UNIVERSITAT DE BARCELONA

Pràctica 2: Jocs amb dos jugadors

Alejandro Guzman

10 octubre 2022

ÍNDEX

1 OBJECTIUS DE LA PRÀCTICA

L'objectiu principal de la pràctica és implementar diversos algoritmes per tal de poder realitzar una simul·lació en la que dos jugadors juguen als escacs. Aquesta simul·lació es realitzarà mitjançant diversos algoritmes de jocs amb oponents.

- **algoritmes de jocs deterministes**: En aquest tipus d'algoritmes cada acció que pren un jugador acaba en un estat el 100% dels cops, és a dir, es sap que realitzant una acció succeïrà tal cosa.
  Es farà servir els algoritmes de Minimax i Poda alfa-beta amb l'objectiu d'implementar una partida d'escacs on cada jugador sap que realitzant un moviment amb una peça, la peça acabarà en una casella determinada del taulell.

- **algoritmes de jocs no deterministes**: Aquests algoritmes, a diferència dels deterministes, tenen un extra afegit, i es que n'hi ha una part de probabilitat en cada acció que es realitza, i es que al realitzar una acció determinada, els estats als quals esdevé el jugador que la realitza, depenen d'una distribució de probabilitat. Per exemple al realitzar una acció potser n'hi han un 50% de probabilitats d'esdevenir un estat, un 25% d'esdevenir en un segon i un 25% d'esdevenenir en un tercer.
  L'algoritme a implementar que afegeix aquest plus d'aletorietat sobre el Minimax serà l'exptectiMinimax.

En la **figura 1.1** es pot observar la diferència entre els dos tipus d'algoritmes presentats anteriorment, i es que els deterministes tenen un únic estat per cada acció, en canvi en els no deterministes en realitzar una acció n'hi han una sèrie d'estats i es pot esdevenir un estat o un altre depenent per exemple d'una distribució probabilística.

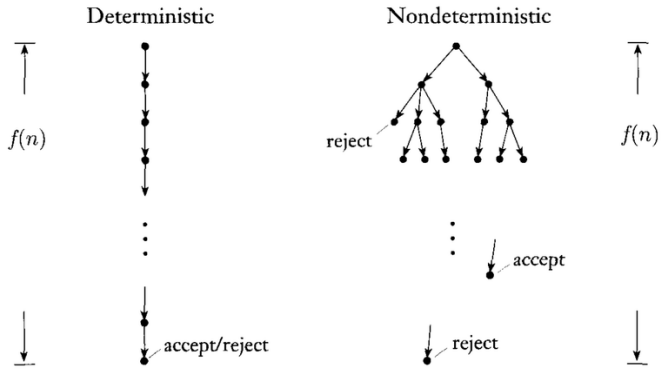

Figura 1.1: Deterministics vs Non-deterministic algorithms

A continuació es mostraràn algunes dificultats que han aparegut durant el desenvolupament de la pràctica i com s'han pogut solucionar.

2 ALGORITMES IMPLEMENTATS

1 Minimax

El Minimax ha sigut el primer algoritme en ser implementar, tractant-se d'un algoritme determinista es sap, com s'ha comentat prèviament, l'estat al qual esdevenirà un jugador en realitzar una acció determinada.

Aquest algoritme és un mètode de decisió per minimitzar la pèrdua màxima esperada en jocs amb adversari i amb informació perfecta. El funcionament de Minimax es pot resumir en com triar el millor moviment per a tu mateix suposant que el teu contrincant triarà el pitjor per a tu.

La pregunta que s'hauria de fer és com es podria implementar aquest algoritme en un joc d'escacs amb una torre i un rei negres i una torre i un rei blancs. La pregunta a aquesta qüestió és simple, es desenvoluparia un bucle per prendre la millor decisió per a negres o blanques (depenent del torn) on a cada iteració es crida al Minimax cercant aquesta millor solució per al jugador.

Tot i així, abans d'implementar el Minimax s'ha de saber quan el joc s'ha donat per finalitzat, és a dir, quan s'ha realitzat escac i mat a un dels dos reis. A part d'implementar una funció d'evaluació per classificar les accions i saber quan una és més bona que una altra.

