DOCUMENTATION TECHNIQUE

Dylan Hochet

Fonctionnement des fonctions du programme Reversi

2019

1. windows.onload = function()

Cette fonction s'exécute au chargement de la page HTML. Dans un premier temps, elle va associer la variable canvas à la balise HTML canvas grâce à l'id « myCanvas ».

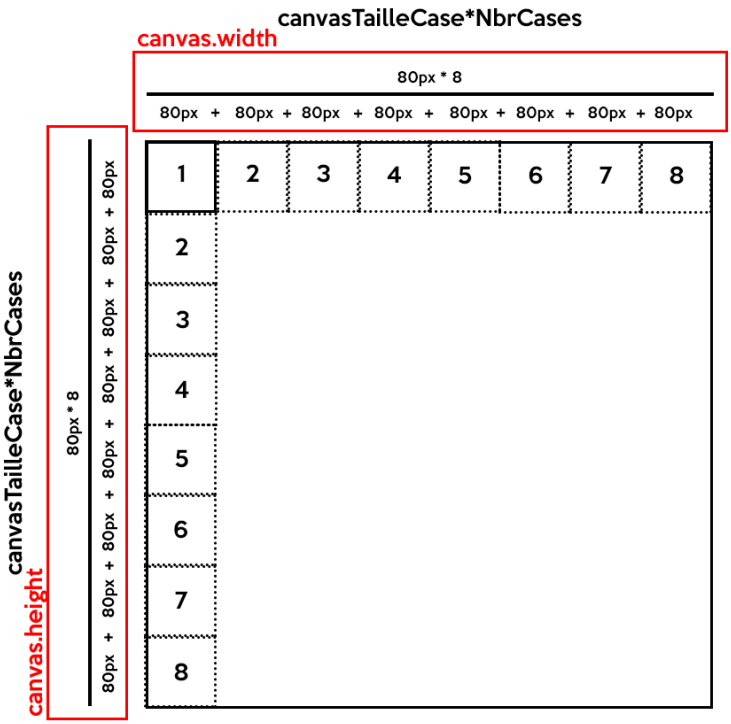
```
canvas = document.getElementById('myCanvas');
```

Une fois l'association faite, la fonction adapte la taille du canvas selon la taille et le nombre de cases

La taille des cases est définie par la variable globale canvasTailleCase

Le nombre de cases est définie par la variable globale NbrCases

```
canvas.width = canvasTailleCase*NbrCases;
canvas.height = canvasTailleCase*NbrCases;
```



La fonction effectue ensuite 2 vérifications :

Si le canvas est bien définie

```
if(!canvas)
```

Si le contexte est bien défini

```
context = canvas.getContext('2d');
if(!context)
```

Dans le cas contraire, la fonction affiche une erreur dans une alerte navigateur

```
alert("Impossible de récupérer le canvas");
return;
```

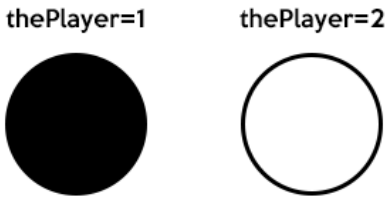
```
alert("Impossible de récupérer le context du canvas");
return;
```

La fonction appelle finalement notre deuxième fonction : initGame()

2. initGame()

initGame() configure les différentes données nécessaires au démarrage de la partie. Dans un premier temps, elle indique que le joueur 1 commence (soit le joueur aux pions noirs).

```
thePlayer = 1 ;
```



Elle crée ensuite un tableau à 2 dimensions. Une première boucle parcourt les « i » de 0 jusqu'à notre NbrCases et créer un tableau à chaque passage. Une deuxième boucle imbriquée dans la première parcourt les « j » et rempli la coordonnée [i][j] d'un 0.

