Desenvolupament d'un joc 2D isomètric d'estratègia per torns.

Àngel Fernández Ibáñez

Resum—Tot i que a Europa no està àmpliament estès, el Go és un dels jocs de taulell més popular a països orientals com la Xina, Corea o el Japó. Aquest projecte té com a objectiu desenvolupar una variant del Go convertint-lo en una illa hexagonal que s'ha de conquerir posant-hi edificis per torns. D'aquesta manera es converteix en un joc més senzill i amè, obrint-lo a més tipus de públic i portant-lo a les plataformes d'avui en dia com els dispositius mòbils. És descriurà tot el procés de creació: com treballar amb taulells hexagonals, com desenvolupar la lògica del joc, quins processos s'han de seguir per poder controlar la lògica del joc, etc. Durant el desenvolupament d'aquest, es farà en tot moment referencia al Go original per entendre el que s'ha fet i les raons de cada decisió. També s'explicarà com desenvolupar una intel·ligència artificial basada en l'algorisme negamax amb poda alfa-beta per donar als jugadors la possibilitat de jugar contra la màquina.

Paraules clau—Go,joc,isomètric,hexagonal,estratègia,ia,pvp,negamax,poda alfa-beta,taules de transposició

Abstract—Despite not being popular in Europe, Go is one of the most popular turn-based board games in the Asian countries like China, Korea or Japan. This project aims to develop a variant of the Go game where the player aims to conquer an hexagonal island by raising buildings. This simplifies the traditional Go game making it simpler and pleasant to play, opening it up to more public and porting it to new platforms such as mobile devices. This project will describe the whole creation process: Working with hexagonal tiles, developing the game logic, what processes must be followed in order to control the game logic. etc During its development it'll reference original Go game soon enough in order to make clear the decisions the were taken and the reasons behind them. It will also explain how to develop an IA using a negamax algorithm with an alpha-beta pruning to grant users the ability to play against a computer.

Index Terms—Go,game,isometric,hexagonal,strategy,ai,pvp,negamax,alfa-beta pruning,transposition tables

1 Introducció

Aquest projecte és basa en la idea i regles del Go [1] per crear un joc de conquesta d'una illa, creant una variant del joc original. El primer i més important canvi és que es treballarà sobre un taulell hexagonal en comptes de quadrat, passant de 4 connexions (veïns) a 6 connexions. D'aquesta manera es diferencia el joc i implica noves estratègies per jugar, evitant així que un jugador del Go tradicional pugui fer servir les seves estratègies sense fer cap rep replantejament.

Aquest projecte també inclou una aproximació a les intel·ligències artificials (en endavant, IA) del Go, a través d'un algorisme de poda alfa-beta.

El Go és un joc oriental mil·lenari, originat a l'antiga Xina, i requereix un alt nivell d'estratègia. Per entendre el propòsit d'aquest Treball de Final de Grau (en endavant, TFG), primer cal entendre el Go i les seves regles.
El Go és un joc de territori, amb un taulell de 19x19 línies que pot ser considerat el territori pel que 2 jugadors han de competir, col·locant en ell peces blanques o negres per tal de reclamar-lo. Els jugadors posen una peça per torn, alternativament, que no tornarà a ser moguda. No obstant, la peça pot ser capturada i eliminada del joc si és completament rodejada per peces de l'enemic.

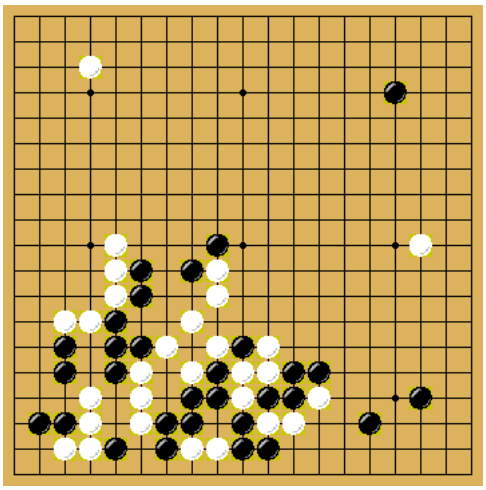

Figura 1. Exemple d'una partida de Go iniciada. Font: Wikipedia

La partida de Go es desenvolupa fins que els dos jugadors estan d'acord en que la puntuació no canviarà més (normalment això es dona quan els dos jugadors passen el torn), moment en el que es fa el recompte final de punts segons el territori i les fitxes de cada jugador (pot variar segons la puntuació xinesa o japonesa), i es decideix el guanyador [1] [2]. En el Annex A es poden veure les regles del Go de forma més extensa.

- *E-mail de contacte: AngelCustodio.Fernandez@e-campus.uab.cat*
- *Menció realitzada: Computació*
- *Treball tutoritzat per: Antoni Gurguí (CVC)*

És important també entendre alguns conceptes claus del Go, ja que es farán servir durant el desenvolupament d'aquest informe i són importants per entendre com s'ha resolt tota la implementació [1]:

Casella Una casella del mapa hexagonal. Pot estar buida (0) o pertànyer a un jugador (1 o 2)

Grups Un grup son caselles del mateix jugador col·locades de forma contigua

Vida Un grup de caselles està viu sempre que tingui com a mínim un espai de llibertat, és a dir, una casella al voltant buida. En cas contrari se'l considera mort i és retirat del taulell.

Suïcidi Fer el moviment sobre una casella buida que implicaria que el grup al que pertany queda automàticament sense espais de llibertat, i per tant mort. No es considera suïcidi si en aquesta jugada s'aconsegueix que un grup de caselles del jugador rival mori.

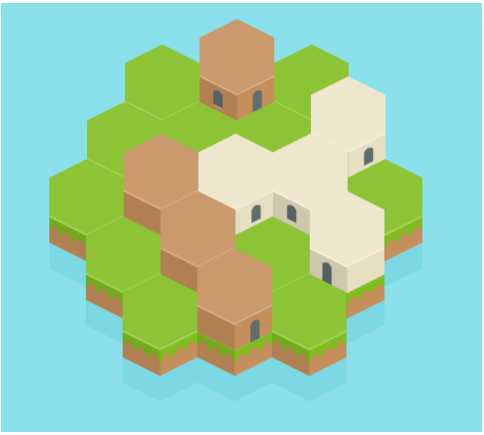

Figura 2. Exemple d'una partida iniciada.