1.1 Checkmate function

Amb l'objectiu de parar el joc en un moment donat, s'implementarà una funció que verifiqui quan un rei es troba en escac i mat.

Realment és una funció més difícil d'implementar del que pugui aparentar a simple vista i, personalment, ha sigut un mètode bastant complicat de dissenyar perquè funcioni per a tots els casos que es poden donar amb només quatre peces en el tauler.

1.2 Evaluation function

1.3 Implementació de l'algoritme

Un cop tenim definides les funcions per tal de saber quan s'acaba el joc i per determinar quan un estat és millor que un altre, queda implementar l'algoritme Minimax el pseudocode del qual és el mostrat en la **figura 2.1**.

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

Figura 2.1: Minimax pseudocode

La part més complicada en aquest punt de la pràctica ha sigut adaptar el pseudocodi al codi proporcionat pel professorat, ja que les funcionalitats de les classes no tenen una explicació

clara i s'ha de relacionar poc a poc cada part del codi per poder implementar l'algoritme a sobre de la lògica de la pròpia aplicació.

Un dels problemes més freqüents ha sigut l'actualització de peces en el tauler, fent coincidir aquesta actualització en temps real als estats actuals i relacionant-lo tot per sempre fer moviments de peces on realment n'hi hagin peces.

Després de molts intents de prova i error, s'ha finalitzat la implementació del Minimax fent jugar a dos jugadors ficticis mitjançant aquest algoritme.

2 α-β prunning

El segon algoritme a implementar i l'últim que forma part dels algoritmes deterministes és la Poda alfa-beta, un mètode basat en el Minimax que té com a principal objectiu reduïr el temps de còmput de l'execució del propi algoritme.

2.1 Implementació de l'algoritme

La reducció de complexitat en relació al temps es realitza amb una sèrie de podes per evitar explorar aquells estats que es sap al 100% que no aportaràn informació rellevant a la cerca per la presa de la millor decisió.

Per a realitzar les podes primer es va calculant el mínim a n (recorrent els fills de n). L'estimació del valor de n va decrementant. Sigui α el valor màxim que MAX pot obtenir a qualsevol elecció al llarg del camí actual, si estimació de n < α podem deixar d'expandir els fills de n, doncs tenim una alternativa millor. Podem definir de la mateixa forma β per a MIN, tal i com es pot observar en la **figura 2.2**.

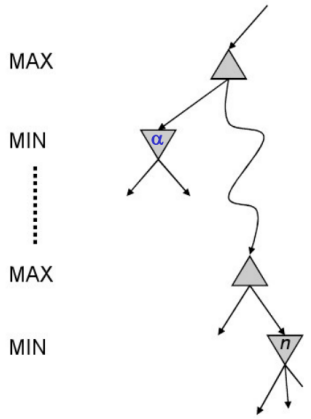

Figura 2.2: Il·lustració de l'explicació realitzada al subapartat 2.1

En la **figura 2.3** es mostra el pseudocodi de la Poda alfa-beta, realment és molt similar al pseudocodi anteriorment vist del Minimax, únicament que s'afegeixen aquestes dues variables α i β per a realitzar la comprovació i efectuar (o no) la Poda.

En el procés d'implementació de l'algoritme no s'han trobat gaires dificultats per tal de portaho a terme, ja que només era ampliar la implementació del Minimax segons el pseudocodi de la Poda alfa-beta mostrat anteriorment.

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figura 2.3: α-β prunning pseudocode

3 Expectiminimax

El tercer algoritme a implementar és un de tipus no determinista, afegint una cert component d'aletorietat al realitzar qualsevol acció.