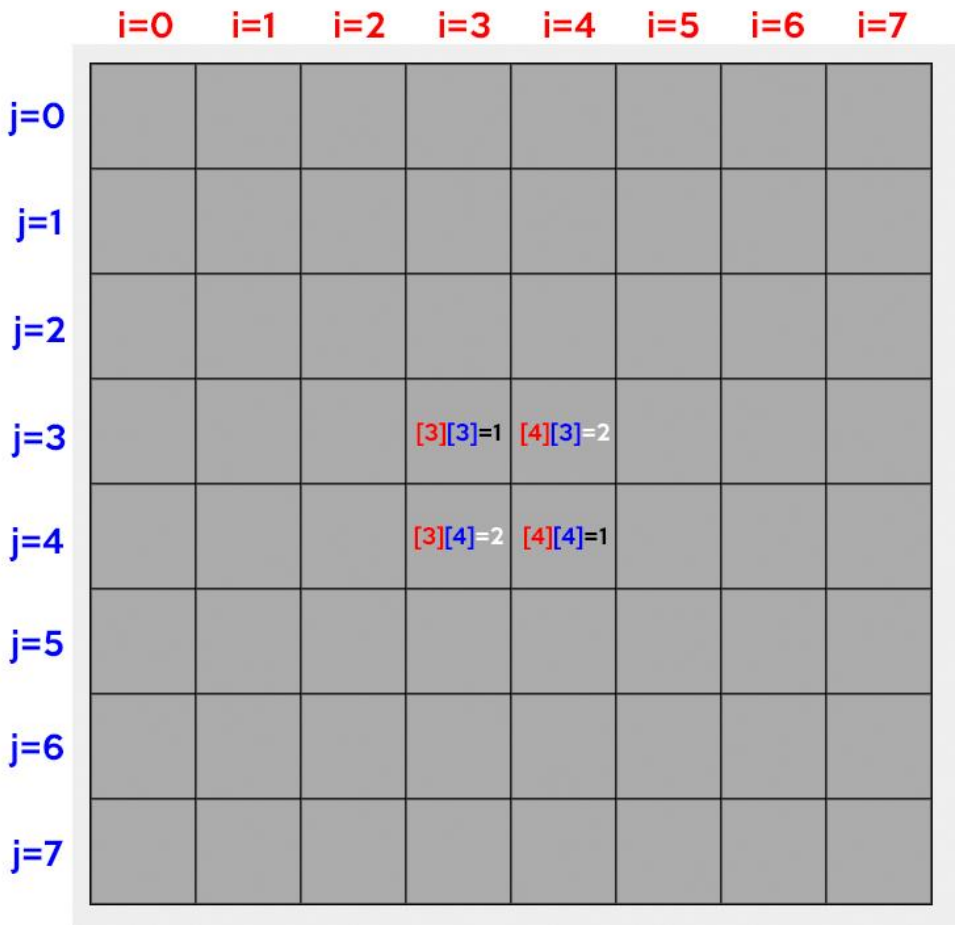
```
theGame = new Array() ;
for (var i=0 ; i < NbrCases ; i++ ) {
    theGame[i] = new Array();
    for (var j=0 ; j < NbrCases ; j++ ) {
        theGame[i][j]=0;
    }
}
```

	i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7
j=0	1	9	etc.	etc.	etc.	etc.	etc.	57
j=1	2	10	etc.	etc.	etc.	etc.	etc.	58
j=2	3	11	etc.	etc.	etc.	etc.	etc.	59
j=3	4	12	etc.	etc.	etc.	etc.	etc.	60
j=4	5	13	etc.	etc.	etc.	etc.	etc.	61
j=5	6	14	etc.	etc.	etc.	etc.	etc.	62
j=6	7	15	etc.	etc.	etc.	etc.	etc.	63
j=7	8	16	etc.	etc.	etc.	etc.	etc.	64

Les valeurs des cases du schéma ci-dessus représente l'ordre du passage lors de la création des tableaux.

La fonction initialise ensuite les pions de départ. Comme vu précédemment, les 1 représente les pions noirs et les 2 les pions blancs.

```
theGame[3][3]=1;
theGame[3][4]=2;
theGame[4][3]=2;
theGame[4][4]=1;
```



Une fois toutes ces données configurées. Il faut afficher graphiquement le résultat pour l'utilisateur. La fonction va appeler tracerDamier().

```
canvas.addEventListener('mousedown',mousePos,false);
```

Finalement, la fonction ajoute un « eventListener » qui va attendre un clic de l'utilisateur (mousedown) et, sur un clic, va lancé la fonction mousePos().

3. mousePos(event)

Le paramètre (event) fait référence à « eventListener » de la fonction initGame() qui va attendre un clic de l'utilisateur pour lancé la fonction mousePos().

