

# Projet

## Jeu De Dames

Nom, prénom	Abir Hsaine
Nom, prénom	Zakaria Raji

### Introduction :

Le but de ce projet est de programmer un jeu de dames en java en se basant sur les règles internationales. Afin d'y parvenir, il nous a fallu analyser tout d'abord le sujet à l'aide de diagrammes UML de classes et nous organiser en nous répartissant les tâches à accomplir, du jeu sur console au jeu graphique voir partie analyse et conception, puis nous sommes passer à l'implémentation du programme à proprement parler, la description des algorithmes se trouve donc dans Réalisation et implémentation, enfin nous avons bien évidemment rencontré beaucoup d'obstacles qui seront listés dans Problèmes rencontrés et solutions apportées.

Afin de mener à bien ce projet, il a donc été décidé d'implémenter premièrement une version console du jeu de dame, pour après programmer une première interface graphique qui nous permet d'enregistrer les pseudos des joueurs, puis une deuxième interface graphique qui contient le plateau et plusieurs paramètres du jeu (nombre de dames, nombre de pions restants, le score, le tour du joueur), et qui permet aussi de quitter la partie de jeu et annoncer le gagnant.

Voici le lien de la démonstration du projet (sur drive):

<https://drive.google.com/file/d/1tbHPSIzZx2XzxBfJV56yqjznWmrR4GcU/view?usp=sharing>

## 1. Objectif :

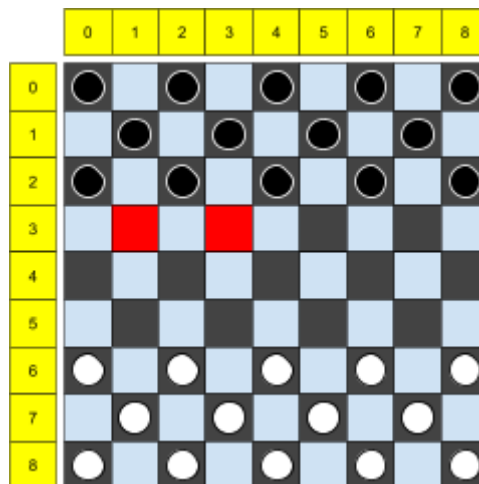
Développer un jeu de dames basique :

- Enregistrer les pseudos des deux joueurs au lancement du jeu.
- Affichage du nombre de pièces restantes pour chaque joueur(nombre de jetons, nombre de dames).
- Souligner le joueur actif.
- Détection de fin de jeu soit par gagnant (automatiquement détecté) ou bien par désistement(en cliquant sur un bouton d'annulation).
- Mettre en place une architecture MVC (Model, vue, controller).

## 2. Concept du jeu :

Dans notre jeu de dames, le plateau est composé de  $9 \times 9 = 81$  cases, alternativement claires et foncées.

Par défaut, le joueur en bas à la main pour faire la première action (déplacer son pion choisi sur une position proposée).



