TREBALL FINAL DE GRAU

# Dificultat adaptativa en jocs de taula a través de Minimax

Albert Medina i Cucurull

Grau en Disseny i Producció de Videojocs

CURS 2020-21

TecnoCampus
Escola Superior Politècnica

*Centre adscrit a la*

upf. Universitat Pompeu Fabra *Barcelona*

TecnoCampus

Centres universitaris adscrits a la

upf. Universitat Pompeu Fabra Barcelona

**Grau en Disseny i Producció de Videojocs**

# Dificultat adaptativa en jocs de taula a través de Minimax

**MEMÒRIA FINAL**

**Albert Medina i Cucurull**

**Tutor: Dr. Enric Sesa Nogueras**

**2020-2021**

TecnoCampus Mataró-Maresme

## Agraïments

A la família i amics, per la paciència i el suport.

A tots els jugadors participants, pel seu temps i dedicació.

A l'Enric Sesa Nogueras, per la seva constant ajuda i suport al llarg de l'elaboració del treball.

## Abstract

The main goal of this project consists in the design and the implementation of an algorithm based on Minimax that adapts its performance dynamically depending on the moves of its opponent, generating an adaptive difficulty. A prototype of the draughts game is also implemented to prove the functioning of this algorithm.

## Resum

L'objectiu principal d'aquest treball consisteix en el disseny i la implementació d'un algoritme basat en Minimax que adapti el seu rendiment de manera dinàmica en funció dels moviments del seu oponent, generant una dificultat adaptativa. També es du a terme l'elaboració d'un prototip del joc de les dames angleses per tal de comprovar el funcionament d'aquest algoritme.

## Resumen

El objetivo principal de este trabajo consiste en el diseño y la implementación de un algoritmo basado en Minimax que adapte su rendimiento de forma dinámica en función de los movimientos de su oponente, generando una dificultad adaptativa. También se lleva a cabo la elaboración de un prototipo del juego de las damas inglesas para comprobar el funcionamiento de este algoritmo.

# Índex

# Índex de figures

# Índex de taules

# 1. Introducció

Un dels reptes més grans amb els que es troben els desenvolupadors a l'hora de dissenyar i implementar els seus jocs és el correcte ajustament de la dificultat. Aquests no només han de tenir en compte el balanceig de tots els paràmetres existents, sinó també el fet que els respectius nivells d'habilitat dels diferents jugadors poden variar d'una manera considerable. Aquest problema, però, podria ser reduït o resolt mitjançant un sistema de dificultat adaptativa que pogués variar de manera automàtica en funció de les accions del jugador, adaptant-se, d'aquesta manera, al seu nivell d'habilitat (Missura i Gaertner, 2010). El cas dels jocs de taula digitals és un bon exemple d'això, doncs el nivell del jugador pot dependre molt d'alguns factors com el coneixement de les regles i possibilitats del joc o l'experiència prèvia, entre d'altres.

Aquest treball proposa el disseny d'un algoritme basat en Minimax, un dels algoritmes més destacats i utilitzats en jocs de taula, que pugui reaccionar a les accions del jugador per tal d'adaptar-ne el nivell de dificultat d'una manera dinàmica, així com la seva posterior avaluació, la qual servirà per posar-lo a prova per tal d'analitzar-ne el funcionament en funció del nivell dels seus oponents.

En el pròxim capítol d'aquest treball se'n detallen els objectius, desglossats en principals i secundaris. A continuació, a través del marc teòric es contextualitza amb més profunditat sobre els aspectes fonamentals d'aquest treball, passant pels jocs de taula, la intel·ligència artificial i Minimax, entre d'altres. El següent capítol mostra diferents referents a tenir en compte per a l'elaboració del treball, i els tres darrers capítols corresponen a la fase de desenvolupament d'aquest, als resultats que se n'extreuen i a les conclusions i valoracions que s'assoleixen finalment.

# 2. Objectius

El principal propòsit d'aquest treball consisteix en l'elaboració d'un algoritme adaptatiu basat en Minimax i en un posterior anàlisi per tal d'avaluar-ne el funcionament. Això podria resumir-se en diversos objectius classificats de la següent manera:

## 2.1. Objectius principals

- Elaborar un algoritme a partir de Minimax que pugui adaptar les seves decisions en funció de la resposta del jugador per tal de crear una dificultat adaptativa.
- Implementar un prototip del joc de les dames angleses per tal de comprovar i analitzar el funcionament de l'algoritme.

## 2.2. Objectius secundaris

- Enfrontar l'algoritme elaborat a jugadors reals, a més de a ell mateix o a altres algoritmes, al joc de les dames per tal d'estudiar situacions equivalents a casos reals.

# 3. Marc teòric

El següent punt del treball inclou tots aquells conceptes teòrics necessaris a estudiar prèviament a la fase desenvolupament. S'estructura en diversos punts que repassen tots els aspectes relatius la matèria del treball.

## 3.1. Jocs de taula

### 3.1.1. Teoria de jocs

Tal com explica Millington (2006), la teoria de jocs es podria definir a grans trets com una disciplina matemàtica que s'encarrega de l'estudi de jocs abstractes. Algunes altres descripcions, però, permeten obtenir més concreció en la definició del concepte. Myerson (1991) i Osborne i Rubinstein (1994), per exemple, defineixen d'una manera més concreta que allò que estudia la teoria de jocs són models matemàtics d'interacció entre dos o més agents racionals i intel·ligents que prenen decisions. D'aquesta manera, la teoria de jocs proporciona diferents tècniques que poden ser utilitzades per analitzar situacions en les quals l'estat en què es troba un determinat individu podrà ser influenciat d'una manera o d'una altra en funció de la decisió presa per un altre individu (Myerson, 1991; Osborne i Rubinstein, 1994).

### 3.1.2. Classificació de jocs

Seguint la teoria de jocs, aquests poden ser classificats en diverses categories diferents en funció de les seves característiques (Osborne i Rubinstein, 1994; Millington, 2006).

A continuació es mostren algunes de les categories més rellevants:

- Classificació segons el nombre de jugadors:

Els jocs poden ser classificats segons el nombre de jugadors que juguin una mateixa partida tot i que, tal com es podrà veure en els propers apartats d'aquest capítol, la majoria d'algoritmes d'intel·ligència artificial ideats per a jocs que funcionen per torns estan pensats per a només dos jugadors (Millington, 2006).

- Classificació segons el benefici dels jugadors:

Un joc és de suma zero quan la victòria o guany d'un jugador suposa la derrota o pèrdua del seu adversari en la mateixa magnitud: d'aquesta manera, no és rellevant si el jugador busca el seu propi benefici o bé pretén aconseguir l'error del seu adversari, doncs el resultat obtingut serà el mateix en ambdós casos. N'és un exemple el pòquer (Millington, 2006).

Aquesta situació forma un patró competitiu en el qual no hi ha la possibilitat de que les dues parts obtinguin una victòria o una derrota: la victòria d'un jugador sempre suposarà la derrota de l'altre, i a l'inrevés. En la representació d'aquest patró els valors de benefici per a cada un dels jugadors seran sempre iguals, però tindran signes oposats. Tots els jocs que segueixen aquest patró, per tant, són jocs de suma zero (Taha, 2021).

- Classificació segons la informació del joc:

Depenent de si els diferents jugadors tenen informació sobre tots els moviments possibles per a cada jugador, el joc serà d'informació perfecta o imperfecta (Osborne i Rubinstein, 1994). D'aquesta manera, un exemple de joc amb informació perfecta és el joc dels escacs, doncs a partir de l'estat en què es troba la partida els dos jugadors poden saber quins son els propers moviments possibles i quin seria el resultat de cada un d'ells (Millington, 2006).

Per contra, qualsevol joc que contingui aspectes que evitin preveure els possibles moviments o els seus resultats, com ara elements ocults o esdeveniments fruit de l'atzar, seran classificats com a jocs d'informació imperfecta (Millington, 2006). El mateix autor posa d'exemple el backgammon, doncs cap jugador pot predir el nombre que sortirà quan el dau sigui llançat.

## 3.2. Intel·ligència artificial

El fet d'intentar trobar una definició precisa per al concepte d'intel·ligència artificial sempre ha estat causant d'una gran discussió i ha generat una certa confusió entre tots aquells autors i investigadors que han provat d'aconseguir-ho. A més, totes aquestes definicions han anat variant i evolucionant amb el pas del temps a

conseqüència del ràpid creixement que ha experimentat aquest camp (Kok, Boers, Kosters, i Van der Putten, 2009).

Aquestes definicions poden ser dividides tenint en compte dues dimensions diferents. La primera dimensió diferencia entre el fet de tenir un comportament i el de tenir un raonament, mentre que la segona dimensió fa una distinció entre mesurar l'èxit equiparant-lo al rendiment propi d'un humà i mesurar-lo segons un concepte ideal d'intel·ligència, el qual pot ser anomenat racionalitat (Russell i Norvig, 1995).

| | **HUMANITAT** | **RACIONALITAT** |
|---|---|---|
| **RAONAMENT** | Sistemes que pensen com a humans | Sistemes que pensen racionalment |
| **COMPORTAMENT** | Sistemes que actuen com a humans | Sistemes que actuen racionalment |

Taula 3.1. Categories de definició d'intel·ligència artificial. Font: Elaboració pròpia a partir de Stuart J. Russell i Peter Norvig, 1995.

En la taula 3.1 es pot observar una definició a grans trets per a cada una de les quatre categories que deriven de les dues dimensions anteriorment esmentades.

En la taula que es mostra a continuació hi apareixen diverses definicions d'intel·ligència artificial per part de diferents autors, classificades de la manera anteriorment indicada en funció de com defineixen el concepte.

| | HUMANITAT | RACIONALITAT |
|---|---|---|
| **RAONAMENT** | "The exciting new effort to make computers think . . . machines with minds, in the full and literal sense" (Haugeland, 1985) | “The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985) |
| | "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..."(Bellman, 1978) | "The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992 |
| **COMPORTAMENT** | "The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990) | "A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990) |
| | "The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991) | "The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993) |

Taula 3.2. Algunes definicions d’intel·ligència artificial. Font: Elaboració pròpia a partir de Stuart J. Russell i Peter Norvig, 1995.

Wang (2019) argumenta que no hi ha una única manera correcta de definir el concepte d’intel·ligència artificial, sinó que això dependrà pròpiament de cada investigador sempre que es clarifiqui la manera en què ha estat interpretat.

### 3.2.1. Intel·ligència artificial als jocs de taula

La primera aplicació que va tenir la intel·ligència artificial al món dels videojocs va ser com a oponent en versions simulades dels jocs de taula més comuns (Millington, 2006). Segons explica aquest mateix autor, existeixen un seguit d’algoritmes bàsics que són aplicables a tots aquests jocs.

Els algoritmes més reconeguts i avançats per a jocs que funcionen per torns estan dissenyats per funcionar amb jocs per a dos jugadors, de suma zero i amb informació perfecta, seguint la classificació de la teoria de jocs anteriorment exposada (Millington, 2006).

### 3.2.1.1. Game Trees

Tal com Redfield, Gaither i Redfield (2008) expliquen, un Game Tree (en català, arbre de joc) és la representació d'una determinada partida d'un joc mitjançant nodes i connexions entre ells. El node principal o base es tracta de l'estat inicial del joc (o bé de l'estat en què aquest es troba en el punt a partir del qual es vol tenir en compte l'arbre), mentre que el contingut de la resta de nodes és l'estat en què aquesta mateixa partida es trobaria després d'un determinat moviment respecte del node que el precedeix. D'aquesta manera, tots els nodes representen un determinat estat de la partida, i cada un dels seus successors representa un moviment diferent que pot ser efectuat a partir d'aquest. Cada nivell del Game Tree pertanyerà als possibles moviments d'un jugador diferent de manera alternada, i si un estat concret de la partida no permet cap moviment, el seu corresponent node no tindrà cap successor; això representarà el final del joc (Redfield, Gaither i Redfield, 2008).

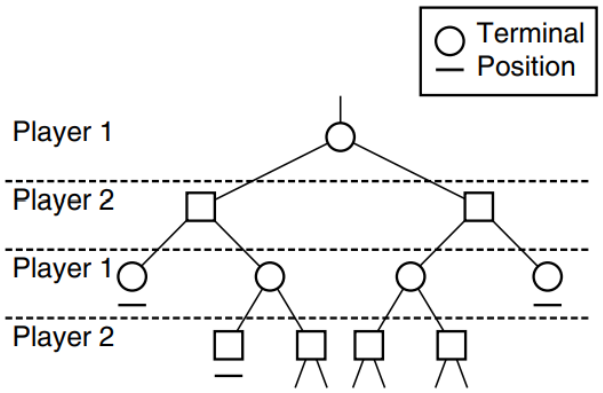


Figura 3.1. Exemple general de Game Tree. Font: Ian Millington, 2006.

En la Figura 3.1, podem comprovar com hi ha certs nodes que no tenen successors. Tal com s'ha esmentat anteriorment, aquest fet és degut a la inexistència de moviments un cop arribat aquest punt de la partida. Millington (2006) els anomena posicions o nodes terminals. També podem observar que, tal com s'ha exposat anteriorment, cada nivell de l'arbre correspon, de manera alternada, al torn d'un

dels dos jugadors. En el cas que mostra la figura, els nodes que corresponen al torn d'un jugador es mostren amb una forma circular, mentre que els que corresponen al seu oponent ho fan amb forma quadrada.

El nombre de branques que surtin d'un mateix node serà equivalent al nombre de moviments que el jugador té la possibilitat de dur a terme. Per tant, en el joc del tres en ratlla, per exemple, el primer jugador a tirar tindrà nou possibilitats diferents, i per tant el node base comptarà amb nou successors (Millington, 2006).

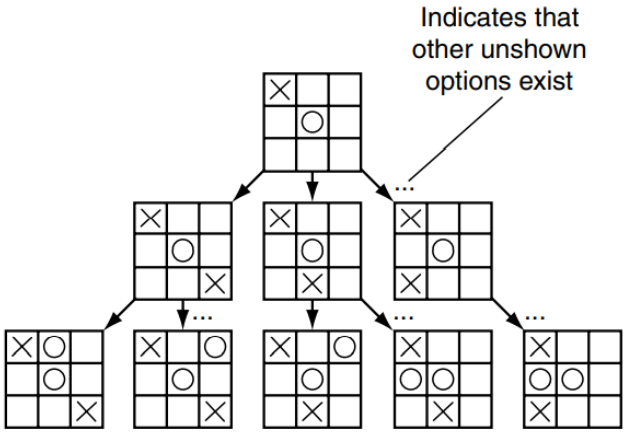


Figura 3.2. Exemple de Game Tree del tres en ratlla. Font: Ian Millington, 2006.

En l'anterior figura es mostra un Game Tree corresponent a una partida de tres en ratlla. En el node base s'hi pot observar que ja hi ha dues fitxes sobre el taulell, la qual cosa significa que l'arbre comença amb el tercer moviment de la partida. Cada cop que s'avança un torn, la profunditat de l'arbre creix un nivell més. Cada una de les bifurcacions suposen una nova possibilitat a partir d'aquell estat de la partida i obren, per tant, un nou camí possible (Millington, 2006).

Les dues principals propietats dels Game Trees son la profunditat i l'amplitud. La profunditat de l'arbre en defineix el nombre de nivells, cosa que significa que com més gran sigui, més extensa serà la simulació. L'amplitud del Game Tree la defineix el factor de ramificació d'aquest, el qual depèn del nombre de moviments possibles en cada torn. Com més elevat sigui aquest valor, més amplitud tindrà l'arbre (Finnson i Björnsson, 2011).

### 3.2.1.2. Funció d'avaluació

La funció d'avaluació permet comprovar si la partida ha acabat en victòria, derrota o empat. En cas que aquesta no es trobi en una situació terminal perquè l'algoritme ha estat aturat abans del final de la partida en comptar amb una profunditat màxima d'arbre, la funció retornarà una determinada puntuació que reflecteixi la situació del jugador segons com de prop es trobi de la victòria (Elnaggar, Gadallah, Aziem i El-Deeb, 2014).

Tal com exposa Millington (2006), en aquest punt és on recau la importància del coneixement del joc, doncs l'algoritme que s'hi apliqui no tindrà en compte cap qüestió estratègica. L'autor explica que, per tant, tota la informació estratègica necessària estarà inclosa en la funció d'avaluació.

Tot i que en la majoria de casos les funcions d'avaluació solen retornar nombres enters per tal de poder treballar millor amb els algoritmes aplicats, aquestes poden retornar qualsevol tipus de nombre de qualsevol mida. En una mateixa funció d'avaluació hi poden haver diversos mecanismes de puntuació actuant a la vegada. Per exemple, la funció pot tenir en compte diferents aspectes com el nombre de peces damunt el taulell o el nivell de perill en el qual es troben aquestes peces i, a continuació, combinar-los tots mitjançant una suma en la qual cada element tingui el seu propi pes (Millington, 2006).

Amb una correcta funció d'avaluació podran ser calculades totes les puntuacions de cada respectiu estat de la partida després de fer cada moviment possible, i per tant l'algoritme comptarà amb la informació necessària per poder prendre les decisions adients (Millington, 2006).

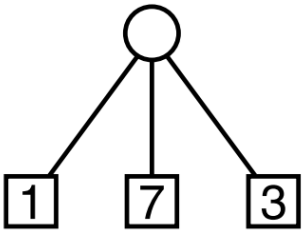


Figura 3.3. Game Tree amb puntuacions resultants. Font: Ian Millington, 2006.

La figura 3.3 mostra els tres moviments possibles que pot realitzar un jugador a partir d'un punt concret, així com la puntuació resultant de cada un d'ells. Aquesta és atorgada per la funció d'avaluació a partir de l'estat en què es troba la partida després del moviment en qüestió. L'arbre compta amb un sol nivell de profunditat.

#### 3.2.1.3. Adversarial Search

L'Adversarial Search (en català, cerca amb adversari) és una tècnica que analitza un joc amb adversari i determina quins moviments haurà de fer un jugador per tal de guanyar la partida (Wilson, Zuckerman, Parker i Nau, 2012).

