UNIVERSITAT DE BARCELONA

Pràctica 2. Adversarial Games

Noah Márquez Vara
Gemma Vallès Fusta

15 novembre 2023

1 OBJECTIUS DE LA PRÀCTICA

L'objectiu principal d'aquesta pràctica és implementar tres algorismes de cerca en jocs amb adversari: dos d'ells seran algorismes de jocs de suma zero de dos jugadors, el Minimax i Poda Alfa-Beta, i el darrer algorisme serà de joc de probabilitats, l'Expectimax.

En el nostre cas els algorismes simularan un joc d'escacs, amb un rei i torre en cada equip, i s'aniran utilitzant els diferents algorismes per a fer proves i comprovar qui guanya en cada situació i a què es deu, tenint en compte també la profunditat de cerca escollida.

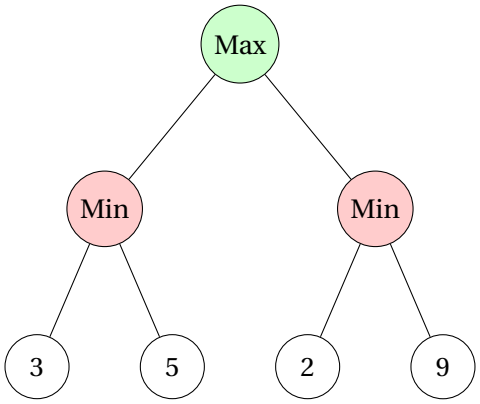
2 ALGORISMES IMPLEMENTATS

1. Minimax

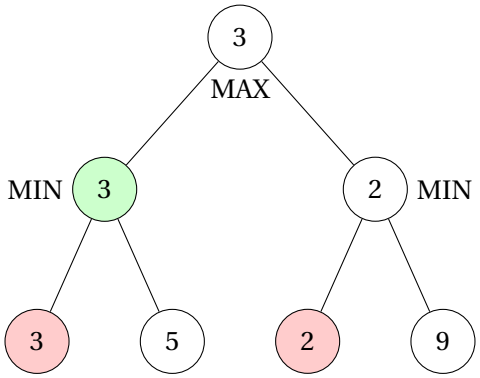
L'algorisme Minimax es basa en analitzar les possibles jugades i escollir la que li donarà una puntuació més alta al nostre jugador, la qual també donarà la puntuació més baixa al nostre oponent.

Per a fer això, escull la jugada amb valor més gran *(max)* d'entre els possibles valors que pot escollir el nostre adversari, el qual escollirà el valor més baix *(min)* d'entre els possibles. Per tant, l'algorisme suposa que el nostre rival realitzarà el moviment més òptim possible.

Per exemple, si tenim l'arbre següent:



Podem observar que en aquest cas el major valor minimax és 3:



Inconvenients: L'algorisme Minimax no és prou eficient com per poder dur a terme problemes grans, ja que en aquests casos el cost computacional és massa elevat.

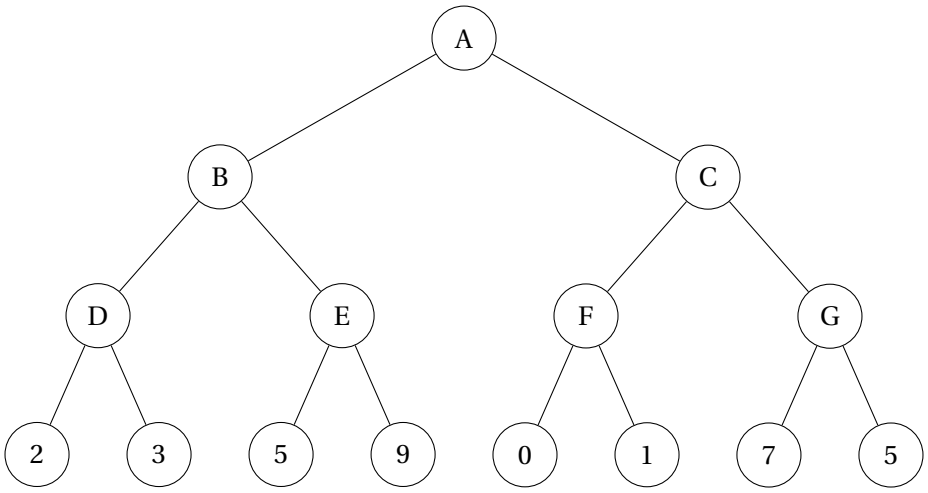
2. Poda Alfa-Beta

L'algorisme Poda Alfa-Beta és una millora del Minimax, en el qual, tal i com el nom indica, es poden nodes per tal de no haver d'expandir-los si no és necessari.