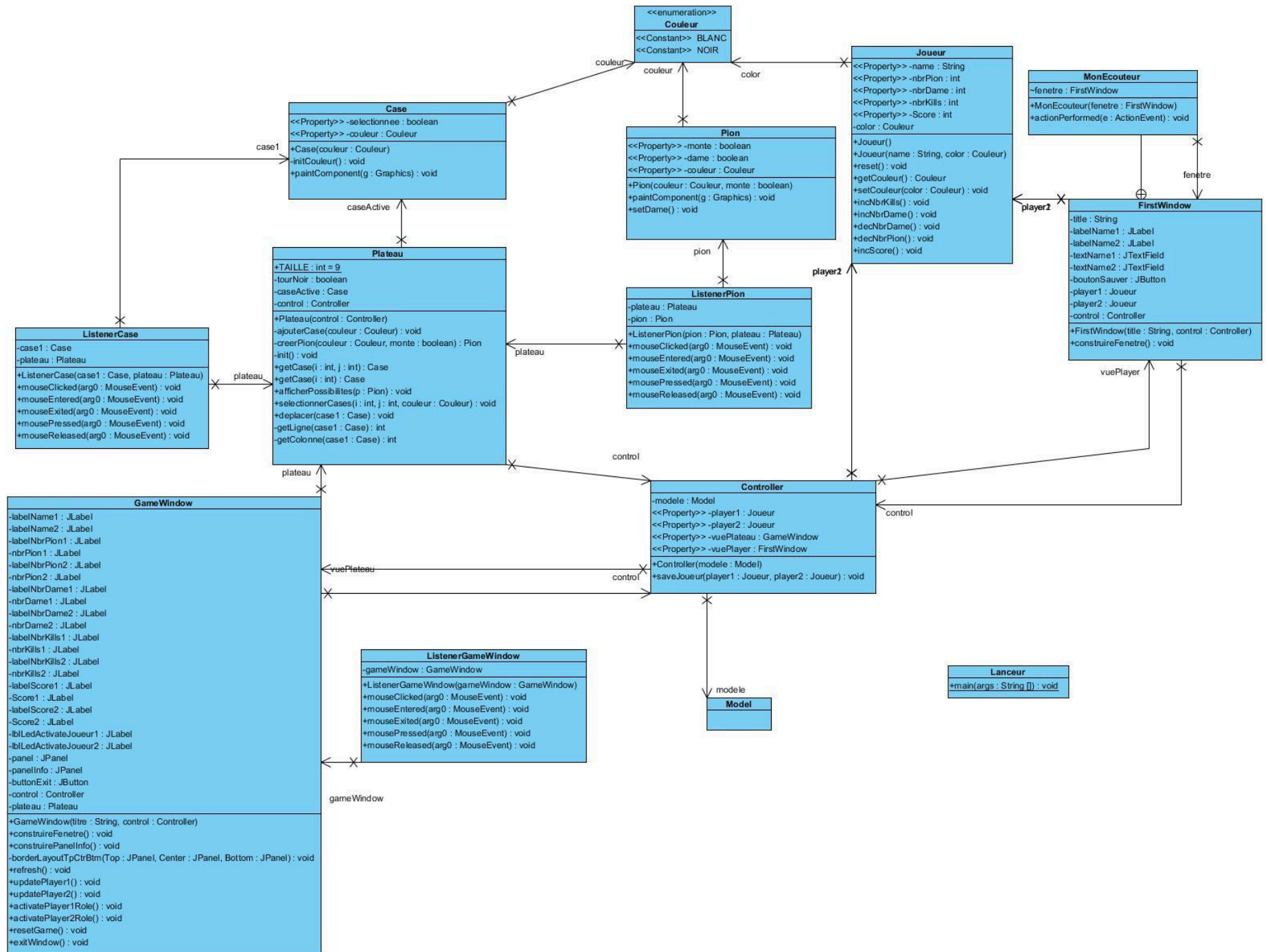
## 3. Perspective - Machine :

A l'oeil nue, le plateau est considéré comme des lignes et des colonnes, c'est comme si on a un tableau de deux dimensions. En revanche, dans notre cas on utilisera qu'un seul tableau a une dimension pour les **cases**.

Par exemple, si la taille est égale à 9, le nombre de cases est de 81 cases. Par la suite, on peut diviser le numéro de chaque case sur la taille pour déterminer la ligne où se trouve la case (EX :  $\text{Case}_{15} / 9 \sim 2 \Rightarrow$  existe dans la 3eme ligne car en comptant 0 aussi).



# Jeu De Dames - Abir - Zakaria



### 4. Lanceur

```
Model modele;  
Controller controle;  
FirstWindow vuePlayer;
```

La classe lanceur contient la méthode `main(String[] args)` qui permet de lancer la première fenêtre du jeu (VuePlayer : **FirstWindow**) où on enregistre les pseudos des deux joueurs.

### 5. FirstWindow

La classe **FirstWindow** permet de générer la première fenêtre qui donne la main à l'utilisateur pour saisir les informations des deux joueurs, elle est définie par :

- Les deux joueur {player1, player2} de Type **Joueur**;
- Un titre de la fenêtre {title};
- Un contrôleur va transmettre les informations de la première fenêtre vers la deuxième.
- Deux Libelles {labelName1,labelName2} et deux champs {textName1,textName2} pour saisir le nom de chaque joueur ainsi qu'un bouton {boutonSauver} pour confirmer et commencer le jeu.

