# PRÀCTICA 2: OTHELLO

PROGRAMACIÓ II GEI

**Treball fet per:**
Pau Esquerda Masip - 48056677V
26/04/2025 PRIMER CURS GEI - UDL

## Classe Position

Aquesta classe s'utilitza per a representar un punt en un espai bidimensional usant una coordenada X (row) i una coordenada Y (column). Es una classe simple que consta d'un constructor i getters per a obtenir les coordenades X i Y. No he experimentat cap dificultat al realitzar aquesta classe.

## Classe Direction

Aquesta classe també es bastant senzilla. Tenim 8 direccions possibles, { `N, NE, E, SE, S, SW, W, NW` } que representen diferents vectors unitaris donats per les direccions verticals, horitzontals i diagonals. El constructor rep dos valors, la variació en la direcció X i la variació en la direcció Y per tal d'establir un vector de direcció.

La classe Direction també consta d'un mètode `move()` que ens permet moure'ns en una direcció determinada. Utilitzem els getters de la classe position per a obtenir la posició inicial, després sumarem la variació en X i en Y per a obtenir la posició resultant de moure'ns en una determinada direcció.

**Funcionament del mètode move()**

- Obtenim les coordenades de la posició actual
- Els sumem la variació en X i en Y
- Retornem una nova posició actualitzada

## Classe Cell

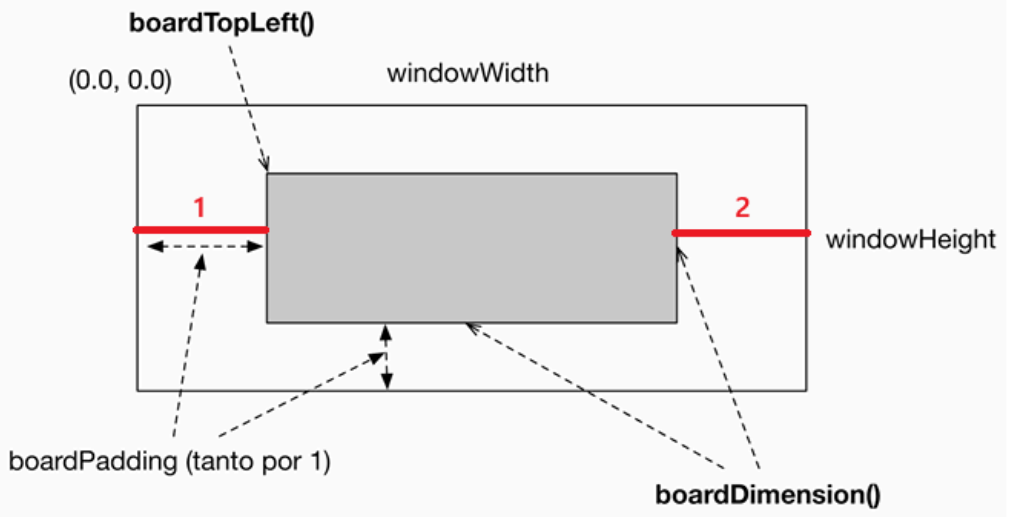
Aquesta classe s'utilitza per a representar cada una de les cel·les del tauler. Aquestes, a diferencia de les classes anteriors, poden canviar d'estat, és a dir, poden passar a ser blanques, negres o simplement estar buides. Aquest estat s'emmagatzema a la variable state. Tenim tres mètodes on cada un retorna cada un dels estats possibles. També tenim definits tres altres mètodes que retornen true si la cel·la a la que s'apliquen té el mateix estat. Aquests son: `isEmpty()`, `isWhite()`, `isBlack()`.

Finalment tenim dos mètodes més per a canviar l'estat d'una cel·la a blanc o a negre, aquests son: `setWhite()` i `setBlack()`. L'últim mètode que hem implementat ha estat `reverse()`. L'única funció d'aquest es canviar l'estat d'una cel·la al contrari.

## Classe Geometry

Aquesta classe serveix per a agrupar i gestionar tots els mètodes relacionats amb les dimensions i posicions dels elements gràfics del joc.

Es aquí on he trobat més dificultats que en les anteriors classes. Primerament, en el mètode `boardDimension()` no vaig tenir en compte que el marge s'aplica en tots els costats, a part tampoc vaig tenir en compte que el padding estava expressat en tant per 1 per tant simplement vaig restar el padding a la mida de la finestra. Després de llegir curosament les instruccions vaig entendre com estava expressat i com s'aplicava aquest al tauler.




```
double totalWidth = windowWidth * (1- 2*boardPadding);
double totalHeigh = windowHeight * (1- 2*boardPadding);
```

Aquí podem observar l'error corretgit, el padding s'ha d'aplicar 2 vegades.

