples per als 3 casos d' actualització, per tal d' entendre l' algorisme en la seva totalitat.

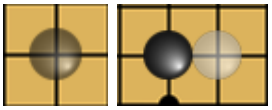


Figura 2.8: Exemples d' actualització amb cap serp aliada adjacent

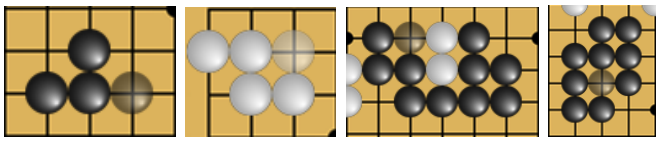


Figura 2.9: Exemples d' actualització amb 1 serp aliada adjacent

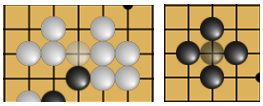


Figura 2.10: Exemples d' actualització amb més d' una serp aliada adjacent

Encara que aquest algorisme sembli més complicat que el més bàsic que es va fer en primera instància, es pot comprovar com la complexitat de còmput ha millorat moltíssim. En els pitjor dels casos, per actualitzar una serp hauríem de recórrer com a molt una vegada, totes les serps adjacents. Així que en un cas molt extrem en que tinguem dues serps ocupant tot el taulell, i posem una pedra per unificar-les, el màxim nombre d' iteracions seria de 361.

La mitjana del temps necessari per a col·locar una pedra va baixar a 0.00008 segons. Si prenem el mateix exemple mencionat anteriorment, es trigaria al voltant d'uns 0.08 segons teòrics per fer 1000 moviments. Això no es del tot cert perquè com veurem més endavant i s' ha dit anteriorment,

haurem d'afegir temps de còmput derivat del propi algorisme de còmput de moviments, i d' una estructura que necessitarem, i encara no s'ha explicat.

No obstant, es pot veure com, en igualtat de condicions, l' algorisme proposat va millorar enormement el temps de còmput. Això sens dubte va ajudar a construir una IA molt més eficient i ràpida.

2.6 Intel·ligència artificial

2.6.1 Introducció

En primer lloc, i abans d 'entrar en detall a la solució que es va implementar, és adequat entendre el perquè de la dificultat d' implementar una IA en el joc del Go.

Simplificant, per trobar una resposta automàtica en un joc d' estratègia, tenim dues opcions: o bé explorar totes les partides possibles, o bé trobar algun model que sàpiga actuar en conseqüència. La primera opció, com ja s' ha dit anteriorment, és totalment inviable amb la capacitat computacional d' avui en dia. La segona opció ens porta a una situació gairebé igual de complicada de resoldre.

Per a que una IA pugui donar una resposta tàctica o estratègica a un joc d' aquestes característiques, necessita tindre coneixement de la "qualitat" d' una jugada. És a dir, necessita associar una puntuació a una determinada jugada, per saber si realment és una bona jugada. Dit d' una altre manera, necessita una funció d' avaluació. Aquest possiblement és el repte més gran al que s' enfronta el Computer Go.

En el Go, la qualitat d' una jugada depèn de moltíssims factors, i cada jugada necessitaria un anàlisis complex per determinar el seu valor. Inclús una jugada pot tindre un valor molt pobre en un moment actual, però ser de vital importància en el futur de la partida. Per a fer-se una idea, una bona jugada podria ser sacrificar un grup de pedres aliades per tal d' aconseguir un benefici major en moments posteriors a la partida.

Tot això fa que construir una funció d' avaluació generalitzada sigui una tasca impossible. Per a això, la breu història del Computer Go ha anat trobant alternatives per a resoldre aquest problema. Desde tècniques de Visió

per computador, fins a bases de dades plenes de patrons de jugades típiques.

Un Tsumego es pot pensar com una versió molt reduïda i simplificada d' una partida de Go. No obstant, encara que la construcció d' una funció d' avaluació segueixi sent extremadament complicada, sota aquestes circumstàncies, hi ha mètodes que han demostrat ser molt eficients.

Seguidament s' explicarà la solució implementada en aquest projecte. Cal destacar que el coneixement del que es disposava, des d' un punt de vista estratègic i tàctic, era bastant bàsic. Es limitava a figures molt bàsiques i l' intuïció que pugui tindre un jugador de nivell principiant. Això és un aspecte molt important a tindre en compte a l' hora de construir la IA, ja que al tindre un coneixement tan bàsic del joc, es va decidir que el més sensat era no fer ús de coneixement expert. Això vol dir que s' havia d' intentar en la mida del possible, no utilitzar regles heurístiques pròpies del joc.

I així es va fer. Es va decidir utilitzar un algorisme anomenat Monte Carlo Tree Search.

2.6.2 Monte Carlo Tree Search (MCTS)

L' MCTS forma part d' una família de mètodes anomenats Mètodes de Monte Carlo, en els que l'idea bàsica consisteix en trobar la descripció del comportament d' un sistema probabilístic, mitjançant moltíssimes simulacions en l' espai de solucions . Quan el nombre de simulacions tendeixi a infinit, s' obtindrà el comportament del sistema de manera acurada.

L'MCTS es una extensió d' aquests mètodes, aplicat a la cerca en arbre. A continuació s' entrarà més en detall, però l' idea bàsica consisteix en fer moltíssimes simulacions fins al final del joc, i anar guardant els seus resultats en forma d' arbre. Quan es decideix acabar el procés (és un mètode iteratiu), s' agafa la fulla del node arrel que millors resultats hagi obtingut. L' avantatge principal d'aquest mètode, i raó per la qual es va escollir, és que no es necessita cap tipus de coneixement expert. A més, en els últims anys ha anat guanyant molta popularitat en el camp del Computer Go. I de fet, algunes de les millors IA's actuals, utilitzen aquest mètode.

Els resultats d' aquest algorisme tenen una extraordinària dependència amb el temps que podem passar fent simulacions. Com més simulacions fem, millors resultats obtindrem. Si es fan 10 simulacions, a no ser que el sistema

sigui extremadament senzill, els resultats que s' obtinguin seran totalment aleatoris. Si per el contrari fem milions de simulacions, obtindrem uns resultats extremadament acurats, i podrem dir amb total seguretat que la solució trobada es la correcta. Amb això es pot entendre el perquè va ser necessari reduir el temps de còmput al col·locar una nova pedra al taulell.

Aproximant-nos al cas que ens ocupa, en un joc de taula, els nodes de l' arbre són les jugades que es van simulant, i per cada node tenim el nombre de partides jugades i guanyades, a partir d' aquella jugada.

Cada simulació del MCTS consta de 4 fases:

- **Selecció**: en aquesta fase, es baixa a través del arbre, seleccionant els moviments que millors resultats hagin donat fins a aquell moment, sent per tant, les jugades més prometedores. Un cop es trobi un node fulla amb moviments no explorats, passarem a la següent fase.
- **Expansió**: en aquesta fase es crea un node fill al node seleccionat a la fase de selecció. Si es donés el cas de que el node seleccionat estigués en un estat de final de joc, aquesta fase s' ignora i es passa directament a la fase de simulació a partir del propi node seleccionat.
- **Simulació**: a partir del node creat, es fan moviments aleatoris fins a arribar al final del joc.
- **Propagació**: finalment, els resultats de la simulació s' expandeixen a través de tots els nodes que s' han recorregut per arribar fins al node actual. És a dir, es suma una partida jugada i una partida guanyada (en el cas que escaigui), a tots els nodes desde el node actual (inclòs), fins al node arrel.

La següent figura mostra l' exemple de les 4 fases d' una simulació:

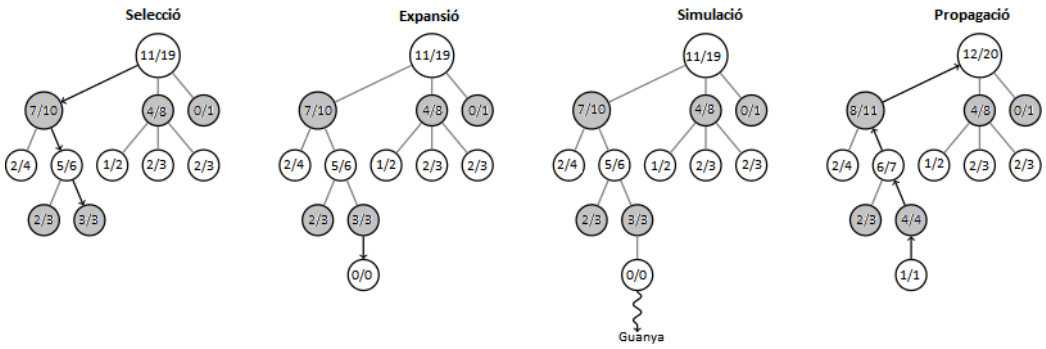


Figura 2.11: Simulació complerta en l' MTCS

Com es pot veure, l' algorisme expandeix els nodes amb els resultats més prometedors, fent que l' arbre creixi en profunditat. Això implica que els moviments que no tenen, o tenen molt poques probabilitats de portar la partida a la victòria, no són gairebé explorats.

