

Checkers!



PROP 2023-24 Q1

Bernat Orellana

Tècniques d'intel·ligència Artificial

- Algoritmes per a jocs:
 - MIN-MAX
 - MIN-MAX amb poda alpha-beta
 - MIN-MAX amb iterative deepening

Dames americanes/ Checkers: El joc

- **Les dames** és un joc clàssic, que trobarem sovint citat com a "checkers" (anglès americà) o "draughts" (en anglès britànic)
- Hi ha dos jugadors, blanc i negre. Comença la partida el jugador negre.
- El tauler es configura inicialment tal i com es mostra a la Figura 1.
- Podeu consultar les regles de la variant americana [aquí](#). Cal destacar que:
 - La captura és obligatòria. No cal capturar el grup més nombrós, però sempre s'han de completar totes les captures possibles.
 - Les dames coronades (reis o reines) és poden moure només 1 posició o capturar en qualsevol direcció diagonal.
 - Si un jugador queda sense moviments, perd.
 - Es pot empatar. En el joc automàtic això succeeix si es donen 16 jugades sense victòria definida a partir del moment en que quedin 4 o menys figures (mínim 1 rei per jugador).

Figura 1: Situació inicial

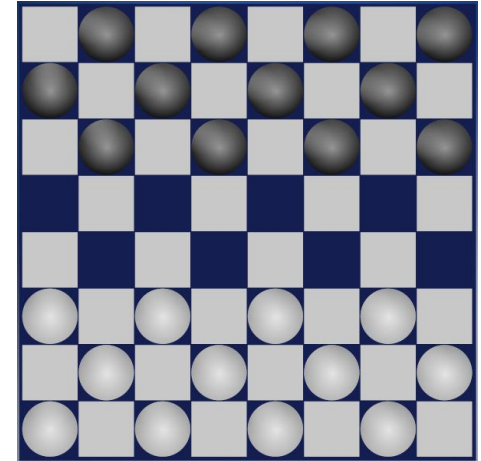
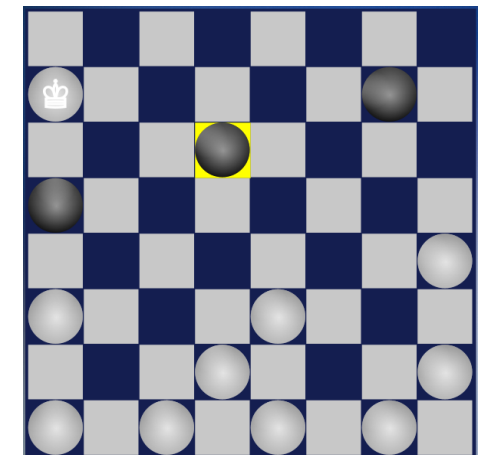
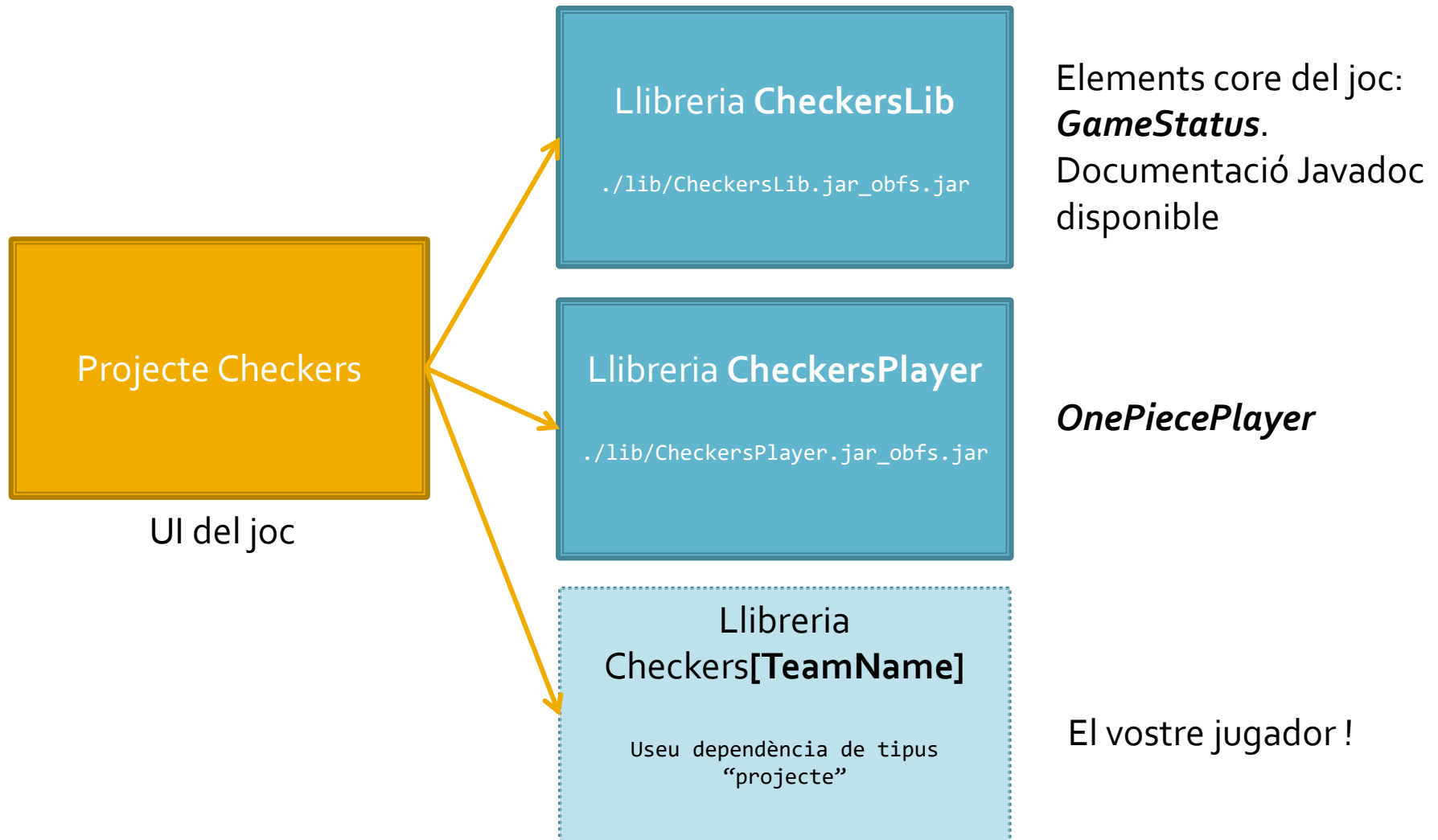