La fonction définie 2 variables locales x et y qui sont les coordonnées du clic sur la page.

```
var x = event.x;    x 296
var y = event.y;    y 88
```

Canvas.offsetLeft et Canvas.offsetTop vont déduire des coordonnées x et y les marges entre les bordures de la fenêtre et le canvas. Ainsi, le coin en haut a gauche de notre canvas correspondra aux coordonnées x=0 et y=0.

```
x -= canvas.offsetLeft;
y -= canvas.offsetTop;
```

Enfin, la fonction divise les coordonnées obtenues par le nombre de cases. Le résultat obtenu étant à virgule, la fonction appelle la méthode Math.floor qui va transformer la coordonnée à virgule en coordonnée entière (arrondi a l'inférieur).

```
coordX = x / canvasTailleCase;
i = Math.floor(coordX);
coordY =  y / canvasTailleCase;
j = Math.floor(coordY);
```

La fonction attribut la nouvelle coordonné x à l'indice j et la nouvelle coordonnée y à l'indice j. Une fois les coordonnée attribuées à une case, la fonction appelle gestionJeu(i,j) avec les paramètres i,j qui sont les coordonnées de notre tableau à 2 dimensions theGame[i][j].

4. gestionJeu(i,j)

Cette fonction va gérer l'état du jeu clic par clic. Les paramètres récupérés (i,j) sont les coordonnées de theGame[i][j]. La fonction va dans un premier temps vérifier si le joueur a cliqué sur une case vide. Dans le cas contraire (clic sur une case avec pion), rien ne se passe.

```
if (theGame[i][j]==0)
```

Elle va ensuite crée une variable adverse (qui représente l'autre joueur) selon le joueur actuel. Si thePlayer=1, adverse sera égal à 2 et si thePlayer=2, adverse sera égal à 1.

```
switch ( thePlayer ) {
    case 1 : adverse = 2 ; break ;
    case 2 : adverse = 1  ; break ;
}
```

Viens ensuite la partie la plus importante de la fonction. Si la fonction verifCoup(i,j,adverse) renvoie vrai (c'est-à-dire si un coup est possible sur la case cliquer), alors on accède à la suite de la fonction. Dans le cas contraire (aucun coup n'est possible) rien ne se passe. En effet, une case peut être vide mais n'avoir aucun coup possible.

```
if (verifCoup(i,j,adverse)==true) {
```

Si un coup est possible, alors on va « switch » entre les joueurs. Dans un premier temps, on va effectuer les changements possibles dans les données du tableau theGame[i][j]. Puis la fonction s'occupe de modifications graphiques mineurs concernant le texte et la couleur de l'affichage du joueur. Enfin, la variable thePlayer devient le joueur adverse. C'est à l'adversaire de jouer.

```
if (thePlayer==1) {
    effectuerChangements(i,j,adverse);
    indicationDuTour.innerHTML = "TOUR DU JOUEUR : BLANC";
    indicationDuTour.classList.add('blanc');
    indicationDuTour.classList.remove('noir');
    thePlayer=adverse;
}
else if (thePlayer==2) {
    effectuerChangements(i,j,adverse);
    indicationDuTour.innerHTML = "TOUR DU JOUEUR : NOIR";
    indicationDuTour.classList.remove('blanc');
    indicationDuTour.classList.add('noir');
    thePlayer=adverse;
}
```