El mateix em va passar amb `tokenDimension()` però al detectar el primer error també vaig corretgir aquest segon.

## Classe Board

Aquesta és la classe encarregada de representar el tauler de joc. Es complementa amb la classe Game per realitzar les variacions dels estats conforme el joc avança.

La informació guardada per aquesta classe és la següent:

- Order: És l'ordre del taulell, l'hem utilitzat per a calcular la mida, el nombre de caselles entre d'altres coses.
- Cells: Es una matriu que emmagatzema les diferents cel·les del taulell, l'hem usat per a definir les 4 fitxes inicials al principi de la partida i per seleccionar fitxes concretes ja sigui per a canviar-les de color, invertir-les, etc.
- Display: S'usa per a accedir a la interfície gràfica.
- Black: Comptador de fitxes negres.
- White: Comptador de fitxes blanques.

La creació del tauler es fa al constructor, on es genera una matriu de dimensions 2*order x 2*order, inicialitzant totes les cel·les com a buides. Mitjançant el mètode `initBoard()`, les peces inicials es col·loquen al centre del tauler seguint les regles típiques d'un joc com Othello.

El mètode `size()` retorna la mida de la matriu, que és útil per realitzar comprovacions dels límits del tauler.

Un altre mètode important és `contains()`, que ens permet saber si una determinada posició es troba dins de la matriu. El mètode s'utilitza sovint per garantir que els moviments en una determinada direcció no es produeixen fora dels límits de la matriu.

Aquí vaig tenir alguna dificultat amb el mètode `initBoard()` ja que aquest ha inicialitzar totes les cel·les a EMPTY exceptuant les 4 del centre que han de tenir 2 peces WHITE i 2 peces BLACK. Això ho vaig aconseguir usant la classe position, trobant les coordenades del centre del tauler amb `cells[][]` i canviant el color de les fitxes amb `setBlack()` i `setWhite()`.

En aquesta classe, a l'igual que en la classe Cell també podem trobar un mètode `reverse()` que ens serveix per canviar de color una determinada posició.

## Classe Game

La classe Game s'encarrega de controlar la lògica del joc i gestionar l'estat general de la partida. Utilitza la classe Board per interactuar amb el tauler i s'encarrega de determinar quins moviments són vàlids, canviar els torns i aplicar les regles per capturar peces.

Un dels mètodes principals és `isFinished()`, que comprova si cap dels jugadors pot continuar jugant. En aquest cas, l'estat del joc canvia a `State.FINISHED`.

Per tal de comprovar l'estat de cada casella respecte al jugador actual, tenim els mètodes `isSame()` i `isOther()` que ens indiquen si una posició pertany al mateix jugador o la contrària, respectivament.

En aquests últims mètodes podem veure clarament la interacció amb la classe Board.

En el mètode isSame, primer es verifica si el jugador és `State.BLACK`. Si és així, s'invoca `board.isBlack()` per determinar si la posició conté una peça negra. Si no hi ha cap peça negra, s'analitza si és blanca mitjançant `board.isWhite()`. Si cap de les dues opcions no coincideix es retorna false. El mètode `isOther()` funciona de manera molt similar.

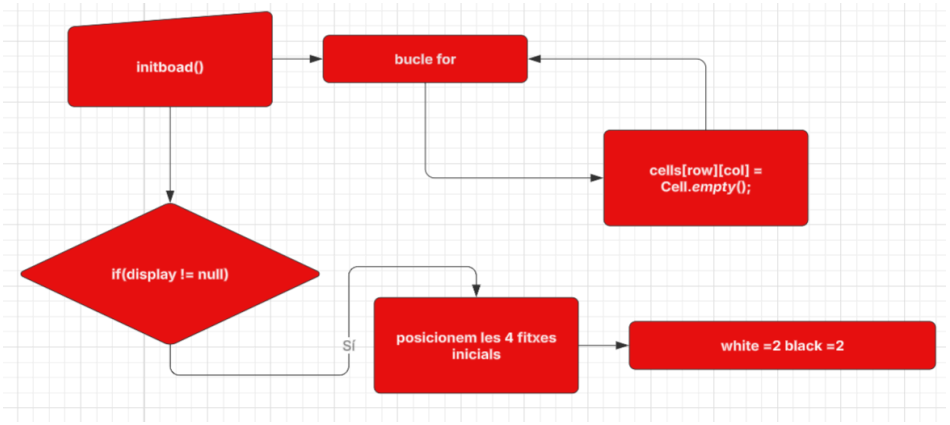
Vaig tenir dificultats a l'hora de programar el mètode `someSame()` ja que no vaig tenir en compte la possibilitat de que el tauler estés buit o de que simplement contingui la següent posició obtinguda mitjançant `direction.move()` i que aquesta no sortís del tauler, però finalment aquests problemes van ser solucionats.