2.6.3 Upper Confidence Bounds

El MCTS sembla un algorisme molt prometedor sobre el paper. Però a la pràctica té dos grans problemes associats a la fase de selecció, que dificultaven la seva implementació més bàsica per a problemes reals.

El primer és que la relació de partides jugades/guanyades en un node que encara no ha estat provat, és 0/0. La solució a aquest problema és trivial: sempre es seleccionaran els nodes que no han estat provats, abans de seguir la cerca en profunditat. Això implica que abans de explorar un nivell inferior, sempre s' hauran provat com a mínim una vegada, tots els moviments del nivell superior.

El segon problema és una mica més complicat de solucionar. Suposem un node, el qual la successió de jugades porta 99 vegades a la victòria, i 1 vegada a la derrota. Es podria donar el cas que a la primera simulació que es fes a través d' aquell node es tingués la mala sort d' arribar a la situació de derrota (recordem que la fase de simulació és totalment aleatòria). Això deixaria al node amb un rati de 0/1, impedint que aquest node torni a ser explorat, i en conseqüència que no sigui escollit com a jugada guanyadora.

És evident que es feia necessari trobar un equilibri entre exploració i explotació, per tal de que aquestes situacions no es passessin. La solució que es va trobar la va proposar un grup d' experts fa uns anys. La fórmula UCB-1 (Upper Confidence Bounds).

$$\frac{W_i}{n_i} + C\sqrt{\frac{ln(t)}{n_i}}$$

On,

W_i és el nombre de partides guanyades a través del node.
n_i és el nombre de partides jugades a través del node.
C és una constant escollida empíricament.
t és el nombre total de partides jugades fins al node.

La primera part del sumand té un valor elevat per a nodes amb un bon rati de partides (explotació), mentre que la segona part, té un valor elevat per a nodes amb poques simulacions (exploració). Si el paràmetre C és 0, s' ignorarà la segona part del sumand, deixant-ho tot a l'explotació dels nodes.

Llavors la fase de selecció es va modificar, fent que si hi havia nodes sense simulacions, donar prioritat a aquests, i que els casos es seleccionessin aplicant l' UCB. D' aquesta manera aconseguíem un equilibri entre exploració i explotació.

2.6.4 Fase de simulació i final de partida

Com s' ha comentat, la fase de simulació fa jugades aleatòries fins al final de partida. Però, quin és el final de partida en un Tsumego? I com podem detectar aquesta situació de manera automàtica? Gorgon es va implementar de manera que detectés dos tipus de final de partida:

- Un dels jugadors ha conseguit formar dos ulls (guanya el jugador defensor).
- Un dels jugadors ha capturat un nombre de pedres suficientment alts com per a que a l' altre jugador no li surti a compte seguir jugant (guanya el jugador atacant).

El primer cas és trivial. Si el jugador defensor fa dos ulls, aconsegueix l' objectiu del Tsumego. La manera de detectar els dos ulls (cosa que no es

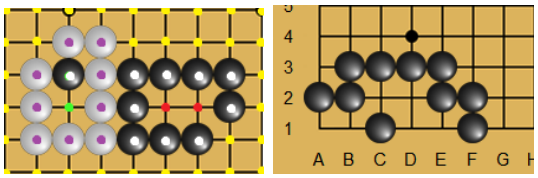
tan trivial), es veurà en els següents capítols. El segon cas es va plantejar d' aquesta manera degut a la dificultat de detectar automàticament una situació en la que el jugador defensor no pogués fer dos ulls de cap de les maneres. Per tal d' aconseguir això s' haguessin tingut que implementar regles heurístiques, cosa que com ja s' ha dit, haguessin dificultat molt l' implementació, degut al poc coneixement que es té del joc des d' un punt de vista tàctic i estratègic.

Per tant, per tal de simplificar la detecció d' aquesta situació, es va definir aquesta regla, derivada del fet de que gairebé tots els Tsumegos d' avui en dia, es poden guanyar sense la necessitat de sacrificar un gran nombre de pedres. Si s' arriba al cas de aconseguir formar dos ulls, el més probable és que no hagi sortit a compte, degut a que el nombre de territori guanyat no compensa el nombre de pedres perdudes. El nombre de pedres necessari per a guanyar el problema en cas de ser el jugador atacant, es va deixar com un paràmetre de l' IA. Per defecte es va posar a 4. És a dir, si un dels dos jugadors aconseguia capturar 4 pedres del equip jugador contrincant, es considerava que havia capturat un nombre de pedres suficient com per a que a l' altre jugador no li surti a compte seguir jugant.

Ara bé, com detectàvem la formació de dos ulls per part del jugador defensor? Abans de donar resposta a això, es veurà una altre estructura que es va fer necessària: els Tàrtars.

2.6.5 Tàrtars

Els Tàrtars es van idear com una estructura que guardava les regions del taulell, d' un mateix color. És a dir, totes les interseccions que tanquen un conjunt de serps del mateix color. A continuació es mostra un exemple gràfic.



(a) Tartar 1 (b) Tartar 2

Figura 2.12: Exemple de dos Tàrtars

A la figura A hi ha 3 serps. La serp negra de la dreta té dos Tàrtars: la regió marcada amb les interseccions vermelles, i la regió formada per les

interseccions liles, verdes, i grogues. La pedra negra de l' esquerra (marcada en blanc), és una pedra aliada i no forma part de cap dels dos tàrtars. En definitiva, tenim la regió interior, i la regió exterior a la serp.

De manera similar, la serp blanca té dos Tàrtars. Les dues interseccions interiors (incloent la pedra negra enemiga marcada amb blanc), i totes les demés interseccions que no formen part de la serp. És a dir, les interseccions marcades en groc, vermell, i les blanques de la serp de la dreta.

És important destacar que les interseccions del color enemic formen part d' una regió tant com ho fan les interseccions buides. És a dir, un Tàrtar tracta de igual manera una intersecció enemiga que una de buida. També es pot veure que una característica dels Tàrtars és que les pedres aliades no formen part de cap Tàrtar del mateix color. Només compten interseccions buides i enemigues.

La figura B és un exemple més senzill on es mostra que les regions aliades són les fronteres que limiten les pròpies regions de la serp a tractar. Així doncs, la serp gran té tres Tàrtars. La regió exterior i les dues interiors. Però la pedra petita a C1 només en té 2, ja que la serp gran fa de frontera a la regió exterior. Per tant, C1 només té els Tàrtars formats per les interseccions A1 i B1, i C2, D2, D1, E1.

Al igual que les serps, aquestes estructures, s' havien d' actualitzant a mida que els jugadors anaven col·locant pedres sobre el taulell. La solució a aquest problema va ser un algorisme per a etiquetar regions, utilitzat en imatges binàries. Encara que en un joc de Go hi ha 3 colors (blanc, negre i buit), gràcies al fet de que els Tàrtars no tenen en compte el color dels enemics, va ser possible considerar el taulell com una imatge binària.

L' algorisme s' anomena algorisme de dos passos, i com el seu nom indica, recorre l' imatge (el taulell en el nostre cas) dues vegades. Treballa a nivell de píxel o intersecció en el nostre cas, i es basa en assignar una etiqueta a totes les interseccions. Totes les interseccions dins de la mateixa regió tindran la mateixa etiqueta al finalitzar l' algorisme. Una de l' informació més important que contenen els Tàrtars, és l' etiqueta corresponent a cadascun d'ells. Per exemple, un Tàrtar podia tindre l' etiqueta "1". Això volia dir que totes les interseccions corresponents a aquell Tàrtar havien de tindre la mateixa etiqueta "1".

En el primer pas de l' algorisme, s' etiqueta cada intersecció depenent

de les etiquetes del seus veïns. Hi ha dues variants d' aquesta fase: la que treballa amb connectivitat a 4, i la que treballa amb connectivitat a 8. En el cas dels Tàrtars vam necessitar la versió de 4, ja que una unió en diagonal de dues interseccions, ja forma una cantonada del Tàrtar. Les regles d' etiquetatge de l' algorisme amb connectivitat a 4 són les següents:

- *L' intersecció de l' esquerra té el mateix color que l' actual?*

  Si: forma part del mateix Tàrtar. Assignar a l' intersecció actual l' etiqueta de la de l' esquerre.

  No: passem a la següent condició

- *Les interseccions superior i esquerre tenen el mateix color però no la mateixa etiqueta?*

  Si: formen part del mateix Tàrtar, però han de juntar-se. Assignem a l' intersecció actual l' etiqueta més baixa de les dues interseccions i guardem la l' equivalència a la taula d' equivalències.

  No: passem a la següent condició

- *L' interseccio de l' esquerra té un color diferent que l' actual, però l' actual té el mateix color que la superior?*

  Si. Assignem a l' interseccio actual l' etiqueta de la superior.

  No: passem a la següent condició

- *Si hem arriba fins aquí es perquè les interseccions superior i esquerre tenen un color diferent de l' actual.*

  Creem una nova etiqueta i l' assignem a l' intersecció actual.