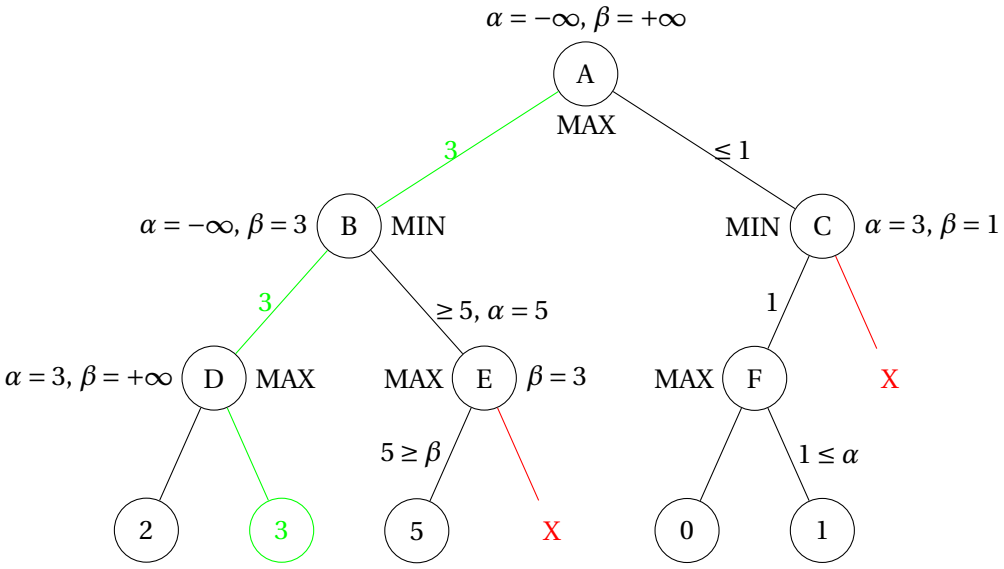
Per a fer això necessitarem dos valors:

- ***alfa*(α) :** és el valor màxim que al maximitzar es pot obtenir en qualsevol elecció al llarg del camí actual.
  Si l'estimació del node actual és menor que α, podem deixar d'expandir els seus fills, perquè ja tenim una alternativa millor.
- ***beta*(β) :** és el valor mínim que al minimitzar es pot obtenir en qualsevol elecció al llarg del camí actual.
  Si l'estimació del node actual és major que β, podem deixar d'expandir els seus fills, perquè ja tenim una alternativa millor.

Per exemple, si tenim l'arbre següent:



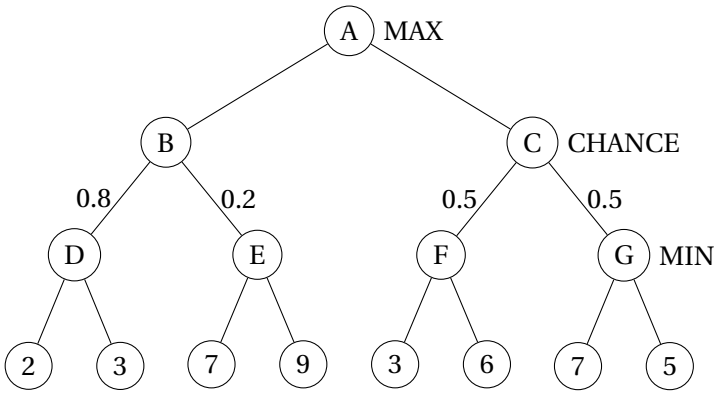
Utilitzant la Poda Alfa-Beta tindríem aquest arbre final:



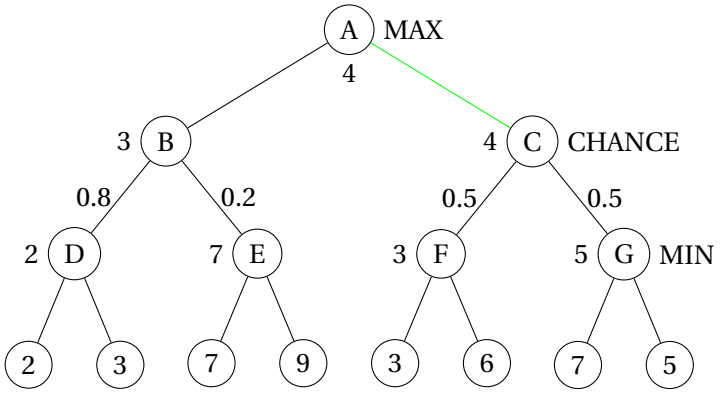
3. Expectimax

L'algorisme Expectimax és una variació del Minimax que és utilitzat en jocs que tenen un factor d'aleatorietat. Els nodes aleatoris utilitzen l'esperança i la resta Minimax. A més a més, necessitem que les magnituds dels valors siguin correctes (e.g. que les probabilitats sumin 1).