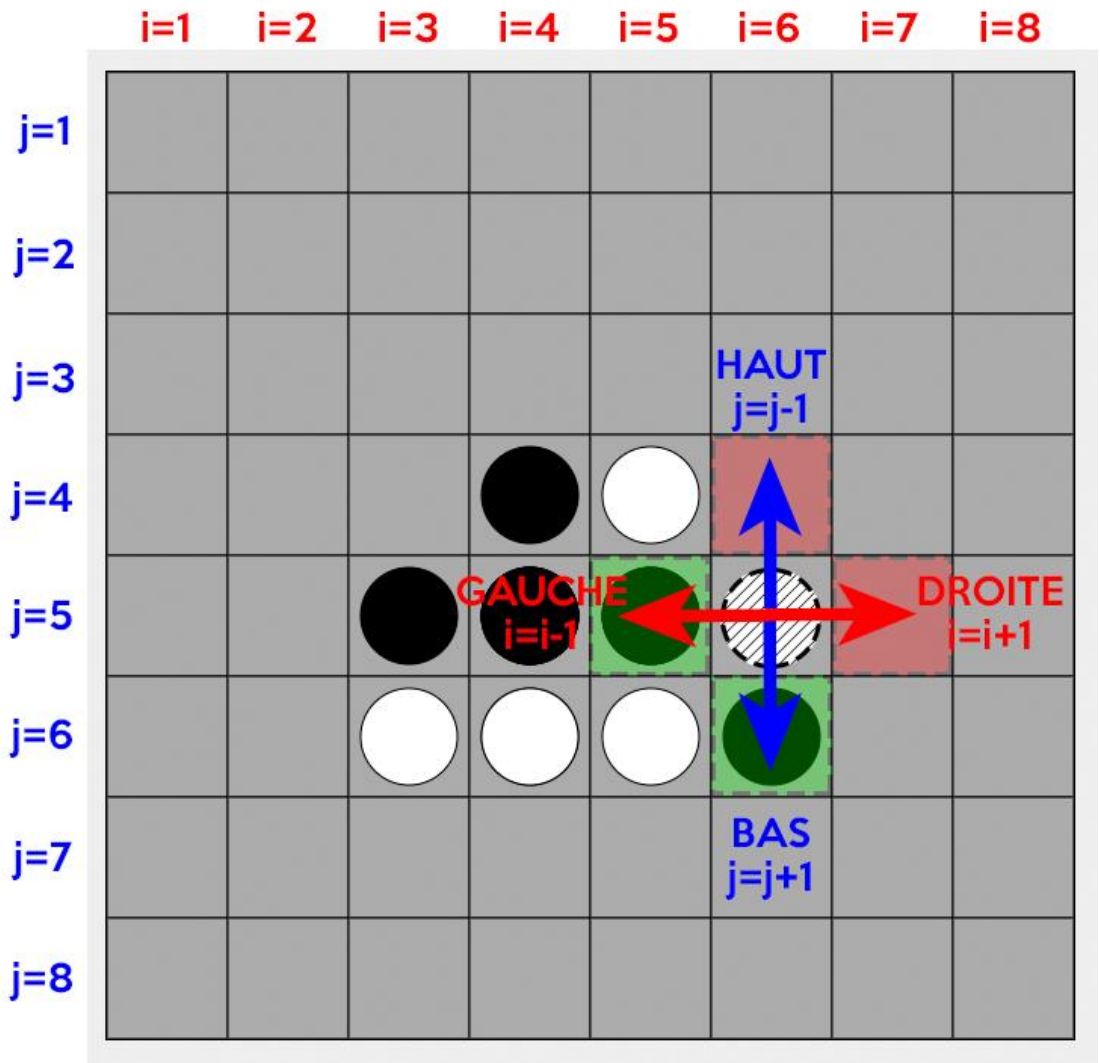
Pour finir, la fonction appelle compterPoints() une fois le tour terminé.

5. verifCoup(caseI,caseJ,adverse)

Cette fonction a pour but de vérifier si un coup est possible selon les paramètres caseI et caseJ reçu en paramètres. Elle va d'abord initialisée la variable booléenne « coupPossible » sur faux. La fonction va parcourir ensuite chacune des 8 directions possibles dans l'ordre suivant : Droite, Gauche, Bas, Haut, Bas-Droite, Bas-Gauche, Haut-Gauche, Haut-Droite.

Quel que soit les directions Haut, Bas, Gauche, Droite, le processus est assez similaire. Une première vérification est faite : Le pion suivant (selon la direction testée) est t-il adverse ? (adverse est une variable reçue en paramètres).