Dans la même classe on a mis en place un listener {MonEcouleur : `actionPerformed`} qui va écouter le clique sur le bouton afin de créer deux instances/objets pour les joueurs et les passer vers la deuxième fenêtre à travers le contrôleur {`control.saveJoueur(player1,player2);`}.

### 6. GameWindow

La classe **GameWindow** représente la fenêtre principale du jeu contient le damier/plateau ainsi que les informations des deux joueurs, elle est définie par :

- Control : Controller, contient tous les informations de deux joueurs ainsi que les deux vue {gameWindow et FirstWindow}
  - Le nom de deux joueurs : **JLabel** {labelName1,labelName2}
  - Nombre de pion de chaque joueur : **JLabel** {nbrPion1, nbrPion2}
  - Nombre de Dame de chaque joueur : **JLabel** {nbrDame1, nbrDame2}
  - Nombre de Kills/pions\_tués : **JLabel** {nbrKills1, nbrKills2}
  - Score : **JLabel** {Score1, Score2};
- + `refresh()` : void, permet d'actualiser les informations des deux joueurs { `updatePlayer1()`, `updatePlayer2()` } sur la fenêtre. Cette fonction est appelée lors de chaque déplacement (avec Kill) dans la méthode `deplacer()` de la classe Plateau.

### 7. ListenerGameWindow

La classe **ListenerGameWindow** est un Listener qui a pour but de quitter la fenêtre lors du clique sur le {buttonExit} de la fenêtre **GameWindow**.

### 8. Case

La classe Case est l'élément principal du jeu de dames, qui va construire le damier/grille, chaque case noir peut contenir q'un seul pion, elle est définie par :

- Couleur : énumérateur {noire, blanche}
- Selectionnee : Boolean, permet de déterminer si la case a été sélectionnée (reference Plateau : afficherPossibilites(..))

### 9. ListenerCase

La classe ListenerCase a pour but d'écouter le clique sur une case du plateau :

- case1 : Case
  - plateau : Plateau
- + mousePressed() : void, permet de déplacer le pion dans la case sélectionnée.

### 10. Couleur

Dans la classe couleur, on a défini les couleurs (blanc,noir) qu'on va utiliser pour les pions ainsi que pour les cases du plateau.

### 11. Joueur

```
private String name;  
private int nbrPion;  
private int nbrDame;  
private int nbrKills;//nbr de pion que l'adversaire a perdu  
private int Score;  
private Couleur color;//noir ou blanc
```

- C'est la classe regroupant les informations d'un joueur à savoir son pseudonyme, le nombre de pions de début de la partie, le nombre de kills (le nombre de pions que le joueur a perdu), le nombre de dames, le score et la couleur de ses pions.
- Cette classe comporte également des méthodes basiques pour incrémenter le score et décrémenter ou incrémenter le nombre de kills, le nombre des pions restants, le nombre de dames ainsi qu'une méthode reset qui nous permet de recommencer de nouveau la partie de jeu en mettant à 0 tous les paramètres du jeu.

```
public void incNbrKills() { this.nbrKills++;}  
public void incNbrDame() { this.nbrDame++;}  
public void decNbrDame() { this.nbrDame--;}  
public void decNbrPion() { this.nbrPion--;}  
public void incScore() { this.Score++;}
```

### 12. Pion

```
private Couleur couleur;  
private boolean monte;  
private boolean dame;
```

Dans cette classe, on a utilisé une méthode `paintComponent(Graphics g)` qui nous permet de dessiner les pions en format de cercle et de choisir la couleur des pions (blanc, noir). Ensuite, on a utilisé une méthode qui nous permet de savoir le status du pion si il est en montée ou pas : `ismonte()` en retournant `true` par la variable `monte` ou non si `false` et une autre pour savoir si le pion est une dame en retournant `true` par la variable `dame` ou non si `false` : `isDame()`.

### 13. ListenerPion

```
public void mouseClicked(MouseEvent arg0)  
public void mouseEntered(MouseEvent arg0)  
public void mouseExited(MouseEvent arg0)  
public void mousePressed(MouseEvent arg0){  
    plateau.afficherPossibilites(pion);}  
public void mouseReleased(MouseEvent arg0)
```

Cette classe contient tous les événements nécessaires pour la souris (click, relacher). L'événement `mousePressed()` est important car on va l'associer à la méthode

afficherPossibilites(pion) qui permet de proposer les possibilités de déplacement du pion et qui sera bien détaillée dans la classe Plateau.