En el segon pas, es re-assignen les interseccions que pertoquin segons la taula d' equivalències. A més, a Gorgon, en aquest pas s' assignava a cada intersecció el Tàrtar corresponent. Si l' intersecció és buida, o enemiga del color que estem calculant, es comprovava si hi havia algun Tàrtar assignat a aquella etiqueta. Si no era així, volia dir que encara no s'havia creat cap Tàrtar corresponent a aquella regió, i creàvem un nou Tàrtar assignant-li aquella etiqueta. Si per el contrari, ja existia un Tàrtar amb aquella etiqueta, volia dir que el Tàrtar corresponent a aquella regió ja existia, i per tant havíem d' assignar aquesta intersecció a aquell Tàrtar.

Una optimització immediata que es va pensar, és que aquestes estructures només tenien sentit en el cas de que un dels jugadors fos la IA. Així doncs, en els Tsumegos sense IA, o partides completes entre dos jugadors, aquestes

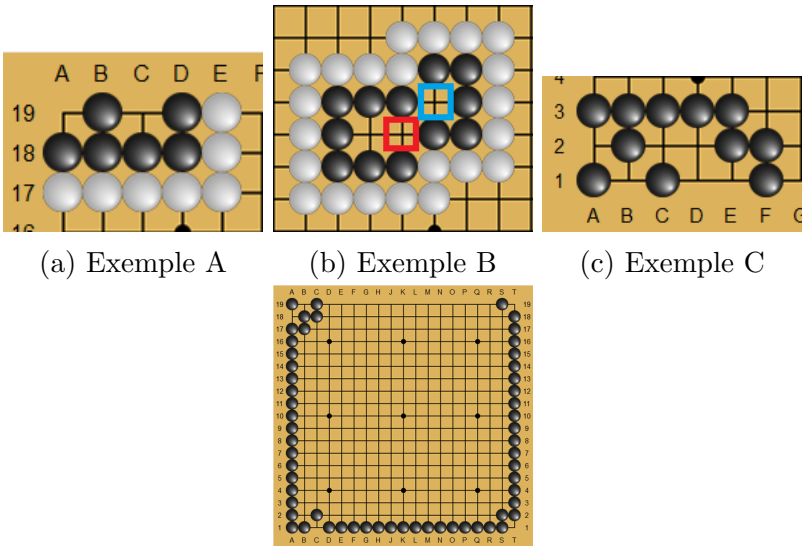
estructures ni tan sols es calculaven. D' aquesta manera el temps per col·locar una pedra en aquests casos, millorava una mica.

2.6.6 Vida incondicional segons Benson

Un cop presentats els Tàrtars, es pot parlar de quina va ser la solució per a detectar automàticament la formació de dos ulls.

David B. Benson va definir fa anys el significat de que una serp estigués incondicionalment viva. Una serp incondicionalment viva vol dir que donada la situació en la que està, si el jugador atacant tingués jugades infinites i el jugador defensor passes el torn a cada una d' elles, seria impossible per al jugador atacant capturar la serp. A la pràctica això es tradueix en si el grup de pedres té una formació de dos dulls reals, com a mínim.

Vegem uns quants exemples.



(a) Exemple A (b) Exemple B (c) Exemple C

(d) Exemple D

Figura 2.13: Exemples de vida incondicional segons Benson

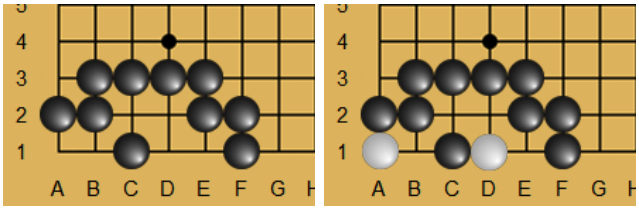
En la figura A, es pot veure com la serp negre no pot ser capturada de cap manera per moltes jugades seguides que puguin fer les blanques. Llavors es diu que la serp negre està incondicionalment viva. És clar que això es així perquè té una formació de dos ulls.

A la figura B en canvi, les serps negres no estan incondicionalment vives. Si les blanques tinguessin jugades infinites i seguides, podrien arribar a capturar la serp negre de la dreta. Ho podrien fer amb dos moviments: el primer a l' intersecció del quadrat vermell, i el segon a l' intersecció del quadrat blau. Havent capturat la serp de la dreta, la de l' esquerra quedaria totalment indefensa i tot el territori passaria a ser de les blanques.
A la figura C es té una situació semblant. Les blanques podrien arribar a capturar la serp formada per l' intersecció C1 si tinguessin jugades seguides infinites. Una seqüència possible seria: D1-¿C2-¿B1. Havent capturat aquesta pedra, es podria capturar fàcilment A1, i impedir que les negres fessin dos ulls al territori.
Finalment a la figura D tenim una barreja dels anteriors casos. La serp formada per l' intersecció A19, i la que conté la C19 estan incondicionalment vives. No hi ha cap manera que les blanques puguin capturar aquestes serps per moltes vegades seguides que juguin. No obstant, les demès serps no compleixen aquesta condició. Les serps S19 i C2 es poden capturar amb poques jugades seguides. Un cop capturades aquestes, la serp gran que conté l' intersecció D1, també es podria capturar amb bastantes jugades (les que rodegen a la serp).

Benson també va formalitzar un algorisme per detectar automàticament la vida incondicional d'un grup de serps. Aquest algorisme utilitza dos conceptes: regions i regions vitals a una serp. Una regió és exactament el mateix que un Tàrtar. I de fet, aquest és el motiu per el que es va haver d' implementar aquest tipus d' estructures. Eren necessàries per al càlcul de l' algorisme de Benson. Es podrien haver anomenat Regions en lloc de Tàrtars perfectament. Però es va decidir seguir amb la nomenclatura relacionada amb la mitologia grega que caracteritza al projecte. A efectes pràctics, una regió és un Tàrtar, i per tant, podem parlar de Tàrtar vital a una serp.

2.6.7 Tàrtars vitals

Una regió (o Tàrtar) es diu que és vital a una serp, si i només si, totes les seves interseccions buides (les enemigues no compten) són llibertats de la serp. Vegem alguns exemples.



(a) Exemple A (b) Exemple B

Figura 2.14: Exemples de Tàrtars vitals

A la figura A tenim dues serps. La serp gran té 3 Tàrtars i la serp petita només 2. El Tàrtar format per les interseccions A1 i B1, és vital per a la serp gran. Totes les interseccions del Tàrtar són llibertats de la serp. Concretament, A1 és llibertat de A2, i B1 llibertat de B2. Ara bé, el mateix Tàrtar, no és vital per la serp formada per la pedra C1. B1 es llibertat de la serp, però A1 no ho és. Per tant, com que hi ha una intersecció al Tàrtar que no és llibertat de la serp, no és un Tàrtar vital per a la serp.

De manera similar, el Tàrtar format per les interseccions C2, D2, D1 i E1, no és vital a la serp gran. La intersecció D1 no és llibertat. Per descomptat, i seguint la mateixa lògica, el Tàrtar exterior no és vital per cap de les dues serps.

La figura B es gairebé igual però amb dues pedres blanques enemigues. Recordem que les pedres enemigues no compten per a calcular si un Tàrtar es vital. Per tant, el Tàrtar format per A1, i B1 si que és vital per a la serp de C1. Ara totes les interseccions buides del Tàrtar (B1), són llibertats de C1. El Tàrtar format per C2, D2, D1 i E1 segueix sense ser vital a C1, perquè ni D2 ni E1 són llibertats de C1.

Seguint el mateix raonament, el Tàrtar format per C2, D2, D1 i E1 si és vital per a la serp gran ara. La intersecció D1 ha estat ocupada i per tant no compta per al càlcul. I tenint en compte que C2, D2, i E1 són llibertats de la serp, el Tàrtar és vital a la serp.

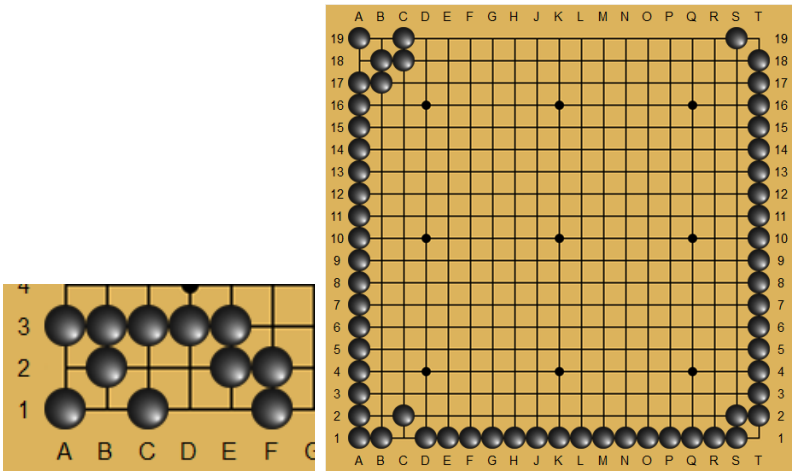
2.6.8 Algorisme de Benson

Coneixent els Tàrtars, la vida incondicional, i els tàrtars vitals a una serp, es pot presentar finalment l' algorisme de Benson per a la detecció automàtica de dos ulls.