A mode de resum, el projecte es pot desglossar en una sèrie d'objectius diferenciats:

- Creació de la lògica d'un joc inspirat en el Go, presentat en un taulell hexagonal en comptes de quadrat. Per tant, té 6 llibertats en comptes de 4 per casella.
- Representació d'aquest taulell amb gràfics isomètrics i visualment agradables
- Possibilitat de jugar a aquest joc per torns contra un altre usuari per torns, en el mateix dispositiu
- Possibilitat de jugar a aquest joc contra la màquina, en diferents nivells de dificultat: Dummy (moviments aleatoris), i contra una IA basada en l'algorisme de poda alfa beta.
- Exportació d'aquest joc a diferents plataformes (escriptori/mòbil).
- Creació d'una petita pàgina web des d'on es podran descarregar els binaris per les diferents plataformes, i a on es podrà consultar el codi font del joc seguint la metodologia del software lliure.

2 Estat de l'art

Actualment existeixen moltes versions del Go per ordinador o dispositius mòbils, i moltes d'aquestes inclouen un mode Humà contra CPU a través d'una intel·ligència artificial.

Tampoc és la primera vegada que es varia el taulell de Go, s'ha parlat i investigat sobre el tema en diverses ocasions. Concretament sobre un taulell hexagonal amb sis connexions s'ha dit que augmenta la dificultat de capturar i converteix el Go en un joc més estratègic i menys tàctic [6] [7]. Fins i tot existeix una aplicació per Android on es pot jugar a un Go hexagonal de 2 a 6 jugadors (però sense possibilitat de jugar contra la màquina) [8].

Les IA desenvolupades per al Go són unes de les més complexes del món de les IA de jocs de taula [3], fins i tot més que la dels escacs. Des dels anys 60 s'ha desenvolupat molt sobre el tema [4] [5], i s'han desenvolupat molts tipus d'algorismes per intentar fer jugar a un ordinador a un nivell acceptable, però a dia d'avui encara no s'ha aconseguit que sigui competent en taulells oficials a nivells professionals degut a les dificultats que presenta el Go, com la mida del taulell (19x19), la majoria de moviments possibles (no com als escacs que moltes jugades son il·legals), o la complexitat progressiva del joc segons es va desenvolupant la partida.
És per això que aquest projecte no pretén endinsar-se en el complexe món de les IA pel Go, sinó simplement arribar a una aproximació amb algorismes de cerca i poda alfa-beta.

3 Metodologia

Durant el desenvolupament d'aquest projecte s'ha utilitzat una metodologia àgil [11] inspirada en l'Scrum [12], però adaptada a la realitat del TFG.

Encaixant amb el calendari, el projecte primer va ser partit en Èpiques [13], coincidint amb les diferents dates d'entrega de seguiment. Al mateix temps, aquestes Èpiques s'han dividit en tasques i s'han realitzat en petits sprints d'una setmana.

El desenvolupament s'ha realitzat sobre un sistema GNU/Linux [18], en el llenguatge de programació Python [17], que a través del framework Kivy [19] permet compilar el joc realitzat a diverses plataformes.

Per altre banda, per tenir un bon control del software, trobar ràpidament errors i gestionar les diferents versions s'ha fet servir el sistema de control de versions Git, primerament hospedat en un repositori privat, i més tard publicat com a software lliure a Github [35].

Els informes s'han desenvolupat en LaTeX, seguint els estàndards proposats per la UAB [15], basats en els estàndards oficials del IEEE [16]. D'aquesta forma, també han quedat dins el control de versions al ser desenvolupats sobre text pla.

3.1 Mapping i veïns

La gran variació del projecte respecte al Go original és que el taulell és hexagonal, permetent 6 connexions entre caselles en comptes de 4. Però les estructures de dades i memòria no permeten treballar d'aquesta forma, i s'ha de fer un mapping respecte al món dels vectors i les matrius quadrades.

La solució és presenta a la figura 3, on es pot observar com és possible fer aquesta correspondència tot i deixar caselles buides a memòria en dues de les cantonades [21].

Cada casella hexagonal té 6 caselles veïnes, en canvi cada casella en una matriu quadrada en té 8. En el mapping, les

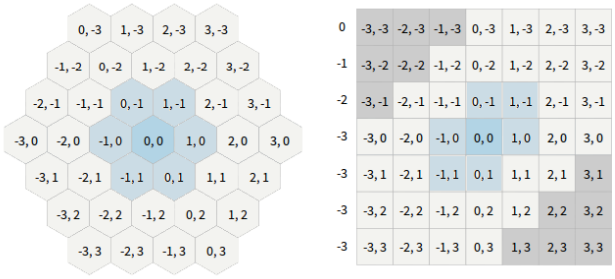

Figura 3. Mapping i veins

posicions [-1,-1] i [1,1] ([x,y]) relatives a una casella [0,0] no són veïnes seves. Es pot veure il·lustrat a la mateixa figura 3, on veiem en blau més claret les caselles veïnes de [0,0].

3.2 Grups i suïcidi

A la figura 4 podem veure un exemple que ajudarà a entendre com funciona la vida i la mort i que implica un suïcidi. Amb els números 1, 2 i 3 estan marcats els diferents grups que hi ha (1 del jugador blanc, 2 i 3 del marró).

El jugador blanc no pot jugar sobre la posició marcada amb una 'a', ja que seria suïcidi (el grup d'una casella que formaria, quedaria completament envoltat i moriria al instant). De la mateixa manera, el jugador marró no pot jugar sobre la posició marcada amb una 'b' ja que el grup 2 quedaria sense espai de llibertat.

En canvi, el jugador blanc sí pot jugar sobre 'b' perquè, tot i que a priori és un suïcidi, en la mateixa jugada està matant el grup 2 (del jugador rival), i per tant es una jugada vàlida i el grup 2 mor.

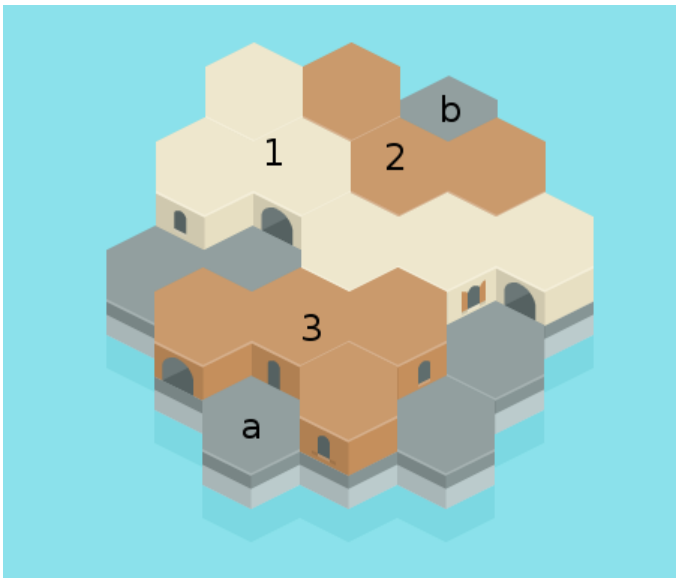

Figura 4. Exemple de grups i suïcidi

3.3 Lògica de torn per a 2 jugadors

A través de tècniques de picking, l'aplicació és capaç de detectar sobre quina casella ha clickat el jugador. Això presenta una dificultat afegida, com es pot veure a la figura 5. Un tile de la illa es realment com es veu en el rectangle extret a fora. Això fa que no es pugui fer servir la eina de picking directa del framework, sinó que s'ha implementat un picking que té en compte només una part del tile. Això es pot veure sobre el tile de la illa, on s'ha ressaltat en blau la part que ha d'activar la jugada en aquella casella i en vermell les parts que no s'han de tenir en compte, ja que pertanyen a altres caselles. Per resoldre això, s'ha marcat com a polígon la part hexagonal superior del tile i a través de tècniques de ray casting es pot determinar si el punt on ha tocat el jugador està dins el polígon [20].