Per exemple, si tenim l'arbre següent:



Quedaria de la següent manera:



3 QÜESTIONS PLANTEJADES

1. Whites start moving. Implement the dynamics of a game in which both, whites and blacks, follow the same Minimax algorithm to try to check-mate each other. Assume that both implement minimax with a depth of 4 moves.
Hem executat l'algorisme varies vegades amb les peces en la posició inicial segons el codi base: el rei negre en la posició [0,4] i la torre negra en posició la [0,7] i el rei blanc en la posició [7,4] i la torre blanca en la posició [7,0].

(a) **Once implemented, run the same game a few times. How often do whites win?**
Podem observar que sempre guanyen les blanques. No importa el nombre de simulacions que provem, sempre guanyen si la profunditat de blanques i negres és igual a 4.

(b) **Provide a justification for that.**
La raó per la qual sempre guanyen les blanques és degut a que en aquesta posició inicial, i tenint en compte que el primer moviment sempre el fan les blanques, el millor moviment segons el nostre algorisme, en profunditat 4, sempre serà que la torre blanca es mengi la torre negra, i com que les negres no ho poden evitar de cap manera, les blanques acaben guanyant degut a l'avantatge que els proporciona tenir una peça més que l'adversari.

No només això, sinó que l'heurística està codificada de tal forma que les negres estan sempre en desavantatge respecte les blanques, ja que sempre es busca tractar d'arraconar a les peces negres. A més, és igual quantes vegades provem el codi ja que l'algorisme no té cap mena d'aleatorietat, llavors sempre donarà el mateix resultat.

2. Now run the same simulations, but varying the depth of the minimax algorithm from 3 to 7 moves both for whites and blacks. Run each possible combination of depths a few times.