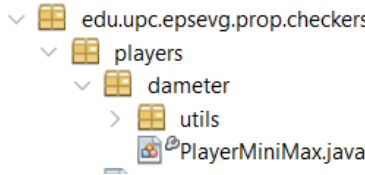
Figura 2: al llarg de la partida



Estructura del programa



IMPORTANT: Com organitzar el codi.

- Tot el vostre codi ha d'estar dins d'un projecte NetBeans de tipus **Java Library** amb el nom **Checkers[team_name]**. Per poder usar el vostre player des del projecte Checkers, cal que l'afegiu com a dependència.
- Heu de posar les vostres classes dins d'un package anomenat :
`edu.upc.epsevg.prop.checkers.players.[team_name]`

 - edu.upc.epsevg.prop.checkers
 - players
 - dameter
 - utils
 - PlayerMinMax.java
- Podeu crear d'altres classes en el mateix package o subpackages.
- El paquet ha de contenir els vostres jugadors, que s'hauran d'anomenar **PlayerMinMax.java** i **PlayerID.java**

Programació bàsica

■ Projecte **Checkers**

- Es subministren els següents Jugadors:
 - *HumanPlayer.java*
 - *RandomPlayer.java*
 - *OnePiecePlayer.java*
- Tots els jugadors implementen la interfície **IPlayer**, i els automàtics també la **IAuto** (vegeu **RandomPlayer**).
- Cal que afegiu el vostre jugador.