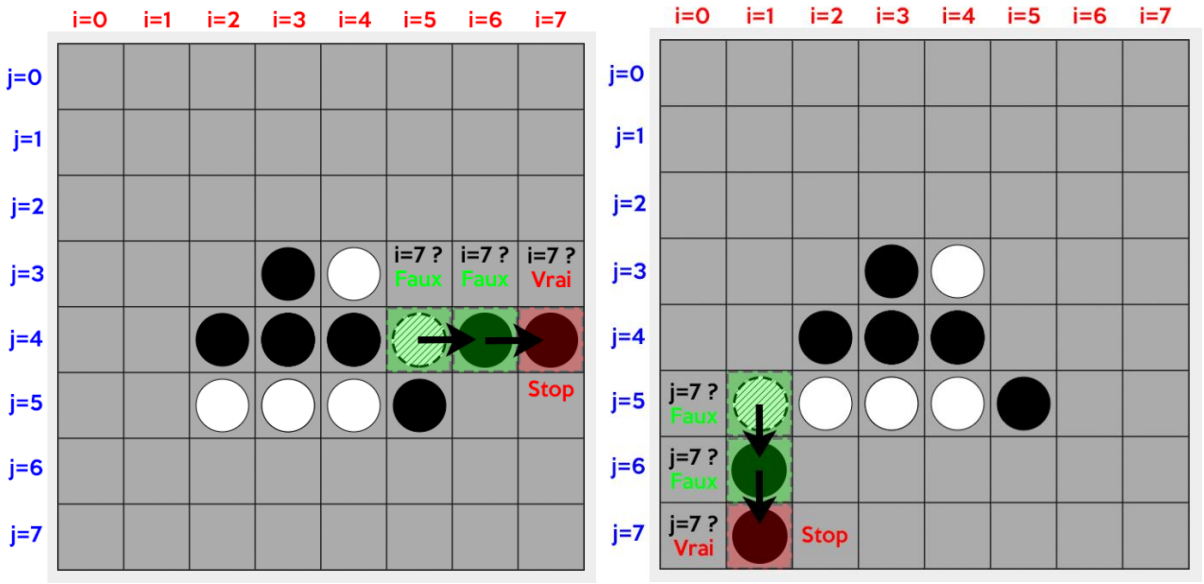
```
for( var i = caseI+1 ; i < NbrCases ; i++ ) { //DROITE
    if (theGame[i][caseJ]==adverse) {
```



Une petite vérification est lancée à chaque coordonnée pour savoir si on est sortie ou non du tableau. Par exemple, le schéma ci-dessous nous montre le fonctionnement sur le côté droit.

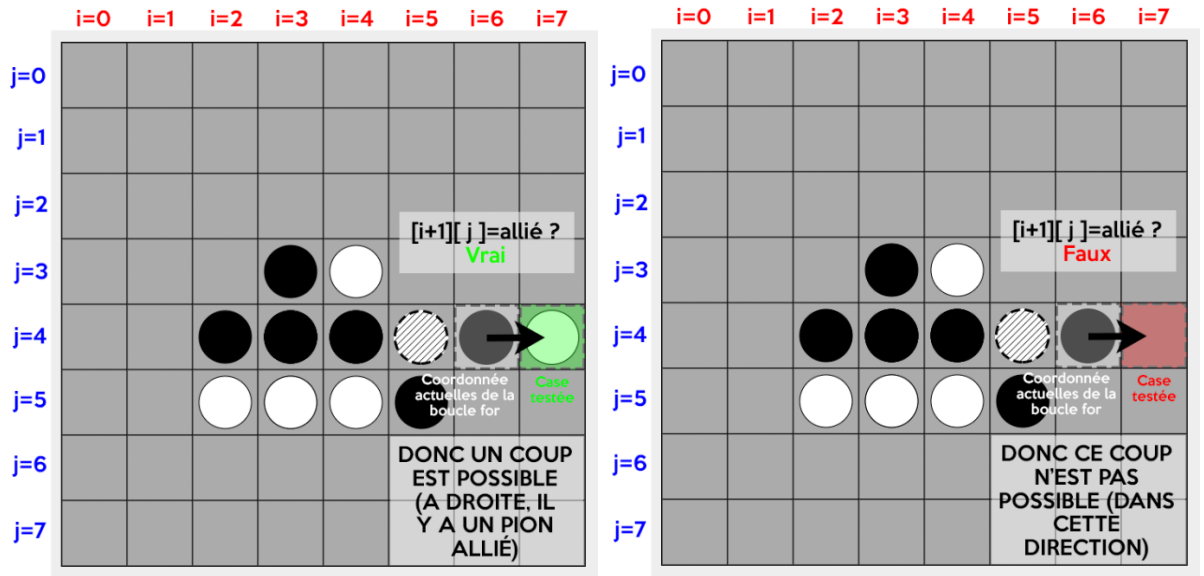
La condition fonctionne de la même manière quel que soit la direction.

```
if (i==7) {
    break;
}
```

