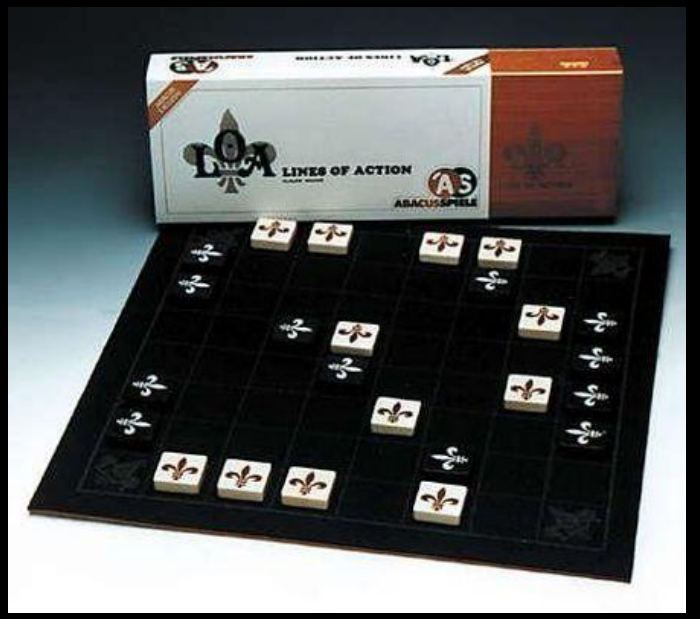
Lines Of Action (LOA)



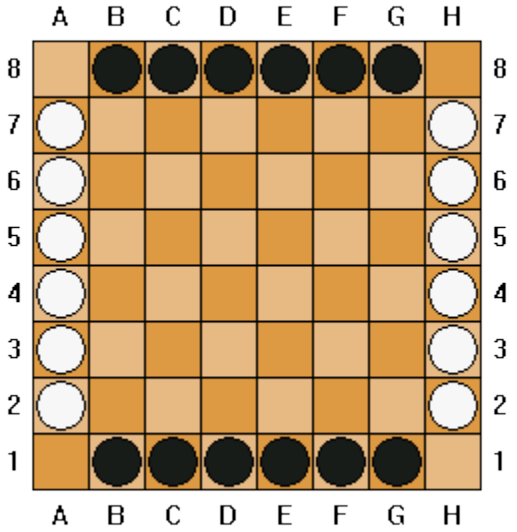
PROP 2021-22 Q1 Bernat Orellana

Tècniques d'intel·ligència Artificial

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

Lines Of Action: El joc

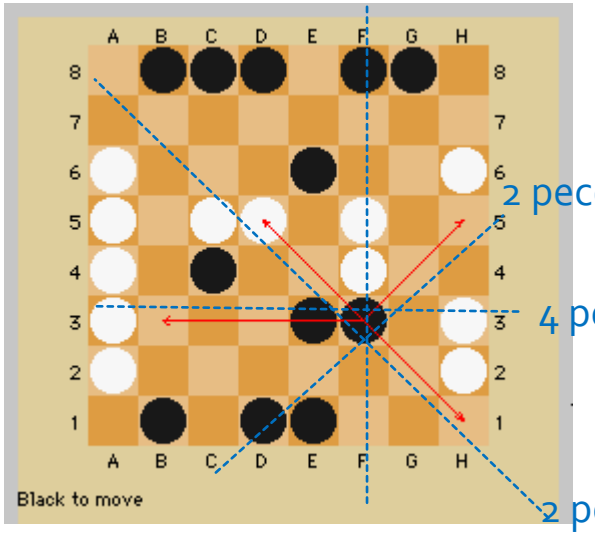
- LOA és un joc de tauler abstracte inventat per Claude Soucie.
- Objectiu dels jugadors és unir les seves peces en un únic bloc connex.
- Es consideren connectades les peces adjacents en horitzontal, vertical i **diagonal**.
- És un joc per torns clàssic, iniciant la partida el jugador que porta les negres.
- Inicialment cada jugador té 12 fitxes, disposades en dos files de 6 a banda i banda del tauler (vegeu figura a l'esquerra)