(a) **Plot the percentage of white wins over the total for each depth value.**

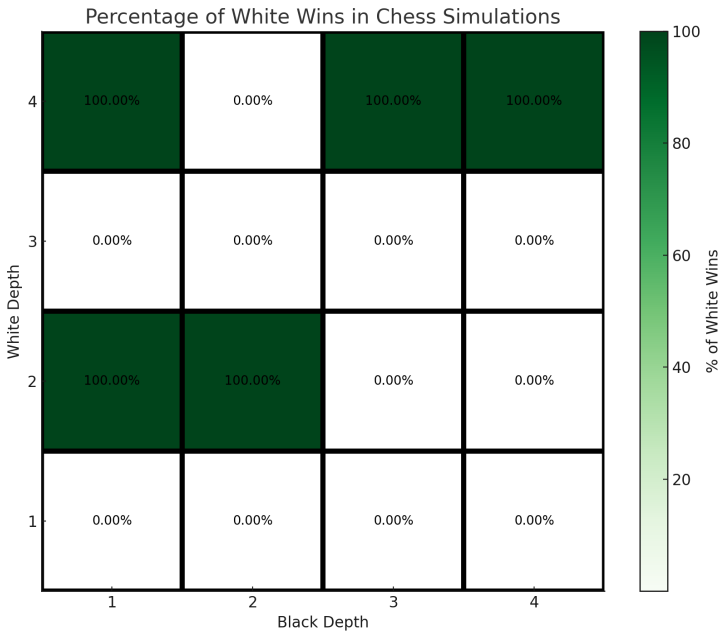


Figura 3.1: Peces blanques

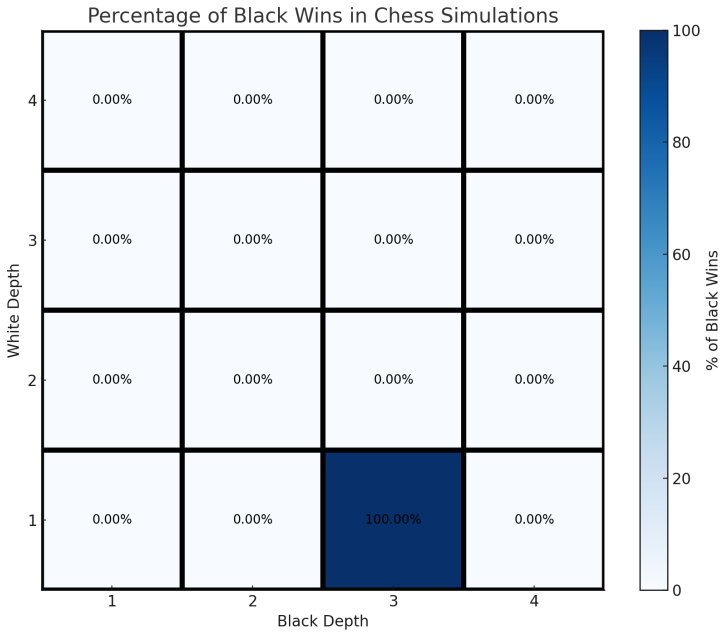


Figura 3.2: Peces negres

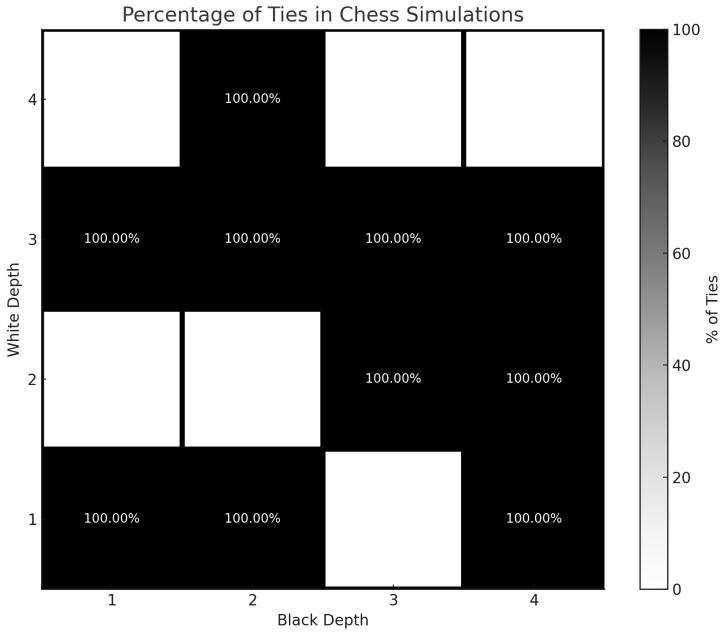


Figura 3.3: Empat

Podem comprovar que les blanques guanyen 5/16 vegades, el qual representa un 31.25%, les negres 1/16, el qual representa un 6.25%, i un empat 10/16, que representa un 62.25%.

(b) **Is the result symmetric. Why is that?**

No, les blanques guanyen molt més que les negres. Això és degut a les blanques tenen

una gran avantatge al ser les que comencen la partida (i també respecte a l'heuristica), les negres només poden perdre o empatar sempre que la profunditat de les blanques sigui major que 1.

3. Implement the alfa-beta pruning for the blacks only, whites still play with minimax.

(a) **Using an equal depth of 4, run the simulation three times. Who is the best of three?**
De la mateixa forma que ho hem comentat en el primer apartat, amb una profunditat igual a 4 per totes dues peces (blanques i negres), sempre guanyen les blanques. La diferència més notable és que amb l'algorisme de poda, el temps d'execució del codi es redueix considerablement.

(b) **Why is that?**
La justificació va en la mateixa línia que en el primer apartat: comencen a jugar les blanques, l'heurística afavoreix a les blanques, i a més no tenim cap mena d'aleatorietat, per tant sempre tindrem el mateix resultat per a unes profunditats donades.

4. Both whites and blacks use the same alfa-beta pruning. Run three simulations each while varying the depth with which each team plays (1-5).