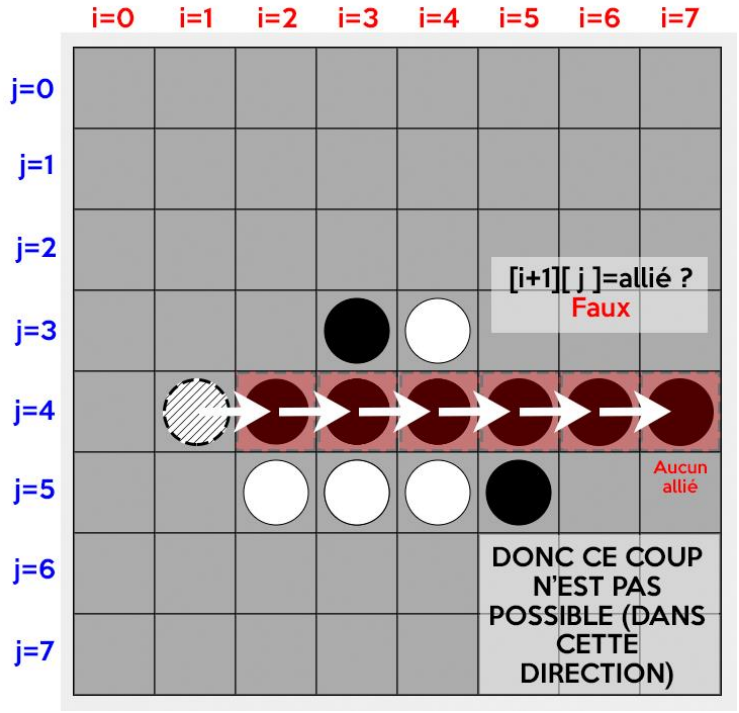


La prochaine condition porte sur le joueur allié. On sait que les coordonnées ou la boucle se trouve comporte un pion ennemi. La fonction vérifie alors si la prochaine case est un joueur allié. Si oui, alors un coup est possible. La variable booléenne devient alors vraie et la boucle se coupe.

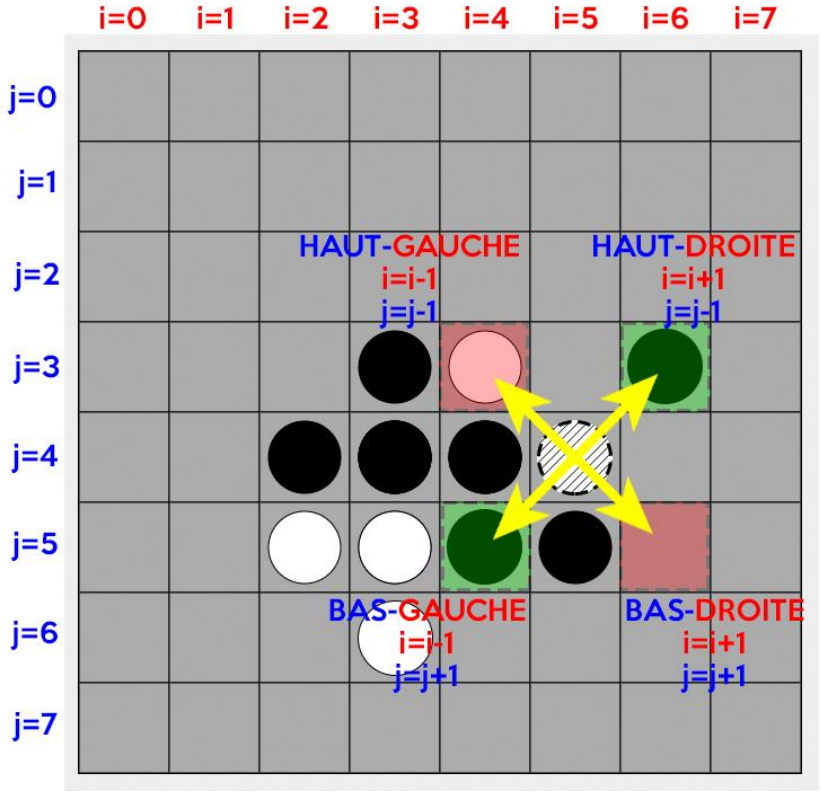
```
if (theGame[i+1][caseJ]==thePlayer) {
    coupPossible=true;
    break;
}
```