Regles de moviment

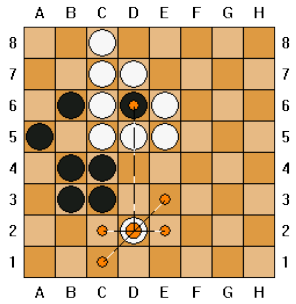
- A cada torn els jugadors han moure una de les seves fitxes segons les regles descrites a continuació.
- Les peces es poden moure en horitzontal, vertical o diagonal. El nombre de caselles que es pot moure en cada direcció ha de correspondre al nombre de fitxes que hi ha al llarg de la recta del moviment (la recta completa, en els dos sentits, de punta a punta del tauler).

- En aquest trajecte podem:
  - Saltar les nostres pròpies fitxes (saltar també conta un pas)
  - No està permès saltar les fitxes de l'adversari.
  - Si que podem **caure** sobre una fitxa de l'adversari. Aquest moviment es considera una **captura,** i la peça capturada es treu del tauler.

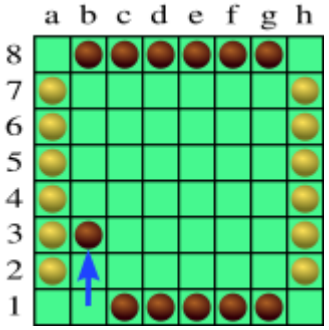


Regles de finalització

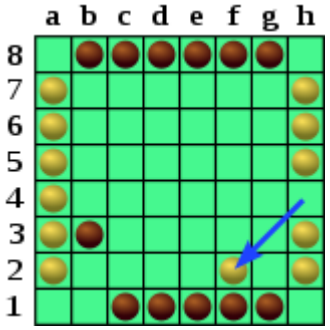
- Un jugador guanya si amb el seu moviment aconsegueix connectar totes les seves peces en un únic grup (connectitat 8-neighbours).
- Si capturem la peça d'un adversari i el seu conjunt de peces es redueix a una, l'adversari guanya.
- Si es dona una situació de moviment guanyador simultàniament pels dos jugadors, guanya el jugador que estava movent aquell torn.



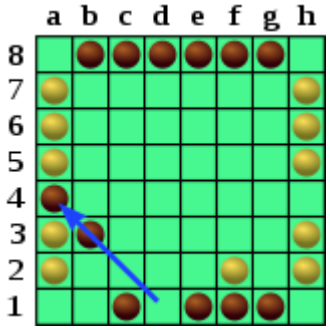
Exemple de partida



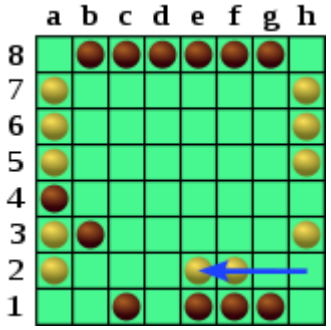
Bloqueig



Bloqueig



Captura – bloqueig – separació de grups



Bloqueig

Mes info a https://www.boardspace.net/loa/english/index.html

Programació bàsica

- Projecte Netbeans
  - Game: classe d'engegada
  - *Board : interfície gràfica del joc*
  - *JControlsPanel: petit panell de control*
  - *Jugadors:*
    - HumanPlayer
    - RandomPlayer
    - ***SurprisePlayer (alliberat properament…)***
  - Llibreria LOALib, que conté les classes "core".

Programació bàsica

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

UI i inici

- **Game.java**
  - Permet triar els jugadors que es volen fer servir.
  - Permet ajustar el *timeout* (en **segons**)de cada jugada.
  - Permet ajustar la dificultat del joc a tres possibles nivells:

```
SwingUtilities.invokeLater(new Runnable() {
    @Override
    public void run() {

        IPlayer player1 = new HumanPlayer("Octopus");
        IPlayer player2 = new RandomPlayer("Crazy Ivan");

        new Board(player1 , player2, 2, Level.DIFFICULT);
    }
});
```

```
public enum Level {
    DIFFICULT,
    MEDIUM,
    EASY
}
```

API: llibreria AmazonLib

- Teniu el javadoc de la llibreria a lib\javadoc

Package edu.upc.epsevg.prop.amazons

Interface Summary

Interface	Description
IAuto	Idenfitica un jugador com automàtic
IPlayer	Interfície que han d'implementar tots els Jugadors (automàtics i manuals) de la nostra aplicació.