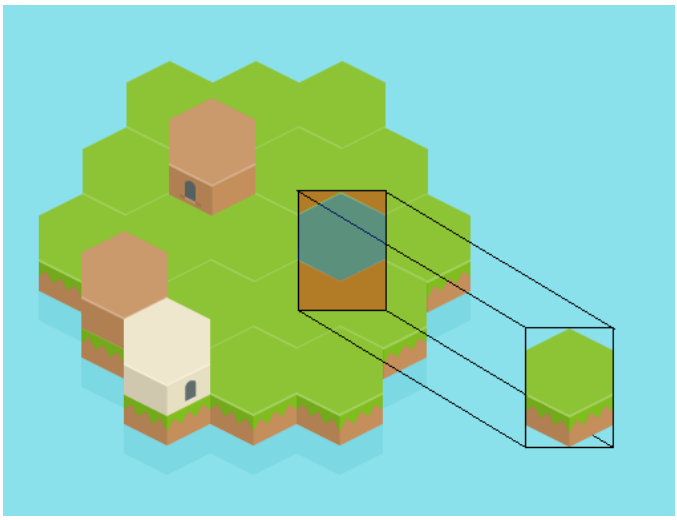

Figura 5. Funcionament del picking

Seguidament, comprova que la jugada es vàlida (la casella està buida i no es suïcidi). Llavors reassigna els grups i comprova si algun grup mor (veure apartat 3.4). Si és així, elimina els grups morts del mapa. Un cop acabat això actualitza el mapa a pantalla i passa el torn al següent jugador.

3.4 Assignar grups de caselles

Un cop vist com trobar caselles veïnes (apartat 3.1)), es necessita un algorisme que donat un mapa sigui capaç de trobar els diferents grups de caselles que hi ha en aquest. Per fer-ho, es recorre el mapa casella a casella, i cada cop que es troba una casella ocupada per un jugador però encara sense grup, s'executa una funció recursiva que s'encarrega d'assignar tot aquell grup.

Podem veure com està implementat a la figura 6, on donat un grup que es vol crear entra en aquesta funció, assigna el grup a la casella, busca caselles veïnes del mateix jugador sense grup (per no repetir feina anterior i tallar la recursivitat) i torna a cridar la funció sobre aquestes caselles.

A part, aquesta funció també aprofita i comprova si el grup que s'està creant està mort, gràcies a marcar-lo per defecte com a mort i si mentre es crea troba alguna casella buida al voltant queda marcat com a viu.

3.5 Comprovació de suïcidi i avís de jugada incorrecta

Durant la fase de lògica del torn es creen els grups i com s'acaba de veure queden marcats els grups morts. Si la

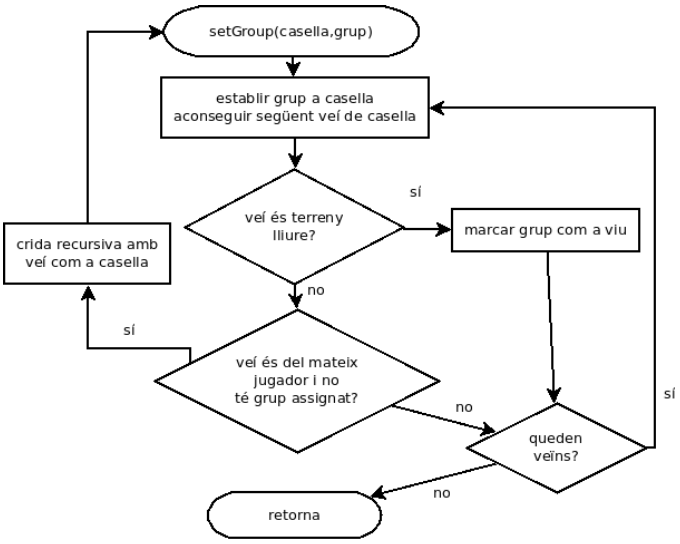

Figura 6. Funció d'assignació de grups

casella que s'ha jugat pertany a un grup mort, es considera suïcidi. S'ha de tenir en compte que si en una jugada de suïcidi mor un grup del contrari, es considera vàlida. Per comprovar això, un cop s'ha detectat el suïcidi, no s'invalida directament la jugada sinó que es guarda una copia del mapa (veure apartat 3.6) i es tornen a generar els grups. D'aquesta manera, si un grup de caselles del contrari ha quedat rodejat, morirà. Un cop s'han tornat a fer els grups es torna a comprovar si la casella pertany a un dels grups que mor. Si és així no ha matat a ningú, es torna a l'estat anterior, s'invalida la jugada i s'avisa de que és suïcidi. Altrament la jugada és vàlida ja que s'ha matat a un grup del jugador contrari.

3.6 Guardat i càrrega de partida

Al final de cada moviment vàlid del joc la partida es guarda, donant la possibilitat de deixar-la a mitges i continuar-la en qualsevol altre moment.

Per fer això, es crea un fitxer amb un objecte Python que conté l'estat actual del mapa, el jugador al qual li toca fer el següent torn, i el número de caselles mortes de cada jugador (per la puntuació final).

Si aquest fitxer existeix, en el menú principal s'activa l'opció de carregar la partida. A més, aquestes funcions serveixen per guardar i carregar estats en moments com la comprovació de suïcidi, o per fer proves de jugades durant la cerca que realitza la IA (veure apartat 3.8).