L'Adversarial Search té lloc quan hi ha un enemic o oponent que intenta canviar l'estat de la partida a cada torn en una direcció contrària a la d'un determinat jugador. Aquest, per tant, pot canviar l'estat de la partida, però perd el control sobre el següent canvi d'estat, el qual serà efectuat per l'oponent d'una manera impredictible i hostil per al primer (Humphrys, s.d.).

Tal com expliquen Yannakakis i Togelius (2018), en aquests casos les accions de cada jugador dependran de manera directa de les accions dels seus contrincants, tenint d'aquesta manera una gran rellevància les unes sobre les altres.

## 3.3. Minimax

Minimax és un mètode de decisió que intenta predir el comportament del jugador oponent segons la situació en què la partida es troba, partint de la teoria que aquest sempre intentarà minimitzar la puntuació rival. (Humphrys, s.d.).

Minimax és el mètode bàsic d'Adversarial Search més comú, i és majoritàriament utilitzat en jocs de taula per a dos jugadors amb informació perfecta (Yannakakis i Togelius, 2018).

### 3.3.1 Funcionament

Segons Millington (2006), Minimax té en compte que, en el seu torn, sempre escollirà el millor moviment que pugui efectuar, aquell que el deixi en la millor posició possible respecte del seu adversari. Segons aquest mateix autor, però, Minimax assumeix que, en el torn del seu contrincant, aquest sempre escollirà

aquell moviment que més el perjudiqui, deixant-lo en la pitjor posició possible. D'aquesta manera, s'assumeix que un jugador sempre intenta maximitzar la seva puntuació, mentre que el seu oponent sempre prova de minimitzar-la al mínim. Aquesta alternança constant entre la maximització i la minimització és el que li dóna el nom a Minimax (Millington, 2006).

Començant per les darreres branques del Game Tree, els valors retornats en aquest punt per la funció d'avaluació aniran sent escollits seguint la regla principal de Minimax: al seu torn s'escollirà el valor més alt i, al de l'oponent, el més baix (Millington, 2006). D'aquesta manera, com explica el mateix autor, un punt s'hagi arribat al node base, s'haurà obtingut la puntuació resultant per a cada moviment possible, cosa que permetrà a Minimax escollir la millor opció a efectuar.

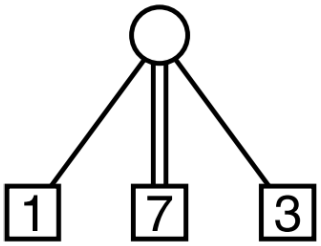


Figura 3.4. Decisió de moviment per maximització. Font: Ian Millington, 2006.

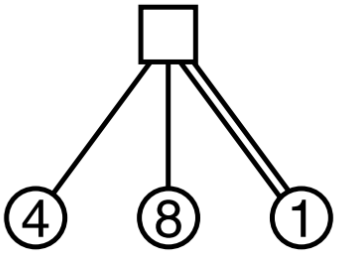


Figura 3.5. Decisió de moviment per minimització. Font: Ian Millington, 2006.

La Figura 3.4 mostra una decisió presa per Minimax en el seu torn, on s'agafa el valor més alt, mentre que la Figura 3.5 mostra una decisió presa en el torn de l'oponent, on s'escull el valor més petit.

### 3.3.2. L'algoritme

Tal com Millington (2006) exposa, l'algoritme de Minimax és recursiu i prova de calcular a cada node de l'arbre el valor de la posició actual del taulell comprovant tots els moviments possibles que poden derivar d'aquesta mateixa.

L'algoritme Minimax pot tenir una cerca exhaustiva o una profunditat màxima fixada. El primer cas pot ser utilitzat per a jocs petits o amb un nombre reduït de moviments, doncs l'algoritme no s'atura fins que arriba a la resolució final del joc. El segon cas, en canvi, és aplicat a jocs d'una magnitud superior i, per tant, amb un nombre més elevat de possibilitats de moviment, i consisteix en limitar la cerca amb una profunditat màxima d'arbre (Humphrys, s.d.).

Aquest segon cas és el de la gran majoria de jocs als quals aquest algoritme és aplicat, on difícilment s'arribarà a la condició de victòria, i per tant la funció d'avaluació haurà de ser aplicada per tal de donar un valor numèric a l'estat de la partida (Yannakakis i Togelius, 2018).

Cal tenir en compte que la funció d'avaluació només haurà de ser aplicada en les posicions terminals, les quals es trobaran en el nivell més baix, doncs el valor dels nivells més alts seran donats per l'algoritme aplicant maximització o minimització en funció del torn (Humphrys, s.d.).

```
def minimax(board, player, maxDepth, currentDepth):

  # Check if we're done recursing
  if board.isGameOver() or currentDepth == maxDepth:
    return [board.evaluate(player), None]

  # Otherwise bubble up values from below

  bestMove = None
  if board.currentPlayer() == player: bestScore = -INFINITY
  else: bestScore = INFINITY

  # Go through each move
  for move in board.getMoves():

    newBoard = board.makeMove(move)

    # Recurse
    [currentScore, currentMove] = minimax(newBoard, player,
                                  maxDepth, currentDepth+1)

    # Update the best score
    if board.currentPlayer() == player:
      if currentScore > bestScore:
        bestScore = currentScore
        bestMove = move
    else:
      if currentScore < bestScore:
        bestScore = currentScore
        bestMove = move

  # Return the score and the best move
  return [bestScore, bestMove]
```

Figura 3.6. Pseudocodi de Minimax. Font: Ian Millington, 2006.

L'anterior figura mostra el pseudocodi de l'algoritme Minimax el qual es basa en una funció que, tal com s'ha explicat anteriorment, serà recursiva per a cada node o estat a analitzar. La funció requereix certa informació per tal de dur a terme un correcte funcionament, com ara l'estat en què es troba el taulell en cada moment, el jugador a qui li toca jugar a cada torn, la profunditat actual de l'arbre, o la màxima profunditat a la qual aquest pot arribar. Alguns d'aquests valors caldrà passar-los com a paràmetre, doncs seran actualitzats a cada repetició de la funció a cada un dels diferents nodes. Aquesta funció comprovarà si el node actual es tracta d'una

posició terminal de la partida o si l'algoritme ha arribat a la profunditat màxima de l'arbre i, en cas de ser així, aplicarà la funció d'avaluació i retornarà la puntuació. En cas contrari, aplicarà aquesta mateixa funció per a tots els seus nodes successors (Millington, 2006).

El següent pas consistirà a comprovar quin és el jugador encarregat de realitzar el moviment en el torn en què la partida es troba en l'actual node. En funció d'això, es durà a terme la maximització o la minimització: en el primer cas, la funció escollirà el valor més gran d'entre tots els rebuts dels nodes successors mentre que, en el segon cas, n'escollirà el més petit. Pel que fa al pseudocodi de la figura 3.6, la funció retorna dues variables: el moviment escollit com a millor i la seva respectiva puntuació. Si el llenguatge de programació emprat no permet que una funció retorni dues variables, s'utilitzarà una única estructura de dades que contingui ambdues variables (Millington, 2006).

En aquest codi mostrat en l'anterior figura, cada instància de la classe Board representa una posició de joc. Aquesta podria representar-se de la següent manera:

```
class Board:
  function getMoves() -> Move[]
  function makeMove(move: Move) -> Board
  function evaluate(player: id) -> float
  function currentPlayer() -> id
  function isGameOver() -> bool
```

Figura 3.7. Pseudocodi de la classe Board. Font: Ian Millington, 2006.

Tal com mostra la figura 3.7, la classe Board conté les funcions per conèixer tota la informació rellevant respecte l'estat de la partida, així com el que en pot derivar. La mateixa classe s'encarrega de calcular tots els moviments possibles i aplicar-los a l'estat de la partida, així com de calcular el valor de l'estat actual de la partida mitjançant la funció d'avaluació. De la mateixa manera, també podrà comprovar si la partida es troba en un estat terminal (Millington, 2006).

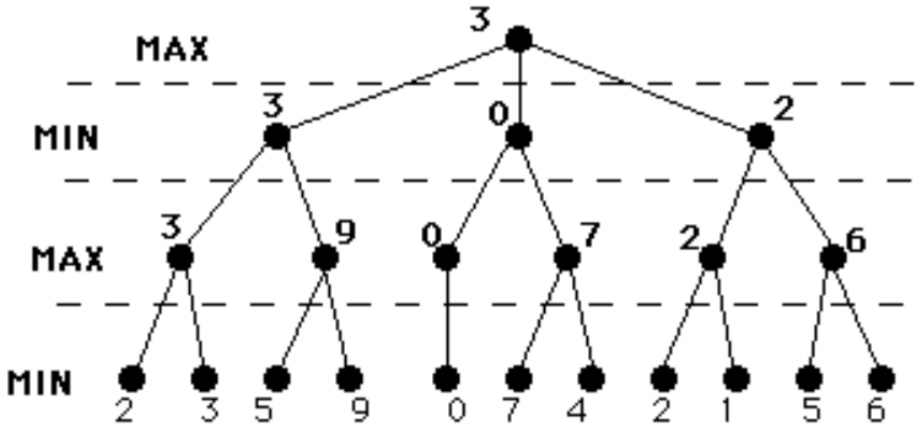


Figura 3.8. Exemple d'aplicació de Minimax. Font: Mark Humphrys, s.d..

L'anterior figura mostra un exemple gràfic d'un Game Tree amb l'aplicació de l'algoritme Minimax. La profunditat màxima en aquest cas és de tres nivells i, per tant, un cop s'arribi a aquesta profunditat d'arbre, es retorna el valor de l'estat de la partida, tant si es tracta de posicions terminals com si no; en aquest segon cas, s'aplica la funció d'avaluació per obtenir-ne la puntuació adequada. Seguidament els valors comencen a ascendir fins a arribar al node base. Els nivells parells corresponen als torns de l'oponent i per tant s'escull el millor moviment aplicant minimització: s'escull el camí que porta al node successor amb una puntuació més baixa. Contràriament, en els nivells imparells s'aplicarà la maximització, escollint els camins que porten al node successor amb la puntuació més alta (Humphrys, s.d.).

### 3.3.3. Alpha-Beta Pruning

Russell i Norvig (1995) expliquen que l'elevat nombre de nodes a analitzar, tenint en compte l'explosió combinatòria present a mesura que la profunditat del Game Tree va augmentant, pot arribar a suposar un problema: si l'algoritme Minimax no és capaç de fer-ho i se n'ha de reduir la profunditat màxima, aquest comptarà amb una menor visió de futur i menys anticipació respecte l'oponent i es convertirà en un rival amb un nivell més baix. Aquests mateixos autors, però, expliquen que és possible obtenir el millor moviment possible utilitzant Minimax sense la necessitat de comprovar cada node del Game Tree a través d'un procés anomenat Pruning.

El Pruning (en català, poda) és el procés de descartar totes aquelles branques del Game Tree que no poden influenciar en la seva decisió final i, per tant, reduir d'aquesta manera el nombre de nodes a analitzar. La tècnica més comuna de Pruning, la qual pot ser pot ser aplicada a Minimax, és l'Alpha-Beta Pruning (Russell i Norvig, 1995).

Tal com expliquen Russell i Norvig (1995), l'Alpha-Beta Prunning rep aquest nom pels dos paràmetres que marquen el límit dels valors que van sorgint a mesura que es desenvolupa la tècnica d'elecció al Game Tree:

- Alpha (α): Es tracta del millor valor per a la maximització, i per tant es tractarà del valor més alt.
- Beta (β): Es tracta del millor valor per a la minimització, i per tant es tractarà del valor més petit.

En aplicar maximització en un determinat node, s'actualitzarà el valor d'alpha amb el valor més gran obtingut dels nodes successors, mentre que en els nodes que s'hi apliqui minimització, el valor actualitzat serà el de beta amb el mínim valor obtingut. En qualsevol d'ambdós casos, s'efectuarà la poda quan el valor d'alpha sigui igual o superior al de beta (Russell i Norvig, 1995).

Tenint en compte que Minimax executa la cerca seguint primer la profunditat del Game Tree, l'Alpha-Beta Pruning només ha de tenir en compte els nodes que segueixen un mateix camí en l'arbre. Per aquest mateix motiu, però, cal tenir en compte que l'efectivitat de l'Alpha-Beta Pruning no serà sempre la mateixa: dependrà de l'ordre en què els nodes successors siguin examinats (Russell i Norvig, 1995).

```
def abMinimax(board, player, maxDepth, currentDepth,
              alpha, beta):

  # Check if we're done recursing
  if board.isGameOver() or currentDepth == maxDepth:
    return [board.evaluate(player), None]

  # Otherwise bubble up values from below

  bestMove = None
  if board.currentPlayer() == player: bestScore = -INFINITY
  else: bestScore = INFINITY

  # Go through each move
  for move in board.getMoves():

    newBoard = board.makeMove(move)

    # Recurse
    [currentScore, currentMove] = minimax(newBoard, player,
                                  maxDepth, currentDepth+1,
                                  alpha, beta)

    # Update the best score
    if board.currentPlayer() == player:
      if currentScore > bestScore:
        bestScore = currentScore
        bestMove = move
        if bestScore > alpha:
          alpha = bestScore
        if beta <= alpha:
          break
    else:
      if currentScore < bestScore:
        bestScore = currentScore
        bestMove = move
        if bestScore < beta:
          beta = bestScore
        if beta <= alpha:
          break

  # Return the score and the best move
  return [bestScore, bestMove]
```

Figura 3.9. Pseudocodi de Minimax amb Alpha-Beta Pruning. Font: Elaboració pròpia a partir de Ian Millington, 2006.

La figura 3.9 mostra el pseudocodi de l'Alpha-Beta Pruning aplicat a Minimax. El funcionament de la funció és el mateix que el de l'algoritme Minimax per si sol, exposat anteriorment en la figura 3.6, tret que en aquest cas es tindran presents dues variables més: l'alpha, la qual serà actualitzada amb el valor més alt en els nivells de l'arbre en què s'apliqui la maximització, i la beta, la qual rebrà el mínim valor en els nodes on s'apliqui minimització. La funció rep aquests dos valors com a paràmetre, doncs podran variar a cada repetició on aquesta sigui cridada.

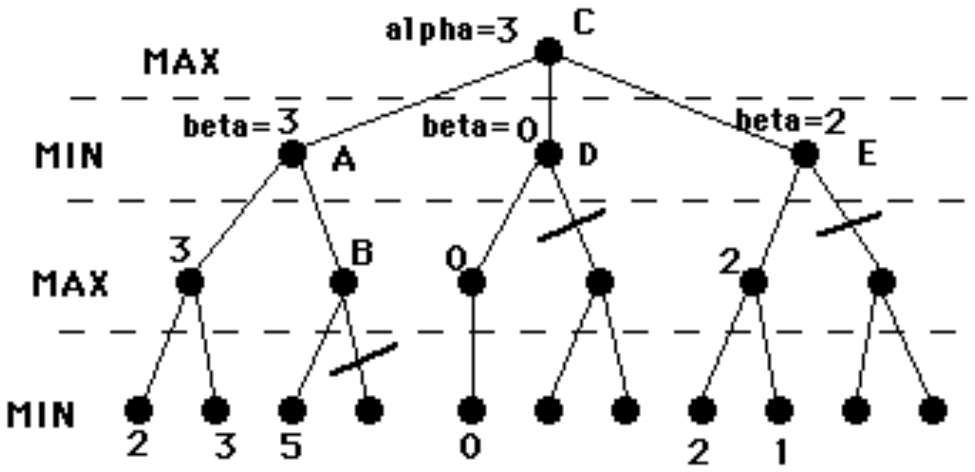


Figura 3.10. Exemple d'aplicació de Minimax i Alpha-Beta Pruning. Font: Mark Humphrys, s.d..

L'anterior figura mostra de manera gràfica un exemple de la utilització d'Alpha-Beta Pruning. Es tracta del mateix Game Tree que en la figura 3.8, però en aquesta ocasió s'utilitza aquesta estratègia de poda per a optimitzar l'ús de Minimax. En el cas de la primera poda que es du a terme es pot observar com el primer successor del node B té un valor de 5, i per tant se li dóna aquest valor a alpha. Com aquest és més gran que beta (3), és sabut en cap cas tindrà cap repercussió a l'hora de completar el camí amb la millor opció, i per tant es du a terme la poda. En el cas de les dues següents podes efectuades, es coneixen els valors dels primers nodes successors de D i E, els quals donaran el valor a beta. Tenint en compte que els dos són inferiors a alpha (3), s'efectua la poda i s'atura la cerca a partir d'aquests nodes (Humphrys, s.d.).

## 3.4. Dificultat adaptativa

Un dels reptes més importants amb el que es troben els desenvolupadors i dissenyadors de videojocs és el fet d'ajustar correctament la dificultat. Per tal d'aconseguir això, el joc ha de ser prou complicat per a generar un cert repte al jugador i que no se li faci avorrit, però a la vegada no ha de tenir una dificultat excessivament elevada per tal que no acabi sent massa frustrant ni descompensat. Els creadors de jocs no poden dependre de cap eina que els ajudi amb aquesta tasca, doncs no existeix una definició clara i acceptada del concepte de dificultat com a paràmetre mesurable. D'aquesta manera, l'ajustament de la dificultat d'un joc es converteix en un procés iteratiu i totalment subjectiu (Aponte, Levieux i Natkin, 2009).

El fet de determinar una certa dificultat estàtica i invariable pot provocar certs desajusts entre el nivell d'habilitat del jugador i el repte que el joc li plantegi. Una alternativa per tal d'evitar aquest problema és l'ajustament dinàmic de la dificultat, el qual planteja sistemes que van responent al llarg de la sessió de joc en funció de les habilitats del jugador. Tenint això en compte, el fet d'automatitzar i adaptar la dificultat d'un joc segons el mateix jugador, a la llarga, pot acabar fent d'aquest joc quelcom més atractiu i interessant per al públic (Hunicke, 2005; Missaura i Gaertner, 2010).

## 3.5. Les dames angleses

Segons la Teoria de Jocs i la seva classificació d'aquests anteriorment exposada, les dames angleses es tracta d'un joc per a dos jugadors, amb informació perfecta i amb suma zero (Millington, 2006).

El joc de les dames angleses es juga amb un taulell de 64 caselles (8x8) que alternen entre el color negre i el blanc, de la mateixa manera que en el cas dels escacs. Cada jugador compta amb 12 peces iguals, les quals poden ser anomenades peons. Les peces dels dos jugadors es distingeixen amb dos colors diferents, que seran els mateixos que els del taulell (Chess & Checkers Games, s.d.).