S = conjunt de serps.
R= conjunt de Tàrtars que encerclen aquestes serps.
Repetir:
Borrar de S totes les serps que tinguin menys de dos Tàrtars vitals a R.
Borrar de R tots els Tàrtars rodejats per alguna pedra que pertanyi a una serp que no està a S.
Mentre s' hagi esborrat algun element de R o S
S = conjunt de serps incondicionalment vives.
R = conjunt de regions que formen part d' alguna serp incondicionalment viva.

Recordem que aquest algorisme és el que ens permetia detectar el final de partida en una simulació del Monte-Carlo Tree Search. Com es pot veure, es un algorisme iteratiu, que acaba quan de cap dels dos conjunts no s' ha esborrat cap element.

A continuació es mostren dos exemples de l' aplicació de l' algorisme.



(a) (b)

Figura 2.15: Exemples per l' algorisme de Benson

Els dos exemples s'han comentat amb anterioritat. S' ha vist que a la figura A no hi ha cap serp incondicionalment viva, ja que la seqüència D1

- C2 - B1 - A2 deixa el territori a favor de les blanques. A la figura B en canvi, estan incondicionalment vives les serps A19 i la serp gran formada per C19. Vegem com arribar de manera automàtica a aquestes conclusions amb l' algorisme de Benson.

Figura A

Suposem la seguent nomenclatura:
SA1 = Serp formada per l' intersecció A1.
SC1 = Serp formada per l' intersecció C1.
SA3 = Serp gran formada per l' intersecció A3 i la resta de pedres.
TA2 = Tàrtar format per l' intersecció A2.
TB1 = Tàrtar format per l' intersecció B1.
TC2 = Tàrtar format per l' intersecció C2 i la resta de les interseccions.
TEX = Tàrtar exterior. El que està format per A4, B4, C4...

1a iteració

Comencem l' algorisme amb el conjunt S={SA1, SC1, SA3}, i el conjunt R={TA2, TB1, TC2, TEX}. SA3 té 2 Tàrtars vitals (TA2, i TB1). La serp A1 té els mateixos Tàrtars vitals, TA2 i TB1. Finalment, la serp C1 només en té un: TB1, ja que D2 no és llibertat seva, i per tant TC2 no és vital a SC1. Així doncs, com que només te un Tàrtar vital, esborrem SC1 del conjunt S.

TC2 i TB1 estan rodejats per SC1, que ja no forma part de S. Per tant, esborrem del conjunt R TC2 i TB1.

Final de la primera iteració: S={SA1, SA3} R={TA2, TEX}

2a iteració

A la segona iteració, la serp SA3, només té un Tàrtar vital: TA2, ja que TB1 ha estat esborrat de R en l' anterior iteració. El mateix passa amb SA1, ja no té TB1 com a Tàrtar vital. Així que les dues serps són esborrades de S, al no tindre dos Tàrtars vitals com a mínim. Com que TA2 està rodejat per SA1 i SA3 que ja no estan a S, esborrem el Tàrtar de R. El mateix passa amb TEX que està rodejat per SA3.

Final de la segona iteració: S={} R={}

Com podem veure, S ha quedat buida. Això vol dir que cap serp està incondicionalment viva, i per tant poden ser capturades amb un nombre determinat de jugades seguides.

Figura B

En aquest exemple usarem la següent nomenclatura:
SA19 = serp formada per la pedra A19.
SC19 = serp formada per la pedra C19 i la resta de pedres.
SC2 = serp formada per la pedra C2.
SS1 = serp formada per la pedra S1 i la resta de pedres.
SS19 = serp formada per la pedra S19.
TB19 = Tàrtar format per l' intersecció B19
TA18 = Tàrtar format per l' intersecció A18
TC1 = Tàrtar format per C1.
TT1 = Tàrtar format per T1.
TT19 = Tàrtar format per T19
TEX = Tàrtar format per totes les interseccions interiors del tauell.

1a iteració

SA19 té dos Tàrtars vitals: TA18 i TB19. SC19 en té tres: TA18, TB19 i TC1. SS19 per la seva banda té com a vitals els Tàrtars TC1, TT19, i TT1. En canvi, SC2 i SS19 només tenen un Tàrtar vital cadascun. Així que aquestes dues serps s' esborren de S.

En conseqüència, els Tàrtars TEX, TT19, i TC1, són esborrats de R, ja que els tres estan rodejats de serps que no estan a S ja.

Final de la primera iteració: S = {SS1, SC19, SA19} R = {TB19, TA18, TT1}

2a iteració

Com que hem esborrat TT19 i TC1, la serp SS1 només té TT1 com a Tàrtar vital. Això vol dir que l' hem d' esborrar de S.

El resultat d' això és l' eliminació del Tàrtar TT1.

Final de la segona iteració: S={SC19, SA19} R={TB19, TA18}

3a iteració

Les dues serps SC19 i SA19 tenen dos Tàrtars vitals: TB19 i TA18.

Com que cap dels dos Tàrtars té cap pedra d' una serp que no està a S, no n' eliminem cap.

Final de la tercera iteració: S={SC19, SA19} R={TB19, TA18}

Com es pot comprovar, en aquesta iteració no hem esborrat cap element de cap dels dos conjunts. Per tant, l' algorisme ha arribat a la seva fi, deixant com a serps incondicionalment vives, SC19 i SA19. R per la seva banda dona els Tàrtars que formen els ulls d' aquestes serps.

Aquest algorisme va ser ideal per a decidir el final de partida en una simulació del MCTS. D' una manera metòdica, podíem determinar quan un dels dos jugadors havia complert l' objectiu que li pertocava. Cal destacar però, que hi han mètodes basats en heurístiques, per a determinar si un grup és mort. Aquests mètodes tenen l' avantatge que poden determinar la supervivència d' un grup en fases anteriors. És a dir, no ha d' haver-hi formació de dos ulls estricta. El problema d' aquests mètodes és que per casos complexes no sempre acerten. L' avantatge és que per a casos no tan complexes, poden determinar molt abans el final de partida, en la fase de simulació del MTCS.

Arribats a aquest punt, fins ara s' ha presentat l' implementació més bàsica de l' IA. A partir d' ara es presentaran algunes optimitzacions que es van plantejar per a millorar l' IA tant en els resultats de l' algorisme com en el rendiment computacional del mateix.

2.6.9 Escollint els punts vitals del Tsumego

Els resultats fins a aquest punt del projecte no van ser molt esperançadors. D' un conjunt de test de 20 problemes, Gorgon en sabia resoldre correctament només 3, i segurament per casualitat. Qué estava passant? El problema era que les simulacions es feien sobre interseccions completament irrellevants per el problema. Suposem el següent exemple:

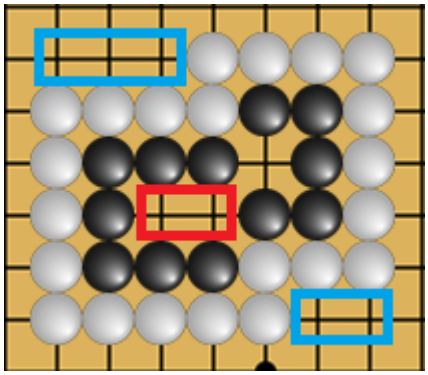


Figura 2.16: Exemple de Tsumego

Aquest Tsumego ja s' ha comentat anteriorment. Qualsevol de les interseccions marcades en vermell porten a la victòria per a les negres. Ara bé, que passa si en una de les simulacions les negres col·loquen sobre una de les interseccions marcades en blau? Es fàcil veure que són jugades completament inútils i sense cap tipus de valor per al problema.

Això suposava un problema greu, ja que aquestes simulacions feien que les blanques arribessin a moltes més jugades guanyadores. Mentre les negres jugaven a interseccions sense cap mena de sentit, les blanques anaven fent jugades amb més sentit, i creant fulles de l' arbre en les que sortien victorioses sense cap tipus de problema. Això suposava que la jugada guanyadora podia inclús estar fora del Tsumego en sí.

Per donar solució a això, es va haver de buscar una manera de que el MCTS només fes simulacions sobre les interseccions més rellevants del Tsumego. És a dir, els punts vitals del problema. Hi ha molts estudis per a determinar aquests punts: heurístiques, tècniques de visió per computador, funcions d' avaluació...