### 14. Controller

```
private Model modele;  
private Joueur player1;  
private Joueur player2;  
private GameWindow vuePlateau;  
private FirstWindow vuePlayer;
```

C'est la classe qui va nous permettre de récupérer les noms des 2 joueurs et de les enregistrer, puis de détruire la première fenêtre vuePlayer et de construire la deuxième fenêtre vuePlateau à l'aide de la méthode saveJoueur().

```
public void saveJoueur(Joueur player1,Joueur player2)  
{ // mise a jour du modele : maj de joueur selectionne  
    this.player1= player1;  
    this.player2 = player2;  
    this.vuePlateau = new GameWindow("jeu de dame",this);  
    //causes the JFrame window to be destroyed and cleaned up by the operating system  
    this.vuePlayer.dispose();  
    this.vuePlateau.construireFenetre();  
}
```

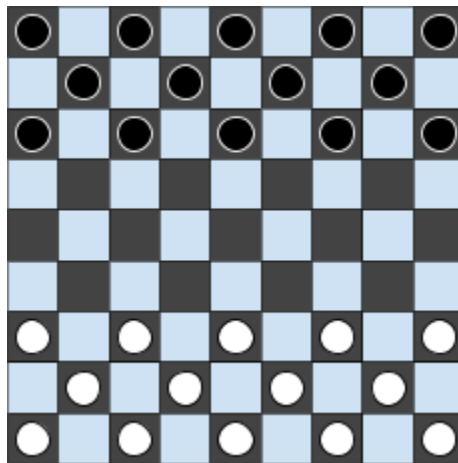


### 15. Plateau

```
public static final int TAILLE=9;  
private Case caseActive; // la case où on va se déplacer  
private boolean tourNoir; // si le tour est du joueur noir return  
true  
private Controller control;
```

#### le constructeur et la méthode init :

-Dans le constructeur de la classe plateau , on a coloré les cases du plateau pour objectif d'avoir une case bleue suivie d'une case blanche avec une méthode d'initialisation init() qui permet de créer les pions dans les cases bleues comme dans la figure ci-dessous:



#### ajouterCase:

Dans cette méthode , on précise la couleur de la case si elle est bleue ou blanche puis on applique un événement à cet objet case1.

#### Creerpion:

Cette méthode permet de nous préciser la couleur et le status du pion si il est en montée ou non puis ajouter un événement à ce dernier.

#### getCase() :

Cette méthode permet d'avoir l'index d'une case quelconque du plateau en fonction de la colonne, la ligne et la taille du plateau (9):

```
(Case) getComponent(j+i*TAILLE) .
```

Il y a une deuxième méthode dans le code getCase() qui renvoie directement l'index de la case du plateau en fonction des lignes.

```
(Case) getComponent(i)
```

#### afficherPossibilites():

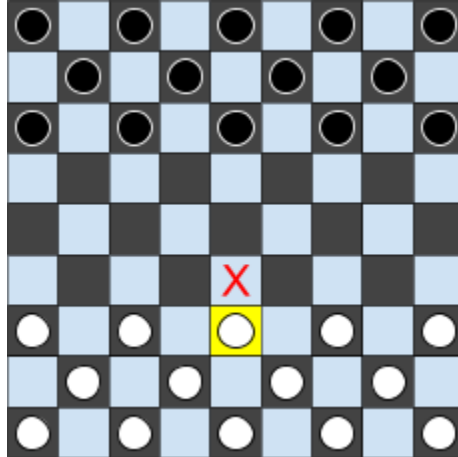
Cette méthode nous permet de calculer les coordonnées(i,j) de la case qui est juste en haut de la case actuel(case active) où on peut se déplacer du bas vers le haut quand il s'agit du tour des blancs .

```
caseActive=getCase(k) ;
```

```
i=k/TAILLE;
```

```
j=k%TAILLE;
```

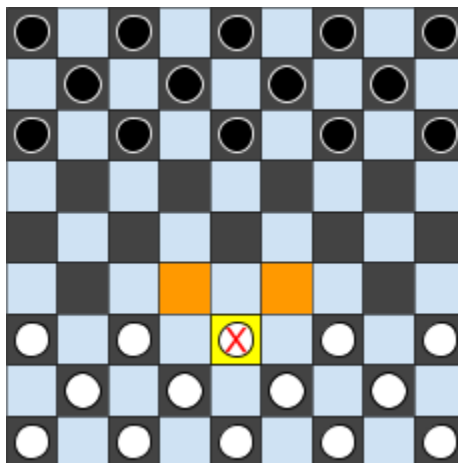
Dans notre exemple la case active est celle colorée en jaune donc les coordonnées qu'on va obtenir seront de la case cochée en rouge.