(a) **Plot the proportion of wins for whites and blacks.**

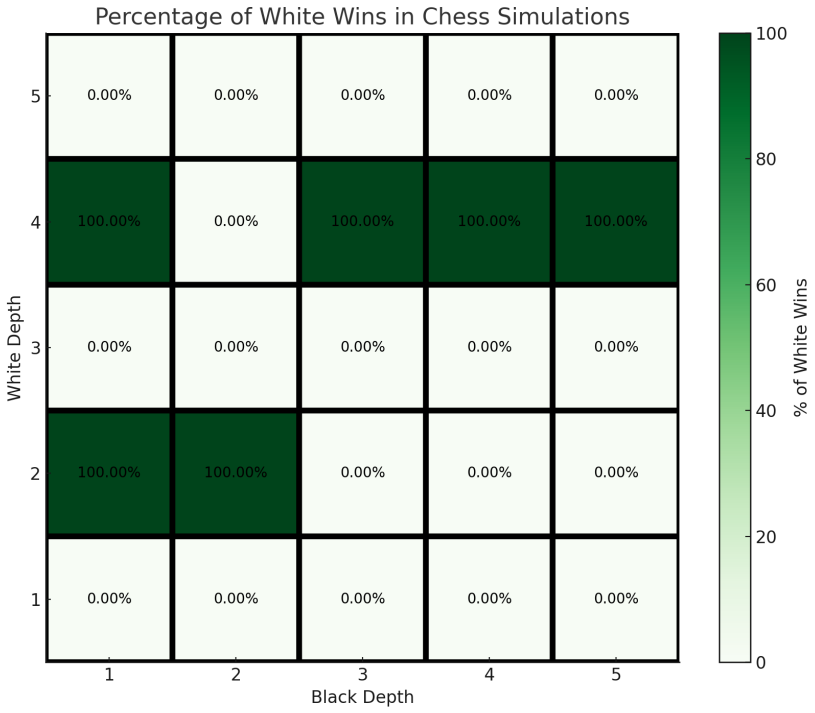


Figura 3.4: Peces blanques

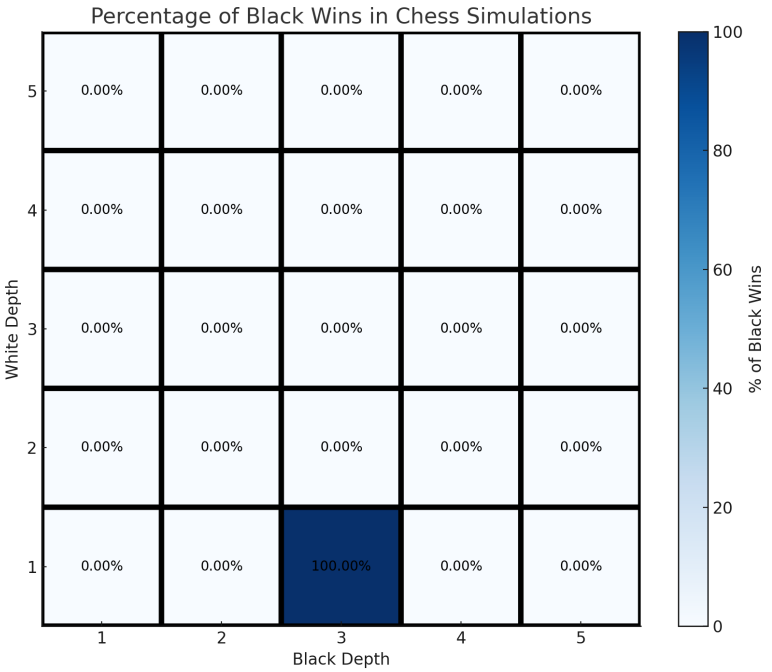


Figura 3.5: Peces negres

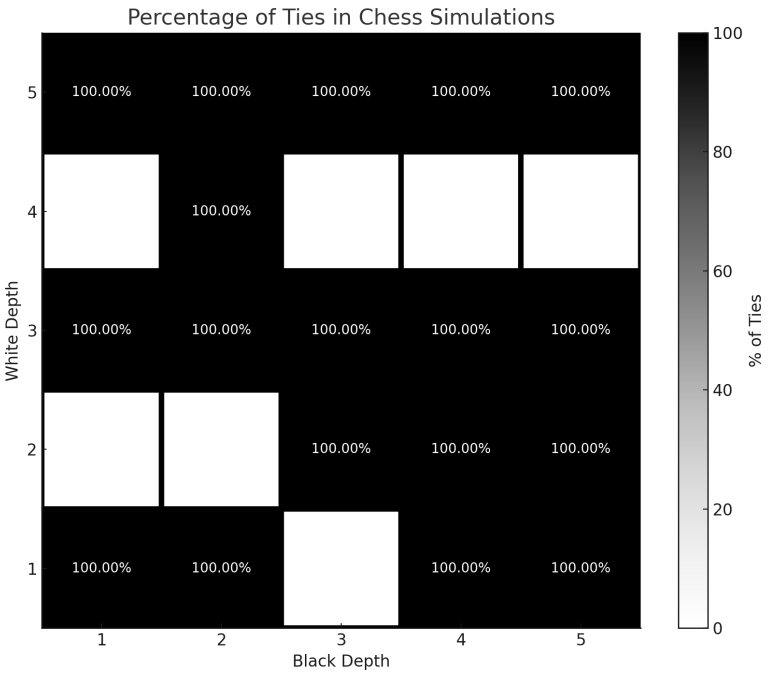


Figura 3.6: Empat

Podem comprovar que les blanques guanyen 6/25 vegades, el qual representa un 24%,

les negres 1/25, el qual representa un 4%, i un empat 18/25, que representa un 72%.

(b) **Comment on the result.**
Podem analitzar en detall els resultats:

- **Victòries de les blanques (6 de 25)**: les victòries de les blanques poden derivar de l'enfocament de l'heurística en el joc agressiu, com ara capturar la torre i arraconar a les peces negres. La baixa taxa de victòria, però, suggereix limitacions potencials en l'agressivitat o l'efectivitat de l'heurística contra la defensa dels negres.
- **Victòria de les negres (1 de 25)**: la rara victòria de les negres implica que l'heurística és menys efectiva per a les negres, i que les estratègies de les blanques estan més alineades amb els criteris de l'heurística.
- **Empats (18 de 25)**: la prevalença dels empats indica que, tot i que l'heurística promou el joc agressiu i a favor de les peces blanques, pot ser que no sigui del tot adequada per guanyar, possiblement posant l'accent en aspectes com la captura de torres o la proximitat del rei.
- **Impacte de la variació de profunditat**: Les variacions en les profunditats (1-5) mostren que les cerques més profundes, tot i que teòricament són més fortes, poden no ser aprofitades completament per l'heurística, la qual cosa condueix a oportunitats de guanyar perdudes.