3.1 Implementació de l'algoritme

El pseudocodi de l'Expectiminimax és el mostrat a la **figura 2.4**, on si s'observa, realment és molt semblant al Minimax, tal i com pasaba amb la Poda alfa-beta.

```
function EXPECTIMAX-VALUE(u, player) :
    if u is a chance node then
        expected ← 0
        for r ∈ moves(u) do
            (move, value) ← EXPECTIMAX-VALUE(result(u, r),
             player)
            expected ← expected + P(r) · value
        end
        return (NULL, expected)
    else if turn(u) = player then
        return MAX-VALUE(u, player)
    else
        // this is a MIN node
        return MIN-VALUE(u, player)
    end
end
```

Figura 2.4: Expectiminimax pseudocode

La implementació de l'Expectiminimax és senzilla, només s'ha d'afegir una distribució de probabilitat als estats disponibles després de moure una peça, per tant afegirem aquesta distribució atribuint a cada estat d'una llista una probabilitat de que aquell mateix surti després de moure una peça determinada.

Amb aquest extra afegit al Minimax, tot i que les peces blanques comencen primer en la implementació, no sempre tindràn les de guanyar, al contrari que pasaba amb el Minimax que el simple fet de fer primer el moviment otorga un avantatge bastant significatiu.

3 EXERCICIS PROPOSATS

1 Exercici 1

Enunciat: Les blanques comencen a moure's. Implementar la dinàmica d'un joc en què tots dos, blancs i negres, segueixen el mateix algoritme de Minimax per intentar fer-se mat entre ells. Suposem que tots dos implementen Minimax amb una profunditat de 4 moviments.

1.1 Apartat a)

Enunciat: Un cop implementat, executa el mateix joc 20 vegades. Quantes vegades guanyen les blanques?
En executar l'algoritme 20 cops, els resultats són exactament els mateixos amb el mateix estat final de les peces. A part d'això s'observa que en el 100% dels cops guanyen les blanques, on l'estat final és un escac i mat amb la torre negra ja eliminada de la partida.

1.2 Apartat b)

Enunciat: Per què això?
Això és degut a que, com les blanques comencen primer, el simple fet de realitzar el primer moviment abans, dona un avantatge superior, a part degut a que les dues torres es troben en la mateixa columna, literalment en el primer moviment la torre blanca pot eliminar a la negra i sentenciar la partida, que és el que efectivament succeeix en l'execució de l'algoritme.

L'estat final del tauler es mostra en la figura 3.1, on podem comprovar que la torre negra ha estat eliminada i les peces blanques han realitzat un escac i mat al rei negre, guanyant així la partida.

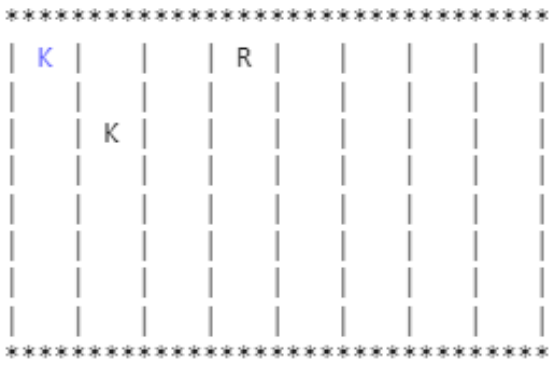

Figura 3.1: Estat final de l'execució del Minimax amb profunditat de 4 moviments

2 Exercici 2

Enunciat: Ara executeu les mateixes simulacions, però variant la profunditat de l'algoritme minimax d'1 a 5 moviments tant per a peces blanques com per a negres. Executeu cada combinació possible de profunditat 10 vegades.

2.1 Apartat a)

Enunciat: Traceu el percentatge de victòries blanques sobre el total de cada valor de profunditat.

Primer de tot executar 10 vegades el mateix codi provoca deu resultats d'execució iguals en el cas del codi desenvolupat, degut a que no n'hi ha cap plus d'aletorietat (no és el cas de l'Expectiminimax). A part, a cada profunditat el resultat és similar, amb victòria en la gran majoria dels casos per part de les blanques, degut a la bona posició que tenen ja que comencen primer i la torre enemiga s'elimina en el primer moviment.