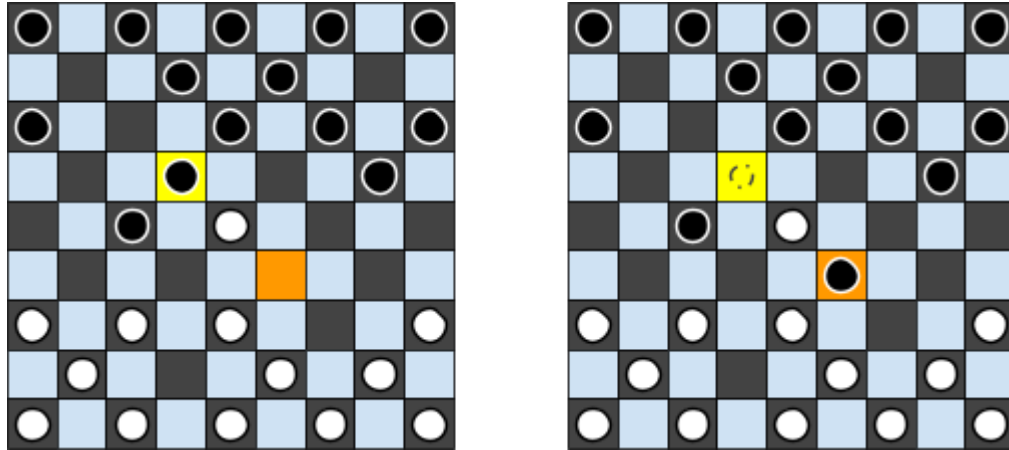
Ces coordonnées on va les implémenter par la suite dans la fonction `selectionnerCases(i,j,couleur du pion)` afin de calculer les positions des cases où on peut se déplacer.

selectionnerCases:

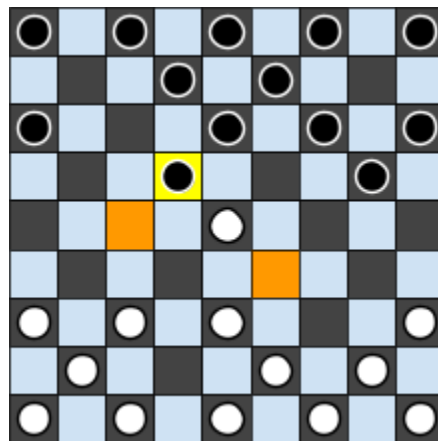
Dans cette méthode on a calculé les coordonnées des cases où on peut se déplacer diagonalement soit vers la gauche soit vers la droite en sautant une case si la case suivante est occupé par un pion de l'adversaire où en passant directement à la case suivante si elle est vide que se soit dans le tour des blancs (en se déplaçant du bas vers le haut) ou dans le tour des noirs (en se déplaçant du haut vers le bas).



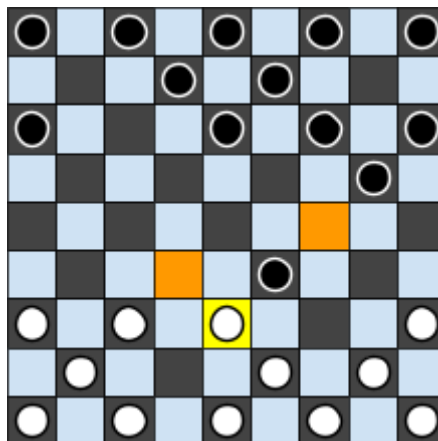
Le tour du blanc deux cases possibles se déplacer d'une case soit vers la gauche soit vers la droite.



se déplacer de deux cases diagonalement vers la droite dans le tour du noir



Le joueur noir a le choix soit de se déplacer à gauche de deux cases soit de se déplacer à droite d'une seule case.