En general, la preferència de l'heurística per estratègies específiques com la captura de les torres i la proximitat del rei pot no conduir constantment a victòries, cosa que suggereix un desequilibri en els seus criteris d'avaluació. L'elevat nombre d'empats indica que l'heurística podria evitar pèrdues, però no té l'agressivitat o la diversitat per assegurar-se constantment les victòries.

5. Implement the expectimax algorithm for whites and blacks. Whites play expectimax, blacks alfa-beta pruning.

(a) **Run three simulations each and plot the proportion of wins for whites/blacks.**

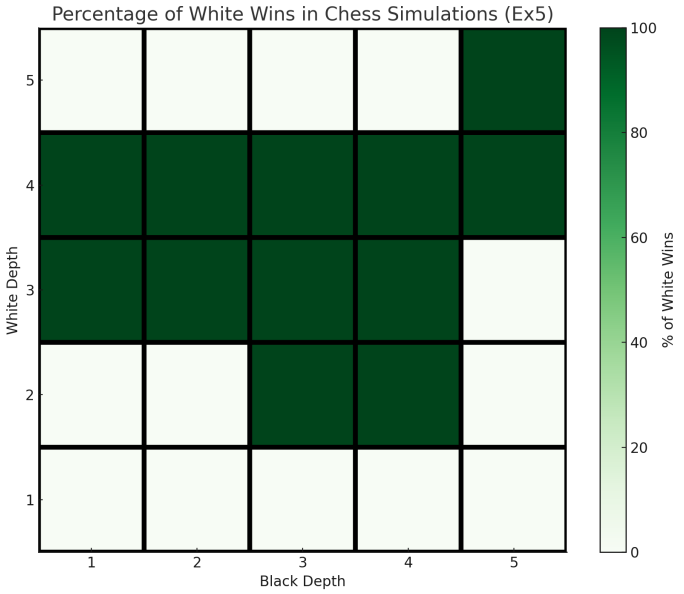


Figura 3.7: Peces blanques

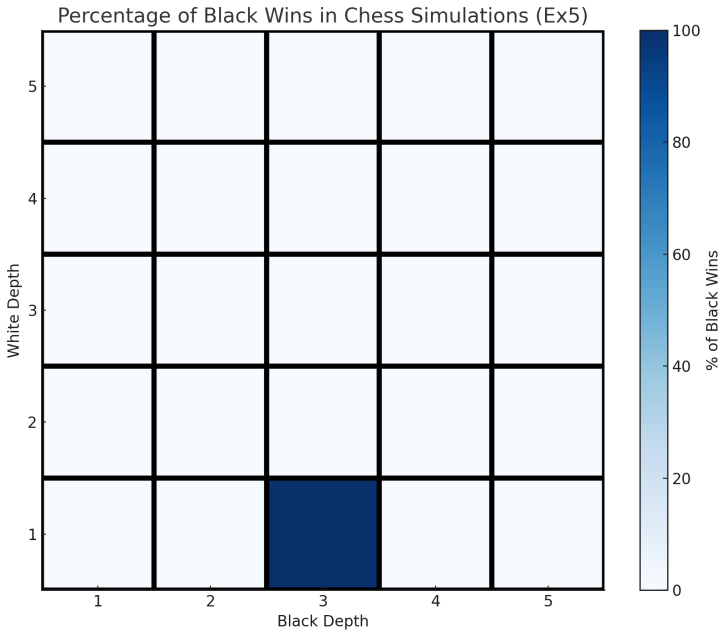


Figura 3.8: Peces negres

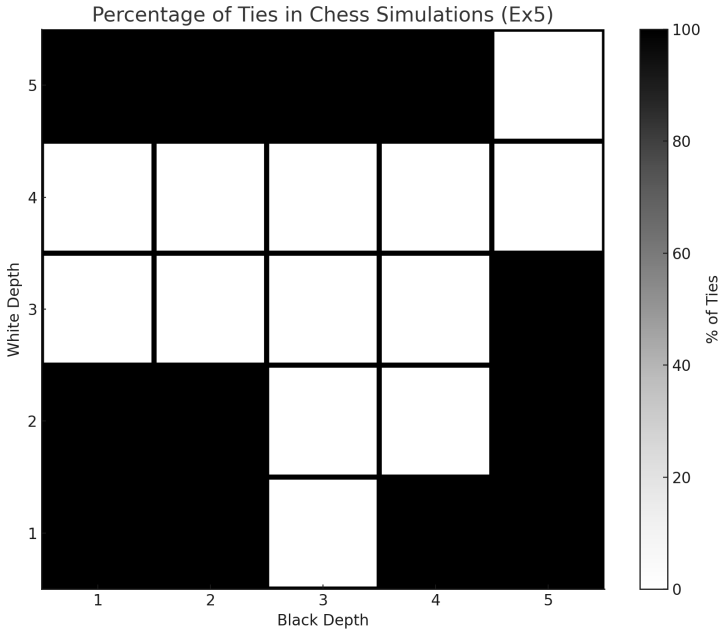


Figura 3.9: Empat

Podem comprovar que les blanques guanyen 12/25 vegades, el qual representa un 48%, les negres 1/25, el qual representa un 4%, i un empat 12/25, que representa un 48%.