Class Summary

Class	Description
GameStatus	Aquesta classe representa l'estat del joc "Amazons Game" en un moment determinat i el jugador al que li toca tirar.
Move	La classe representa un moviment complet del joc

Enum Summary

Enum	Description
CellType	Tipus de continguts d'una cel·la.
Level	Nivells de joc
SearchType	Tipus de cerca

API: llibreria AmazonLib

Interface IPlayer

All Known Implementing Classes:

CarlinhosPlayer

`public interface IPlayer`

Interfície que han d'implementar tots els Jugadors (automàtics i manuals) de la nostra aplicació.

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
java.lang.String	getName()	Retorna el nom del jugador que s'utlilitza per visualització a la UI
Move	move(GameStatus s)	Decideix el moviment del jugador donat un tauler i un color de peça que ha de posar.
void	timeout()	Ens avisa que hem de parar la cerca en curs perquè s'ha exhaurit el temps de joc.

HINT: MIREU LA IMPLEMENTACIÓ DE **RandomPlayer**

API: llibreria AmazonLib

Package edu.upc.epsevg.prop.loa

Class Move

java.lang.Object
    edu.upc.epsevg.prop.loa.Move

public class **Move**
extends java.lang.Object

La classe representa un moviment complet del joc i dades de l'exploració realitzada per obtenir-lo.

Constructor Summary

Constructors

Constructor
`Move(java.awt.Point from, java.awt.Point to, int numerOfNodesExplored, int maxDepthReached, SearchType searchType)`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method
java.awt.Point	`getFrom()`
int	`getMaxDepthReached()`
long	`getNumerOfNodesExplored()`
SearchType	`getSearchType()`
java.awt.Point	`getTo()`
void	`setMaxDepthReached(int maxDepthReached)`
void	`setNumerOfNodesExplored(long numerOfNodesExplored)`

API: llibreria AmazonLib

- GameStatus

Constructor Summary

Constructors

Constructor	Description
`GameStatus(int[][] matrix)`	Constructor per a testing.
`GameStatus(GameStatus copyFrom)`	Fa la copia completa (deep-copy) de l'estat de joc.
`GameStatus(Level level)`	Crea un estat de joc buit amb un nivell de dificultat indicat.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`CellType`	`getCurrentPlayer()`	Retorna el jugador actual (PLAYER1 o PLAYER2)
`int`	`getEmptyCellsCount()`	Ens diu quantes caselles queden buides al tauler.
`java.util.ArrayList<java.awt.Point>`	`getMoves(java.awt.Point from)`	Donada una posició on hi ha una peça, retorna tots els possibles destins de moviment
`int`	`getNumberOfPiecesPerColor(CellType currentPlayer)`	Ens diu quantes peces té el jugador del color passat per paràmetre
`java.awt.Point`	`getPiece(CellType t, int n)`	Retorna la posició de la peça del color i índex indicat.
`CellType`	`getPos(int x, int y)`	retorna la peça que està a la posició inidicada per les coordenades x i y.
`CellType`	`getPos(java.awt.Point p)`	Retorna la peça que està a la posició inidicada per les coordenades del punt
`int`	`getSize()`	Retorna la mida del costat del tauler (és 8...però podria canviar!)
`CellType`	`GetWinner()`	Retorna el guanyador (només té sentit si isGameOver()==true)
`boolean`	`isGameOver()`	Permet saber si el joc s'ha acabat (s'actualitza després de fer movePiece())
`boolean`	`isHighlighted(java.awt.Point p)`	Utilitzat per UI - no rellevant
`void`	`movePiece(java.awt.Point currentPosition, java.awt.Point newPosition)`	El jugador actual (getCurrentPlayer()) mou una peça de la casella indicada per la coordenada currentPosition a newPosition.
`void`	`setHighlight(java.awt.Point selectedPiecePos)`	Utilitzat per UI - No rellevant
`java.lang.String`	`toString()`	Retorna una representació senzilla del tauler en format cadena
`void`	`validateCordinates(int i, int j)`	Valida les coordenades de joc