Le joueur blanc a le droit de sauter 2 cases diagonalement à droite pour aller à la case orange sélectionnée.

GetLigne :

Cette méthode permet de récupérer le numéro de la ligne où se trouve la case active (où on se positionne actuellement ) compris entre 0 et 8 (taille 9).

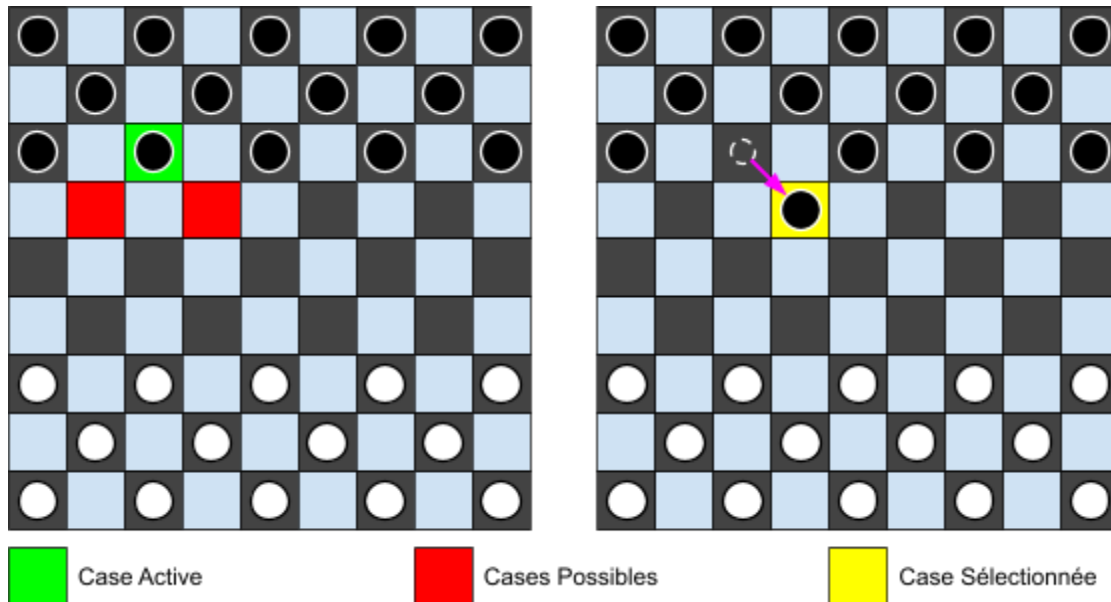
### GetColonne:

Cette méthode permet de récupérer le numéro de la colonne où se trouve la case active (où on se positionne actuellement ) compris entre 0 et 8(taille 9).

Deplacer(Case) : void, cette méthode permet le déplacement d'un pion d'une case active vers une autre sélectionnée en se basant sur plusieurs critères.

Avant de vous montrer les critères de déplacement, il faut prendre en compte qu'un seul pion est stocké dans une case x, on peut récupérer le pion de la case active à travers

**caseActive.getComponent(0)**, puis on peut insérer ce pion{component} dans une autre case sélectionnée(case doit être vide) **case1.add(caseActive.getComponent(0))**;



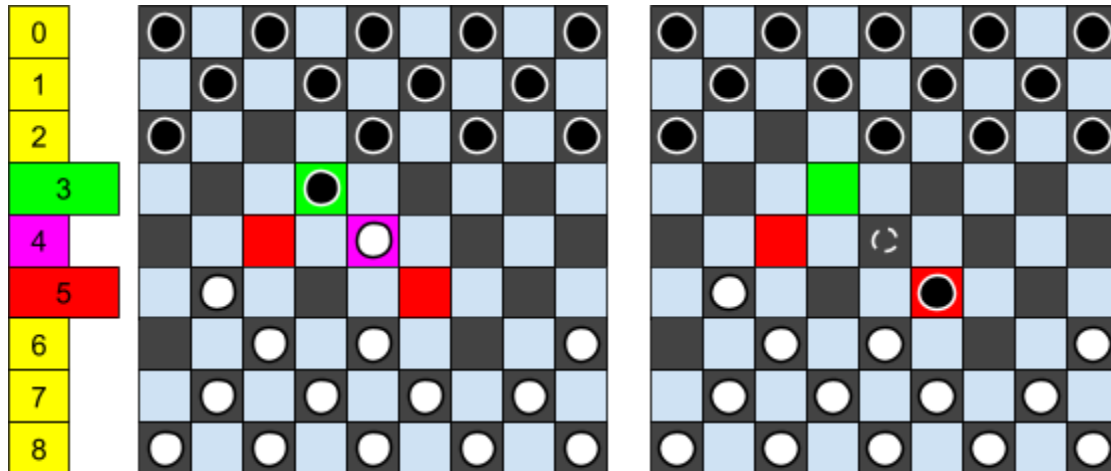
Case active est la case où on peut se déplacer diagonalement en sautant une case soit vers la gauche ou vers la droite.