3.7 Recompte de puntuació final

En el Go original, hi han 2 mètodes principals de recompte de puntuació [22]:

- Area Scoring: Puntuen les fitxes que té cada jugador més el número de caselles buides que estan rodejades només per fitxes del seu color.
- Territory Scoring: Número de caselles buides rodejades només per fitxes d'un jugador, menys número de caselles buides rodejades en *seki* (rodejades per grups dels dos jugadors de forma que cap dels dos pot morir sempre que no jugui en aquestes caselles [23]), menys número de caselles que un jugador ha matat de l'altre

En ambdós mètodes, es necessita que prèviament els dos jugadors es posin d'acord en quines fitxes estan inevitablement mortes (és a dir, que si la partida continués acabarien eliminades) i les retirin del joc. Si no es posen d'acord, es continua jugant.

Per simplificar l'experiència dels nous jugadors, en aquest projecte s'ha decidit fer una puntuació basada amb l'Àrea Scoring però afegint la suma de caselles que eliminades del contrari. Per tant la puntuació és basa en la suma de:

- Número de fitxes que cada jugador té sobre la illa
- Número d'espais lliures que envoltats principalment per un jugador. Aquesta part està simplificada respecte al Go original, no cal que els espais lliures estiguin completament envoltats per un jugador, sinó que simplement se l'emporta el que més fitxes té al voltant de l'espai buit. En cas d'empat no compta per ningú.
- Número de fitxes eliminades del jugador contrari

Els jugadors tampoc han de posar-se d'acord en quines fitxes estan inevitablement mortes. Al ser taulells petits comparats amb el del Go original, és més fàcil que acabin de fer les jugades necessàries, simplificant el joc i el que s'ha d'aprendre per jugar-lo.

Tot això s'ha implementat fent ús dels mètodes per trobar veïns i crear grups de caselles, modificats per tenir en compte també els grups de caselles buides. D'aquesta manera es comprova quin és el jugador que les rodeja majoritàriament.

En cada torn, s'assigna una puntuació a cada jugador.

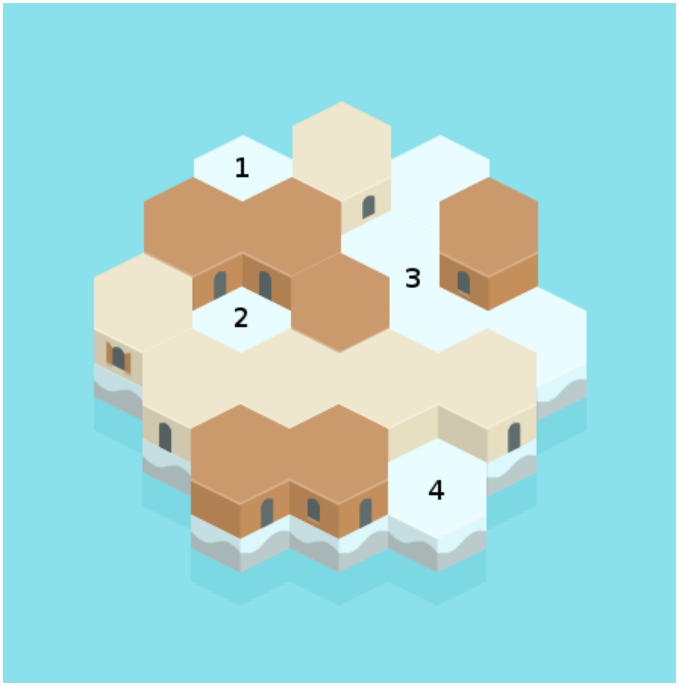

Figura 7. Puntuació

En la figura 7 es pot veure un exemple de partida d'on es posarà un exemple de puntuació. Hi ha un empat en quant a número de fitxes de cada jugador, ambdós en tenen 6. Pel que fa a puntuació de terreny (caselles buides), la cosa canvia.

1) Compta un punt al jugador marró, ja que té dues fitxes rodejant la casella buida i el jugador blanc només en té una
2) No compta per ningú, ja que ambdós jugadors tenen tres fitxes rodejant la casella buida i hi ha un empat.
3) Aquest grup de caselles tampoc compta per ningú, ja que també toca amb tres fitxes de cada jugador
4) Rodejat per dues fitxes blanques i una marró, compta un punt pels blancs

Per tant, suposant que encara no hi ha hagut morts, hi ha un empat de 7 punts entre els dos jugadors. Una bona jugada de les blanques seria jugar sobre l'espai buit 4, matant les dues fitxes marrons de la fila inferior. D'aquesta forma aconseguiria pujar fins a 11 punts: 7 punts de fitxes, 2 punts de caselles buides (els espais que ocupava el marró que queden buits i completament rodejats pel jugador blanc) i 2 punts per les caselles eliminades del rival.

3.8 Intel·ligència Artificial

Uns dels objectius més representatius del joc era tenir una petita Intel·ligència Artificial perquè un usuari pogués fer partides al joc sense tenir a un amic amb ell.

En aquest projecte s'ha implementat IA senzilla, una cerca estàndard de qualsevol joc de taula per a dos jugadors.

Per cobrir aquesta necessitat, s'ha creat una estructura preparada per rebre un estat del taulell en un punt concret i retornar un moviment escollit per la màquina.

A la figura 8 podem veure un esquema de mòduls i fluxe del projecte. Com veiem, un cop estem en el joc principal (quadrat inferior esquerra), a cada torn el mòdul de IA comprova si la partida es contra una IA. En aquest cas, fa els càlculs necessaris i realitza el moviment del jugador 2 automàticament.

Aquesta IA es una caixa negra pel joc, que rep un estat concret del joc (taulell, puntuacions actuals, etc.) i retorna una jugada (ja sigui movent fitxa o passant). Està preparada per tenir diferents formes de pensar aquesta jugada, i és aquí on radica la configuració del nivell (dificultats o comportaments) de la IA.

Tot això es fa en un thread a part per no bloquejar el bucle principal del joc, i també mostra un missatge en pantalla per informar al jugador que la CPU està calculant el següent moviment.

Sota aquesta estructura el primer que s'ha desenvolupat és una IA de prova sota el nom de *Dummy*, que l'únic que fa es escollir un moviment aleatori entre els possibles, i en cas de no existir cap moviment legal passa el torn.

A la figura 9 és pot veure un exemple clar de com fer ús del mòdul d'IA. Es pot contemplar la importància de les funcions *getState* i *loadState* (veure apartat 3.6) per poder fer moviments sobre la classe del joc sense afectar-lo, carregant al finalitzar l'estat inicial.

Sota aquesta estructura s'ha desenvolupat una segona IA basada en un algorisme tipus minimax amb poda alfa-beta.