UI : inici

■ Game.java

- Permet triar els jugadors que es volen fer servir.
- Permet ajustar el *timeout* (en segons) de cada jugada.
- Es pot activar el mode interactiu per pausar el joc entre IAs.

```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(new Runnable() {  
        @Override  
        public void run() {  
            IPlayer player1 = new HumanPlayer("Human");  
            IPlayer player2 = new OnePiecePlayer(1);  
            new Board(player1, player2, 5, false);  
        }  
    });  
}
```

GB de
RAM usats

Mode
interactiu

API: llibreria CheckersLib

- Teniu el javadoc de la llibreria a [lib\javadoc](#)

Package edu.upc.epsevg.prop.checkers

package edu.upc.epsevg.prop.checkers

All Classes and Interfaces	Interfaces	Classes	Enums
Class	Description		
CellType	Tipus dels possibles continguts d'una cel·la.		
GameStatus	Aquesta classe representa l'estat del joc "Checkers" en un moment determinat, f		
IAuto	Identifica un jugador com automàtic		
IPlayer	Interfície que han d'implementar tots els jugadors (automàtics i manuals) de la n		
Level	Nivells de joc		
MoveNode	Classe per representar un node (moviment o tirada) dins de l'arbre de possibles i		
PlayerMove	La classe representa un moviment complet del joc i dades de l'exploració realitza		
PlayerType	Tipus de jugador: PLAYER1 o PLAYER2		
SearchType	Tipus de cerca		

API: llibreria CheckersLib

■ GameState

Class GameState

```
java.lang.Object*  
edu.upc.epsevg.prop.checkers.GameStatus
```

```
public class GameState  
extends Object*
```

Aquesta classe representa l'estat del joc "Checkers" en un moment determinat, format per: - el tauler de joc i la disposició de les peces - el jugador al que li toca tirar. - si el joc és terminal (hi ha un guanyador)

Constructor Summary

Constructors

Constructor	Description
GameState()	Crea un estat de joc amb un tauler buit i amb PLAYER1
GameState(int[][] matrix)	Constructor per a testing.
GameState(GameStatus copyFrom)	Fa la còpia completa (deep-copy) de l'estat de joc.

Method Summary

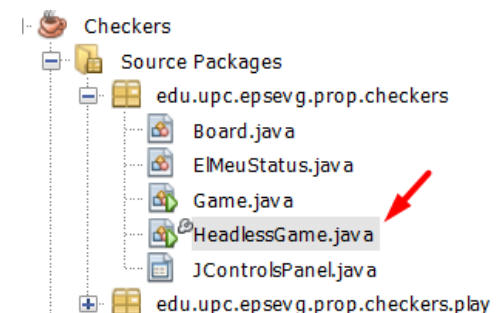
All Methods	Instance Methods	Concrete Methods	
Modifier and Type	Method	Description	
boolean	checkGameOver()	Verifica si el joc ha acabat i retorna cert si ja tenim un guanyador.	
boolean	currentPlayerCanMove()	Ens diu si el jugador actual pot moure	
boolean	equals(Object* obj)	Compara els estats dels taulers (revisa	
void	forceLoser()		
PlayerType	getCurrentPlayer()	Retorna el jugador actual (PLAYER1 o PLAYER2)	
int	getEmptyCellsCount()	Ens diu quantes caselles queden buides al tauler.	
List*MoveNode*	getMoves()	Retorna tots els possibles moviments	
MoveNode	getMoves(Point* p, PlayerType player)		
CellType	getPos(int x, int y)	retorna la peça que està a la posició indicada per les coordenades x i y.	
CellType	getPos(Point* p)	Retorna la peça que està a la posició indicada per les coordenades del punt	
int	getScore(PlayerType player)	Retorna el nombre de fitxes d'un dels jugadors.	
int	getSize()	Retorna la mida del costat del tauler (és 8...però podria canviar!)	
PlayerType	GetWinner()	Retorna el guanyador (només té sentit si isGameOver()==true)	
int	getVDirection(PlayerType player)	Direcció natural de moviment de cada jugador en l'eix de la Y.	
boolean	isGameOver()	Permet saber si el joc s'ha acabat (s'actualitza després de fer movePiece())	
void	movePiece(List*MoveNode* path)	El jugador actual (getCurrentPlayer()) mou una peça de la casella indicada per la coordenada to.	
String*	toString()	Retorna una representació senzilla del tauler en format cadena	
boolean	validateCoordinates(int i, int j)	Valida les coordenades de joc d'una casella	
boolean	validateCoordinates(Point* p)	Valida les coordenades de joc d'una casella	