### 3.5.1. Regles del joc

L'objectiu de cada jugador és aconseguir que el seu oponent es quedi sense peces damunt el taulell o bé que no pugui efectuar cap moviment amb les que li queden. El primer a assolir aquesta fita es convertirà en el vencedor de la partida (Chess & Checkers Games, s.d.).

A l'inici, cada un dels dos jugadors col·locarà les peces distribuïdes en les tres files que té més a prop ocupant només les caselles de color negre, de manera que comptarà amb quatre peons a cada una de les files. Per al joc de les dames angleses, només seran utilitzades les caselles d'aquest mateix color (Chess & Checkers Games, s.d.).

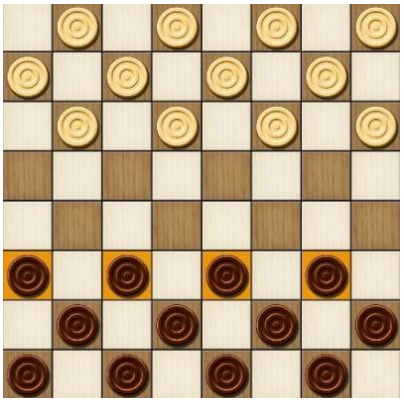

Figura 3.11. Joc de les dames: posició inicial. Font: Draughts (Chess & Checkers Games, 2015)

El jugador que té les peces negres és l'encarregat d'efectuar el primer moviment. Cada jugador disposa d'un sol moviment per a cada torn, i només podrà moure les seves pròpies peces. Els peons poden desplaçar-se en diagonal i sempre cap endavant sense opció a retrocedir, avançant una sola casella en cada moviment. El moviment només es podrà realitzar sempre que la casella de destí no estigui ocupada per cap altre peça. En cas que un peó travessi tot el taulell i arribi a l'última fila de caselles possible, aquella que queda més a prop de l'oponent, aquest es convertirà en una dama. La dama també es mourà desplaçant-se en diagonal per les caselles del taulell d'una en una, de la mateixa manera que el peó. Aquesta, però, no només podrà fer-ho cap endavant, en direcció a l'oponent, sinó que també podrà retrocedir cap enrere. En algunes variants, la dama pot desplaçar-se un

nombre il·limitat de caselles, tant cam endavant com cap enrere, sempre i quan ho faci en diagonal i cap altre peça obstaculitzi el seu pas (Chess & Checkers Games, s.d.).

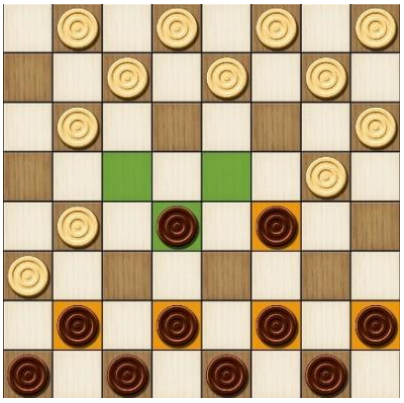

Figura 3.12. Joc de les dames: moviment del peó. Font: Draughts (Chess & Checkers Games, 2015)

Figura 3.13. Joc de les dames: moviment de la dama. Font: Draughts (Chess & Checkers Games, 2015)

En les dues anteriors figures, els requadres pintats de verd remarquen la peça que realitzarà el moviment, així com totes les caselles a les quals es podrà desplaçar. La figura 3.12 mostra el moviment d'un peó: tot i tenir lliures les quatre diagonals, només pot moure's cap endavant. La dama, en canvi, pot desplaçar-se en qualsevol de les quatre diagonals mentre aquestes no estiguin obstaculitzades per cap altra peça, tal com mostra la figura 3.13, doncs aquesta té la possibilitat de moure's tant endavant com enrere (Chess & Checkers Games, s.d.).

Per tal d'eliminar una peça del seu oponent, el jugador haurà de saltar per damunt d'ella amb una de les seves peces. Aquesta acció es podrà dur a terme si la peça rival es troba a una casella de distància en diagonal de la peça del jugador, i la següent casella seguint la mateixa direcció es troba buida. Cada un dels dos tipus de peces poden eliminar peces rivals en la mateixa direcció en què tenen permès moure's: el peó únicament cap endavant i la dama tant endavant com enrere. En cas que en eliminar una peça rival la peça del jugador quedi en una posició que li permetria saltar una altra peça adversària, també eliminarà aquesta segona peça seguint el mateix procediment de la primera. Aquesta successió de captures no es podrà interrompre mentre hi hagi la possibilitat de seguir eliminant peces. D'altra banda, sempre que un jugador tingui la possibilitat d'eliminar una o més peces, estarà obligat a fer-ho (Chess & Checkers Games, s.d.).

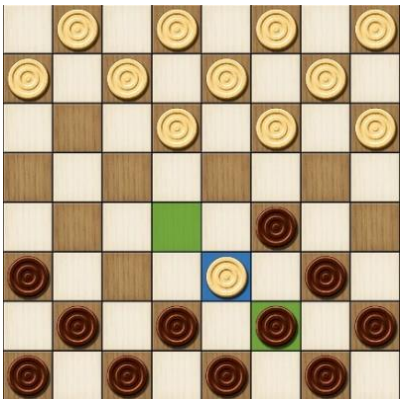

Figura 3.14. Joc de les dames: captura d'una peça rival. Font: Draughts (Chess & Checkers Games, 2015)

Figura 3.15. Joc de les dames: captura múltiple. Font: Draughts (Chess & Checkers Games, 2015)

Les dues figures anteriors mostren dos exemples de captures de peces rivals. En color verd es mostren les caselles on comença i acaba el moviment que realitzarà la peça que durà a terme la captura, mentre que en color blau es mostren la casella o caselles que saltarà la peça en qüestió per efectuar el moviment. En la figura 3.14, el jugador amb les peces negres realitza una captura d'una de les peces rivals. En la figura 3.15, s'observa una captura múltiple: de la manera que s'ha exposat anteriorment, es capturaran dues peces rivals amb un sol moviment (Chess & Checkers Games, s.d.).

El joc de les dames no sempre acaba en victòria per a un dels dos jugadors, sinó que també compta amb la possibilitat que la partida acabi en empat. Aquest fet es dona en dos casos diferents: quan es realitzen vint moviments consecutius només ama dames i sense eliminar cap peça rival, o bé quan la mateixa situació de taulell es dona tres vegades en la mateixa partida (Chess & Checkers Games, s.d.).

### 3.5.2. Funció d'avaluació del taulell

Tot i que no hi ha un consens general sobre l'opció més adequada per a la funció d'avaluació per al joc de les dames, en aquest apartat es posen diversos exemples d'autors que han proposat diferents models per a aquesta funció, tenint en compte diferents elements i característiques a analitzar.

En primer lloc, tal com expliquen Schaeffer, Lake, Lu i Bryant (1996), Chinook és el nom que rep la primera intel·ligència artificial que va enfrontar-se a un campió mundial del joc de les dames per tal d'intentar arrabassar-li aquest títol i, posteriorment, la primera a aconseguir-ho. Segons expliquen aquests mateixos autors, que van ser-ne els desenvolupadors, Chinook compta amb una funció d'avaluació que s'encarrega d'analitzar la situació que es troba sobre el taulell.

Aquesta funció d'avaluació analitza certs aspectes del taulell atorgant-hi una puntuació, la qual és multiplicada per un valor predeterminat que correspon al seu grau de rellevància en comparació a les altres característiques del taulell analitzades i que, per tant, en determinarà el seu pes en la suma de puntuació final. Alguns dels aspectes que l'algoritme té en compte i als quals dona una major rellevància són el recompte de peons de cada jugador, el recompte de dames de

cada jugador, a qui dels dos jugadors li toca tirar a continuació i quines peces es troben atrapades sense cap opció de moure's (Baba i Jain, 2001).

Per una altra banda, la funció d'avaluació que utilitzen Shuqin, Weiming i Xiaohua (2015) per al joc de les dames només té en compte tres aspectes del taulell: el nombre de peons de cada jugador, el nombre de dames de cada jugador i el nombre de peces en perill per a cada jugador. A més, aquests mateixos autors destaquen la importància del fet que cada un dels diferents valors obtinguts tingui el seu propi pes en funció de la seva rellevància i sigui tingut en compte a l'hora d'avaluar la situació del taulell.

La funció d'avaluació per a aquest mateix joc que, d'altra banda, proposen Evans i Sable (s.d.) contempla, de major a menor importància, els següents aspectes de la situació de la partida: quin jugador té millors peces sobre el taulell donant major pes a les dames, quins peons es troben més a prop de convertir-se en dames, el recompte total de les peces que es troben sobre el taulell i quin jugador té les dames més ben posicionades.

Finalment, Kusiak, Waledzik i Mandziuk (2007) proposen per al seu treball una nova funció d'avaluació que analitza dos tipus de característiques diferents: les simples, i les de posició. En el primer bloc, l'avaluació del taulell té en compte per a cada un dels jugadors el nombre de peces de cada tipus, quines d'elles es troben assegurades, quines tenen possibilitat de moure's i quins peons es troben a prop de convertir-se en dames. Pel que fa a les característiques de posició del taulell, la funció té en compte alguns altres aspectes com ara el nombre de peces que es troben en posició atacant, el nombre de peces que es troben en posició defensiva o el nombre de peces que es troben en una posició central (Kusiak, Waledzik i Mandziuk, 2007).

A continuació, la taula 3.3 mostra de manera resumida les diferents característiques analitzades per les funcions d'avaluació proposades per cada un dels autors esmentats.

| Autors | Elements analitzats per la funció | Utilització de pesos |
|---|---|---|
| Schaeffer, Lake, Lu i Bryant (1996); Baba i Jain (2001) | - Nombre de peons<br>- Nombre de dames<br>- Torn<br>- Peces immobilitzades | Sí |
| Shuqin, Weiming i Xiaohua (2015) | - Nombre de peons<br>- Nombre de dames<br>- Peces amenaçades | Sí |
| Evans i Sable (s.d.) | - Valoració de les peces<br>- Peons a prop de ser dames<br>- Recompte de peces<br>- Posició de les dames | Sí |
| Kusiak, Waledzik i Mandziuk (2007) | *Simples*:<br>- Nombre de peons<br>- Nombre de dames<br>- Peces assegurades<br>- Peces movibles<br>- Peons a prop de ser dames<br>*Posicionals:*<br>- Posició atacant<br>- Posició defensiva<br>- Posició central | No especificat |

Taula 3.3. Propostes de funció d'avaluació per al joc de les dames. Font: Elaboració pròpia

# 4. Referents

En els seus treballs, alguns autors presenten estudis realitzats envers el concepte de dificultat adaptativa o dinàmica en jocs que poden ser utilitzats com a referents. N'és un exemple el treball efectuat per Andrade, Santana, Furtado, Leitão i Ramalho (2003), que en el seu estudi presenten un agent de dificultat adaptativa basat en Reinforcement Learning (en català, aprenentatge per reforç), el qual aprèn a escollir quines accions realitzar en un determinat entorn per tal de maximitzar-ne la recompensa obtinguda.

A través del Q-Learning, una tècnica derivada del Reinforcement Learning, aquests autors anteriorment anomenats utilitzen un agent que controla la dificultat del joc. Ordinàriament, el Q-Learning defineix el conjunt d'accions disponibles que hi ha en un determinat estat del joc i les classifica en funció de quines opcions donen un millor resultat, escollint-ne, finalment, la millor. En aquest cas, però, l'agent proposat pels autors tria l'opció en funció del nivell de dificultat que percep segons comportament del jugador. D'aquesta manera, si, per exemple, el joc sembla ser massa difícil per al jugador, l'agent tria una opció pitjor que l'anteriorment triada per tal d'ajustar la dificultat a un nivell més fàcil (Andrade, Santana, Furtado, Leitão i Ramalho, 2003).

Per tal de comprovar l'efectivitat de l'agent, aquest és enfrontat a altres agents basats en algoritmes existents en un joc de lluita per combats. Tot i que el resultat resulta satisfactori, es remarquen alguns problemes associats al Reinforcement Learning, com ara la gran quantitat de temps que l'agent tarda a aprendre del jugador al qual s'enfronta (Andrade, Santana, Furtado, Leitão i Ramalho, 2003).

Un altre referent pel que fa a la dificultat adaptativa en jocs és l'estudi realitzat per Ilicim, Wang, Missura i Gärtner (2012), en el qual s'investiga l'efectivitat d'un algoritme per a un ajustament dinàmic d'aquesta. Els jocs triats per a aquest estudi són les dames i els escacs xinesos, mentre que l'algoritme utilitzat serà el Partially Ordered Set Master, també anomenat POSM per les seves sigles, presentat per dos dels autors esmentats en un treball dut a terme anteriorment per ells mateixos.

A grans trets, l'objectiu de l'algoritme POSM és predir la dificultat correcta a partir d'un conjunt parcialment ordenat de configuracions o paràmetres possibles dins el joc. Aquest algoritme compta només amb tres possibles respostes: positiva si l'ajustament dels paràmetres resulta massa fàcil per al jugador, negativa si és massa difícil, i neutre si és adequada. En els dos primers casos, els paràmetres es reajustaran per tal d'adequar-se a la dificultat prevista (Ilicim, Wang, Missura i Gärtner, 2012).

Tant en el cas de les dames com en el dels escacs xinesos, l'experiment es du a terme enfrontant l'algoritme POSM a altres algoritmes. En ambdós casos els resultats són favorables i es pot concloure que l'ajustament dels paràmetres i de la dificultat del joc s'ha efectuat de manera correcta. Això no s'ha complert, però, en certs casos excepcionals on l'algoritme POSM s'ha enfrontat a un algoritme que ha actuat de manera aleatòria o que ha suposat ser superior a ell.

Els dos anteriors treballs serveixen de referents pel que fa a la cerca d'un agent o algoritme que adapti la dificultat d'un joc en funció de les accions del jugador. Tot i que el segon cas utilitza jocs de taula per a dur a terme l'experiment, l'algoritme dissenyat no deriva de Minimax. Ambdós casos, però, serveixen de referent pel que fa a la metodologia seguida a l'hora d'avaluar el funcionament de l'algoritme dissenyat. També cal tenir en consideració, però, que en cap dels dos casos s'avalua l'enfrontament de l'algoritme contra jugadors reals.

Pel que fa a la dificultat adaptativa d'un joc de taula utilitzant Minimax, un referent a tenir en compte és l'estudi dut a terme per Missura i Gaertner (2010). En aquest treball proposen un agent adaptatiu que s'adequa segons les accions del jugador durant el transcurs de la mateixa partida per tal d'avaluar els resultats obtinguts d'aquest.

El joc escollit per Missura i Gaertner (2010) per a aquest esmentat treball és el quatre en ratlla. Tenint en compte que l'agent adaptatiu que utilitzen per a aquest estudi parteix de l'algoritme Minimax, ells l'anomenen AdaptiveMiniMax. Per a aquest estudi, parteixen de la consideració que l'algoritme comptarà amb una cerca exhaustiva, la qual cosa vol dir que aquest utilitzarà un Game Tree complet sense cap profunditat màxima, quan només arribarà a la fi quan es doni una condició de

victòria, derrota o empat, la qual suposi la fi de la partida, comptant com a neutral qualsevol altra situació.

Per tal d'obtenir els resultats, Missura i Gaertner (2010) enfronten l'AdaptiveMiniMax en partides al joc del quatre en ratlla contra jugadors i també contra altres algoritmes. Mentre que en aquest segon cas els resultats obtinguts mostren una correcta adaptació de l'agent envers cada un dels diferents algoritmes contra els quals juga, en el cas dels jugadors humans els resultats no mostren cap correlació entre el nivell real del jugador i el nivell que l'agent interpreta, doncs aquest fet no es veu reflectit en la ràtio de victòries d'aquest. Els autors exposen a les conclusions que creuen que aquest fet es pot donar al fet que, tot i que això no acaba tenint cap repercussió per al resultat de la partida, als inicis del joc els jugadors efectuen moviments allunyats del més òptim, doncs amb una situació del taulell poc desenvolupada pel que fa al transcurs de la partida és complicat preveure a llarg termini per part del jugador.

Aquest darrer cas, a diferència dels altres dos estudis exposats anteriorment, sí que parteix de Minimax per tal de dissenyar el nou algoritme. D'aquesta manera, per tant, no només és referent per la metodologia que utilitza durant l'experiment i avaluació del funcionament de l'algoritme, sinó que també serveix de referent pel que fa als resultats obtinguts, doncs a partir d'aquests es pot encarar el disseny del prototip tenint en compte els errors i punts negatius que s'han donat per tal d'enfocar des d'un bon principi la manera d'evitar-los.

# 5. Disseny metodològic i cronograma

En aquest capítol del treball s'explica la metodologia a seguir durant la fase pràctica d'aquest, així com la planificació a seguir per tal de dur-ho a terme.

## 5.1. Metodologia

La metodologia que s'aplicarà per tal de dur a terme aquest treball constarà principalment de dues fases diferents. La primera fase tractarà de l'elaboració de l'algoritme, així com de la seva implementació en un prototip del joc de les dames angleses. La segona fase tractarà de l'avaluació de la versió final de l'algoritme mitjançant el prototip implementat, per tal d'analitzar-ne el funcionament i poder treure'n les respectives conclusions.

### 5.1.1. Disseny de l'algoritme i implementació del prototip

Aquesta primera etapa consistirà en la preparació i implementació d'allò que serà necessari per efectuar la posterior anàlisi: l'algoritme i el prototip del joc.

Prèviament a tot això, però, caldrà un estudi més a fons de tots els referents que poden ser útils pel que fa a aquest aspecte, destacant el treball de Missura i Gaertner (2010), per tal d'avaluar, a partir dels seus resultats i conclusions, en quins punts i aspectes cal focalitzar l'atenció. A continuació, s'implementarà el prototip amb l'algoritme plantejat.