Cases Possibles seront indiqués par la fonction **afficherPossibilites(Pion p)**

Case Sélectionnée est la case cible, la nouvelle position pour le pion sélectionné.

Pour s'assurer qu'on a bien mangé un pion adverse, il faut comparer la ligne de la case active et la ligne cible/sélectionnée si la différence entre les deux est égale à 2 donc on a bien mangé un pion adverse qui est identifiée par deux indices i et j.

```
if(Math.abs(getLigne(case1)-getLigne(caseActive))==2) //get the index (i,j) of pion => kill
int i = (getLigne(case1)+getLigne(caseActive))/2; //Calculate the index i
int j = (getColonne(case1)+getColonne(caseActive))/2; //Calculate the index j
```



Ligne de la Case Active est 3 et la ligne de la case sélectionnée est 5  
 $\Rightarrow 5-3 = 2 \implies$  recuperer la position de ○ KilledPion(i=4,j=4)

```
//selecting the pion that will be killed by the opponent ==> remove from the Plateau
Pion p = ((Pion)getCCase(i,j).getComponent(0));
//the current pion has the same color as the first player -==> NOIR
//decrement Pion Number and increment Kills number
//if the current pion is a dame decrement the dame's number
if(p.getCouleur() == Couleur.NOIR){//Reverse Players if p.getCouleur() == Couleur.BLANC.
    control.getPlayer1().decNbrPion();
    control.getPlayer2().incNbrKills();
    if(p.isDame()) {
        control.getPlayer1().decNbrDame();
    }
    //refresh the panel information ==> update player1 information
    control.getVuePlateau().refresh();
}
```

Après la modification de différents paramètres des deux joueurs, on actualise IHM puis on supprime le pion adverse.

```
// effacer et valider l'événement de la case absorbé (i,j)
getCCase(i, j).removeAll();
getCCase(i, j).validate();
getCCase(i, j).repaint();
```

Lors de chaque déplacement on inverse le tour de chaque joueur (  $\text{tourNoir} = !\text{tourNoir}$ ;  $\Rightarrow$  par default le  $\text{tourNoir} = \text{false}$  ). Afin de préciser le tour de quel joueur, une LED sur IHM sera allumée en vert pour le joueur actif.

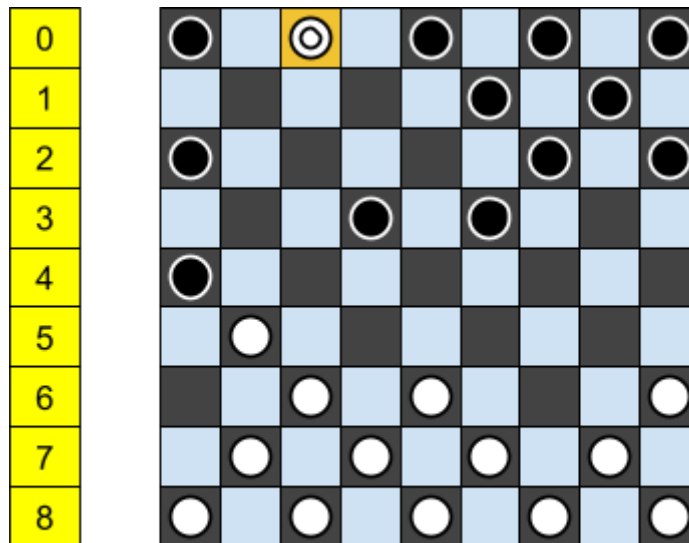
```
if(tourNoir) control.getVuePlateau().activatePlayer1Role();//LED joueur1 ⇒ vert
else        control.getVuePlateau().activatePlayer2Role();//LED joueur2 ⇒ vert
```


Enfin il faut supprimer le pion de la case active puisqu'il était déplacé vers une autre case.

```
caseActive.removeAll();
caseActive.repaint();
caseActive=null;
```

Cas particulier :

- Si le pion devient Dame, on incrémente le nombre de dames du joueur actif.