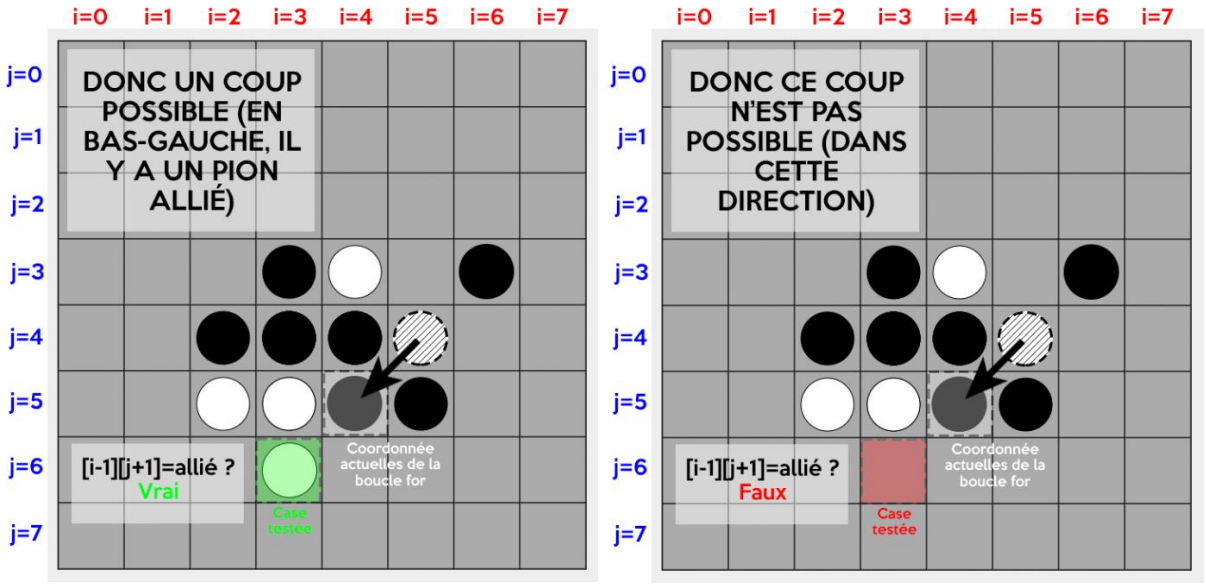
Le cas particulier suivant est possible. Si un ennemi est trouvé tout le long de la direction mais aucun allié n'est présent. La boucle effectue alors un break quand i=7 comme vu précédemment dans la fonction.



La fonction continue d'appliquer le même principe pour les diagonales, quel que soit la direction. Une variable « jBoucle » est incrémenter dans la boucle for pour pouvoir se diriger verticalement en même temps que la boucle horizontale.



Encore une fois, la fonction effectue les mêmes vérifications pour les diagonales.



Une fois les 8 directions testée. La variable booléenne coupPossible est finalement testée. Si elle renvoie vraie, un coup est possible et inversement si elle renvoie faux.

```
if (coupPossible==true) {
    return true;
}
else {
    return false;
}
```

Selon le résultat renvoyer, la fonction gestionJeu() va poursuivre ou non la suite de son code.

6. effectuerChangements(caseI,caseJ,adverse)

La fonction effectuerChangement(caseI,caseJ,adverse) va effectuer exactement le même code que verifCoup() à quelques exceptions près.

Au lieu de renvoyer vrai ou faux, la fonction une fois appelée va effectuer des changements sur les données du tableau theGame[i][j] ainsi que le changement graphique des pions pour les utilisateurs.

Dans un premier temps, la fonction associe une couleur à la variable coulPion selon le joueur actuel.

```
switch ( thePlayer ) {
    case 1 : coulPion = CouleurPionNoir ; break ;
    case 2 : coulPion = CouleurPionBlanc  ; break ;
}
```

Puisque lorsque cette fonction est appelé, un coup est forcément possible, effectuerChangement() va directement attribué un pion au joueur sur la case cliqué. Les coordonnées en question du tableau theGame[i][j] sont alors modifié selon le joueur et un pion est tracé à la couleur du joueur grâce à la fonction tracerRond() qui est appelé.

```
theGame[caseI][caseJ]=thePlayer;
tracerRond(caseI,caseJ,coulPion);
```