L'eina que s'utilitzarà per a l'elaboració del prototip serà Unity, tenint en compte que es tracta d'un motor que destaca per, entre d'altres, la seva adequació al prototipatge de jocs, doncs permet fer-ho d'una manera eficient i flexible. El llenguatge de programació utilitzat per a l'algoritme serà C#, doncs és el llenguatge suportat per aquest motor. Aquest, a més, es tracta d'un llenguatge de programació orientat a objectes, la qual cosa facilitarà treballar amb classes.

Durant aquesta primera fase de desenvolupament, s'utilitzarà la metodologia Waterfall. Aquesta metodologia consisteix en un procés de desenvolupament seqüencial, el qual comença amb unes fases inicials d'anàlisi i disseny, a continuació segueix amb el desenvolupament, i per últim finalitza amb les fases de

testeig. D'aquesta manera, es podrà seguir un ordre i una determinada cronologia a l'hora de dissenyar i implementar tant l'algoritme com el prototip. A més, dins d'aquestes fases, els temps aniran marcats per dates límit que serviran per portar al dia cada un dels punts a realitzar. Durant la darrera part d'aquesta primera fase, la qual es destina al testeig, alguns trets de l'algoritme o de la funció d'avaluació del taulell podran canviar i ser redissenyats de manera iterativa per tal d'obtenir el millor balanç possible entre l'eficiència i l'eficàcia d'aquest.

### 5.1.2. Avaluació de l'algoritme

Un cop realitzat el prototip, podrà tenir lloc la segona fase del projecte, la qual consistirà en l'avaluació de la versió definitiva de l'algoritme a través d'aquest i l'obtenció i anàlisi de resultats.

Seguint una metodologia similar a la d'estudis referents com el d'Andrade, Santana, Furtado, Leitão i Ramalho (2003) o el de Missura i Gaertner (2010), a través del prototip creat, l'algoritme implementat serà enfrontat al joc de les dames a alguns rivals. Aquests rivals podran ser, per una banda, algoritmes com ara ell mateix amb una determinada dificultat preestablerta o el Minimax ordinari i, per una altra banda, jugadors humans de diferents nivells. L'algoritme s'enfrontarà a cada oponent diverses vegades.

A partir d'aquestes partides, s'avaluarà el rendiment de l'algoritme de la següent manera:

- En el cas dels algoritmes amb una dificultat predeterminada, s'observarà si l'algoritme en qüestió adequa la dificultat al mateix nivell que la de l'oponent.
- En el cas dels oponents humans s'avaluarà, per una banda, si la dificultat que l'algoritme en qüestió estableix per a un determinat rival coincideix en les diferents partides que ha jugat contra ell, i per una altra banda, si existeix alguna correlació entre el nivell mitjà que l'algoritme li atorga a cada jugador i el percentatge de partides que aquest ha aconseguit vèncer.

A partir dels resultats que s'obtinguin d'aquest estudi, es podrà dur a terme una anàlisi i extreure'n les conclusions pertinents.

## 5.2. Planificació

Seguint el calendari acadèmic i tenint en compte el temps del qual es disposa per a l'elaboració d'aquest projecte, es compta amb una planificació a seguir. Aquesta planificació es pot veure reflectida gràficament en la figura 5.1, mostrada a continuació, la qual presenta un cronograma que estableix els diferents punts a treballar i fases del treball en el marc temporal adequat.

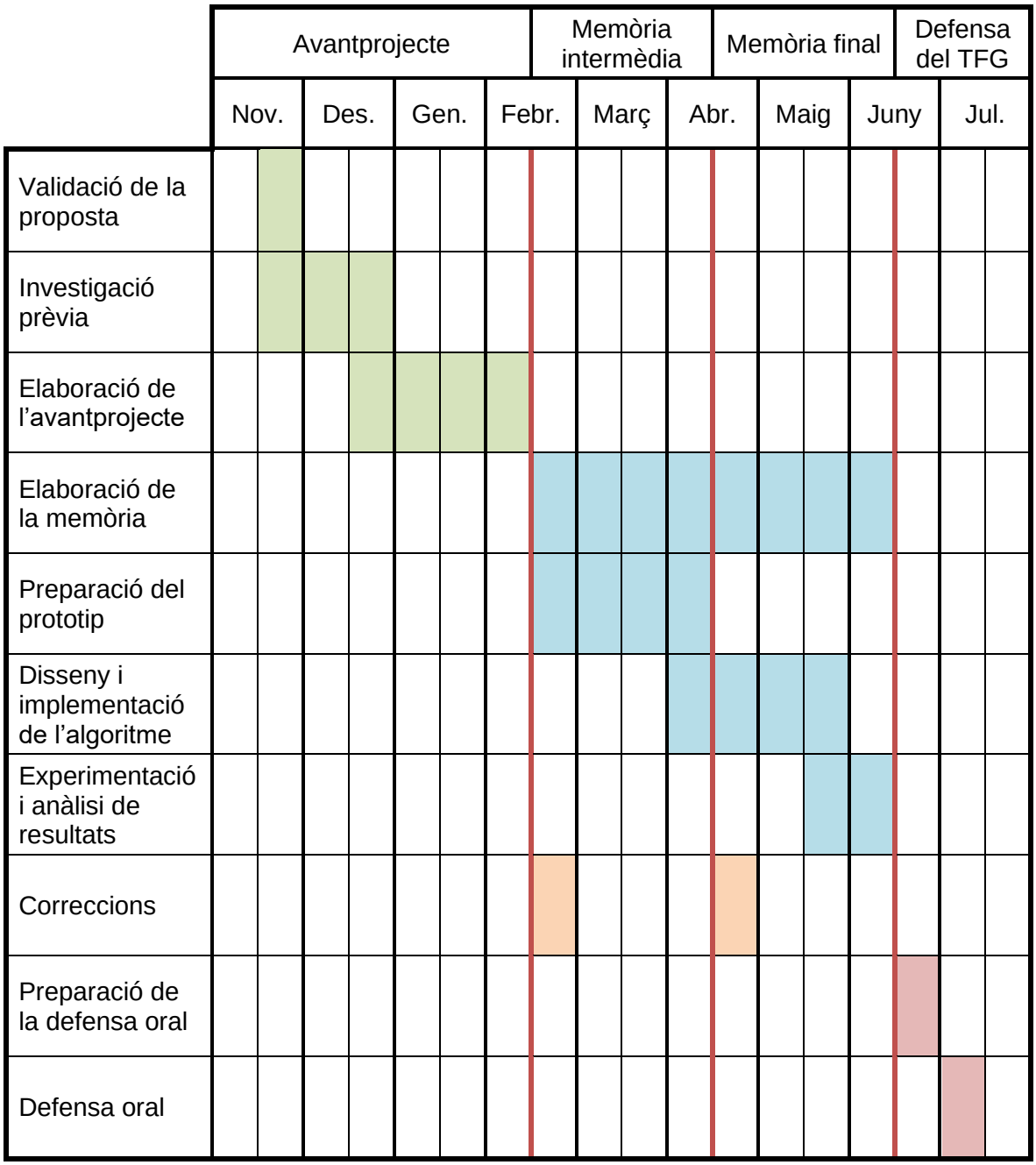


Figura 5.1. Cronograma. Font: Elaboració pròpia

# 6. Desenvolupament

En aquest apartat del treball es mostra el desenvolupament dut a terme per tal d'assolir els objectius plantejats, així com els resultats obtinguts a través de les diferents anàlisis efectuades per a comprovar el seu funcionament. D'aquesta manera, per tant, es distingeixen les diferents fases i versions de l'algoritme amb les seves respectives proves, i es mostra la iteració duta a terme sobre el mateix per tal d'obtenir els millors resultats possibles.

## 6.1. Preparació del prototip

La primera tasca que s'ha dut a terme per a la part de desenvolupament d'aquest treball és la implementació d'un prototip preparat per ser funcional un cop s'hi afegeixi un determinat algoritme que jugui contra el jugador o, en la seva absència, contra ell mateix o un altre algoritme. Tal com s'ha esmentat anteriorment en aquest mateix treball, l'eina utilitzada per a la realització d'aquest prototip és el motor de joc Unity i el llenguatge de programació emprat és C#. El paquet d'*assets* utilitzat per a la part visual del joc es tracta de *Board Games*, proveït pel creador de contingut *Jarst* (2019) i distribuït per la *Unity Asset Store*, el mercat d'*assets* del mateix motor.

L'arquitectura de classes que s'utilitza per a la implementació del prototip s'estructura de la següent manera:

- Classe *GameManager*:

La classe *GameManager* es tracta de l'única classe *MonoBehaviour*, doncs és l'encarregada d'executar el codi durant la partida i de gestionar aquesta a tall de controlador. Aquesta diferencia entre tres modes de joc diferents: el jugador amb peces negres contra l'algoritme, el jugador amb peces blanques contra l'algoritme, i l'algoritme contra ell mateix o contra un altre algoritme, deixant l'usuari com a espectador. La partida serà gestionada en funció del mode de joc seleccionat, alternant entre ambdós torns a cada moviment tot seguint el bucle de joc. L'estat de la partida es trobarà desat en una instància de la classe *Board*, la qual anirà sent actualitzada al llarg de la partida a cada moviment que s'efectuï.

```
def Update():

  # Update depending on the game mode
  switch gameMode
    case GameMode.Black_VS_AI:
      if gameBoard.currentTurn == gameBorad.Turn.Black:
        PlayerTurn()
      else:
        AITurn()
      break
    case GameMode.White_VS_AI:
      if gameBoard.currentTurn == gameBorad.Turn.White:
        PlayerTurn()
      else:
        AITurn()
      break
    case GameMode.AI_VS_AI:
      AITurn()
      break

def PlayerTurn():

  # Check if the player selects a piece and get its moves
  if pieceSelected:
    if jumpsAvailable:
      selectedPieceMoves = gameBoard.GetPieceJumps(piece)
    else:
      selectedPieceMoves = gameBoard.GetPieceMoves(piece)

  # Check if the player selects a square where the selected
    piece can be moved
  if squareSelected:
    for move in selectedPieceMoves:
      if square == move.to:
        gameBoard.MakeMove(move)
        UpdateBoard(move)
        ChangeTurn()

def AITurn()

  # Choose and make the best move if any
  chosenMove = Algorithm.Minimax()
  if chosenMove == null:
    GameOver()
  else:
    gameBoard.MakeMove(move)
    UpdateBoard(move)
    ChangeTurn()
```

```
def ChangeTurn()

  gameBoard.ChangeTurn()

  # Game Over if the player cannot keep moving
  if playerTurn and noMoves
    GameOver()
```

Figura 6.1. Pseudocodi de la classe *GameManager*. Font: Elaboració pròpia

Aquesta classe, a més, s'encarrega de mostrar tant l'estat de la partida com tot el que hi succeeix de forma visual. D'aquesta manera, actualitzarà l'estat del taulell a cada moviment de manera que el jugador o espectador pugui seguir la partida. De la mateixa manera, també proporcionarà el feedback visual necessari dels inputs del jugador durant el seu torn per tal que aquest pugui efectuar els moviments pertinents.

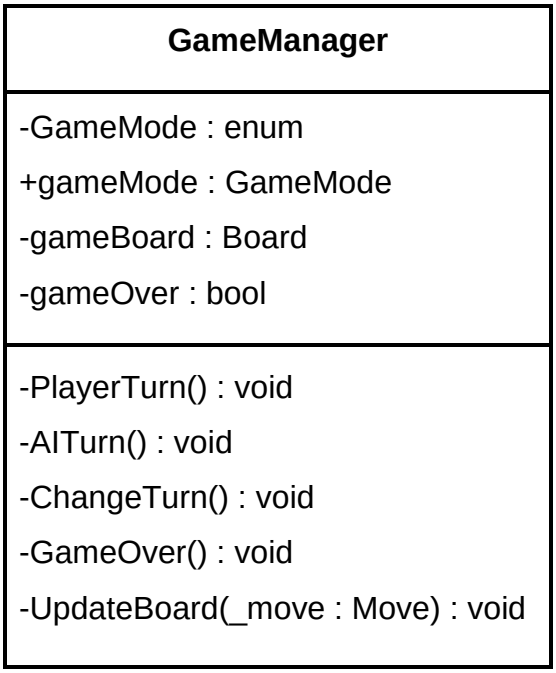


Figura 6.2. Classe *GameManager* en UML. Font: Elaboració pròpia

- Classe *Move*:

La classe *Move* és la que conté tota la informació necessària perquè qualsevol moviment pugui ser efectuat sobre el taulell en qualsevol estat de la partida: la

posició inicial del moviment, la posició final i, en cas de ser un moviment de salt, una llista amb les posicions de les peces oponents que són saltades.

Cada moviment efectuat o simulat per l'algoritme resultarà una instància d'aquesta classe.

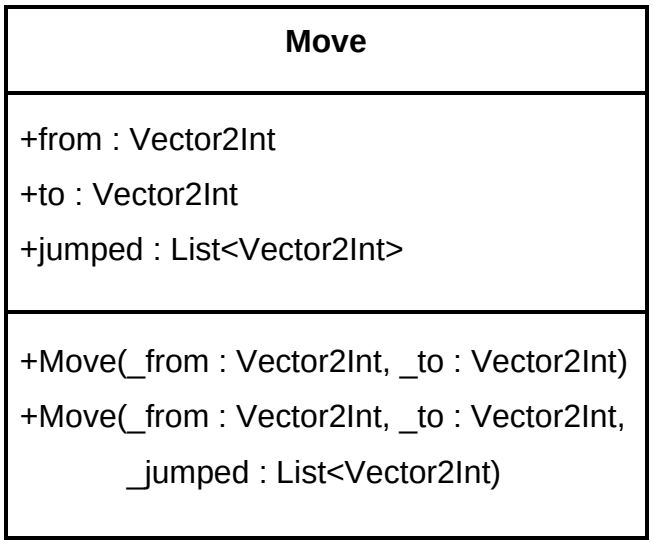


Figura 6.3. Classe *Move* en UML. Font: Elaboració pròpia

- Classe *Board*:

Aquesta classe conté tota la informació necessària sobre una determinada situació de taulell. En formen part alguns paràmetres rellevants com ara a qui li toca moure peça a continuació, amb quines peces juga el jugador i quina és la posició de cada peça. D'altra banda, també és la mateixa classe la que analitza totes les possibilitats disponibles que poden derivar en la propera tirada.

Cada situació de taulell simulada per l'algoritme, així com la situació real de la partida gestionada per la classe *GameManager*, resultarà una instància d'aquesta classe. Per tant, tal com s'ha esmentat anteriorment, l'algoritme o el *GameManager* accediran a la pròpia instància per tal d'obtenir la informació desitjada sobre el que pot derivar d'aquest estat de la partida. La classe *Board* gestiona aquest de manera que, per cada peça disponible de l'oponent a qui li toca moure, s'obtenen tots els salts possibles o, en cas de no haver-n'hi, tots els moviments simples possibles. En el primer cas, tenint en compte que els salts es poden encadenar formant una combinació d'aquests en un sol moviment, la funció utilitzada serà recursiva.

Un altre punt important que depèn d'aquesta mateixa classe es tracta de la funció d'avaluació del taulell, la qual atorga una puntuació numèrica a la situació en la qual es trobi el taulell. Per a aquest primer prototip aquesta funció no és definitiva, doncs anirà evolucionant i variant els seus paràmetres en funció dels resultats i del rendiment a mesura que també es vagi treballant en l'algoritme.

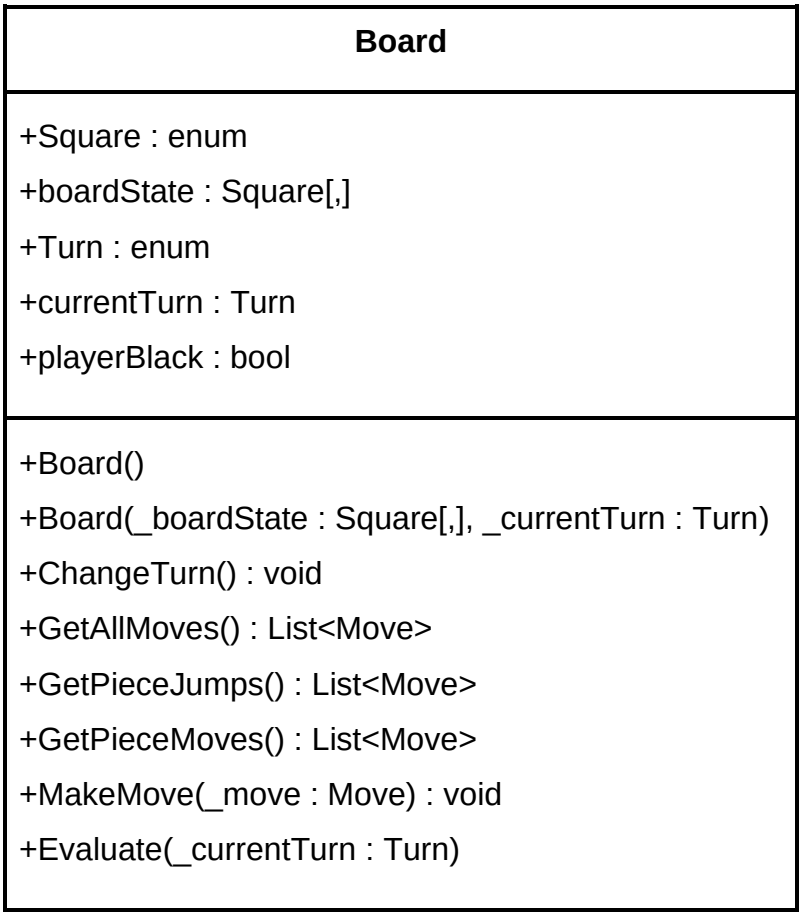


Figura 6.4. Classe *Board* en UML. Font: Elaboració pròpia

- Classe *Algorithm*:

Aquesta classe contindrà l'algoritme, o algoritmes, en cas d'haver-hi diverses versions commutables entre elles, que serà dissenyat i finalment utilitzat com a resultat d'aquest treball. En aquest prototip inicial aquesta classe compta amb una versió bàsica de l'algoritme Minimax, implementat tant amb Alpha-Beta Pruning com sense, per tal de comprovar el correcte funcionament del prototip abans de començar amb el disseny de l'algoritme definitiu.