Desgraciadament, no es comptava amb massa temps per fer un estudi sobre aquests mètodes, així que es va optar per una solució més fàcil: agafar com a punts vitals, les interseccions buides dels Tàrtars que no fossin el Tàrtar exterior (el que ocupa la major part del taulell, i per tant, no és rellevant per al problema). A continuació es veuen alguns exemples de punts vitals (marcats en vermell):

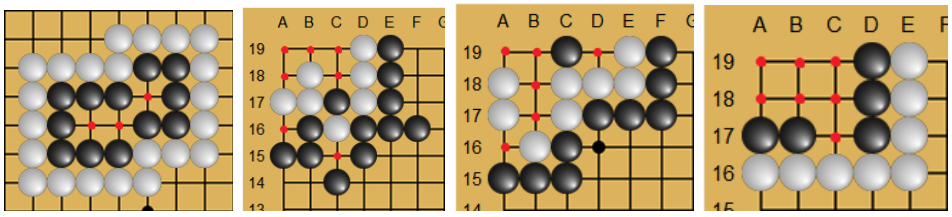


Figura 2.17: Alguns Tsumegos i els seus punts vitals segons Gorgon

Com es pot intuir, aquest punt és el que va suposar la limitació més forta per al projecte. Per a aplicar aquesta lògica, es necessita que el problema tingui com a mínim 2 Tàrtars, i que la serp que els contingui, encercli el Tsumego en sí. Això va limitar molt els Tsumegos que podia resoldre Gorgon. Els següents problemes Gorgon no era capaç de resoldre'ls correctament, ja que només hi ha un Tàrtar en tot el taulell, per a qualsevol dels dos jugadors:

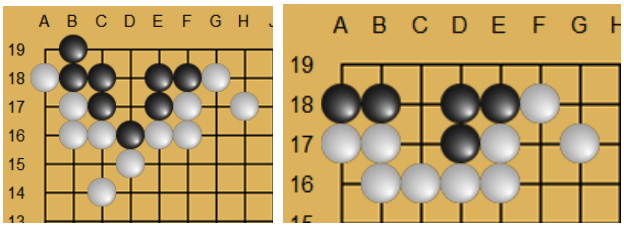


Figura 2.18: Exemples que Gorgon no es capaç de resoldre

No obstant, es va deixar com a treball futur la cerca i estudi d' un mètode més eficient per trobar aquests punts vitals.

2.6.10 MCTS paral·lelitzable

La següent optimització que es va portar a terme, va ser la paral·lelització d' una part del MCTS. L' MTCS és un algorisme altament paral·lalitzable, i de fet, hi ha tres tipus de paral·lelització. En aquest projecte es va implementar el tipus més senzill.

El mètode s' anomena Paral·lelització per fulla. Consisteix en que per cada iteració de l' algorisme, es facin múltiples fases de simulació a la vegada. La fase de selecció es fa de manera normal, i quan s' arriba a una fulla no explorada, es fan múltiples simulacions amb múltiples fils d' execució. La següent figura mostra aquest mètode d' una forma gràfica.

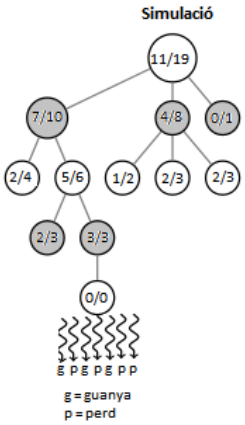


Figura 2.19: MCTS Paral·lelitzable

Com es pot intuir, la fase de propagació també inclou aquesta paral·lelització. Per cada simulació feta en els N fils d' execució, es genera una fase de propagació fins al node arrel, guardant els resultats d' aquella simulació per tot l' arbre. D' aquesta manera, amb una iteració del MCTS, es generen N simulacions.

Com s' ha comentat aquest és el mètode més fàcil d' implementar, ja que els fils d' execució no necessiten gairebé comunicació entre ells. El principal problema d' aquest mètode és que fins que l' última simulació no ha acabat, no es pot començar una altre iteració de l' algorisme. Però degut a que un Tsumego arriba a una situació de final de partida relativament ràpid, l' impacte d' aquest problemàtica va ser petit.

Es va utilitzar OpenMP per a dur a terme aquesta optimització. Es va plantejar d' utilitzar CUDA, i fer una paral·lelització massiva per a GPU. Però CUDA té unes certes limitacions respecte a les estructures que es poden utilitzar dins dels Kernels. Com que aquesta optimització no es va planejar fins a fases avançades del projecte, molt del codi implementat no complia aquestes regles. Per tant, i degut a la falta de planificació, es va decidir que no era viable fer una paral·lelització a nivell de GPU amb el temps del que es disposava.

Els resultats d' aquesta optimització no obstant, van ser molt bons. Es va passar de que l' IA trigués uns 20 segons de mitja, a que trigués 2 segons de mitja.

Existeixen dos mètodes més de paral·lelització per l' MCTS. Són bastant més complicats, però teòricament donen millors resultats. No es van plantejar la seva implementació a Gorgon degut a la falta de temps, però es deixa com a treball futur intentar explorar aquestes dues alternatives, i reorganitzar el codi per a poder utilitzar la GPU d' una manera relativament senzilla.

2.6.11 Simulacions completament aleatòries?

Finalment, l' última optimització que es va fer, deriva del fet que l' MCTS bàsic utilitzi simulacions completament aleatòries. Suposem el cas estudiat anteriorment:

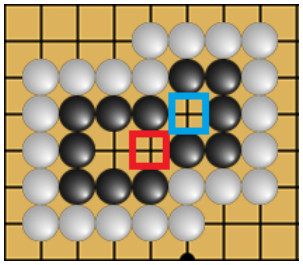


Figura 2.20: Un Tsumego recurrent

Com es pot veure, les negres obtenen dos ulls simplement jugant al quadrat vermell. Però qué passa si en la fase de simulació es col·loca una pedra negra al quadrat blau? Recordem que les jugades són completament aleatòries. El resultat d' això seria que les negres es quedarien sense la possibilitat de fer els dos ulls, i les blanques acabarien guanyant la simulació. Degut a això, és possible que la jugada arrel de la simulació, anés guanyant cada vegada més força, i acabés sent el sub-arbre de la jugada guanyadora, i retornada per el MCTS. Però realment, no té molt de sentit que una de les jugades de les negres sigui tancar-se un dels ulls potencials. Si realment el jugador acaba jugant així, el propi algorisme ja trobaria la resposta adequada, però aquesta jugada no es tindria que donar mai en la fase de simulació, per cap dels dos jugadors.

Es per això que es va implementar la regla de que en la fase de simulació, no es dongúes una jugada en la que el propi jugador bloqueja un dels propis ulls potencials. Entenem per ulls potencials les interseccions sense llibertats (és a dir, una intersecció aïllada), que són punts clau per formar dos ulls. Així, per exemple, el Tàrtar del quadrat vermell de la figura anterior no seria un ull potencial a considerar fins que tingués una sola intersecció.

Per a determinar si una intersecció és ull potencial el que es va fer és comprobar que l' intersecció estigués completament rodejada de pedres aliades, que NO estiguessin en atari. Això es tradueix a les interseccions que formen part d'un territori aliat, el qual tingui alguna llibertat, i no sigui l'intersecció que estem mirant.

És un algorisme relativament poc precís. Hi ha bastants casos que aquest algorisme no contempla i serien ulls potencials. Però tenint en compte que per a cada pedra en la fase de simulació s' ha de comprovar si és ull potencial o no, es va pensar que era millor un algorisme no molt pesat per tal de permetre a l' MCTS fer més simulacions amb menys temps. Una de les millores potencials del projecte és buscar un algorisme alternatiu més precís, però igual de ràpid.

Capítol 3

Resultats

Per a validar la qualitat de l' IA implementada, es va utilitzar un conjunt de test de 20 Tsumegos. Aquests Tsumegos complien la condició de tindre un Tàrtar rodejant al problema en sí, per a poder trobar els punts vitals. Es van categoritzar en 3 categories. Entre els 20 Tsumegos hi havia 15 de dificultat fàcil, 4 de dificultat mitjana, i 1 de dificultat molt elevada.

El bon funcionament d' un problema es va determinar seguint dos criteris diferents:

- IAN: Jugant l' IA amb el color negre, sabia trobar el camí crític per a que aquestes guanyin. És a dir, sabia resoldre el problema.
- IAB: Jugant amb el color blanc, jugava de manera lògica, ajudant al jugador a comprendre millor el problema.

Seguin aquests dos criteris, es van fer diverses proves amb diferents mides de simulació i de valors del paràmetre empíric per al UCB.

Amb un tamany de simulació fixe a 500 partides aleatòries per moviment

UCB	IAN	IAB
0	50%	55%
0.25	65%	70%
0.50	80%	80%
0.75	55%	60%
1	10%	40%

Amb la constant UCB fixe a un valor de 0.50

Nº partides	IAN	IAB	Càlcul seqüencial	Càlcul OPENMP
50	10%	20%	3 segons	menys d'un segon
500	75%	80%	8 segons	menys d'un segon
1000	65%	70%	18 segons	1 segon
2000	60%	70%	39 segons	2 segons
5000	55%	65%	1.5 minuts	4 segons
10000	55%	60%	3 minuts	7 segons
20000	45%	55%	més de 5 minuts	10 segons

Com es pot comprovar, l' augment de simulacions no va provocar la millora esperada. Això es així degut a la falta de de precisió a l' hora de detectar ulls durant les simulacions. Això provoca que com més simulacions es facin, hi hagi més possibilitats de que un dels jugadors es tanqui un dels seus ulls potencials, i l' altre jugador comenci a trobar victòries on no hi haurien d' haver-hi.

Una millora potencial és trobar una manera alternativa de trobar aquests ulls potencials, però sense que suposi un overhead massa gran en temps de còmput.

Els millors resultats es van trobar amb una constant de UCB a 0.50, i 500 simulacions per moviment.