Le pion  (0,2) est Dame

```
// si la case 1 est dans la première ligne du plateau donc
//on peut pas faire monter le pion car on va sortir du plateau
if(getLigne(case1) == 0){//dans le cas inverse on prend getLigne(case1)==TAILLE-1
    Pion p=(Pion)(case1.getComponent(0));
    if(p.getCouleur() == Couleur.BLANC){
        p.setDame();// p becomes dame
        control.getPlayer2().incNbrDame();
        control.getVuePlateau().refresh();
    }
    p.setMonte(false);
}
```

Le traitement de cas de gagne automatique :

- A chaque déplacement, on vérifie le nombre de pions restant de chaque joueur afin de déterminer le gagnant (incrémenter le score), puis afficher un message pour indiquer le gagnant et rejouer/quitter au choix de l'utilisateur.

```
//==> winning case based on the PION number = 0
if(control.getPlayer1().getNbrPion()==0) {
    control.getPlayer2().incScore();
    int btn_Yes = JOptionPane.showConfirmDialog(this,
        "Le gagnant est : " + control.getPlayer2().getName()+"\nVoulez-vous
recommencer une nouvelle partie ?",
        "Congratulation",
        JOptionPane.YES_NO_OPTION);
    if(btn_Yes == JOptionPane.YES_OPTION) {
        System.out.println("Yeas btn_Yes : " + btn_Yes);
        control.getVuePlateau().resetGame();
    }else { //exit
        control.getVuePlateau().exitWindow(); // close window
    }
}
}
```

**Les perspectives du projet:**

- On peut améliorer le déplacement de la dame de telle façon qu'elle peut sauter plusieurs cases diagonalement et pas que deux cases au plus .
- On peut améliorer l'interface du jeu en utilisant des librairies java pour des interfaces dynamiques.
- On peut ajouter l'option IA dans le jeu si on veut jouer avec un ordinateur en se basant sur un principe extrêmement simple : l'aléatoire. L'algorithme de jeu du joueur virtuel correspond presque exactement à celui du joueur réel, à l'exception que celui-ci appelle des fonctions de choix de cases à la place de demander au joueur ce qu'il veut jouer. Ces fonctions de choix peuvent être divisées en trois niveaux, facile, normal et difficile, selon le niveau de difficulté sélectionné en début de partie. Chacune de ces trois fonctions commence par sélectionner au hasard une des coordonnées qui lui sont transmises en paramètres dans un vecteur. A la suite de ce choix, la fonction facile et la fonction difficile détermine si ce choix entraînera l'élimination du pion courant au tour suivant. Si c'est le cas, le joueur facile prend cette décision, et joue de manière à être éliminé, tandis que le joueur difficile lui teste les autres possibilités, s'il en existe une moins risquée, il opte pour celle ci.

### **Difficultés du projet :**

- La programmation sous java en utilisant des IHM avec le modèle MVC nous était challengeante, nous programmons notre jeu de dames en même temps que nous prenions nos premiers cours de java.
- La gestion des cases ainsi que les pions lors de chaque déplacement était abstraite au début; cependant le travail en binôme nous a permis de bien comprendre le concept de damier ⇒ communication plus fluide.

### **Conclusion :**

De notre côté, ce projet a développé nos compétences dans le domaine de la programmation et du travail en groupe. Nous avons développé un esprit de travail en équipe et d'adaptation, qui nous seront utiles lors des projets futurs dans notre vie professionnelle. Nous avons de plus grâce à ce projet approfondi nos compétences en programmation avec le langage Java (Swing) en appréhendant les interfaces graphiques. Nous avons enfin pu approfondir des notions vues en cours qui étaient mal assimilées comme le modèle MVC en Java.