Tal com s'ha exposat en apartats anteriors d'aquest treball, Millington (2006) explica que la funció recursiva d'aquests algoritmes ha de retornar tant el moviment a efectuar com la seva respectiva puntuació atorgada per la funció d'avaluació i que, en cas que el llenguatge no permeti retornar dues variables, s'haurà d'emprar una estructura que contingui les dues variables. Tot i que C# permet en les funcions l'ús de paràmetres *out,* els quals poden ser emprats per simular un retorn múltiple, per a aquest cas s'utilitza una estructura que contingui ambdós paràmetres per tal d'organitzar les variables dins el codi amb una millor claredat. La classe *Algorithm*, per tant, també defineix aquesta estructura.

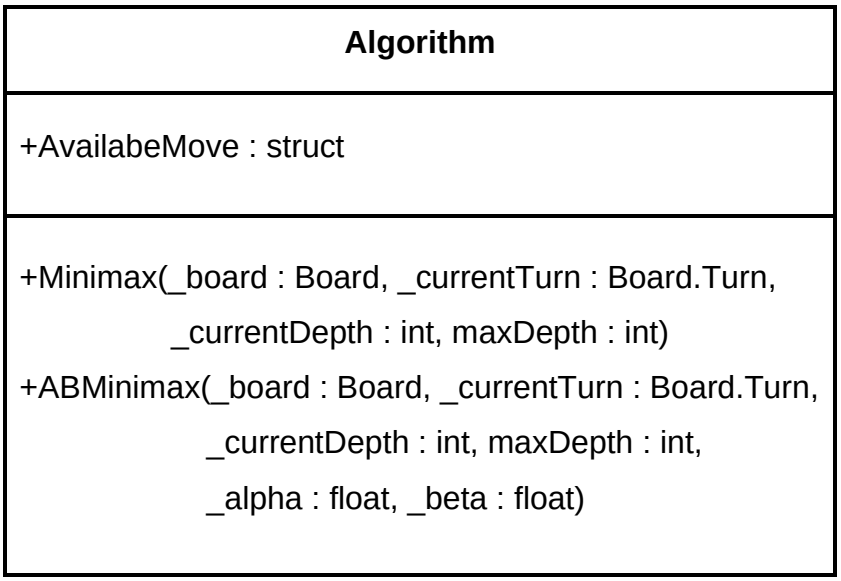


Figura 6.5. Classe *Algorithm* en UML. Font: Elaboració pròpia

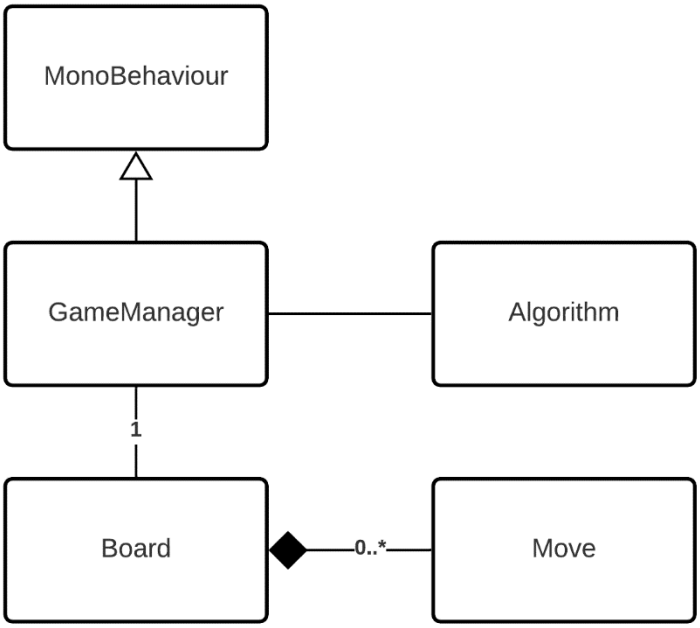


Figura 6.6. Relacions entre classes en UML. Font: Elaboració pròpia

## 6.2. Disseny i implementació de l'algoritme

Un cop acabada la preparació d'aquesta primera versió del prototip, es compta ja amb una base sobre la qual poder treballar en el disseny i la implementació de l'algoritme i, de la mateixa manera, poder provar el seu funcionament amb el joc de les dames angleses.

El comportament bàsic de l'algoritme dissenyat consisteix, a grans trets, a adaptar les seves accions en funció del rendiment del jugador al qual s'enfronta a través dels moviments que aquest va efectuant al llarg de la partida. D'aquesta manera, compta principalment amb dues parts: l'anàlisi dels moviments que du a terme el seu adversari d'entre tots els moviments que tenia la possibilitat de fer, i la decisió de quin moviment realitzar en el seu torn depenent d'aquest fet.

A continuació, en els següents punts d'aquest apartat, s'explica i es descriu la implementació i el funcionament d'aquests aspectes comentats, així com d'altres punts rellevants amb relació a l'algoritme en qüestió.

### 6.2.1. Anàlisi del rendiment de l'adversari

Per tal de poder triar quin de tots els moviments possibles ha de realitzar, l'algoritme necessitarà conèixer el rendiment del seu adversari. Aquest valor serà representat en forma de percentatge tot indicant, del zero al cent, l'encert del jugador en els seus moviments. Un bon rendiment del jugador mantindrà aquest valor elevat, mentre que unes decisions menys encertades devaluaran aquesta ràtio.

Per tal de calcular aquest valor, serà necessari tenir constància del darrer moviment realitzat pel jugador, així com una llista de tots els moviments que tenia la possibilitat de realitzar juntament amb les seves respectives puntuacions atorgades per la funció d'avaluació del taulell. Tant si l'algoritme juga contra un jugador real com si s'enfronta a qualsevol altre algoritme, el càlcul es realitzarà durant el seu torn a través d'una funció que, mitjançant l'algoritme Minimax amb poda Alpha-Beta, retornarà una llista de tots els moviments que poden ser efectuats per l'adversari amb les seves respectives puntuacions.

Un cop conegudes aquestes dades, després del moviment de l'adversari es calcularà aquesta ràtio que determinarà la dificultat del joc i, per tant, el nivell amb el qual haurà de jugar l'algoritme.

El primer que es fa és localitzar el moviment realitzat d'entre la llista de moviments possibles per tal de conèixer-ne la respectiva puntuació. Seguidament, s'inicialitza una llista que contingui totes les puntuacions resultants de la llista de moviments, ordenant-les segons el seu valor, de menor a major, i eliminant aquelles puntuacions repetides. A continuació, es localitza en aquesta llista la puntuació del moviment efectuat per l'adversari, anteriorment trobada i desada, i la seva posició en aquesta determinarà el valor del seu rendiment en el darrer moviment efectuat: l'índex de l'element en qüestió respecte de l'índex del darrer element de la llista equivaldrà al percentatge que representarà el valor del rendiment del jugador en aquest últim moviment. La següent fórmula (6.1) mostra aquest càlcul:

$$r = \frac{i * 100}{n-1} \tag{6.1}$$

En l'anterior fórmula, *r* representa el rendiment obtingut, mentre que *i* indica l'índex de l'element de la llista pertinent a la puntuació esmentada i *n* correspon a la longitud de la llista.

Tal com s'ha mostrat, el càlcul del rendiment del darrer moviment realitzat s'efectua a partir d'una llista en la qual cada puntuació apareix de manera única sense ser repetida. Aquest procediment es du a terme d'aquesta manera perquè, en cas de no ser així, en cas d'haver-hi més d'un moviment amb una mateixa puntuació, el valor resultant de l'operació variaria en funció de la posició del moviment en la llista, la qual cosa no seria correcte tenint en compte que dos moviments amb la mateixa puntuació han d'equivaldre a un mateix rendiment.

Un cop obtingut el rendiment del jugador en l'últim moviment, és desat en la llista que conté els valors corresponents a tots els moviments previs i es calcula la ràtio general a partir dels elements d'aquesta. En una primera aproximació, aquesta ràtio és calculada a partir de la mitjana aritmètica de tots els valors de la llista, però a través del testeig realitzat amb jugadors s'acaba comprovant el punt exposat per Missura i Gaertner (2010): els primers moviments de la partida per part del jugador acostumen a ser més aleatoris tenint en compte que la situació poc desenvolupada del taulell dificulta preveure a llarg termini. És per això que s'opta per provar de calcular la ràtio de diferents formes que donin més importància als moviments més recents. La fórmula definitiva (6.2) calcula una mitjana utilitzant pesos, donant a cada valor un pes més gran com més a prop es trobi del final de la llista, és a dir, del moment actual de la partida.

$$r = \frac{\sum_{i=1}^{n} x_{i-1} * i}{n(n+1)/2} \tag{6.2}$$

En la fórmula anterior, *r* és la ràtio de dificultat obtinguda, *x* equival a cada un dels valors, *i* representa l'índex de cada element de la llista, i *n* es tracta de la longitud de la llista que conté tots aquests valors.

Tot aquest seguit de càlculs que determinen la ràtio de dificultat amb la qual l'algoritme haurà d'actuar quan hagi d'escollir el moviment a efectuar, però, només tindrà lloc en cas que hi hagi més d'una puntuació derivant de la llista de moviments

que el jugador tenia la possibilitat d'efectuar. El fet d'haver-n'hi una de sola significaria que tots aquests moviments porten a situacions de taulell que comparteixen una mateixa puntuació, sent tots equivalents i no havent-n'hi cap de més ni menys òptim. En aquest cas, per tant, no es podria arribar a cap conclusió pel que fa a l'acció del jugador. És per això que, en cas d'haver-hi una sola puntuació, la ràtio de dificultat es manté invariable respecte el torn anterior.

```
def UpdateDifficultyRate(chosenMove, movesList,
                         lastDifficultyRate,
                         ref playerPerformancesList):

  # Locate the chosen move in the list and save its score
  for availableMove in movesList:
    if availableMove.move == chosenMove:
      chosenScore = availableMove.score

  # Create a list with unique and ordered scores
  for availableMove in movesList:
    scoresList.Add(availableMove.score)

  scoresList = scoresList.Distinct()
  scoresList.Sort()

  # If just one score, return the last difficulty rate
  if scoresList.Count == 1:
    return lastDifficultyRate

  # Get the current performance and add it to the list
  id = scoresList.IndexOf(chosenScore)
  currentPerformance = id * 100 / (scoresList.Count - 1)
  playerPerformancesList.Add(currentPerformance)

  # Update the rate using the formula and the list
  for i = 1, i <= playerPerformancesList.Count, i++
    sum += playerPerformancesList[i-1] * i

  n = playerPerformancesList.Count *
      (playerPerformancesList.Count + 1) / 2

  return sum / n
```

Figura 6.7. Pseudocodi de la funció *UpdateDifficultyRate*. Font: Elaboració pròpia

### 6.2.2. Elecció del moviment a efectuar

Un cop obtinguda la ràtio de dificultat amb què l'algoritme haurà de jugar, la qual serà actualitzada a cada torn de l'adversari, l'algoritme pot escollir quin moviment realitzar en funció del seu valor. Per fer això, primer cal conèixer tots els moviments possibles a efectuar amb les seves respectives puntuacions.

Aquesta part de l'algoritme es planteja com una funció que, de la mateixa manera que Minimax, tal com s'explica en el marc teòric d'aquest treball, genera situacions de taulell de manera recursiva a través dels diferents moviments possibles per a cada torn fins a arribar a una situació terminal de la partida o bé a una profunditat màxima de cerca, on es retornarà la puntuació del taulell a partir d'una funció d'avaluació. Un cop obtinguts els moviments i puntuacions, la funció retornarà el moviment escollit a partir de la ràtio de dificultat.

Per tal de dur a terme aquesta elecció, les diferents puntuacions de tots els moviments obtinguts s'afegeixen a una nova llista de manera ordenada, de menor a major o a la inversa depenent del torn en què l'algoritme, i eliminant totes aquelles repeticions d'un mateix valor. Això es fa perquè, de la mateixa manera que s'ha explicat anteriorment en el càlcul del rendiment del jugador, el fet de tenir en compte dues puntuacions iguals en la mateixa llista fixaria una diferència entre dos moviments que en realitat són igual d'òptims. D'aquesta manera, per tant, el que s'obtindrà en una primera instància no és el moviment a escollir, sinó el valor de la puntuació que ha de tenir el moviment a ser escollit.

La puntuació senyalada es trobarà d'entre la llista mitjançant un càlcul invers al seguit en l'obtenció del rendiment del jugador, explicat anteriorment en aquest mateix capítol: l'índex de la puntuació escollida respecte de l'índex del darrer element de la llista de puntuacions ha de ser equivalent al valor del percentatge que expressa la ràtio de dificultat. Aquest càlcul es representa mitjançant la fórmula (6.3) mostrada a continuació:

$$i = \frac{r * (n - 1)}{100} \tag{6.3}$$

En la fórmula anterior, *i* arrodonida al nombre enter més proper correspon a l'índex de la puntuació escollida en la llista de puntuacions úniques ordenades, mentre que *n* equival a la longitud d'aquesta. Per la seva banda, *r* correspon a la ràtio de dificultat obtinguda del rendiment de l'adversari.

L'algoritme escull el moviment a realitzar a partir d'aquesta puntuació: el primer element de la llista de moviments disponibles que comparteixi aquest mateix valor, serà el moviment a efectuar. Abans d'això, però, el contingut de la llista haurà estat reordenat de manera aleatòria per tal que, en cas de comptar amb més d'un moviment amb aquesta mateixa puntuació, l'elecció entre ells es doni a través de l'atzar.

A través del testeig de l'algoritme es comprova que, escollint els moviments a efectuar de manera recursiva a partir de la ràtio calculada, el comportament d'aquest acaba sent erroni. Aquest fet és degut al fet que en cada nivell de profunditat de cerca de l'algoritme la puntuació és escollida d'aquesta mateixa manera, basant-se en la ràtio de dificultat. Això provoca que un cop arribat a l'elecció de la capa zero de profunditat, és a dir, la decisió del moviment més adequat per a la situació de taulell actual, les diferents puntuacions no sigiuin aquelles més òptimes seguint la lògica de Minimax, sinó aquelles que s'han considerat properes a ser escollides pel jugador a tall predicció.

Per tal de posar solució a aquest fet, s'opta per canviar la lògica de l'algoritme fent que el moviment només sigui escollit a partir de la ràtio de dificultat en la darrera elecció, és a dir, en la profunditat de cerca zero. Pel que fa a la resta de capes de profunditat, s'opta per escollir els moviments seguint la metodologia pròpia de Minimax, escollint el més òptim o el menys encertat en funció del torn. D'aquesta manera, la decisió final del moviment a efectuar es realitza a partir de les puntuacions reals dels millors moviments possibles. L'algoritme escollit per a complementar l'algoritme adaptatiu és Minimax amb poda Alpha-Beta.

D'altra banda, en les següents tandes de testeig s'observa que, en moltes partides en les quals l'algoritme s'enfronta a altres algoritmes, es generen situacions repetitives, responent ambdós de la mateixa manera. Aquest fet es deu a la utilització de Minimax, doncs aquest algoritme sempre tria el primer millor o pitjor

moviment trobat, només actualitzant-lo en cas que aquest sigui superat. D'aquesta manera, en cas que es trobin diversos moviments amb una mateixa puntuació, l'algoritme sempre es quedarà amb el primer moviment trobat, obviant tota la resta.

Per tal de posar solució a aquest problema, s'opta per plantejar una variant de l'algoritme Minimax, tant amb Alpha-Beta com sense, en la qual, en cas d'existir més d'un moviment amb la màxima o mínima puntuació, s'escull el moviment a retornar de manera aleatòria. Per tal de dur això a terme, quan durant la iteració es troba un moviment millor a l'anterior trobat, aquest s'afegeix en una llista. D'aquesta manera, si es troba un moviment amb la mateixa puntuació, s'afegeix a aquesta mateixa llista. D'altra banda, cada cop que es trobi un moviment més adient per la seva puntuació, la llista es buidarà i se li afegirà aquest darrer. Aquest procés es repetirà fins al final de la iteració. D'aquesta manera, en cas que la llista compti amb més d'un moviment, s'escollirà quin serà retornat per la funció mitjançant l'atzar.

Tot i que arribats a aquest punt s'obté un resultat força similar a l'esperat, a través del testeig es reflecteix com, quan l'algoritme juga contra un usuari, aquest darrer sol guanyar totes les partides de manera desproporcionada si no efectua sempre els millors moviments possibles. Després de fer-ne una valoració, es comprova que aquest problema pot derivar del fet que la llista contingui totes les puntuacions possibles, incloses les més baixes. Els moviments que corresponen a aquestes puntuacions no només no pretenen guanyar la partida sinó que, per contra, pretenen ajudar a guanyar-la al seu adversari, sent aquestes les pitjors decisions a prendre, aquelles menys òptimes.

Tenint això en compte, s'opta per eliminar un petit percentatge de les puntuacions més petites de la llista, doncs per poc òptims que hagin de ser els moviments escollits per l'algoritme, en cap cas es pretén que busquin de manera directa la derrota, provocant expressament la victòria de l'usuari. Optant per aquesta solució s'aconsegueix que, per molt que el rendiment de l'adversari sigui baix, l'algoritme cerqui la victòria obviant aquells moviments que el condueixen de forma més directa a una derrota assegurada, doncs el fet de deixar guanyar l'usuari de manera deliberada també seria contraproduent.

Al llarg de les partides disputades tant amb aquest algoritme com amb els anteriorment implementats, el temps de processament és molt variable en funció de l'estat de la partida: si hi ha moltes peces sobre el taulell o moltes opcions diferents de moviment, el temps que tarda l'algoritme a assolir una determinada profunditat de cerca és més elevat, mentre que quan queden poques peces o hi ha poca varietat de moviments a realitzar, la mateixa profunditat s'assoleix més ràpidament.