(b) **Who wins the most. Why is that?**

A partir dels resultats de les nostres simulacions, podem observar que les peces blan-

ques, utilitzant l'algoritme d'expectimax, guanyen molt més sovint que les negres, que utilitzen la poda alfa-beta. Aquest resultat, on les blanques guanyen 12 de 25 vegades, les negres només guanyen una vegada, i hi ha 12 empats, està influenciat per diversos factors relacionats amb els algorismes i l'heurística específica implementada.

- **Diferències d'algorisme**: L'expectimax i la poda alfa-beta són fonamentalment diferents en el seu enfocament a l'avaluació de l'arbre del joc. L'expectimax és un algorisme probabilístic, tenint en compte el resultat mitjà de tots els moviments possibles (per tant, és de naturalesa més exploratòria), mentre que la poda alfa-beta és un algorisme determinista que elimina les branques que no poden influir en la decisió final (per tant, és més selectiu). Aquesta diferència podria portar a expectimax a trobar estratègies més òptimes a nivell global sobre la poda alfa-beta a la configuració específica del nostre joc.
- **Funció heurística**: la funció heurística que se'ns ha proporcionat està adaptada per capturar una àmplia gamma d'elements estratègics del joc (com la posició de reis i torres, escac i mat i seguretat de les peces). L'efectivitat d'aquestes heurístiques depèn en gran mesura de com representen el valor real d'un estat del joc, cosa que podria afavorir l'algoritme d'expectimax, ja que avalua més estats de joc de mitjana en comparació amb la poda alfa-beta.
- **Detalls de la implementació**: els detalls de la implementació d'ambdós algorismes, com ara la profunditat de la cerca i com s'avaluen els llaços o les amenaces, poden afectar significativament el seu rendiment. Per exemple, si l'expectimax s'implementa per explorar més a fons o està millor optimitzat per a l'escenari de joc específic, podria tenir un cert avantatge.

En conclusió, les peces blanques guanyen amb més freqüència a causa de la naturalesa exploratòria de l'algorisme expectimax juntament amb la funció heurística específica, que proporciona una avaluació més completa dels estats del joc en comparació amb la naturalesa selectiva i determinista de la poda alfa-beta utilitzada per les peces negres. A més, els detalls de la implementació i la profunditat de la cerca també contribueixen a aquest resultat.

6. The situation generated by confronting a white king and a black king plus a rook each may be considered even.

(a) **Is it really the case? Justify your answer.**
La situació en què cada jugador té un rei i una torre no està igualada. El resultat depèn en gran mesura de la posició de les peces i del nivell d'habilitat dels jugadors. En general, els jocs finals d'escacs que impliquen un rei i una torre contra un rei i una torre es consideren molt atractius, però hi ha diversos matisos:

- **Coordinació de les peces**: l'eficàcia del treball conjunt entre el rei i la torre és crucial. Els atacs i les defenses ben coordinats poden donar lloc a un avantatge significatiu.
- **Seguretat dels reis**: la posició dels reis pot influir molt en el joc. Un rei més centralitzat i implicat activament en el joc pot ser un factor decisiu.
- **Activitat de les torres**: el posicionament i l'activitat de les torres també són vitals. Una torre activa pot exercir pressió i control sobre caselles clau, mentre que una torre passiva pot ser una responsabilitat.
- **Habilitat del jugador**: a la pràctica, el resultat depèn en gran mesura de les habilitats finals del joc dels jugadors, inclosa la seva capacitat per evitar errors, forçar intercanvis o crear amenaces.

(b) **In your opinion, what makes this situation of particular interest for the study of adversarial games?**
Aquest escenari final del joc d'escacs és especialment interessant per estudiar jocs adversaris per diversos motius:

- **Equilibri de poder**: tots dos jugadors tenen el mateix material, però la dinàmica del joc pot canviar ràpidament en funció de l'estratègia i la tàctica, el que el converteix en un excel·lent cas d'estudi per entendre l'equilibri en la teoria de jocs.
- **Complexitat des de la senzillesa**: malgrat el nombre limitat de peces, l'escenari ofereix riques possibilitats estratègiques i tàctiques, il·lustrant com la complexitat pot sorgir de sistemes simples.
- **Predictibilitat versus incertesa**: el joc ofereix una combinació de predictibilitat (a causa del material limitat) i incertesa (a causa de la gran varietat de possibles posicions i moviments), el que el converteix en un bon model per explorar la presa de decisions sota incertesa.
- **Estratègia del final del joc**: és un escenari excel·lent per estudiar els principis del final del joc, que sovint tracten de maximitzar el potencial dels recursos limitats, un concepte aplicable a diversos jocs estratègics i situacions de la vida real.
- **Eficiència de l'algoritme**: provar algorismes d'IA com expectimax i poda alfa-beta en aquest escenari pot proporcionar informació sobre la seva eficiència, eficàcia i com gestionen situacions igualment equilibrades però canviants dinàmicament.