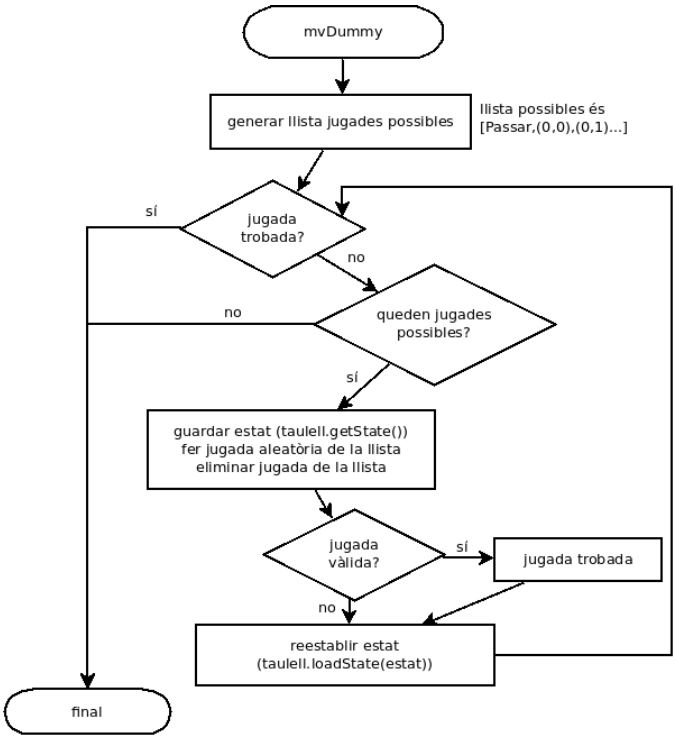

Figura 9. Funció de moviment Dummy

Concretament s'ha implementat l'algorisme **negamax** [25], una variant de l'algorisme minimax [24] que en comptes de alternar entre mínim i màxim, alterna entre màxim i màxim negat. Al buscar sempre un màxim, el codi es simplifica lleugerament.

A més, per intentar optimitzar al màxim la cerca de la millor jugada de la IA, s'ha implementat poda alfa-beta [26] per reduir els nodes explorats.

Per aquesta mateixa raó s'ha implementat també una taula de transposició [27], una tècnica de programació dinàmica que guarda l'avaluació heurística d'un estat per evitar analitzar-la de nou.

A la figura 10 és pot veure el funcionament del mètode de IA negamax de fora esquemàtica.

On la clau és ply [28], que específica el número màxim de jugades que es poden explorar, és a dir, la profunditat màxima que pot assolir l'algorisme recursiu.

3.9 IA Interactiva

Un cop implementada la IA amb negamax, poda alpha-beta i programació dinàmica, s'han fet diferents proves per comprovar fins a quina profunditat podem jugar sense perdre la interactivitat amb el jugador, és a dir, que no hagi d'estar massa temps esperant a que la màquina faci el seu torn.

Amb un taulell hexagonal de 5x5 tenim 19 caselles possibles on jugar. Sempre comença el jugador humà, per tant la CPU tindrà 18 caselles on jugar. Analitzem quantes jugades hauria d'analitzar amb diferents profunditats de joc, sempre en el pitjor dels casos (no poda i no fa servir la taula de transposició)

- Amb profunditat 2, aproximadament $\frac{18!}{16!} = 306$ jugades

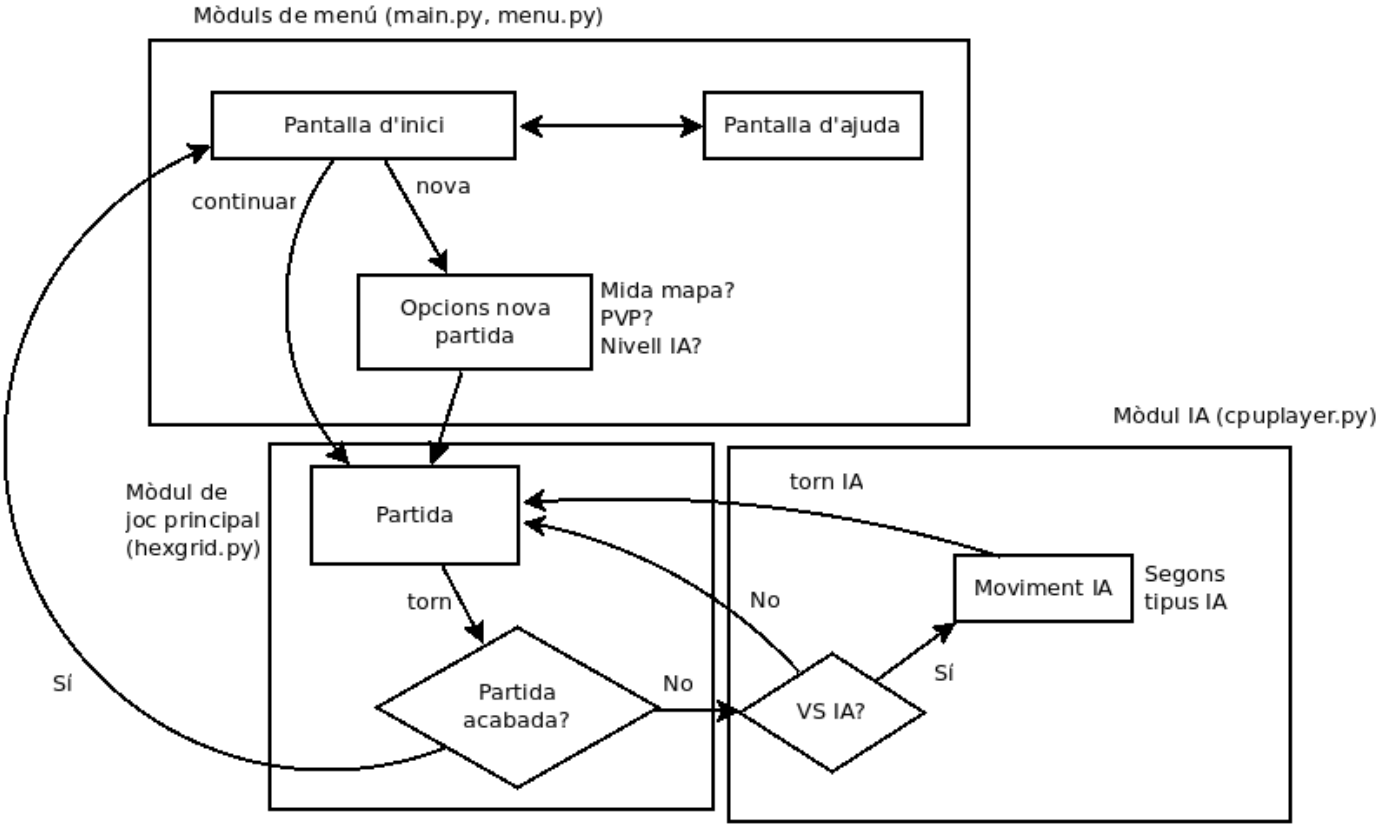

Figura 8. Diagrama del projecte

- Amb profunditat 4, aproximadament $\frac{18!}{14!} = 73440$ jugades
- Amb profunditat 6, aproximadament $\frac{18!}{12!} = 13366080$ jugades
- Amb profunditat 8, aproximadament $\frac{18!}{10!} = 1764322450$ jugades

Com veiem, segons augmentem la profunditat el número de jugades a analitzar augmenta de forma exponencial. A més, tot això es vàlid si el taulell es va omplint, però aquests càlculs no tenen en compte que si un grup de caselles mor, torna a haver-hi espais lliures al taulell que analitzar.