Per tal de trobar la profunditat de cerca adequada per a cada moment de la partida, s'ha optat per dinamitzar-ne el valor. Si el temps que passa des que es crida la funció per primer cop fins que es retorna el moviment resultant supera un màxim determinat, la profunditat de cerca disminueix. Per contra, si aquest procés es completa en un temps per sota d'un mínim establert, la profunditat de cerca augmenta. Per una altra banda, també es defineix un temps màxim de processament per evitar temps de processament excessius: si en superar aquest límit l'algoritme encara no ha retornat el moviment esperat, aquest serà interromput i es retornarà el millor moviment trobat en aquest punt.

```
def AdaptiveABMinimax(board, currentTurn, currentDepth,
                      maxDepth, alpha, beta,
                      startingTime, maxThinkingTime,
                      difficultyRate, movesToDraw):

  # Check if the maximum depth has been reached
  if currentDepth >= maxDepth:
    return new AvailableMove(){null,
                               board.Evaluate(currentTurn)}

  # Check if the game ends in a draw
  if movesToDraw <= 0:
    return new AvailableMove(){null, 0f}

  # Check if the current player can move
  if moves.Count == 0:
    return new AvailableMove(){null,
                               board.Evaluate(currentTurn)}

  # Go through each move. Add the result to the list
  for move in moves:
    if board.currentTurn == Board.Turn.Black:
```

```
    newBoard = new Board(board.boardState,
                         Board.Turn.White,
                         board.playerBlack)


  else:
    newBoard = new Board(board.boardState,
                         Board.Turn.Black,
                         board.playerBlack)


  newBoard.MakeMove(move)

  # Update the "moves to draw" counter
  if movedPiece == king and noJumpedPieces:
    currentMovesToDraw = movesToDraw - 1
  else
    currentMovesToDraw = 20

  currentMove = ABMinimax(newBoard, currentTurn,
                          currentDepth + 1, maxDepth,
                          alpha, beta, startingTime,
                          maxThinkingTime,
                          currentMovesToDraw)

  availableMoves.Add(new AvailableMove(){move,
                                         currentMove.score})

# If there is just one move, return it
if availableMoves.Count == 1:
  return availableMoves[0]

# Create a list with unique and ordered scores
for availableMove in availableMovesList:
  scoresList.Add(availableMove.score)

scoresList = scoresList.Distinct()
scoresList.Sort()

# Update the list discarding the lowest values
startingId = scoresList.Count / discardedMoves
for i = startingId, i < scoresList.Count, i++
  highestScoresList.Add(scoresList[i])

# Get the score of the chosen move from the list
id = (highestScoresList.Count - 1) * difficultyRate / 100
chosenScore = scoresList[id]
```

```
# Shuffle the moves in the list and choose the first one
# with the chosen score

ShuffleList(ref availableMoves)



for move in availableMove:
  if move.score == chosenScore:
    chosenMove = chosenScore
    break;

return chosenMove
```

Figura 6.8. Pseudocodi de la funció *AdaptiveABMinimax*. Font: Elaboració pròpia

### 6.2.3. Funció d'avaluació del taulell

La funció d'avaluació del taulell és un aspecte important a tenir en compte per a l'algoritme, doncs permet avaluar el taulell i donar un valor numèric en aquelles situacions terminals de la partida o bé quan aquest arriba a la profunditat màxima de cerca. Seguint els exemples exposats al marc teòric d'aquest mateix treball, aquesta funció és programada per retornar cada valor de cada aspecte del taulell a avaluar multiplicat pel seu propi pes que dependrà de la rellevància d'aquest respecte la resta.

Es contempla implementar una funció d'avaluació que mantingui un equilibri entre l'eficàcia i l'eficiència: la funció ha de complir amb el seu propòsit per tal que l'algoritme funcioni correctament i es pugui avaluar de forma encertada cada situació de taulell, però també ha d'evitar una prolongació massa llarga en el temps de càlcul tenint en compte l'elevat nombre de situacions de taulell a analitzar, creixent exponencialment per a cada nivell de profunditat de cerca.

Per tal d'avaluar diverses opcions, es fa una funció diferent per a cada aspecte del taulell a avaluar, on se'n multiplica el valor pel seu determinat pes. La funció d'avaluació retornarà la suma d'aquest valor retornat per aquelles funcions corresponents a aquells aspectes que siguin tinguts en compte. A partir del testeig de diverses versions diferents, finalment s'opta per una avaluació del taulell de

caràcter més neutral, analitzant els aspectes que poden ser mesurats de manera més objectiva en comptes dels més relatius a l'estratègia. D'aquesta manera, els aspectes que s'analitzen en avaluar el taulell són el recompte de dames, el recompte de peons, les peces que es troben en perill, les peces que tenen la possibilitat de moure's i les peces que es troben a prop de convertir-se en dama. Els pesos dels diferents aspectes avaluats es distribueixen seguint aquest mateix ordre de prioritat, tal com mostra la següent taula:

| **Aspectes analitzats** | **Pes (magnitud)** | **Pes (proporció)** |
|---|---|---|
| Nombre de dames | 3 | 60% |
| Nombre de peons | 1 | 20% |
| Peces en perill | 0,5 | 10% |
| Peces movibles | 0,25 | 5% |
| Peons prop de ser dames | 0,25 | 5% |

Taula 6.1. Funció d'avaluació: aspectes a analitzar i pesos. Font: Elaboració pròpia

## 6.3. Finalització del prototip i avaluació de l'algoritme

Un cop finalitzada la implementació definitiva de l'algoritme, es completa l'elaboració del prototip per poder donar pas a la fase d'experimentació que forma part dels objectius secundaris d'aquest mateix treball. S'implementa un sistema de menús per poder escollir si l'algoritme s'enfronta a un jugador o a un altre algoritme, les peces amb què jugarà el jugador o a quin algoritme s'enfrontarà, entre d'altres. També s'ha afegit en la interfície d'usuari tota la informació necessària per sobre el rendiment de l'algoritme durant la partida. D'altra banda, s'ha millorat el feedback visual per tal que el jugador pugui interactuar còmodament amb el prototip del joc.

Malgrat que l'experiment es durà a terme des del mateix ordinador i amb els mateixos valors de temps límit pel que fa a la dinamització de la profunditat de cerca per tal d'evitar cap diferència de rendiment de l'algoritme en les diferents partides, també s'ha afegit al prototip un menú d'opcions que permet definir tant els temps mínim i màxim de cerca com altres paràmetres vinculats a aquest aspecte.

Tot i que durant la fase de desenvolupament s'ha testejat contínuament el prototip a través del joc amb diferents usuaris per poder iterar sobre el disseny de l'algoritme amb els millors resultats possibles, aquesta darrera fase servirà per fer un estudi i una avaluació més a fons sobre el funcionament d'aquest.

Pel que fa a altres algoritmes, l'algoritme dissenyat podrà enfrontar-se a Minimax amb poda Alpha-Beta o sense i a ell mateix amb una ràtio de dificultat invariant definida des del menú. Amb relació als dos primers casos, s'utilitza la versió d'aquests que s'explica a l'apartat anterior d'aquest capítol, la qual escull el moviment a realitzar de manera aleatòria en cas de trobar-ne més d'un amb un mateix resultat. Quant al tercer cas, s'utilitzarà una versió de l'algoritme en qüestió que, a diferència d'aquest, no obviarà els moviments amb menys puntuació a l'hora de triar l'adequat. D'aquesta manera, comptarà amb el mateix ventall de possibilitats amb les quals comptaria un jugador real, simulant de la manera més propera possible el comportament que aquest podria tenir.

D'altra banda, pel que fa als jugadors, l'algoritme s'enfrontarà a ells sense mostrar el rendiment calculat ni cap altre valor per tal que aquesta informació no pugui interferir en les decisions ni en la manera de jugar de cada usuari.

# 7. Resultats

En aquest apartat del treball es parla dels resultats obtinguts, tant de manera general pel que fa al compliment dels objectius i del desenvolupament de l'algoritme en qüestió com de les dades obtingudes de la darrera fase del treball, corresponent a l'experimentació.

## 7.1. Resultats generals del treball

Els dos objectius principals d'aquest treball consistien en el disseny i implementació d'un algoritme amb dificultat adaptativa basat en Minimax, així com la creació d'un prototip del joc de les dames per tal de poder comprovar-ne el funcionament. Al final d'aquest treball, ambdós objectius han estat complerts. Es compta, per tant, amb un algoritme adaptatiu totalment funcional i un prototip del joc de les dames angleses preparat per utilitzar-lo i enfrontar-lo a oponents.

D'altra banda, es proposava com a objectiu secundari un estudi més a fons del comportament de l'algoritme en situacions reals de joc, enfrontant-lo tant a ell mateix com a altres algoritmes i a jugadors reals. Aquest estudi s'ha dut a terme en una darrera fase del treball dedicada a l'experimentació, i s'han agafat les dades corresponents als resultats de les diferents partides per tal de, posteriorment, poder extreure'n les respectives conclusions.

## 7.2. Resultats de la fase d'experimentació

Pel que fa a l'enfrontament de l'algoritme contra altres jugadors, s'ha comptat amb vuit usuaris que han jugat dotze partides cadascú, la meitat amb les peces blanques i l'altra meitat amb les peces negres. Tal com s'ha mostrat al capítol anterior, l'algoritme calcula el rendiment del jugador al llarg de cada partida a partir dels seus moviments, mitjançant la fórmula exposada (6.2). De la comparació d'aquest rendiment mitjà obtingut d'aquest càlcul amb el nombre de partides guanyades, n'obtenim el següent gràfic:

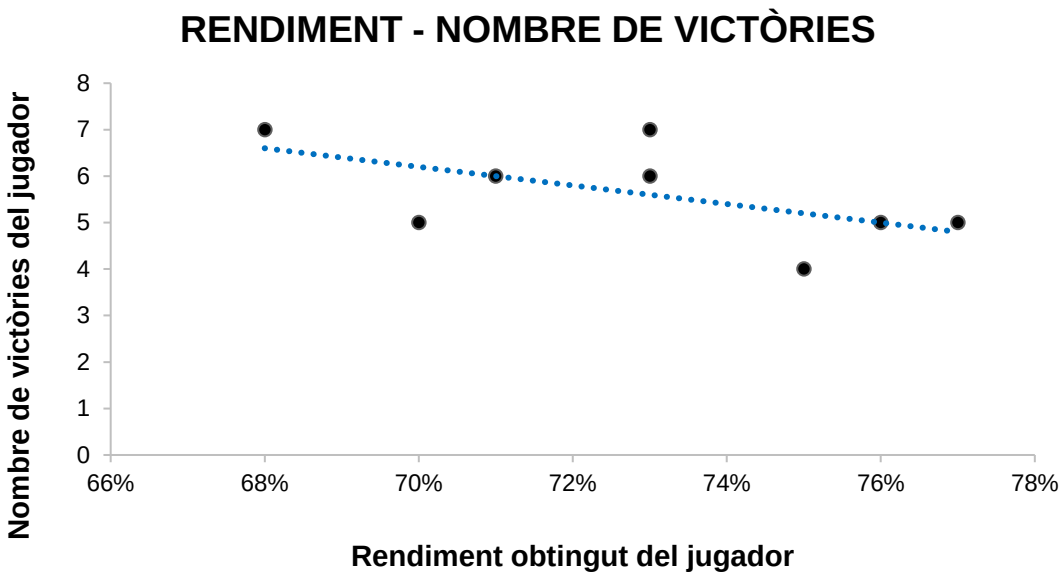


Figura 7.1. Relació rendiment - nombre de victòries. Font: Elaboració pròpia

En aquest gràfic es pot observar com el nombre de victòries obtingudes per cada un dels jugadors es manté aproximadament en la meitat del nombre de partides jugades. A més, tal com indica la línia de tendència, el nombre de partides guanyades pels jugadors tendeix a disminuir com més alt és el valor del rendiment obtingut. Això compleix amb les expectatives esperades, doncs l'obtenció d'una victòria creix en dificultat paral·lelament amb el nivell del jugador calculat per l'algoritme.

D'altra banda, però, també destaca el reduït nombre de partides perdudes pels jugadors en gran part dels casos, doncs la majoria d'enfrontaments que no han acabat amb victòria pel jugador han finalitzat en empat, i en cap cas el nombre de derrotes obtingudes ha superat ni igualat al de victòries. Aquest fet podria ser solucionat elevant la dificultat de la resposta de l'algoritme als moviments del jugador, augmentant el percentatge de moviments amb puntuació baixa descartats.

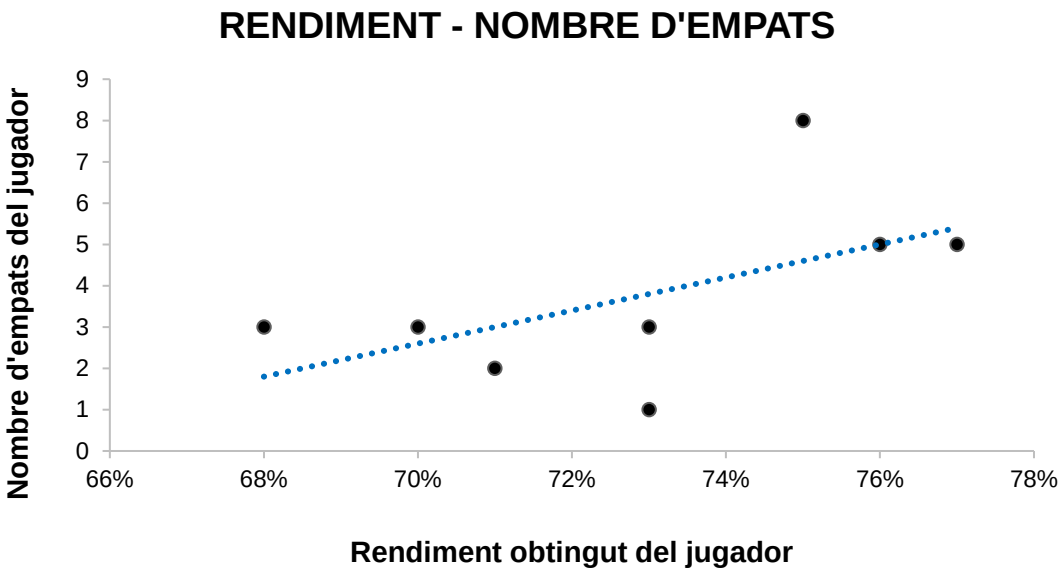


Figura 7.2. Relació rendiment - nombre d'empats. Font: Elaboració pròpia

La línia de tendència de l'anterior gràfic mostra com tendeix a haver-hi més empats a mesura que el rendiment obtingut del jugador augmenta. Tot i que aquest fet resulta positiu tenint en compte que l'algoritme ha de posar més resistència al seu adversari com més alt és el nivell, el nombre de partides empatades és força elevat tenint en compte el generalment reduït nombre de derrotes per part dels jugadors.

Amb relació a les partides en les quals l'algoritme s'ha enfrontat tant a altres algoritmes com a ell mateix, també s'han dividit equitativament els enfrontaments disputats entre partides amb peces blanques i partides amb peces negres. Els resultats obtinguts en els enfrontaments contra els algoritmes Minimax amb poda Alpha-Beta i Minimax sense poda coincideixen en gran manera amb els esperats. En el primer cas, el rendiment mitjà obtingut en les diferents partides disputades és del 100%, fet totalment esperat tenint en compte que l'algoritme adaptatiu també realitza la cerca de moviments a través de la mateixa lògica. Pel que fa al segon cas, el rendiment mitjà obtingut dels diferents enfrontaments és del 94%. Tot i ser un resultat alt, la diferència entre aquest valor i el del rendiment màxim es deu a la impossibilitat de Minimax d'arribar a la mateixa profunditat de cerca en el mateix temps que ho faria utilitzant la poda en aquelles situacions de taulell més complexes i que poden derivar en una quantitat més elevada de moviments diferents.

Cal tenir en compte, a més, que tant en els enfrontaments contra Minimax amb poda Alpha-Beta com en les partides contra Minimax sense poda, el balanç entre partides guanyades i partides perdudes s'ha mantingut força equilibrat, destacant amb una gran mesura el nombre d'empats entre ambdós adversaris respecte dels de victòries i derrotes. Aquest fet també coincideix amb l'escenari que es contemplava de manera predictiva.

D'altra banda, en la fase d'experimentació en què l'algoritme s'ha enfrontat a ell mateix, el resultat també ha estat notablement proper a l'esperat. La relació entre el valor predeterminat que se li ha adjudicat a l'algoritme adversari a tall de rendiment i el valor del rendiment obtingut per l'algoritme principal en aquestes partides es pot veure reflectida en el gràfic que es mostra a continuació:

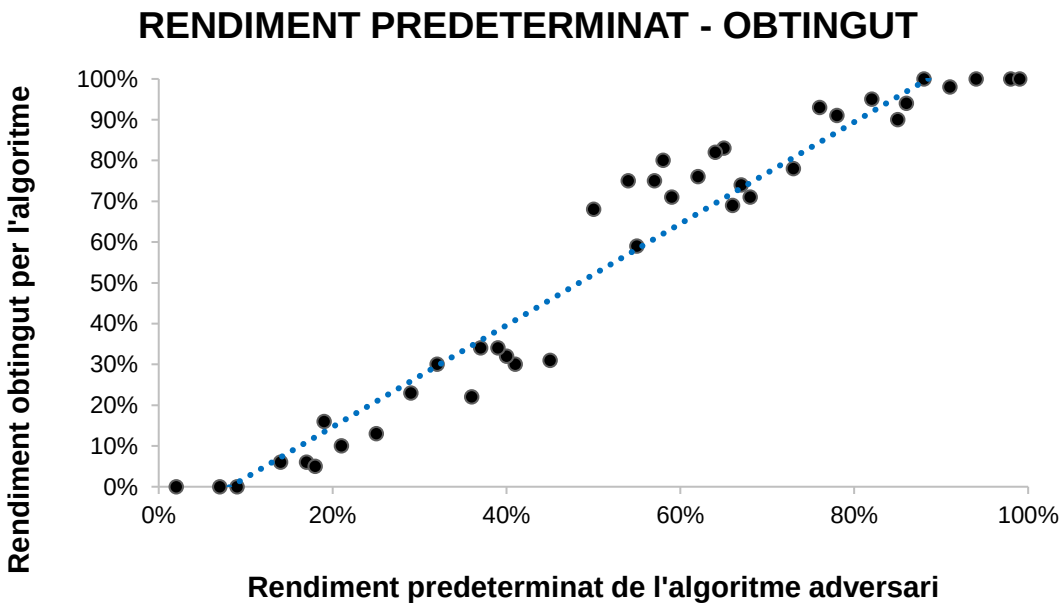


Figura 7.3. Relació valor predeterminat - obtingut. Font: Elaboració pròpia

Tal com es pot observar en el gràfic a través de la distribució de valors i de la línia de tendència, el rendiment obtingut coincideix de manera destacable amb el rendiment predeterminat que se li ha donat a l'algoritme adversari en cada una de les partides, el qual s'ha escollit de manera aleatòria. L'error mitjà entre ambdós valors ha estat d'un 10%. Cal tenir en compte que aquest petit error entre el rendiment predeterminat i el rendiment obtingut era també esperable, doncs l'algoritme adversari no obté aquest valor de cap moviment realitzat pel seu

oponent. És per això que, en moltes situacions de partida, aquest valor equivaldria a un moviment inexistent, el qual s'ubicaria entre dos dels moviments possibles: un corresponent a una puntuació superior i un altre corresponent a una puntuació inferior. És per aquesta mateixa raó que els enfrontaments on s'ha assignat un valor més elevat solen desviar-se de la tendència cap amunt, mentre que les partides amb un valor predeterminat més baix solen desviar-se'n cap avall.