# 3 Funcions a destacar:

### initBoard()

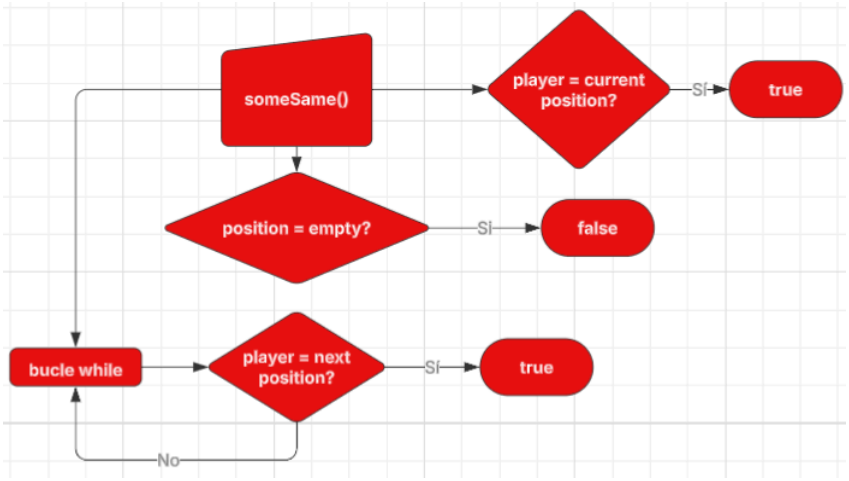
Com hem comentat anteriorment, aquesta funció és l'encarregada de inicialitzar el tauler com es faria al principi d'una partida.

- Usant un bucle for recorrem el tauler de joc assegurant que cada cel·la es troba en l'estat EMPTY, és a dir, sense cap fitxa.
- Després calculem el centre del tauler dividint size() entre 2.
- Si display no es null, és a dir, el tauler és present establim al centre d'aquest 2 fitxes WHITE i 2 fitxes BLACK.
- Ajustem el contador de fitxes white i black a 2.



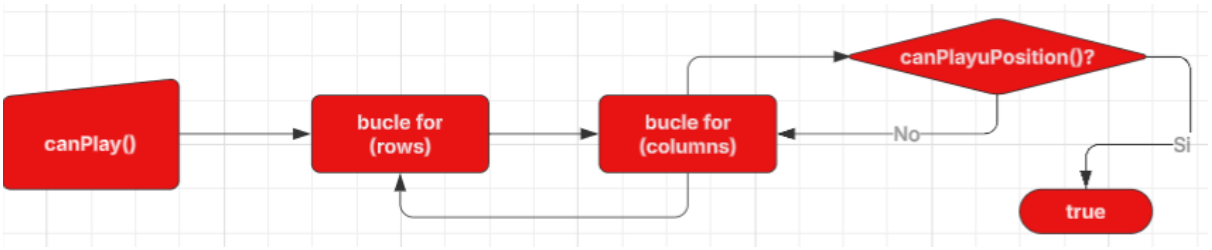

### someSame()

- Si la posició actual és del mateix color que la que hem introduït retornem true
- Si la posició està buida retornem false
- Iterem en la direcció donada amb un bucle while fins a trobar una posició amb el mateix color que l'introduït.
- Si no la trobem retornem false.



**canPlay()**

- recorrem tot l'array de cel·les cells[][]
- comprovem una a una si la posició es jugable
- si ho és retornem true, si finalmente no n'hem trobat cap retornem false




## Altres observacions:

- La complexitat dels algorismes usats es bastant reduïda ja que es limiten a recórrer un tauler bidimensional. Això implica una complexitat de (O(n^2) per a un n petit).
- Els tests han estat de gran ajuda per a trobar l'origen i resoldre problemes relacionats amb les classes Geometry, Game i Board. Tot i així, el SimulatedGame1Test no ha acabat de funcionar encara que el joc es comporta de manera correcta, s'inicia bé i la partida té un transcurs normal.

## Aprenentatge personal:

Aquesta pràctica m'ha ensenyat a treballar amb coordenades i entorns gràfics. He entès com s'estructura de manera ordenada i entenedora un programa d'aquest estil. Cada classe fa la seva funció i juntes es complementen per a oferir un resultat final complet i funcional.

He après també a trobar errors encara que aparentment sembli que el codi estigui funcionant correctament i he entès com cada un d'aquests afecta a tot l'entorn i com al solucionar-lo es posen a lloc altres seccions del programa. En general ha estat una pràctica molt profitosa.