Capítol 4

Conclusions i treball futur

S' ha implementat una intel·ligència artificial per resoldre Tsumegos, amb una tassa d' èxit relativament bona. No obstant, hi ha moltes vies futures per intentar millorar l' aplicació. Tant des de el punt de vista de Gorgon i la seva IA, com per a l' interfície GoGUI.

4.1 Conclusions

Gorgon es va pensar com un projecte de final de carrera bastant ambiciós. El resultat obtingut va ser satisfactori, i es van poder extreure les següents conclusions:

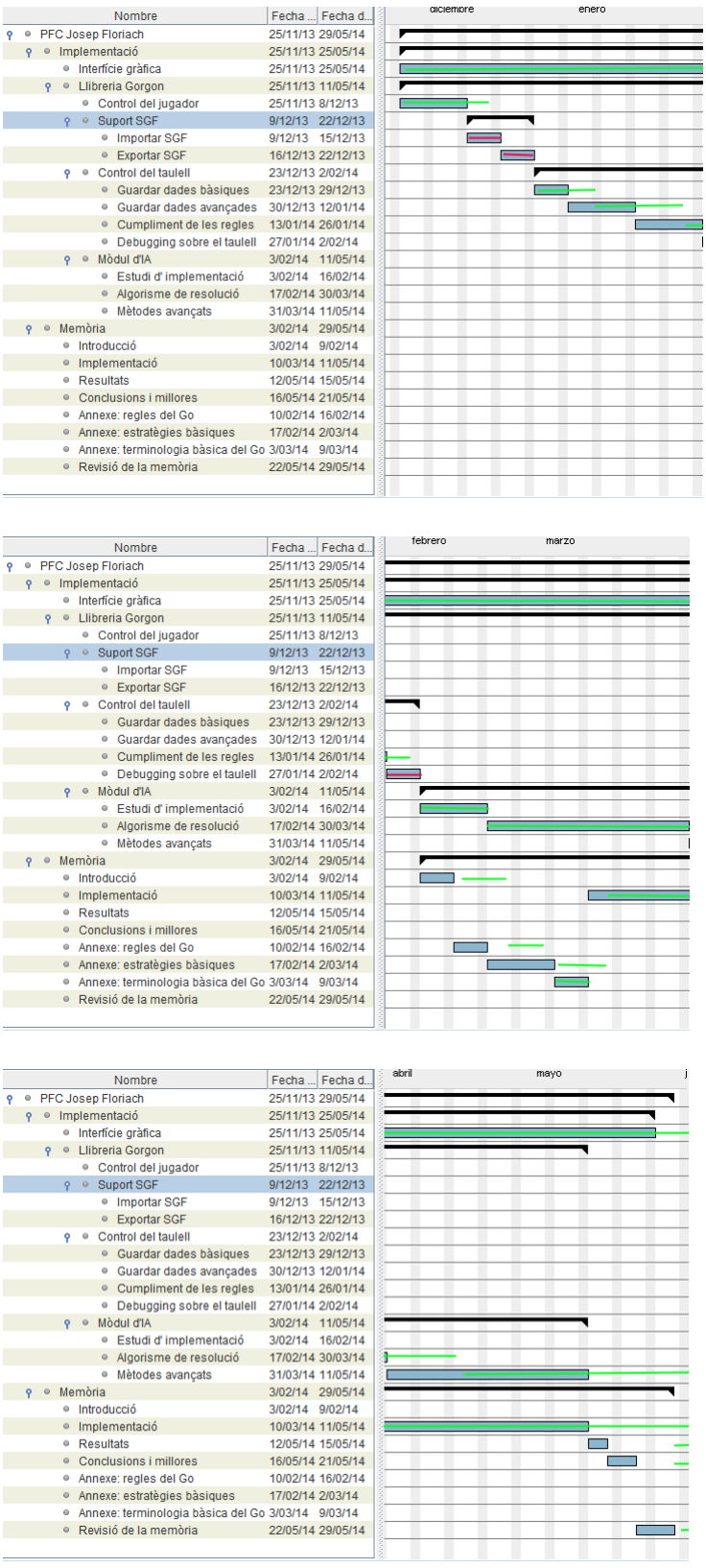
- L' implementació d' una IA, com ja es suposava és una tasca complexa. Però el que s' ha aprés amb aquest projecte, és que no es tan sols des de el punt de vista tècnic, sinó que també des d'un punt de vista de planificació de projecte. El fet de no planificar algunes fases del projecte, va fer que no es pogués implementar tot el que s' hagués volgut. Un clar exemple d' això és el fet de no poder paral·lelitzar amb CUDA degut a les seves limitacions amb les estructures a tractar.

- L' MCTS bàsic és un algorisme prometedor sobre paper, però a la realitat fan falta bastantes ampliacions per a poder funcionar correctament sobre un projecte real.

- L' ús d' heurístiques hagués millorat molt el rati de victòries per part de la màquina. En aquest projecte l' única heurística implementada ha estat el fet de que l' MCTS no simulés sobre un ull potencial. Però existeixen moltes heurístiques associades a un coneixement més profund del joc, que no s' han pogut portar a terme, tant com per falta de temps, com per falta de coneixements sobre el joc del Go.

- La paral·lelització de l' MCTS ha estat un punt vital per a aconseguir un bon rendiment. Es va passar de més de 20 segons per moviment, a 1 segon. Tot això amb 500 simulacions per jugada.

- A l' hora de planificar un projecte de tamany mitjà, és molt important ser relativament pessimista a l' hora de calcular la duració de les tasques. En aquest projecte es va dur a terme un càlcul massa optimista i han hagut moltes coses que no es van poder dur a terme. A continuació es posa el diagrama de Gantt planificat inicialment, i la comparativa amb el que van durar realment les tasques. Les línees vermelles simbolitzen les tasques planificades que no es van poder dur a terme per falta de temps. Les verdes, el temps real que es va necessitar per completar aquella tasca.



Nombre	Fecha ...	Fecha d...
PFC Josep Floriach	25/11/13	29/05/14
Implementació	25/11/13	25/05/14
Interfície gràfica	25/11/13	25/05/14
Llibreria Gorgon	25/11/13	11/05/14
Control del jugador	25/11/13	8/12/13
Suport SGF	9/12/13	22/12/13
Importar SGF	9/12/13	15/12/13
Exportar SGF	16/12/13	22/12/13
Control del taulell	23/12/13	2/02/14
Guardar dades bàsiques	23/12/13	29/12/13
Guardar dades avançades	30/12/13	12/01/14
Cumpliment de les regles	13/01/14	26/01/14
Debugging sobre el taulell	27/01/14	2/02/14
Mòdul d'IA	3/02/14	11/05/14
Estudi d' implementació	3/02/14	16/02/14
Algorisme de resolució	17/02/14	30/03/14
Mètodes avançats	31/03/14	11/05/14
Memòria	3/02/14	29/05/14
Introducció	3/02/14	9/02/14
Implementació	10/03/14	11/05/14
Resultats	12/05/14	15/05/14
Conclusions i millores	16/05/14	21/05/14
Annexe: regles del Go	10/02/14	16/02/14
Annexe: estratègies bàsiques	17/02/14	2/03/14
Annexe: terminologia bàsica del Go	3/03/14	9/03/14
Revisió de la memòria	22/05/14	29/05/14

Figura 4.1: Tasques planificades vs tasques realitzades

No obstant, a pesar de totes les dificultats trobades, es creu que el projecte va ser un èxit, i va aconseguir el seu propòsit, que era comprovar de primera mà el repte que suposa fer una aplicació d' aquest tipus.

4.2 Treball futur

Les possibles millores s' han dividit en les que es poden dur a terme a GoGUI i en les que es poden dur a terme a Gorgon.

4.2.1 GoGUI

GoGUI es va idear com l' aplicació final amb la que l' usuari interactuaria. Però per falta de temps es va quedar en un estat relativament pobre. Per començar, els primers punts a atacar són els que es van planificar i no es van poder dur a terme. Aquests són:

- Poder modificar els paràmetres de l' IA.
- Grabar partides i reproduir-les.

A part, a mida que s' anava treballant amb l' aplicació es van anar veient millores potencials que en un principi i s' havien plantejat, i al final del projecte semblen coses indispensables per a la bona interacció de l' usuari amb l' aplicació. Algunes d' aquestes són:

- Un mode d' edició de Tsumegos. Per crear un nou problema a GoGUI, s' havia d' anar alternant jugades i passant el torn si feia falta, fins a arribar al estat del taulell desitjat. La primera millora potencial i no planificada, és que per crear un Tsumego es pugui posar totes les pedres seguides d' un color, després de l' altre, i guardar. És un mètode molt més còmode, ràpid i pràctic.
- Poder desfer la jugada. A mida que s' anaven fent proves, hi havia vegades que per error es posava una pedra a un lloc on no es pretenia. Una funcionalitat casi obligatòria que es va pensar després, és el fet de poder desfer l' última jugada feta i tornar a l' estat anterior del taulell. Això sembla una tasca trivial, però rés més allunyat. Tal com funciona Gorgon, s' hauria de pensar un algorisme per desfer l' actualització de serps.

4.2.2 Gorgon

Gorgon per la seva banda, té moltes millores potencials i ampliacions. La majoria d' elles orientades al fet de millorar els resultats de l' MCTS.