HeadlessGame

- Podeu llançar proves massives fer estadístiques de resultats enfrontant a jugadors automàtics amb *HeadlessGame*.

```
public static void main(String[] args) {  
  
    IPlayer player1 = new OnePiecePlayer(1);//GB  
    IPlayer player2 = new RandomPlayer("Kamikaze 2");  
  
    HeadlessGame game = new HeadlessGame(player1, player2, 1/*s timeout*/, 10/*games*/);  
    GameResult gr = game.start();  
    System.out.println(gr);  
  
}
```

```
=====
===== RESULTS =====
=====
PLAYER 1 (                OncePiece):      wins 10  ties:0  looses:0
PLAYER 2 (                Random(Kamikaze 2)): wins 0   ties:0  looses:10
```



Objectiu

- Crear un jugador de **checkers** basat en l'algorisme minimax amb poda alfa-beta.
- Farem dues versions:
 - A) **PlayerMinimax**: El jugador parametritzat amb el nombre de nivells que explora.
 - B) **PlayerID**: El Jugador que utilitza el mecanisme de timeout previst per la interfície gràfica. Useu Iterative-Deepening per poder tallar en “qualsevol” moment la cerca.
- L'heurística és la part crítica d'aquesta activitat. Cal pensar molt bé què tenir en compte i com calcular-la de forma eficient.

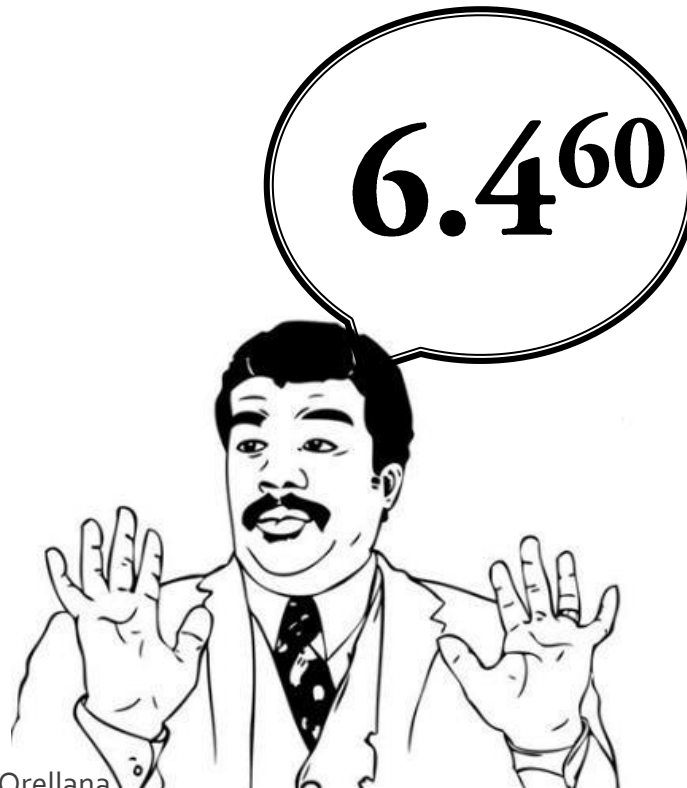
Objectiu

- **No** es demana un programa que apliqui una ~~estratègia~~ coneguda, sinó un algorisme que faci una **exploració minimax** sobre l'arbre de joc i heurística.
- Podeu usar coneixement del joc per millorar l'heurística.



Tingueu en compte que...

- “Having an average branching factor of about 6.4 and an estimated average game length of 60 ply.”