2.2 Apartat b)

Enunciat: El resultat és simètric. Per què és això?
El resultat és pràcticament simètric, amb una mitjana de la meitat dels cops amb victòria de les blanques i meitat amb empat amb cada valor de profunditat, però el que és gairebé impossible que passi és que guanyin les negres degut al desequilibri comentat.

3 Exercici 3

Enunciat: Implementeu la poda alfa-beta només per a les peces negres, les blanques encara juguen amb minimax.

3.1 Apartat a)

Enunciat: Amb una profunditat igual de 4, executeu 10 simulacions. Qui guanya més?
En aquest cas passa exactament el mateix, continuen guanyant les blanques degut a que comencen primer amb connexió directa torre amb torre.

3.2 Apartat b)

Enunciat: Justifica el teu resultat.
Això és degut a que, com hem comentat, en el primer moviment ja es sentencia la partida eliminant la torre blanca a la negra, llavors amb la poda passarà el mateix que amb el minimax ja que l'únic canvi és que és més ràpida, pero la millor solució continuarà sent la mateixa, que en el primer torn és eliminar a la torre negra.

4 Exercici 4

Enunciat: Tant les peces blanques com les negres utilitzen la mateixa poda alfa-beta. Executeu deu simulacions cada vegada variant la profunditat amb la que juga cada equip (1-5).

4.1 Apartat a)

Enunciat: Grafiqueu la proporció de victòries de blancs i negres.
En la poda tal i com hem comentat en els apartats anteriors passa el mateix que amb el minimax, resulta que les blanques al començar primer doncs tenen un avantatge molt important, de fet és aquest avantatge el que fa que guanyin en alguns casos.

El resultat amb l'execució de profunditat 2 és el mostrat a continuació, on les blanques guanyen.

Les blanques guanyen en dos dels cinc casos, amb profunditat 2 i 4.

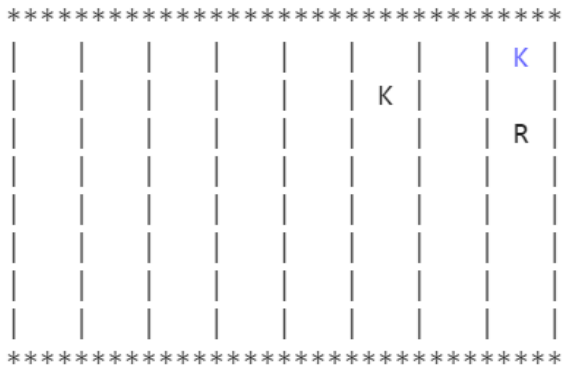

Figura 3.2: Estat final de l'execució de la poda amb profunditat de 3 moviments

4.2 Apartat b)

Enunciat: Comenta el resultat.
Tot depèn dels moviments escollits i si el rei negre es capaç de poder escapar, llavors en aquest cas es produeix l'empat, quan els dos reis s'enfronten però realment cap dels dos li pot guanyar a l'altre.

5 Exercici 5

Enunciat: Implementeu l'algorisme expectimax per a peces blanques i negres.

5.1 Apartat a)

Enunciat: Les peces blanques juguen amb l'expectimax, les negres amb la poda alfa-beta.

i. Executeu 10 simulacions cadascuna i dibuixeu la proporció de victòries per a blancs/negres.
En totes les iteracions la victòria no se la emportat cap jugador, per tant han quedat empat en totes i cadascuna de les simulacions. **ii. Qui guanya més. Per què és això?**
Com hem comentat, no guanya cap dels dos jugadors, degut a que la blanca tot i que té l'avantatge significatiu inicial, que hem comentat que és començar primer, té el desavantatge de que juga amb l'Expectiminimax, per tant els seus moviments tindràn un plus d'aletorietat i per tant no n'agafarà el millor. Llavors com hem comprovat aquest avantatge i desavantatge es confronten per finalment provocar l'empat en totes les partides.