API: llibreria AmazonLib

- CellType

Enum Constant Summary

Enum Constants

Enum Constant	Description
EMPTY	
PLAYER1	
PLAYER2	

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method	Description
static CellType	fromColor01(int c)	Convertir 0 i 1 a CellType
static CellType	opposite(CellType currentPlayer)	Donat un jugador, retorna el contrari.
static int	toColor01(CellType c)	Convertir CellType a 0 o 1
static CellType	valueOf(java.lang.String name)	Returns the enum constant of this type with the specified name.
static CellType[]	values()	Returns an array containing the constants of this enum type, in the order they are declared.

Methods inherited from class java.lang.Enum

compareTo, describeConstable, equals, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

Objectiu

- Crear un jugador de LOA basat en l'algorisme minimax amb poda alfa-beta. Farem dues versions:
  - El jugador parametritzat amb el nombre de nivells que explora.
  - El Jugador que utilitza el mecanisme de timeout previst per la interfície gràfica. Useu Iterative-Deepening per poder tallar en "qualsevol" moment.
- L'heurística és la part crítica d'aquesta activitat. Cal pensar-la molt bé:
  - què cal tenir en compte.
  - com calcular-ho eficientment.

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.

Objectiu

- Valoreu en la versió limitada en profunditat si podem ordenar l'exploració dels nodes de forma que es faciliti la poda alfa-beta.
- Valoreu quan feu la versió amb iterative deepening si podem usar informació de passades anteriors per millorar la poda.
- Use Zobrist-Hashing per evitar expandir cicles.

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.
  - Eviteu System.out en certs punts (excepte en mode depuració), doncs fa molt més lenta l'execució.

Qüestions prèvies

- Donades les dimensions del tauler i una estimació del nombre mig de moviments que poden tenir les fitxes, quantes jugades avalua un arbre minimax de profunditat 4?

Activitat

- Entrega a Atenea el 24 de Desembre a les 23:55h
- Grups de 2 persones
- Zip incloent 3 arxius:
  - **Projecte.zip**: Projecte NetBeans enzipat, incloent el vostre player i classes accessòries.
  - **Documentació.pdf**: Document PDF amb documentació
  - **Javadoc.zip**: documentació enzipada.
  - **Readme.txt**: Fitxer de text amb nom de l'equip, noms i DNIs dels alumnes.
- El nom del zip són els noms dels alumnes de l'equip:
  - [Cognom1Nom1]_[Cognom2Nom2].zip

Activitat

- 30% de la nota final de PROP !
- Fins a 1 punt extra a la nota final pel guanyador de la competició a temps fixat. (1 pel guanyador, 0.5 segon lloc, 0.25 semifinalistes)
- Nota
  - 70% Codi
    - Implementació del Jugador parametritzat amb profunditat.
    - Implementació del Jugador amb timeout (iterative deepening)
    - Resultats de joc
      - Contra Aleatori
      - Contra Humans
      - Contra jugador sorpresa.
        - modalitat A: profunditat fixada.
        - modalitat B: temps fixat.
    - Estratègies per optimitzar la poda.
    - Zobrist-Hashing
    - Valoració del disseny i dels aspectes d'implementació.
    - Demostració de l'ús de Git per la realització de la pràctica.
  - 30% Documentació
    - Javadoc
    - Document d'anàlisi, incloent:
      - URL del projecte a GitHub/Gitlab o altres repo Git. Proporcioneu clau si és privat.
      - Descripció de l'heurística.
      - Detalls rellevants de la vostra implementació.
      - Estudi del nombre de nivells baixats segons el temps disponible de tirada. Recordeu: el nivell "efectiu" del IDS és l'últim completat, no el del que queda tallat a mitges.
      - Influència de la poda en el nombre de nodes avaluats / nivells baixats.
      - Influència de la informació d'iteracions anteriors del IDS en el nombre de nodes avaluats / nivells baixats.
      - 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)
- **El codi ha de ser original vostre.**
- **(!) Activitat validada amb Prova de Validació a l'examen final.**