En aquesta fase de l'experimentació en què l'algoritme s'enfronta a ell mateix no se li dona rellevància al nombre de victòries, empats i derrotes, doncs pel mateix motiu explicat anteriorment, aquest balanç queda afectat. A més, cal tenir en compte que en l'algoritme adaptatiu es descarten els moviments amb una puntuació més baixa per tal de no buscar una derrota provocada de manera deliberada. És per això que gairebé totes les partides en què s'ha donat a l'algoritme adversari un rendiment predeterminat inferior al percentatge de moviments descartats han acabat amb una derrota per part d'aquest.

# 8. Conclusions i valoracions finals

Aquest darrer capítol del treball consisteix en una valoració del transcurs de l'elaboració d'aquest tenint en compte el calendari definit durant el seu plantejament, així com dels resultats finalment obtinguts.

## 8.1. Valoració de la realització treball

Tot i que generalment el calendari establert des del plantejament del treball s'ha pogut seguir, algunes dates o terminis s'han vist afectats i han acabat sent lleugerament modificats a conseqüència de certs imprevistos o canvis de plantejament durant la fase de desenvolupament.

Per una banda, durant la fase de desenvolupament es va considerar que, per tal de detectar possibles errors i millores per a l'algoritme, seria necessari un testeig més exhaustiu, per la qual cosa alguns usuaris van començar a prendre contacte amb el prototip abans d'iniciar la fase d'experimentació. Posteriorment, precisament aquest fet va permetre la detecció d'alguns errors que van endarrerir les tasques i, conseqüentment, van provocar que aquesta mateixa fase d'experimentació també comencés més tard de l'inicialment previst.

D'altra banda, tot i que al cronograma establert (figura 5.1) es mostra com la preparació del prototip tenia lloc íntegrament abans de començar amb el disseny i la implementació de l'algoritme, alguns aspectes d'aquesta van ser prorrogats fins a tenir més avançada la següent fase. Alguns exemples són el sistema de menús i la interfície d'usuari que es mostra en pantalla durant les partides, doncs la informació que mostren deriva del funcionament de l'algoritme.

Un altre aspecte que cal esmentar amb el que no es comptava és la manca d'usuaris participants en la fase d'experimentació, doncs finalment no es va poder comptar amb tots aquells jugadors amb qui es tenia previsió de fer-ho. Nogensmenys, tot i veure's reduït el nombre de participants, aquells usuaris que van col·laborar en l'experiment van poder dedicar-hi prou temps per a realitzar unes sessions de joc prou llargues per jugar un nombre de partides suficientment rellevant com per poder treure'n conclusions. En qualsevol cas, un estudi més a

fons amb una mostra d'usuaris més nombrosa podria ser una bona proposta de continuïtat per aquest treball.

## 8.2. Valoració dels resultats obtinguts

En primer lloc, cal destacar que tot i que el resultat obtingut amb l'algoritme dissenyat és satisfactori i compleix amb els objectius plantejats i amb les expectatives previstes en el plantejament del treball, aquest és susceptible a possibles millores. Això ha estat comprovat durant la darrera fase del treball, dedicada a l'experimentació amb algoritmes i usuaris reals.

Dues debilitats amb les quals compta l'algoritme, a part d'aquelles comentades en el capítol anterior, són l'aleatorietat i el nivell alt per part dels jugadors. La primera es dona quan el jugador, pel motiu que sigui, fa moviments molt aleatoris, els quals acaben derivant en un càlcul incorrecte del seu rendiment. Això pot provocar, per exemple, que a un jugador totalment novell li sigui atorgat un rendiment elevat pel simple fet d'experimentar amb el joc. Tot i que no ha pogut ser comprovat en aquest estudi, la segona debilitat es pot deure al fet que el jugador sigui capaç d'analitzar les situacions del taulell amb més profunditat que l'algoritme. En aquest cas, l'algoritme calcularia un rendiment més baix al jugador interpretant que les seves decisions són menys òptimes quan, en realitat, ho són més.

La primera qüestió comentada podria ser tractada en un possible treball posterior a aquest, doncs és un escenari que queda obert i pendent de ser estudiat. Tot i que aparentment la segona qüestió comentada només podria tenir com a solució la utilització de l'algoritme en un dispositiu amb la suficient capacitat de processament per a poder arribar a cercar amb més profunditat que el seu adversari, també pot ser estudiada i investigada amb jugadors més experimentats o amb un nivell més elevat al joc en qüestió, en aquest cas les dames angleses, tenint en compte que aquest aspecte no ha pogut ser comprovat en aquest treball.

# 9. Referències

Andrade, G., Santana, H., Furtado, A., Leitão, A., & Ramalho, G. (2003). Online Adaptation of Computer Games Agents: A Reinforcement Learning Approach. *II Workshop de Jogos e Entretenimento Digital*, 105-112.

Aponte, M., Levieux, G., & Natkin, S. (2009). Measuring the level of difficulty in single player video games. *Entertainment Computing*, 205-213.

Baba, N., & Jain, L. (2001). *Computational Intelligence in Games.* Heidelberg: Physica-Verlag.

Bellman, R. (1978). *An Introduction to Artificial Intelligence: Can Computers Think?* San Francisco: Boyd & Fraser Publishing Company.

Charniak, E., & McDermott, D. (1985). *Introduction to Artificial Intelligence.* Reading: Addison-Wesley.

Chess & Checkers Games. (2015). *Draughts.* (Android) [Videojoc]. Varsòvia, PL: Google Commerce Ltd.

Chess & Checkers Games. (sense data). *Basic Draught Rules*. Recollit de Draughts for Android: https://www.draughtsforandroid.com/basic-draughts-rules.html

Elnaggar, A., Gadallah, M., Aziem, M., & El-Deeb, H. (2014). A Survey of Game Tree Searching Methods. *International Journal of Advanced Computer Science and Applications*, 68-77.

Evans, D., & Sable, C. (sense data). *Alpha-Beta Pruning and Checkers*. Recollit de Department of Computer Science, Columbia University: http://www.cs.columbia.edu/~devans/TIC/AB.html

Finnsson, H., & Björnsson, Y. (2011). Game-Tree Properties and MCTS Performance. *International Joint Conferences on Artificial Intelligence* (p. 23-30). Barcelona: AAAI Press.

Haugeland, J. (1985). *Artificial Intelligence: The Very Idea.* Cambridge: MIT Press.

Humphrys, M. (sense data). *Adversarial Search*. Recollit de DCU - School of Computing: https://computing.dcu.ie/~humphrys/Notes/AI/adversarial.search.html

Hunicke, R. (2005). The Case for Dynamic Difficulty Adjustment in Games. *Advances in Computer Entertainment Technology (ACE)* (p. 429-433). València: ACM International Conference Proceeding Series (ICPS).

Ilici, L., Wang, J., Missura, O., & Gärtner, T. (2012). Dynamic Difficulty for Checkers and Chinese chess. *Computational Intelligence and Games (CIG)* (p. 55-62). Granada: IEEE.

Kok, J., Boers, E., Kosters, W., & Van der Putten, P. (2009). Artificial Intelligence: Definition, Trends, Techniques, and Cases. A J. Kok, *Artificial Intelligence* (p. 1-20). Oxford: Eolss Publishers.

Kurzweil, R. (1990). *The Age of Intelligent Machines.* Cambridge: MIT Press.

Kusiak, M., Waledzik, K., & Mandziuk, J. (2007). Evolutionary Approach to the Game of Checkers. *Adaptive and Natural Computing Algorithms* (p. 432-440). Varsòvia: Springer.

Luger, G., & Stubblefield, W. (1993). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving.* Redwood City: Benjamin/Cummings.

Millington, I. (2006). *Artificial Intelligence for Games.* San Francisco: Elsevier Inc.

Missura, O., & Gaertner, T. (2010). Online Adaptive Agent for Connect Four. *Proceedings of the 4th international conference on games research and development cybergames*, 1-8.

Myerson, R. (1991). *Game Theory. Analysis os Conflict.* Cambridge: Harvard University Press.

Osborne, M., & Rubinstein, A. (1994). *A Course in Game Theory.* Massachusetts: MIT Press.

Redfield, C., Gaither, D., & Redfield, N. (2009). COTS Computer Game Effectiveness. A R. Ferdig, *Handbook of Research on Effective Electronic Gaming in Education* (p. 277-294). Hershey: IGI Global.

Rich, E., & Knight, K. (1991). *Artificial Intelligence.* New York: McGraw-Hill.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence. A Modern Approach.* Englewood Cliffs: Prentice Hall.

Schaeffer, J., Lake, R., Lu, P., & Bryant, M. (1996). Chinook. The World Man-Machine Checkers Champion. *AI Magazine*, 21-29.

Schalkoff, R. (1990). *Artificial Intelligence: An Engineering Approach.* New York: McGraw-Hill.

Shuqin, L., Weiming, X., & Xiaohua, Y. (2015). Study on the Evaluation Function Parameters of the Checkers Game Program on Weka Platform. *International Journal of New Technology and Research (IJNTR)*, 1-5.

Taha, W. (2021). Game Theory. A W. Taha, A. Taha, & J. Thunberg, *Cyber-Physical Systems: A Model-Based Approach* (p. 113-128). Cham: Springer.

Wang, P. (2019). On Defining Artificial Intelligence. *Journal of Artificial General Intelligence*, 1-37.

Wilson, B., Zuckerman, I., Parker, A., & Nau, D. (2012). Improving Local Decisions in Adversarial Search. *Frontiers in Artificial Intelligence and Applications*, 840-845.

Winston, P. (1992). *Artificial Intelligence.* Reading: Addison-Wesley.

Yannakakis, G., & Togelius, J. (2018). *Artificial Intelligence and Games.* Cham: Springer.

TecnoCampus

Centres universitaris adscrits a la

upf. Universitat Pompeu Fabra Barcelona

**Grau en Disseny i Producció de Videojocs**

# Dificultat adaptativa en jocs de taula a través de Minimax

## ANNEX

**Albert Medina i Cucurull**

**Tutor: Dr. Enric Sesa Nogueras**

**2020-2021**

TecnoCampus Mataró-Maresme

# Índex

# Índex de figures

# 1. Localització d'annexos

## 1.1. Projecte de Unity i codi font

Nom: *TFG_AdaptiveMinimax_Checkers*

Format: Carpeta comprimida (*zip*)

Ruta: *Medina_Cururull_DificultatAdaptativaMinimax_TFG\Annex*

El codi font es troba dins la carpeta del projecte, seguint la següent ruta:

*TFG_AdaptiveMinimax_Checkers\Assets\Scripts*

## 1.2. Executable del prototip

Nom: *TFG_AdaptiveCheckers*

Format: Carpeta comprimida (*zip*)

Ruta: *Medina_Cururull_DificultatAdaptativaMinimax_TFG\Annex*

L'executable (*exe*) es troba dins d'aquesta carpeta i rep el mateix nom.

## 1.3. Vídeo demostratiu

Nom: *TFG_Video*

Format: Vídeo (*mp4*)

Ruta: *Medina_Cururull_DificultatAdaptativaMinimax_TFG\Annex*

## 1.4. Recull de dades de la fase d'experimentació

Nom: *TFG_DadesExperiment*

Format: Full de càlcul de Microsoft Excel (*xlsx*)

Ruta: *Medina_Cururull_DificultatAdaptativaMinimax_TFG\Annex*

# 2. Material de tercers

Per a l'elaboració del prototip han estat utilitzats els següents *assets*:

- *Board Games* de *Jarst*, obtingut de *Unity Asset Store*.
  Enllaç: *Board Games - Jarst (2019)*

# 3. Obtenció de l'executable

L'executable del prototip podrà ser obtingut a través del projecte de Unity, present en aquesta mateixa carpeta de Drive, en la ruta anteriorment indicada. El prototip s'ha desenvolupat amb la versió de Unity 2020.3.2f1.

Un cop obert el projecte, caldrà navegar a la pestanya "*File*" i posteriorment a "*Build Settings...*". Després de comprovar en la finestra emergent que l'escena "*AdaptiveABMinimax*" es trobi correctament marcada a l'apartat "*Scenes In Build*", caldrà seleccionar la plataforma adequada i prémer "*Build*".

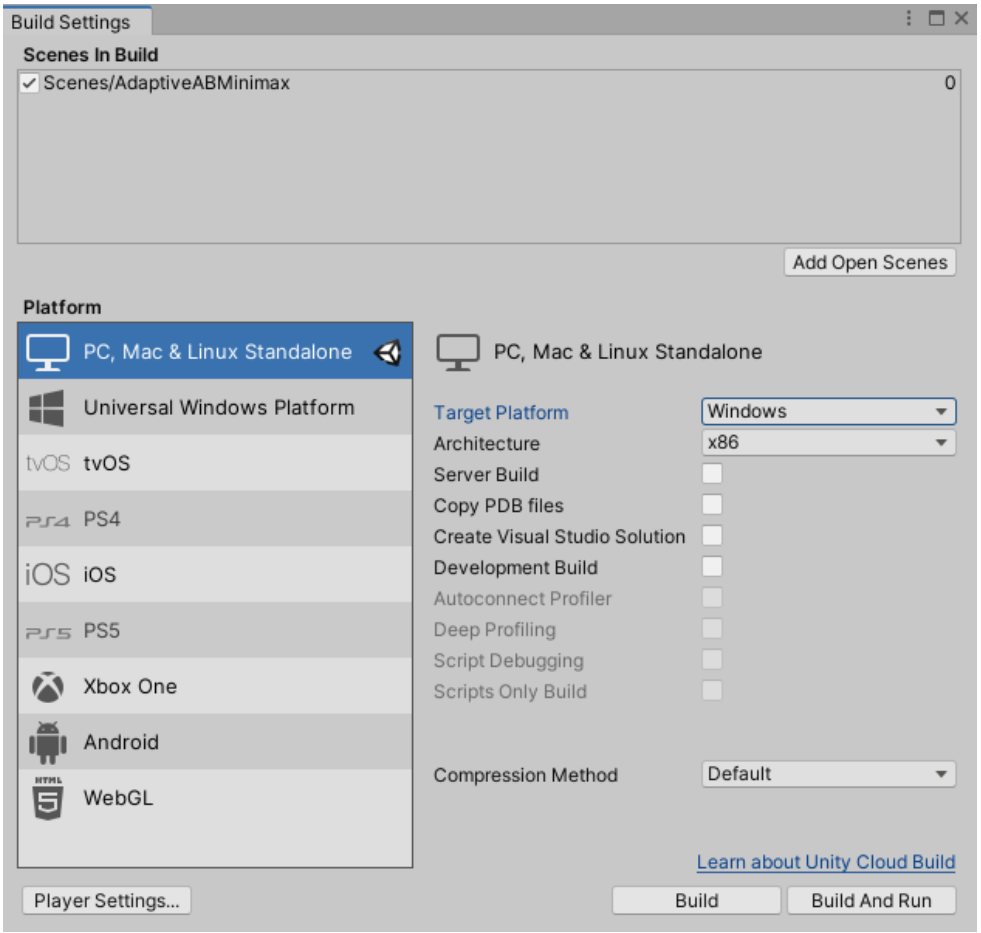


Figura 3.1. Obtenció de l'executable des de Unity. Font: Elaboració pròpia

# 4. Instruccions d'ús del prototip

## 4.1. Menú principal

Per tal de començar a jugar una partida, aquesta s'haurà de configurar prèviament des del menú principal.

El primer selector que apareix permet escollir les peces del jugador. D'aquest fet dependrà si un cop començada la partida realitzarà ell el primer moviment o ho farà el seu adversari: seguint les regles del joc de les dames, ho farà aquell jugador amb les peces negres.

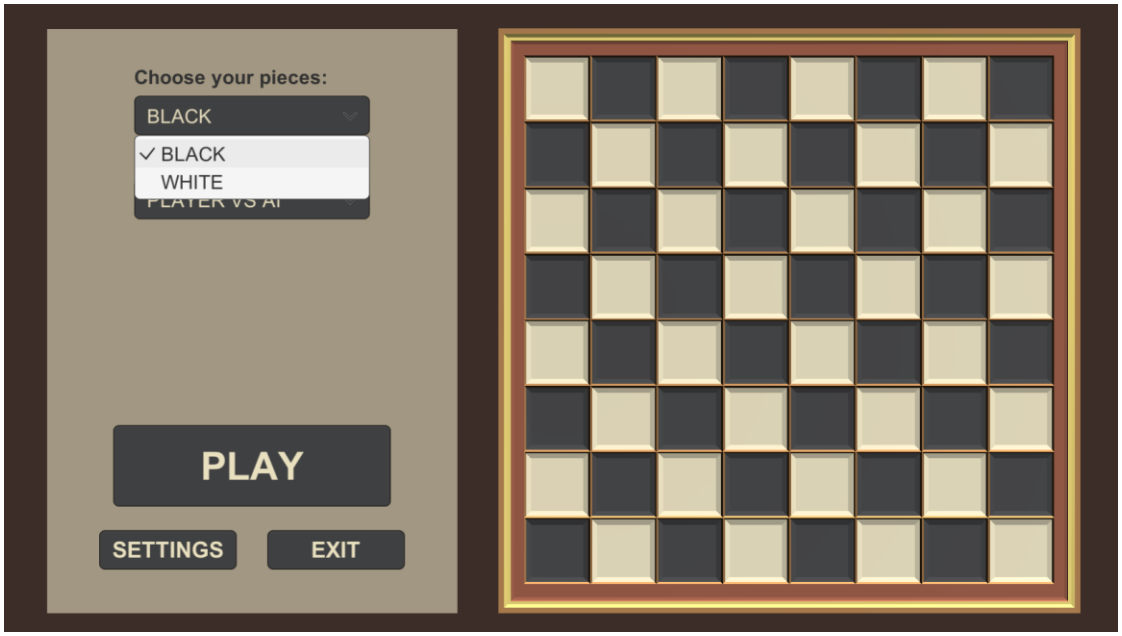


Figura 4.1. Selecció de les peces del jugador. Font: Elaboració pròpia

El segon selector permet determinar si les peces del jugador seran mogudes per ell mateix o per un algoritme que jugarà per ell. En aquest segon cas, el jugador no interactuarà amb el taulell durant la partida.