5.2 Apartat b)

Enunciat: Ara les blanques juguen amb la poda alfa-beta, els negres amb l'expectimax.

i. Torna a executar 10 simulacions cadascuna i dibuixa la proporció de victòries blanques/negres.
En aquest cas continuen haver-hi empats però també les blanques guanyen en algun dels casos. Tot depén amb la profunditat que cada jugador jugui, ja que amb millor profunditat es realitzaran millors moviments. Per tant amb una profunditat de 2 queden en empat un total de 4 cops i les blanques en guanyen 6. **ii. Qui guanya més. Per què és això?**
Guanyen més les blanques degut a que parteixen de l'avantatge de començar primer amb la connexió de torre amb torre i així poder eliminar en el primer torn a la torre enemiga.

5.3 Apartat c)

Enunciat: Si hi ha diferències entre a. i b., si us plau, comenta per què. Les diferències que ens trobem principalment són que en el primer apartat empaten en tots els casos degut a que les blanques s'equilibren amb les negres al utilitzar l'Expectiminimax, en canvi en el segon apartat es torna a desequilibrar ja que l'Expectiminimax l'útilitzen les negres.

6 Exercici 6

Enunciat: La situació generada en enfrontar-se un rei blanc i un rei negre més una torre cadascú es pot considerar justa.

6.1 Apartat a)

Enunciat: És realment el cas? Justifica la teva resposta.
En l'escac pot ser el cas depenent sobretot de la posició de les quatre peces, ja que com s'ha vist en aquesta pràctica, que les dues torres estiguin directament conectades per la columna en el primer torn, provoca que el primer jugador sempre eliminarà la torre enemiga, per tant això no és gens just, en canvi si es tenen les torres dels llocs oposats del taulell, és a dir, [0,0] i [7,7] amb els reis en les seves posicions inicials allunyades, llavors sí que és una situació justa on la gran majoria dels cops depenent de la implementació de l'algorisme acabarà la partida en empat.

6.2 Apartat b)

Enunciat: Al seu parer, què fa que aquesta situació sigui d'especial interès per a l'estudi de jocs amb adversaris?
Aquesta situació és interessant sobretot perquè amb un simple canvi de posició de la torre es pot passar d'un joc totalment equilibrat a un joc on la partida està sentenciada inclús abans de començar.

Quan el joc està equilibrat és molt interessant saber quines probabilitats té cada jugador de guanyar, encara que les blanques (les que comencen abans la partida), segons estudis estadístics, tenen més probabilitats de guanyar, degut a que precisament 'obren' la partida, iniciant elles mateixes les possibilitats de joc.

4 CONCLUSIONS

La pràctica s'ha sentit com una continuació de la primera pràctica, degut a que hem utilitzat les mateixes classes implementant algoritmes amb una complexitat superior, afegint més peces.

Un dels passos importants respecte a l'anterior pràctica és que no s'ha hardcodejat cap dada inicial. Si es fa memòria, en els algoritmes de cerca es trobava el camí a una posició que sempre era la mateixa per a totes les execucions, en el cas dels algoritmes de jocs amb dos jugadors això no passa i es que la posició inicial pot variar i el programa continua funcionant amb normalitat, tot i que en les nostres execucions sempre han sigut les mateixes posicions per un interès en particular, i es que les dues torres es troben enfrontades en iniciar el joc.

Per la part de la dedicació en hores, ha sigut una pràctica molt llarga on la majoria del temps ha sigut per poder implementar correctament el pseudocodi amb el codi que ofereix el professorat, de forma que primer s'ha d'analitzar el codi per posteriorment implementar els mètodes a sobre d'aquest mateix codi.

El resultat obtingut a sigut satisfactori, ja que s'ha pogut adquirir els objectius que oferia la pràctica, amb la millora del temps de còmput implementant la Poda.
Personalment, també potser perquè he fet la pràctica sol, al principi he trobat diversos problemes que m'han costat continuar amb el desenvolupament del primer algoritme, però després de diversos intents he aconseguit solucionar-ho i després no he trobat gaire problemes en implementar la resta.

Finalment, ha sigut una pràctica interessant, amb la qual s'ha aprés a optimitzar un algoritme per tal de que trobi la mateixa solució en un temps de còmput reduït, a part de continuar aprenent programació en Python.