La poda i ús de programació dinàmica redueix de forma considerable els nodes analitzats (per exemple, per profunditat 4 analitza només unes 1000 jugades en comptes de les 73440 que hem analitzat).

S'ha de tenir en compte que la taula fa servir la memòria, i que es guarden diferents estats del joc també segons es va baixant en l'arbre per poder tornar a estats anteriors del joc i provar diferents combinacions. Tot això té un elevat cost de memòria.

Amb tot això, si per exemple fem la primera jugada amb profunditat 6, analitza unes 30000 jugades i triga uns 13 segons en fer-ho. Això ens diu que triga aproximadament mig mili-segon en analitzar cada jugada. Per tant, si volem mantenir la interactivitat, no podem passar de profunditat 4.

En canvi, en moments finals del joc, quan queden poques caselles en les que jugar, el cost d'augmentar la profunditat a 6 o fins i tot a 8 és molt menor, i ens podem permetre que en moments finals del joc la IA intenti jugar millor.

Per tant, el que s'ha fet per millorar la interactivitat és definir el *ply* segons el nombre de caselles lliures:

$$ply = \begin{cases} 4 & si\, casellesLliures > 10 \\ 6 & si\, casellesLliures > 5\ < 10 \\ 8 & si\, casellesLliures <= 5 \end{cases}$$

Tot i que això té un perill, i és que una de les ultimes jugades s'analitzi amb profunditat 8, però un moviment impliqui matar un grup gran de caselles. En aquest cas, analitzarà amb profunditat 8 més caselles de les esperades, i afectarà notablement en la interactivitat. Per tant, una millora per realitzar seria tenir en compte aquests casos per evitar pèrdua d'interactivitat.

4 RESULTATS

El resultat d'aquest projecte és el propi joc i tot el que l'envolta (binaris, pàgina web, source code, etc.)
L'estructura final del joc es resumeix en el diagrama de la figura 8.

Per veure el resultat gràfic, podem veure a la figura 11 la pantalla d'inici i d'ajuda, i a la figura 12 altres pantalles com la de nova partida, un torn del jugador humà i un torn de la CPU.

La millor forma de veure el resultat d'aquest joc és provar-lo. Hi han varies opcions:

- Descarregar el codi font de https://github.com/afibanez/hexland [35] i executar-lo. A més, al ser software lliure, també es pot modificar i millorar, o fer variacions del joc

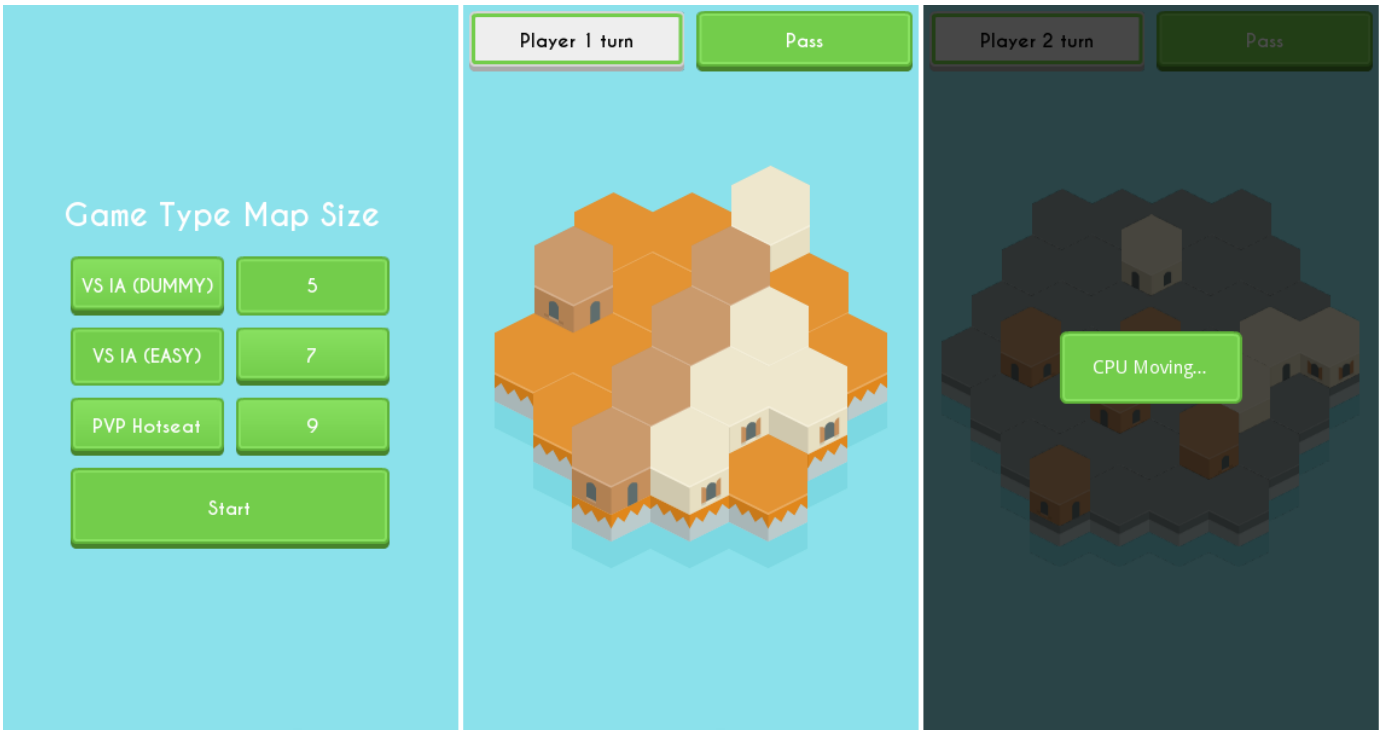

Figura 12. Varies pantalles del joc

- Jugar-hi a un mòbil o tauleta amb sistema operatiu Android, descarregant-lo des de https://play.google.com/store/apps/details?id=org.afibanez.hexland [37]
- Descarregar el binari per escriptori (Windows) a la pàgina web del projecte: http://afibanez.github.io/hexland/ [36]

4.1 APK Play Store

Per poder generar l'aplicació per Android i pujar-la al Play Store s'ha fet servir Buildozer [32], una eina que incorpora Kivy per exportar a Android o iOS.