- La primera millora seria trobar un mètode alternatiu per poder trobar els punts vitals d' un Tsumego. Això permetria a Gorgon resoldre Tsumegos de més complexitat i varietat.
- Trobar un mètode alternatiu per a la detecció d' ulls en la fase de selecció de l' MCTS. Això provocaria segurament un augment elevat dels Tsumegos resolts correctament.
- Millorar el codi implementat fins ara, per optimitzar el seu rendiment, i permetre a l' MCTS fer més simulacions. A part, organitzar el codi en estructures que permetin paral·lelitzar amb CUDA.
- Estudiar més en profunditat el joc del Go, per a poder aplicar heurístiques al propi MCTS.
- Estudiar ampliacions diverses del mètode de Monte Carlo. Existeixen diverses variants. Una d' elles és RAVE (Rapid Action Value Estimation). El RAVE té com a objectiu que l' arbre comenci a trobar els moviments més prometedors abans.
- Ampliar la resolució de Tsumegos a la resolució d' una partida complerta. Això és un repte encara molt llunyà, però hi ha tècniques en estudis avançats. Un exemple d' ells és fer us de Xarxes neurals convolucionades. La més prometedora no obstant, segueix sent aplicar el mètode de Monte-Carlo a taulells de 19x19. El problema és que es necessita una capacitat de còmput en entorns distribuïts per tal de poder fer simulacions massives en poc temps. I a part, per poder jugar una partida complerta entren en joc heurístiques i bases de dades amb coneixement expert sobre el joc.

Apèndix A

Regles del Go

A.1 Introducció al Go

El Go es juga en un taulell de 19x19 interseccions. A cada torn, els jugadors van col·locant pedres del seu color en una intersecció buida qualsevol del taulell.

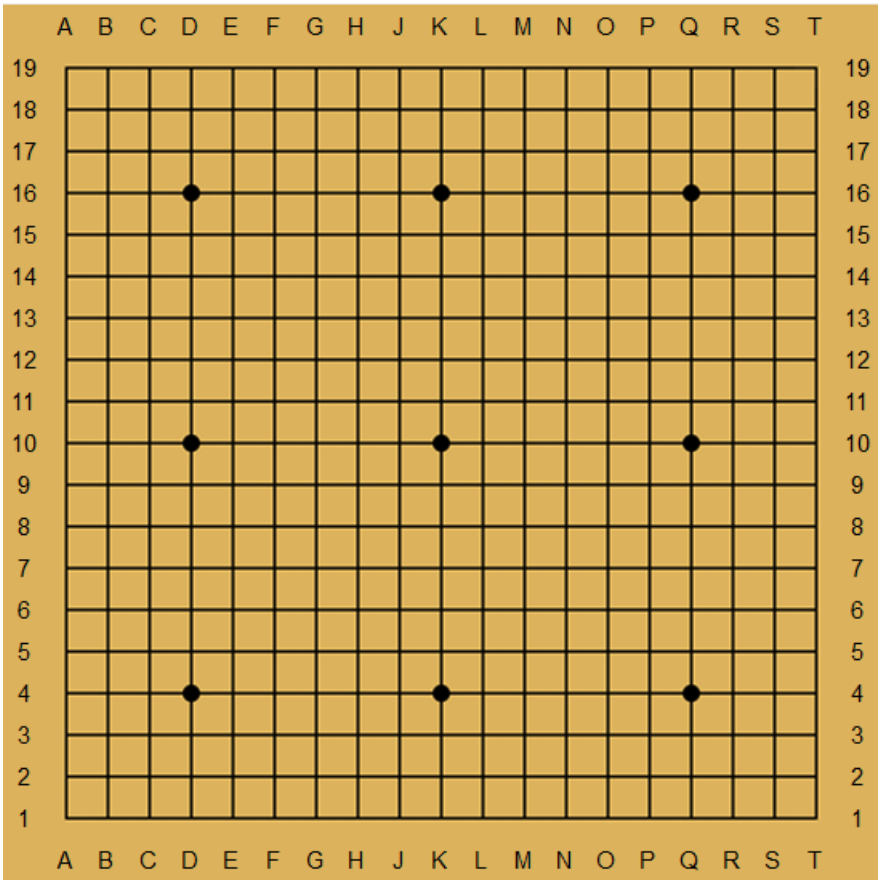


Figura A.1: Taulell de Go

Les normes que defineixen el joc són dues: la regla de les llibertats, i la regla del KO.

A.2 Regla de les llibertats

Cada intersecció està formada per un conjunt de línies que surten del propi encreuament. Aquestes línies s'anomenen llibertats. Les cantonades del taulell només en tenen 2, i els extrems superior, inferior, esquerre i dreta, en tenen 3. La resta d' interseccions en tenen 4. Si una pedra es queda sense llibertats, es diu que és capturada. Es retira immediatament del taulell i passa a formar part de les pedres capturades de l' adversari. Aquestes pedres suposaran un avantatge per a l' adversari, al final del joc. La següent figura mostra el cas més basic possible de captura.

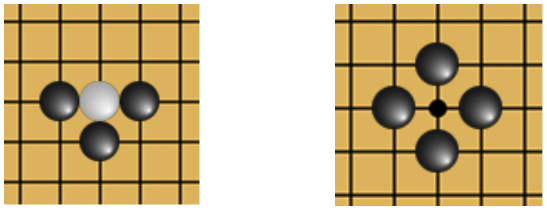


Figura A.2: El cas més bàsic possible capturant una pedra

Aquesta norma s' aplica també a conjunts de pedres. Si un conjunt de pedres es queda sense cap llibertat (no hi ha cap llibertat en cap pedra que forma el conjunt), tot el conjunt és capturat.

Quan es col·loca una pedra que amenaça a una pedra o grup de pedres de l'equip contrari, deixant-les només amb una sola llibertat, és diu que s' ha fet "Atari". No obstant, es una pura qüestió de terminologia. El jugador no està obligat a informar a l' adversari d' aquesta situació. I de fet, no es recomana, per raons obvies. A continuació es veuen alguns exemples de grups de pedres blanques en Atari, i la seva captura per part de les negres.

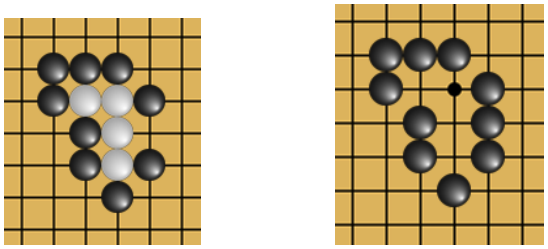


Figura A.3: Exemple 1 de captura múltiple

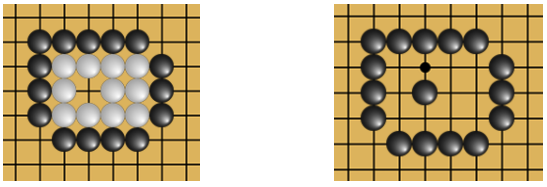


Figura A.4: Exemple 2 de captura múltiple

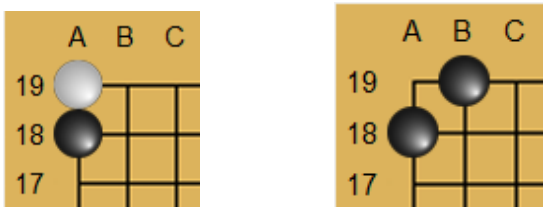


Figura A.5: Exemple 3 de captura múltiple

Figura A.6: Exemple 4 de captura múltiple

A.3 Regla del KO

La norma del KO estableix que l' estat del taulell no es pot repetir en 2 jugades consecutives.

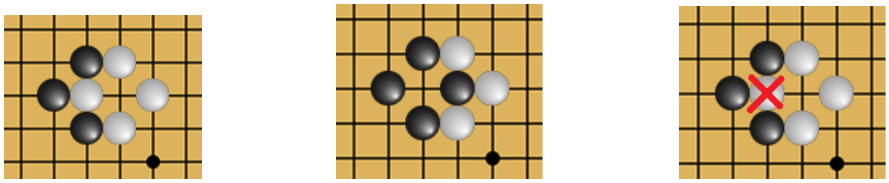


Figura A.7: KO

En les imatges anteriors, l' adversari captura una pedra. Al capturar-la, deixa exposada la pròpia pedra que acaba de col·locar (Atari). Però si el

jugador defensor, la tornés a capturar, el taulell tornaria a l' estat del torn anterior. Això podria inicialitzar un bucle infinit per part dels dos jugadors. La regla del KO impedeix aquesta situació, no permetent al segon jugador col·locar una pedra en la posició en qüestió, fins al seu següent torn.

A.4 Objectiu del joc

L' objectiu del joc consisteix en ser el jugador que controli més territori al final de la partida. Els territoris són interseccions buides rodejades per un dels dos jugadors. Cal tindre en compte que la captura de pedres és només un medi, no una meta. La captura de pedres ens donaran una puntuació extra al final de partida, però ni molt menys suficient per guanyar, si l' enemic controla més territori que nosaltres.