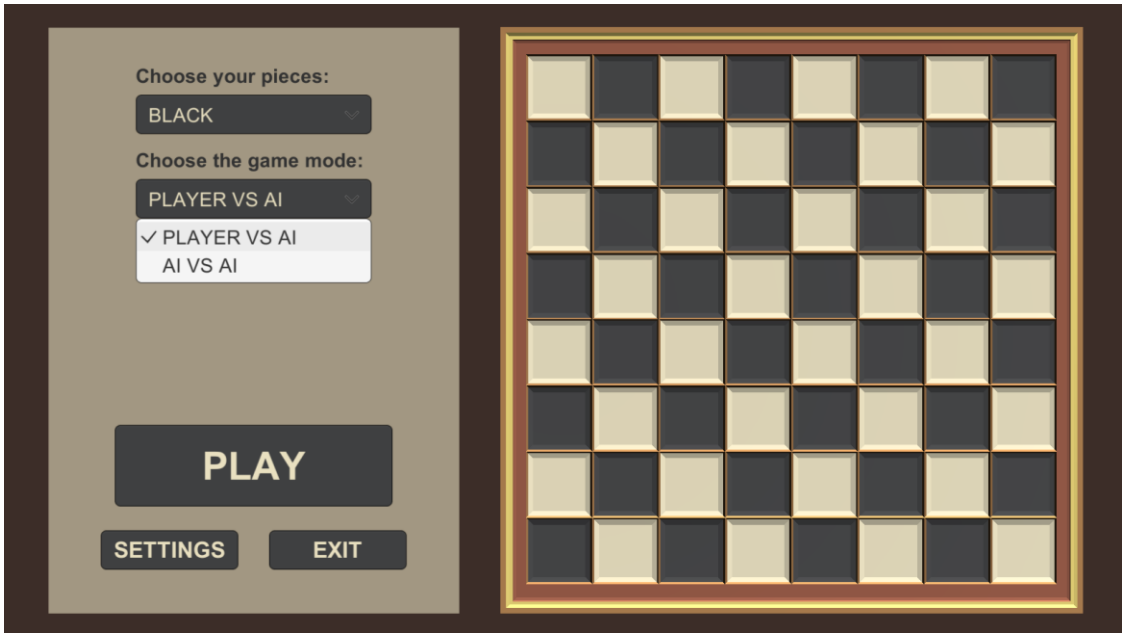


Figura 4.2. Selecció del mode de joc. Font: Elaboració pròpia

El segon selector permet determinar si les peces del jugador seran mogudes per ell mateix o per un algoritme que jugarà per ell. En aquest segon cas, el jugador no interactuarà amb el taulell durant la partida.

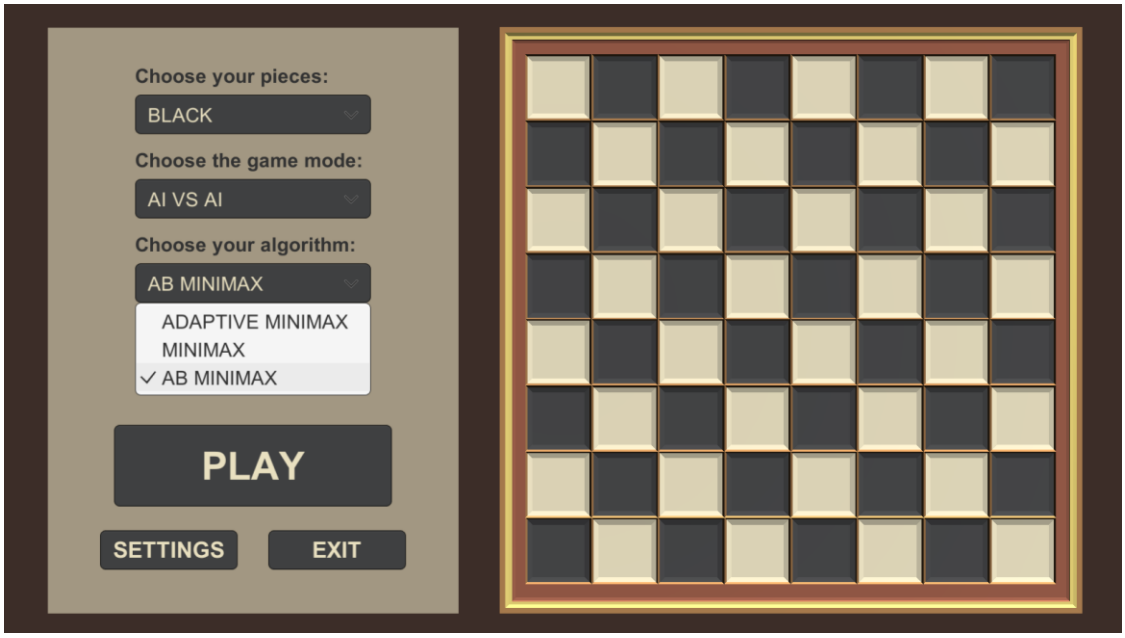


Figura 4.3. Selecció de l'algoritme. Font: Elaboració pròpia

Si en el selector anterior s'ha optat per triar la primera opció, la qual permetrà a l'algoritme adaptatiu enfrontar-se a ell mateix amb una ràtio predeterminada, caldrà determinar el valor d'aquesta.

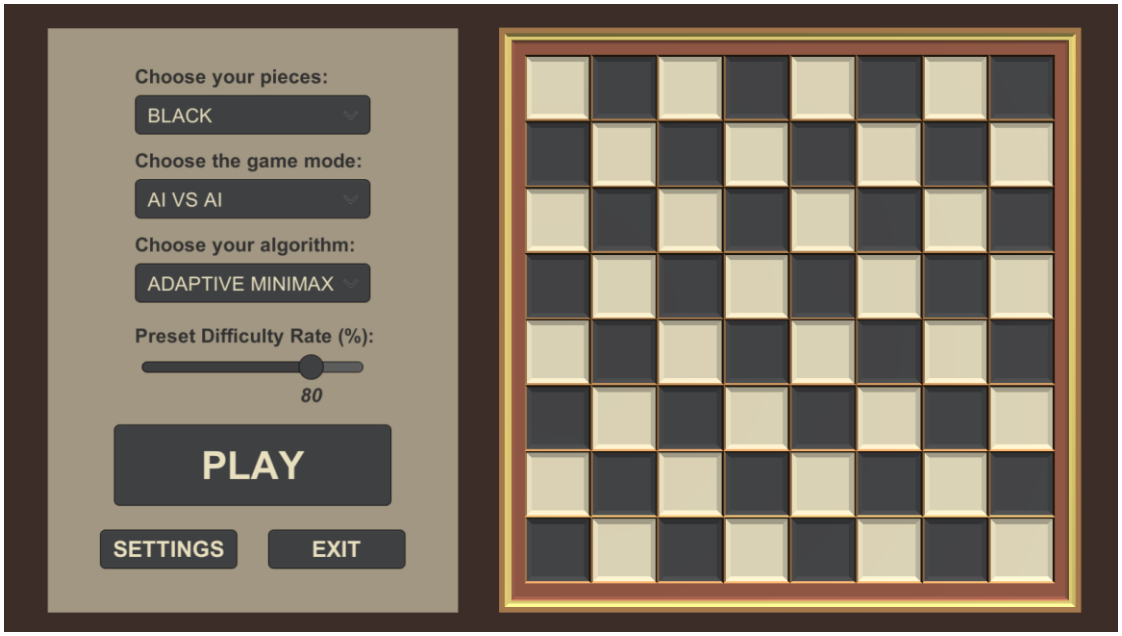


Figura 4.4. Selecció de la ràtio de l'algoritme adaptatiu. Font: Elaboració pròpia

Per últim, a través del botó "*Settings*" es podrà accedir a un segon menú d'opcions, explicat en el següent apartat. Un cop realitzada la configuració de la partida, aquesta podrà ser iniciada.

## 4.2. Menú d'opcions

Des d'aquest menú d'opcions es podran variar els diferents paràmetres amb relació al temps de processament i a la profunditat de cerca de l'algoritme o algoritmes presents en la partida.

Els dos primers paràmetres corresponen als temps mínim i màxim per variar la profunditat màxima de cerca dels algoritmes de manera dinàmica durant la partida: un temps de resposta inferior al mínim farà pujar-ne el valor, mentre que un temps superior al màxim farà abaixar-lo.

El tercer paràmetre correspon al temps d'interrupció de l'algoritme: si el temps de processament supera aquest valor, la cerca de l'algoritme serà interrompuda i aquest retornarà la millor opció trobada fins al moment.

El quart i últim paràmetre correspon a la profunditat de cerca inicial de la partida, amb la qual s'efectuarà el primer o primers moviments, abans de ser variada de manera dinàmica.

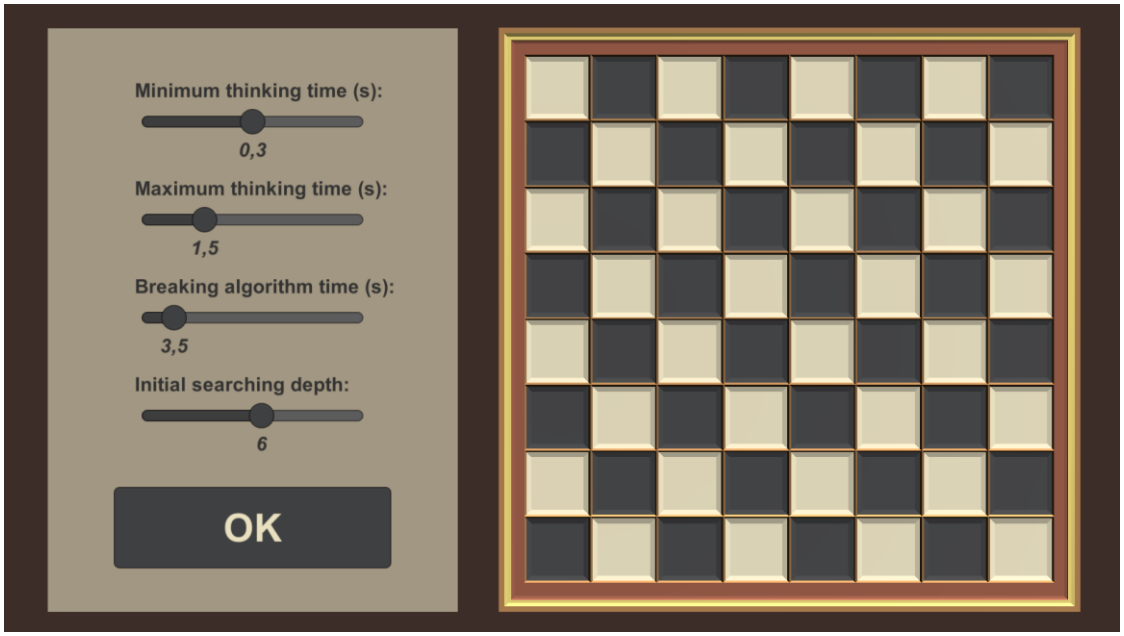


Figura 4.5. Menú d'opcions. Font: Elaboració pròpia

## 4.3 Interacció amb el joc

Durant el transcurs de la partida pot ser activada i desactivada l'opció "*Move Automatically*". Si aquesta opció està activada, l'algoritme o algoritmes presents en la partida iniciaran el seu moviment quan sigui el seu torn. En cas contrari, l'usuari haurà de prémer l'espai cada cop que vulgui que un dels algoritmes efectuï el moviment. Mentre que es recomana l'activació d'aquesta opció quan el jugador s'enfronta a l'algoritme, pot convenir que aquesta estigui desactivada si es vol seguir més còmodament el fil de la partida en cas que l'algoritme adaptatiu s'enfronti a ell mateix o a un altre algoritme.

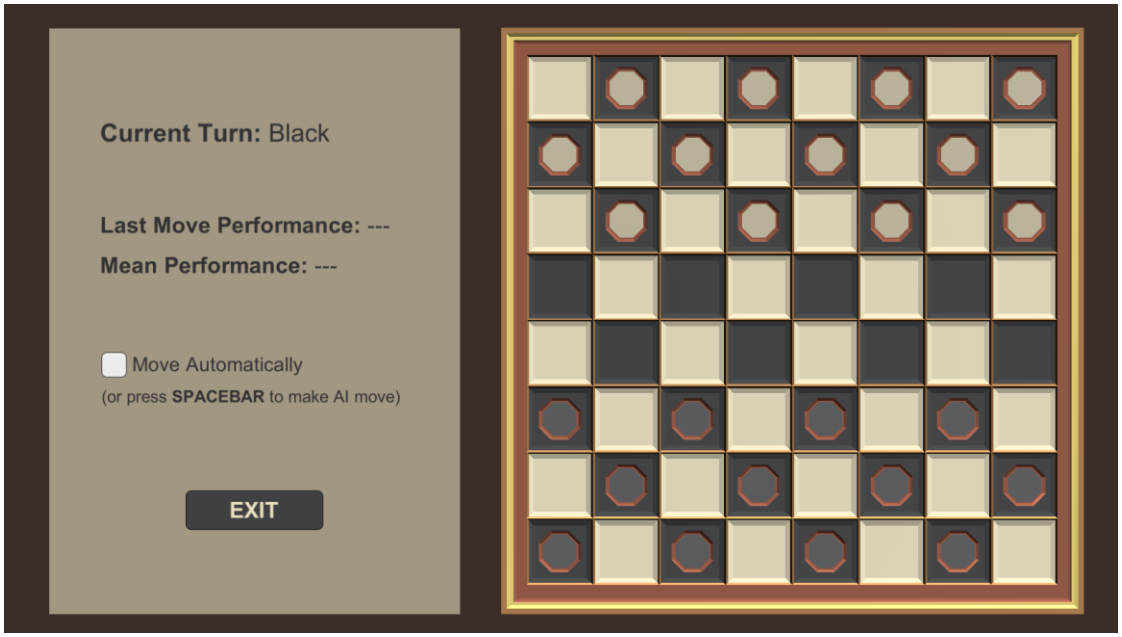


Figura 4.6. Situació inicial de partida. Font: Elaboració pròpia

En cas de ser el jugador qui s'enfronti a l'algoritme, aquest haurà d'interactuar amb les seves peces per efectuar els moviments durant el seu torn. Per tal de fer això, haurà d'utilitzar el ratolí per seleccionar la peça que vol moure. Si la peça en qüestió pot ser moguda, la casella on es troba es marcarà amb color verd, així com aquelles caselles a les quals pot desplaçar-se. En cas contrari, la casella de la peça seleccionada es marcarà amb color vermell.

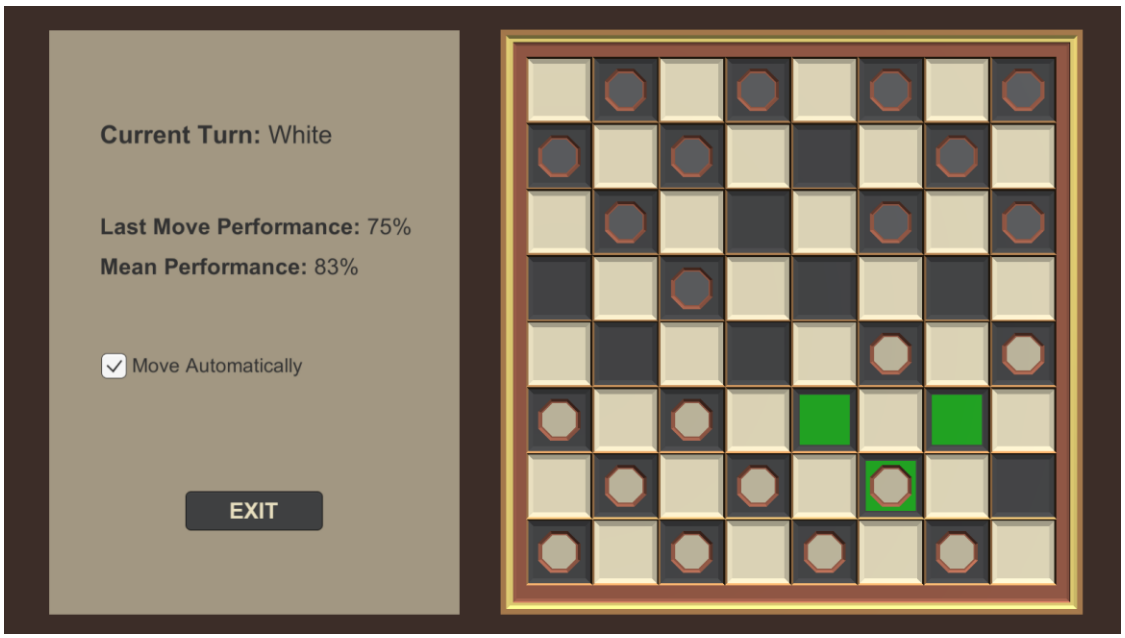


Figura 4.7. Possibles moviments de la peça seleccionada. Font: Elaboració pròpia

En cas que el moviment de la peça seleccionada elimini una o més peces enemigues, les caselles on aquestes estiguin situades es marcaran amb color vermell.


Figura 4.8. Moviment amb eliminació d'una peça rival. Font: Elaboració pròpia

Per tal de realitzar el moviment escollit, caldrà clicar amb el ratolí sobre la casella de destí. Si l'usuari vol canviar de peça i seleccionar-ne una altra, podrà fer-ho fent clic sobre aquesta nova peça.