Buildozer simplifica la feina d'exportació, convertint el codi generat sota Python i Kivy en codi compatible amb Android, a través de l'Android NDK [33]. Amb un fitxer de configuració es poden especificar les opcions del *manifest* d'Android, com el nom del paquet, la versió, icona, etc.

Per poder pujar una aplicació a Google Play, es necessita que l'apk estigui signada amb un certificat vàlid. Buildozer no fa aquesta feina, però sí descarrega les eines de l'Android SDK [34], necessàries per fer-la.

4.2 Binaris PC

Pyinstaller [29] és una eina que permet generar binaris *standalone* (és a dir, sense necessitat de dependències extra) per a plataformes d'escriptori com Windows, GNU/Linux o Mac OS X.

Generar un binari per Windows només es pot fer des de la mateixa plataforma Windows, i implica tenir una serie d'eines instal·lades com el propi Kivy per Windows i pyinstaller [30].

En el cas de GNU/Linux python ve instal·lat per defecte, per tant no existeix la necessitat de generar un binari *standalone*. Els usuaris de GNU/Linux poden jugar al joc només clonant el repositori de Github i instal·lant les dependències del readme (que es redueixen a Kivy, el motor de joc).

5 Conclusions

Tot i que s'han complert tot els objectius del projecte, hi han moltes futures millores que m'agradaria fer un cop acabat el projecte.

Per exemple, estaria molt bé millorar la IA, que tot i que compleix el seu objectiu, no és una IA ideal per jocs tipus Go, ja que actualment s'utilitzen algorismes tipus Monte Carlo, com Monte Carlo Tree Search [31], que milloren notablement l'eficiència i el temps de la IA.

Un altre objectiu interessant que no es va planificar perquè el temps no donava per tant seria donar la opció de jugar online contra amics, per no haver d'estar físicament amb algú per poder jugar la versió *player vs player*.

Però tot això son extres, l'important és que la planificació va ser correcte segons la realitat i el temps del TFG i els objectius principals estan complerts.

Agraïments

Vull mostrar el meu agraïment a l'Antoni Gurguí, tutor d'aquest treball de fi de grau, per deixar-me la llibertat de portar el projecte a la meva manera però sense parar de recordar-me la importància de les coses que sempre deixo més de banda, com la redacció de qualsevol informe. La seva ajuda i insistència ha estat clau pel document que esteu llegint.

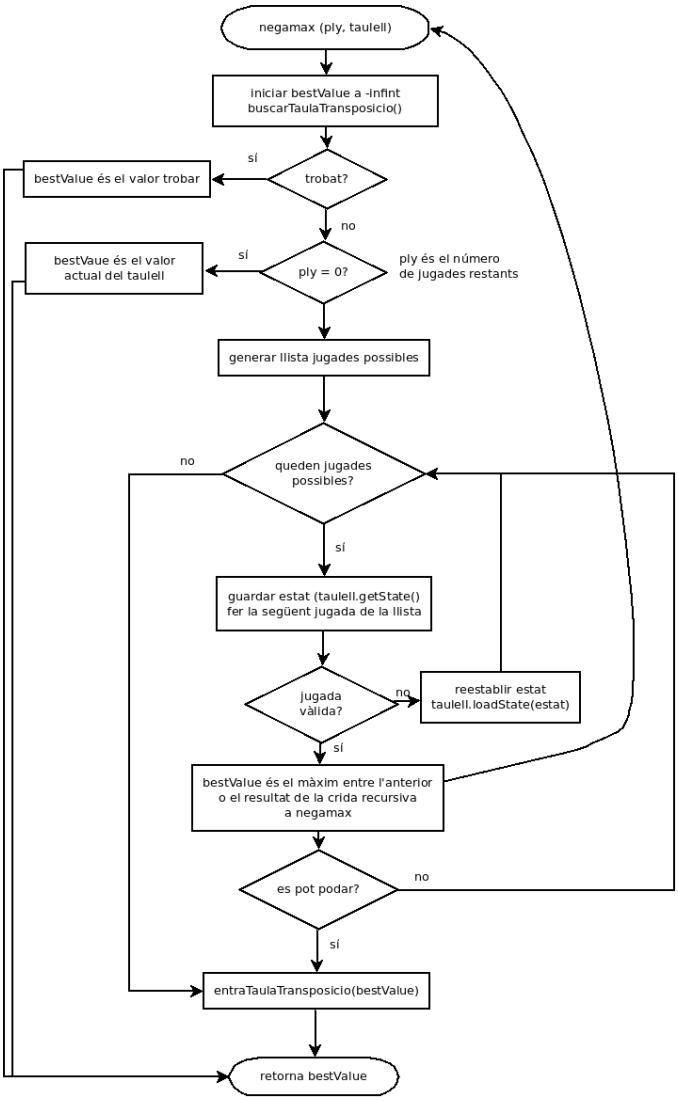

Figura 10. Funció de moviment Negamax

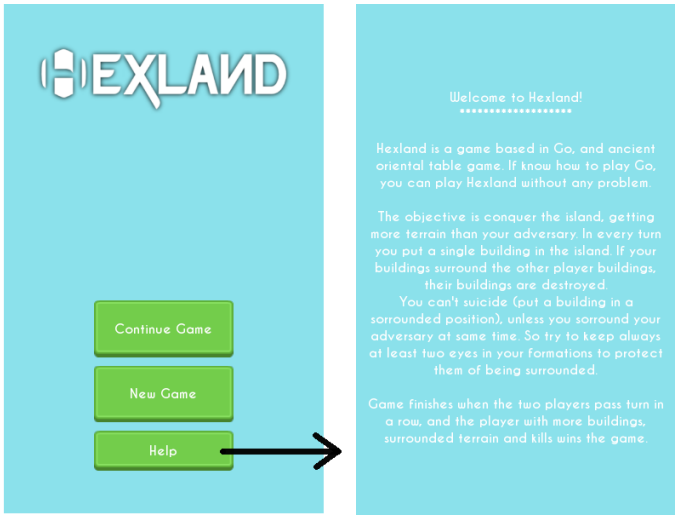

Figura 11. Pantalla d'inici i d'ajuda

REFERÈNCIES

[1] Diversos Autors, pàgina "Go (game)" de Wikipedia, http://en.wikipedia.org/wiki/Go_(game). 2015

[2] J. A. Andújar, J. C. del Río, *El Juego del Go*. Autoeditat, pp. 5-9, 2004. http://en.wikipedia.org/wiki/Go_(game). 2015.