Possibles millores sobre minimax alfa-beta.

- Valoreu per la versió bàsica limitada en profunditat si podem ordenar l'exploració dels nodes de forma que es potenciï la poda alfa-beta. Useu heurístiques per determinar “les bones jugades” i provar-les abans.
- Useu taules de transposició per:
 - Per cada jugada explorada, recordar el millor node fill per prioritzar-lo si el tornem a trobar en iteracions següents, millorant així la poda alfa-beta.
 - Recordar heurístiques (o talls alfa/beta) ja calculades anteriorment de taulers idèntics i no tornar a fer la cerca.
- Taula d'obertures o finals.
- Pels més valents:
 - Implementar minimax amb paral·lelisme usant la idea de [Feldmann's Young Brothers Wait Concept](#) (YBWC)

Algunes recomanacions...

■ Recomanacions

- Seguir bones pràctiques en el desenvolupament.
- Programar tests per a cada funció en la mesura del possible.
- Estructura de packages.
- Tenir clara la responsabilitat de cada classe.
- Assegureu-vos que no useu ***System.out.println*** en els punts crítics de l'algorisme, sobretot quan distribuïu la vostra IA per competir, doncs fa molt més lenta l'execució.

Lliurament

- Entrega a Atenea segons data planificada.
- Grups de 2 persones.
- Només lliura un dels components del grup.
- Zip incloent 4 arxius:
 1. **Projecte.zip**: El vostre projecte NetBeans enzipat (Checkers[team_name]), esborreu les carpetes *dist* i *build*.
 2. **Documentació.pdf**: PDF amb documentació de la pràctica.
 3. **Javadoc.zip**: javadoc enzipat.
 4. **README.md**: Fitxer de text amb:
 - nom de l'equip, noms i DNIs dels alumnes.
 - URL GitHub del projecte
- El nom del zip està format pels noms dels alumnes de l'equip:
 - [Cognoms1Nom1]_[Cognoms2Nom2].zip

Activitat

- Fins a 1 punt extra a la nota final pel guanyador de la competició. (1 pel guanyador, 0.5 segon lloc, 0.25 tercer lloc, 0.1 quart)
- Puntuació del projecte (30% de la nota final de l'assignatura)
 - 70% Codi
 - Implementació del Jugador parametritzat amb profunditat.
 - Implementació del Jugador amb timeout (Iterative Deepening)
 - Resultats de joc
 - Contra Aleatori
 - Contra OnePiece
 - Contra Humans
 - Contra la resta de IAs dels companys:
 - modalitat A: profunditat fixada.
 - modalitat B: temps fixat.
 - Estratègies per optimitzar la poda.
 - Valoració del disseny i dels aspectes d'implementació.
 - 30% Documentació
 - Documentar la/es heurística/ques usades, i el mètode per provar-les / seleccionar-les
 - Comparar els temps d'execució de minimax i minimax amb iterative deepening per a un número de nivells fixat.
 - Gràfica de número de nivells baixats segons el temps disponible.
 - Explicar les estratègies d'optimització utilitzades:
 - Idea
 - Detalls d'implementació.
 - Justificació quantitativa de la incidència de la millora en els resultats.
- **(!) Activitat validada amb Prova de Validació a l'examen final.**

Activitat

- Condicions
 - El projecte ha de ser un treball original vostre
 - El codi ha de ser correcte
 - Netbeans no reporta errors
 - El projecte compila i funciona.
 - No es donen excepcions al llarg de la partida.
- Entregar documentació:
 - Documentació codi (javadoc)
 - Document explicatiu que inclogui:
 - URL del projecte a GitHub/Gitlab o altres repo Git.
 - Heurística dissenyada pel jugador:
 - Explicació detallada dels conceptes aplicats.
 - Explicació de com s'ha implementat.
 - Anàlisi de l'efecte de la poda sobre els resultats.
 - D'altres aspectes que es vulguin remarcar, particularitats del problema i de la vostra aproximació.
 - Petita taula que mostri la implicació de cada membre de l'equip en les diferents parts del projecte (indicar també percentatge de participació)