En resum, aquest escenari final del joc d'escacs, tot i que sembla equilibrat pel que fa al material, ofereix un ric terreny per explorar la teoria de jocs adversaris, tant des d'una perspectiva humana com d'IA.

4 PROVES REALITZADES

Les proves realitzades per tal de dur a terme tots els exercicis que es demanaven al document de la pràctica les hem deixat comentades i visibles dins del propi codi de *aichess.py*.

Per cadascun dels exercicis hem creat una funció (e.g. ***def exercici1(self, aichess)***) que executa les simulacions demanades, i en el cas dels exercicis que demanava fer un gràfic (i.e. exemple exercici 2) hem fet un petit codi que generava un fitxer *.csv* on es guardava per cada profunditat estudiada qui havia guanyat.

Per tal d'executar qualsevol de les proves només cal descomentar al *main()* el codi just a sota del comentari de *Python* de l'estil: ***### Run exercise X ###***.

5 HEURÍSTICA

Una heurística és una estimació de quant pròxim es troba un cert node actual de l'objectiu del problema. Cada problema necessitarà una heurística específica, una mateixa heurística no ens serveix per a dos problemes diferents.

1. La nostra heurística

En el nostre cas concret l'objectiu és arribar a l'escac i mat, per tant necessitarem una heurística que calculi quin moviment ens apropa més a aquest objectiu.

Per a dur això a terme, anirem sumant o restant diversos valors segons si ens apropen o allunyen de l'objectiu. Per simplificar-ho assumim que estem calculant l'heurística de les blanques i al final de tot ho multipliquem per −1 si és que estem en el cas de les peces negres.

Primer de tot comprovem si la torre negra ha estat eliminada, si és així sumem 50 punts al valor de l'heurística:

```
if brState == None:
    value += 50
```

A més a més, si la torre negra ha estat eliminada però la blanca no, i la distància entre els reis és superior o igual a 3, sumem també la distància entre la torre blanca i el rei negre dividida entre 10:

```
fila = abs(filaBk - filaWk)
columna = abs(columnaWk - columnaBk)
distReis = min(fila, columna) + abs(fila - columna)
if distReis >= 3 and wrState != None:
    filaR = abs(filaBk - filaWr)
    columnaR = abs(columnaWr - columnaBk)
    value += (min(filaR, columnaR) + abs(filaR - columnaR))/10
```

Si la torre negra ha estat menjada li sumem 7 menys la distancia entre els reis al valor, ja que volem que el rei blanc estigui el més a prop possible del negre:

```
value += (7 - distReis)
```

Si la torre negra ha estat eliminada, també mirarem si el rei negre està en una paret, prioritzant que estigui en una cantonada, per a facilitar l'escac i mat, i si no està en una paret prioritzarem que hi estigui a prop:

```
if bkState[0] == 0 or bkState[0] == 7 or bkState[1] == 0 or bkState[1] == 7:
    value += (abs(filaBk - 3.5) + abs(columnaBk - 3.5)) * 10
else:
    value += (max(abs(filaBk - 3.5), abs(columnaBk - 3.5))) * 10
```

Després repetirem el mateix codi però al revés, mirant si és la torre blanca la que ha estat eliminada i restant els valors en comptes de sumar-los.

Finalment, restarem 20 punts si el rei blanc està amenaçat i en sumarem 20 punts en cas de que sigui el negre el que ho està:

```
if self.isWatched(currentState, color):
    # White threatened
    if color:
        value += -20
    # Black threatened
    else:
        value += 20
```

Tal i com ja hem comentat al principi, en cas de que estiguem calculant l'heurística de les negres multiplicarem el valor per −1 abans de retornar-lo:

```
if not color:
    value = (-1) * value
```

6 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 fet ús de cap dada *hard-coded* inicial. Si fem 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 i final poden variar i el programa continua funcionant amb normalitat.

En vers la dedicació en hores, ha estat una pràctica molt llarga on la majoria del temps s'ha utilitzat per poder implementar correctament el pseudocodi que ens va oferir el professorat per cadascun dels diferents algorismes, de forma que primer havíem d'entendre correctament com funcionaven tot i cadascun dels algoritmes, per poder després passar a implementar el codi fixant-nos en el pseudocodi proporcionat.

El resultat obtingut ha estat molt satisfactori, ja que s'ha pogut adquirir els objectius que oferia la pràctica, amb la millora del temps de còmput implementant la Poda.

En conclusió, ha sigut una pràctica molt 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 més d'haver implementat tres grans algorismes de cerca.