[3] Diversos Autors, pàgina "Computer Go" de Wikipedia, http://en.wikipedia.org/wiki/Computer_Go. 2015.

[4] I J Good, "The Mystery of Go", http://www.chilton-computing.org.uk/acl/literature/reports/p019.htm. 2015.

[5] Millen, Jonathan K, "Programming the Game of Go", http://archive.org/stream/byte-magazine-1981-04/1981_04_BYTE_06-04_Future_Computers#page/n101/mode/2up. 1981.

[6] João Pedro Neto, "Go Board Game on Hexagonal and Other Grids", http://xahlee.info/math/go_board_variations.html. 1965.

[7] João Pedro Neto, "Variants on Go: Other Boards", http://www.di.fc.ul.pt/ jpn/gv/boards.htm. 2013.

[8] Rad Radish, "Hex Go" a la Play Store, https://play.google.com/store/apps/details?id=com.radradish.hexgo. 2015.

[9] Kenney, "Hexagon Tiles", http://www.kenney.nl/assets/hexagon-tiles. 2015.

[10] Kenney, "Hexagon Buildings", http://www.kenney.nl/assets/hexagon-buildings. 2015.

[11] Diversos Autors, "Agile Manifesto", http://www.agilemanifesto.org/iso/ca/. 2001.

[12] Diversos Autors, "Scrum (software development)", http://en.wikipedia.org/wiki/Scrum_%28software_development%29. 2015.

[13] Diversos Autors, "Epic", http://advancedtopicsinscrum.com/glossary-agile-scrum-terms/epic/. 2015.

[14] Raspberry Pi Fundation, "Raspberry Pi ", http://www.raspberrypi.org/. 2015.

[15] UAB, "Treball Fi d'Enginyeria", https://tfe.uab.cat. 2015.

[16] IEEE, "Article Templates and Instructions", http://www.ieee.org/publications_standards/publications/authors/author_templates.html. 2001.

[17] Python Sofware Foundation, "Python", http://python.org/. 2015.

[18] Diversos Autors, pàgina "Linux" de la Wikipedia, https://en.wikipedia.org/wiki/Linux. 2015.

[19] Diversos Autors, "Kivy", http://kivy.org/. 2015.

[20] D. Finley, "Determining Whether A Point Is Inside A Complex Polygon", http://alienryderflex.com/polygon/. 2007.

[21] Amit Patel, "Hexagonal Grids", http://www.redblobgames.com/grids/hexagons/. 2013.

[22] Diversos Autors, "(Go) Scoring", http://senseis.xmp.net/?Scoring. 2014.

[23] Diversos Autors, "(Go) Seki", http://senseis.xmp.net/?Seki. 2015.

[24] Diversos Autors, pàgina "Minimax" de Wikipedia, http://en.wikipedia.org/wiki/Minimax. 2015

[25] Diversos Autors, pàgina "Negamax" de Wikipedia, http://en.wikipedia.org/wiki/Negamax. 2015

[26] Diversos Autors, pàgina "Alpha-beta pruning" de Wikipedia, http://en.wikipedia.org/wiki/Alpha-beta_pruning. 2015

[27] Diversos Autors, pàgina "Transposition Table" de Wikipedia, http://en.wikipedia.org/wiki/Transposition_table. 2015

[28] Diversos Autors, pàgina "Ply" de Wikipedia, http://en.wikipedia.org/wiki/Ply_(game_theory)

[29] Diversos Autors, "Pyinstaller", https://github.com/pyinstaller/pyinstaller/wiki. 2015

[30] Diversos Autors, "Packaging for Windows", http://kivy.org/docs/guide/packaging-windows.html. 2015

[31] Diversos Autors, pàgina "MCTS" de Wikipedia, http://en.wikipedia.org/wiki/Monte_Carlo_tree_search

[32] Diversos Autors, "Buildozer", http://buildozer.readthedocs.org/en/latest/. 2015.

[33] Google, "Android NDK", https://developer.android.com/tools/sdk/ndk/index.html. 2015.

[34] Google, "Android SDK", https://developer.android.com/sdk/index.html. 2015.

[35] A. Fernández, "Hexland Github", https://github.com/afibanez/hexland. 2015.
[36] A. Fernández, "Hexland Github Page", http://afibanez.github.io/hexland/. 2015.
[37] A. Fernández, "Hexland in Play Store", https://play.google.com/store/apps/details?id=org.afibanez.hexland. 2015.

APÈNDIX A
REGLES DEL GO

1) El tauler de go consisteix en una xarxa amb 19 línies horitzontals, i 19 de verticals. Per principiants, o per partides ràpides, es pot jugar amb taulers de 13x13 o de 9x9, amb les mateixes regles. A l'inici de la partida el tauler està buit. Les negres juguen primer. Després ambdós jugadors juguen alternativament. Una jugada consisteix a posar una pedra en una intersecció buida.
2) Quan un jugador fa una jugada que priva la seva última llibertat a una pedra o formació de l'oponent, ha de treure les pedres envoltades del tauler i guardar-les separadament fins al final de la partida.
3) No està permès de fer una jugada ocupant l'última llibertat a l'interior d'una formació enemiga (suïcidi), llevat que aquesta jugada capturi una o més pedres enemigues.
4) No es pot repetir una situació anterior del taulell. Aquesta regla serveix únicament per evitar que en una posició com la de la figura 13, Blanc i Negre capturin i recapturin indefinidament la pedra marcada; cal jugar a una altra banda del tauler abans de recapturar la pedra en situació de ko.
5) En lloc de posar una pedra, un jugador pot passar (perdre un torn). Quan els dos jugadors passen consecutivament s'acaba la partida.
6) Una vegada finalitzada la partida, es retiren les pedres mortes de cada bàndol i s'afegeixen a les capturades.

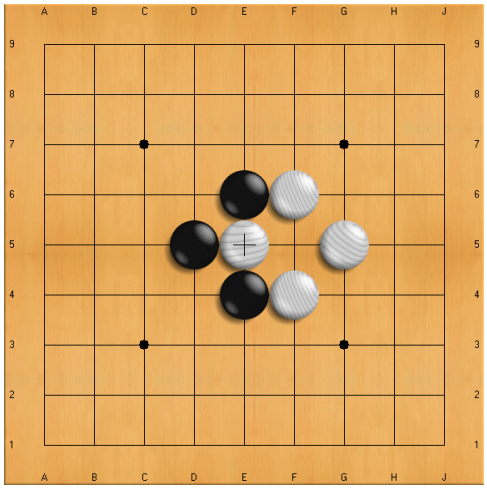

Figura 13. Exemple de situació de ko