A.5 Final de partida

La partida acaba quan els dos jugadors es posen d' acord. Això pot semblar molt desconcertant per a un nou jugador. Però si es pensa bé, té sentit. Arribarà un moment de la partida, en el que les úniques opcions que tindrà un jugador seran dues:

- Col·locar una pedra en un territori propi. Això suposa la pèrdua del propi territori, obtenint menys puntuació al final de partida
- Col·locar una pedra en territori enemic per intentar reduir-lo. Això és completament contraproduent a vegades. Si l' adversari té un mínim de nivell, pot impedir molt fàcilment que el jugador disminueixi el seu territori. Això comporta el sacrifici de moltes pedres, i un gran avantatge per a l' adversari.

Per tant, per a que s' acabi la partida, els dos jugadors han de passar el seu torn. Llavors es diu que s' han posat d' acord, i la partida acaba. En aquest moment es comptabilitzen els territoris dels dos jugadors. Cada intersecció d' un territori es compta com un punt per al jugador en qüestió, i a més, qualsevol pedra o conjunt de pedres enemigues, aïllades dins d' aquest territori, queden capturades automàticament. Aquesta fase del joc és potser la que més confusió genera per als nous jugadors, així que seguidament es mostren alguns exemples de territoris controlats i territoris no controlats.

A.6 Handicaps

Els handicaps són una manera de donar avantatge als jugadors menys experimentats. Consisteix en col·locar un nombre de pedres al principi de la partida per tal de començar amb un mínim d' influència sobre el taulell (veure annex 3, glossari en el Go). Els handicaps es col·loquen en les interseccions de tipus "Hoshi" o en la de "Tenggen".

A.7 Fluxe d' una partida de Go

En una partida de Go hi ha tres fases ben diferenciades entre sí, cadascuna amb un objectiu diferent, però totes apuntant a la victòria de la partida.

- **Apertura**: és la fase inicial de la partida, on cada jugador intenta definir el seu territori inicial i guanyar influència sobre el taulell. Existeixen diverses estratègies d' apertura anomenades "Fuseki", cadascuna orientada a un joc més agressiu o defensiu.
- **Middle-game**: és la fase més llarga del joc i està formada per invasions i atacs per parts dels dos jugadors, per tal de restar territori a l' enemic, i sumar el propi.
- **End-game**: en aquesta fase els jugadors intenten consolidar els territoris conquistats en les dues fases anteriors.

És important destacar que una partida de Go pot quedar sentenciada en la primera fase del joc. Si un jugador no ha sigut capaç d' expandir-se lo suficient a lo llarg del taulell, en la primera fase, és pràcticament impossible fer-ho en les següents. En una partida de Go no es pot ser ni massa conservador, ni massa ambiciós. Si es juga massa defensiu, l' adversari conquistarà la major part del taulell. Si s' és massa ambiciós, l' adversari no tindrà problemes en frenar el nostre atac.

Apèndix B

Dos ulls en el Go

La formació de dos ulls és un dels pilars centrals del projecte, i el concepte més important en el que es basa l' estratègia de qualsevol partida de Go. Ara bé, qué vol dir exactament la formació de dos ulls, i perquè és tan important? El que un grup de pedres tingui una formació de dos ulls, vol dir que té assegurades les llibertats internes, i per tant, les pedres que les rodegen no podran ser capturades de cap de les maneres.

A continuació es mostren exemples de formacions de dos ulls per part de les negres.

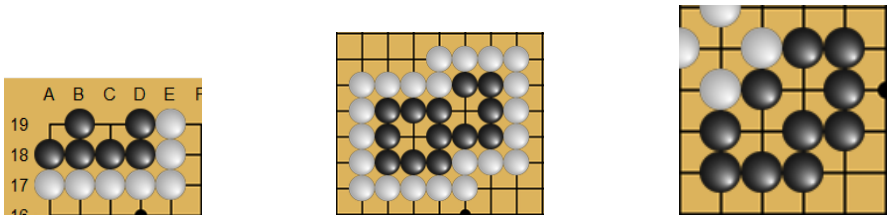


Figura B.1: Exemples de dos ulls.

Com es pot comprovar, la força d' aquestes figures està directament relacionada amb la regla de les llibertats. Com que a un jugador no li està permès el suïcidi, no podrà jugar mai en una zona en la que no pugui capturar cap pedra enemiga, i a més tingui 0 llibertats.

És important destacar que un ull no té perquè estar format de una intersecció només. Pot ser composat per dues o més. Així doncs, encara que el jugador enemic pugui col·locar pedres, si el grup realment té aquesta formació seran pedres virtualment mortes. Això es així perquè arribarà un moment en que el jugador defensor podrà capturar-les sense posar en risc cap de les seves pedres. El següent exemple mostra aquesta situació.

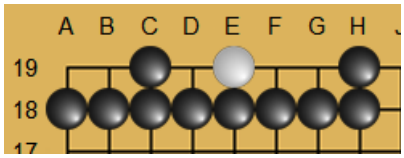


Figura B.2: Dos ulls múltiples

Un error molt comú en els jugadors principiants és confondre una formació de dos ulls, amb una formació que realment està potencialment amenaçada. Aquest tipus de formacions s' anomenen falsos ulls, i són interseccions on l' oponent pot jugar a les llibertats internes, capturant les pedres que formen els falsos ulls.

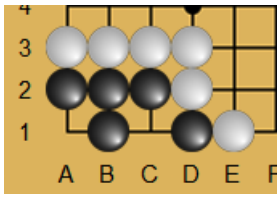


Figura B.3: Exemple d' ull fals

En aquest exemple, un jugador principiant podria pensar que C1 forma dos ulls juntament amb A1. Però les blanques tenen rodejada D1, amb la qual cosa és fàcil veure que poden capturar el grup sencer jugant a C1 per capturar D1, i seguidament a A1 (les negres no poden tornar a jugar a D1 ja que trencaria la regla del KO).

En resum, i com es pot veure, si un jugador domina la formació de dos ulls, domina gran part del joc. La principal dificultat d' això, és que hi ha centenars de patrons i formes que tenen com a objectiu final aquesta formació de dos ulls, i cadascuna d' elles té les seves debilitats i els seus punts forts. Una de les habilitats de un bon jugador de Go, és saber quina d' aquestes formes jugar a cada moment, i amb quin objectiu.

Bibliografia

[1] Article de Wikipedia Computer Go *Computer Go*. en.wikipedia.org

[2] Article de Wikipedia sobre Tsumegos *Life and death*. http://en.wikipedia.org

[3] Article sobre els algorismes més utilitzats en el Computer Go *Computer Go Algorithms*. http://senseis.xmp.net/?ComputerGoAlgorithms

[4] Article sobre l'algorisme de Benson *Benson's Definition of Unconditional Life*. http://senseis.xmp.net/?BensonsDefinitionOfUnconditionalLife

[5] Article de Wikipedia sobre els mètodes de Monte Carlo *Monte Carlo method*. http://en.wikipedia.org

[6] Article de Wikipedia sobre Monte Carlo Tree search *Monte Carlo tree search*. http://en.wikipedia.org

[7] Article amb un exemple del MCTS *What is MCTS?*. http://mcts.ai/about/index.html

[8] Article amb un exemple del MCTS *Monte Carlo Tree Search*. https://sites.google.com/a/lclark.edu/drake/courses/drakepedia/monte-carlo-tree-search

[9] Article de Wikipedia sobre l'algorisme de dos passos *Connected-component labeling*. http://en.wikipedia.org

[10] Web i documentació oficial d'OpenMP *OpenMP*. http://openmp.org/wp/

[11] Web i documentació oficial de CUDA *CUDA toolkit documentation*. http://docs.nvidia.com/cuda

[12] Web i documentació oficial de QT *QT Project*. http://qt-project.org/doc/

[13] Web i documentació oficial del projecte GNU Go *GNU Go*. https://www.gnu.org/software/gnugo/

[14] Web d'ajuda a la programació *stackoverflow*. http://stackoverflow.com/

Resum

S'ha implementat una intel·ligència artificial relativament senzilla per a resoldre Tsumegos en el joc del Go. Per a això, s'ha fet un estudi per a saber quins eren els algorismes més interessants, tenint en compte sempre un equilibri entre complexitat i resultats, degut al temps del que es dispossa.

Com a conseqüència, també s'ha implementat una interfície gràfica per a poder jugar partides complertes entre dos jugadors, i poder veure els resultats de l'intel·ligència artificial.

Resumen

Se ha implementado una inteligencia artificial relativamente sencilla para resolver Tsumegos en el juego del Go. Para ello, se ha llevado a cabo un estudio para saber cuales eran los algoritmos más interesantes, teniendo en cuenta siempre un equilibrio entre complejidad y resultados, debido al tiempo del que se dispone.

Como consecuencia, se ha implementado también una interfície gráfica para poder jugar partidas completas entre dos jugadores, y poder ver los resultados de la inteligencia artificial.

Abstract

We have implemented a relatively simple artificial intelligence to solve Tsumegos in Go Game. To this end, it has been conducted a study to find out what was the most interesting algorithms, always considering a balance between complexity and performance due to the available time.

As a result, it has also implemented a graphical interface to play complete games between two players, and see the results of the artificial intelligence.