Comme expliquer précédemment, la fonction effectue le même code que verifCoup(). Elle va alors vérifier quelles directions sont possibles une par une. Cependant, au lieu de modifié une variable booléenne, elle va directement interagir avec le tableau theGame[i][j] pour chaque pion ennemi trouvé dans une direction possible. Ainsi, chaque pion ennemie devient allié et la fonction tracerRond() est appelé pour chaque changement de pion afin d'afficher un nouveau pion allié.

```
if (theGame[i+1][caseJ]==thePlayer) {
    for( i=caseI+1 ; i < NbrCases ; i++ ) {
        if (theGame[i][caseJ]==adverse) {
            theGame[i][caseJ]=thePlayer;
            tracerRond(i,caseJ,coulPion);
        }
        else {
            break;
        }
    }
    break;
}
```

7. compterPoints()

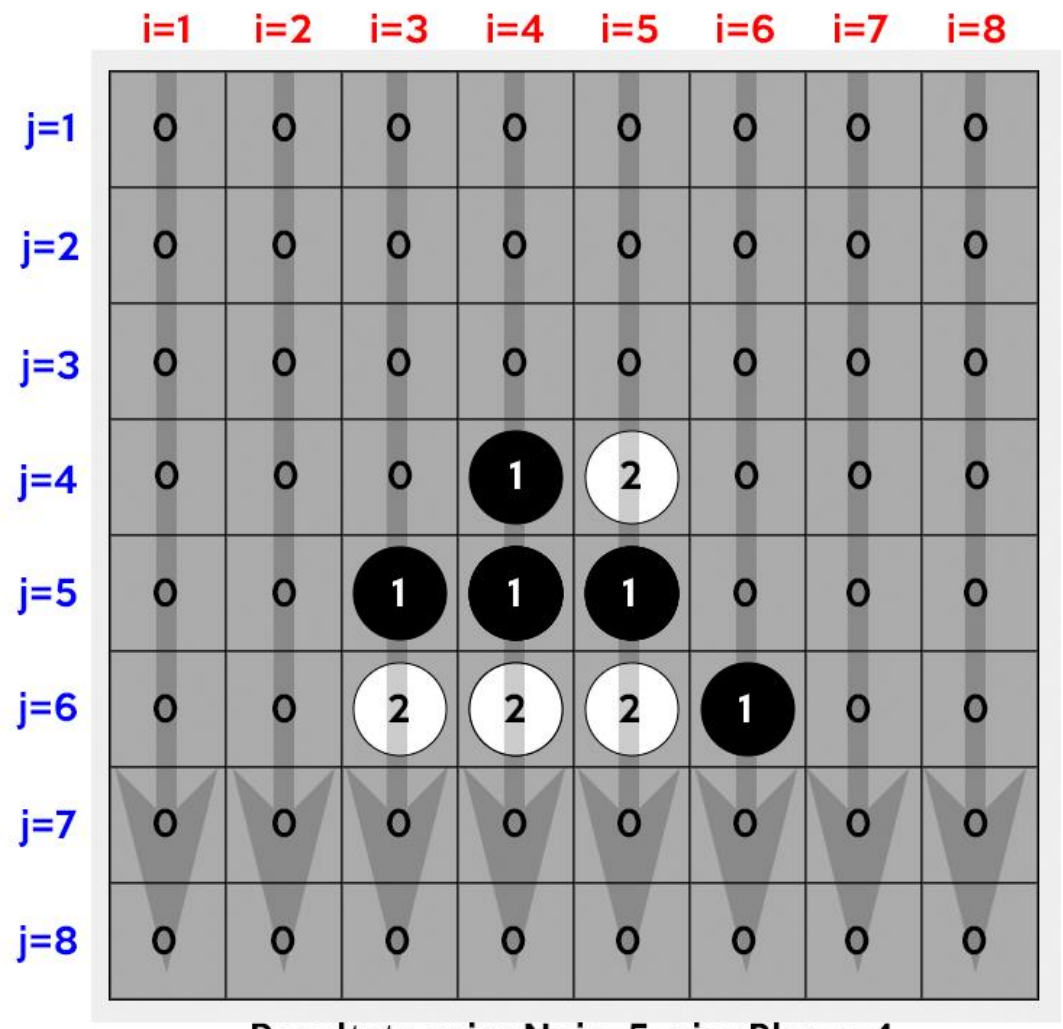
La fonction créer 2 variables locales : pionNoir et pionBlanc qui servirons de compteur. Elle va ensuite parcourir l'ensemble du tableau theGame[i][j].

Si un pion noir est trouvé (soit theGame[i][j]=1), on incrémente pionNoir de 1

Si un pion blanc est trouvé (soit theGame[i][j]=2), on incrémente pionBlanc de 1

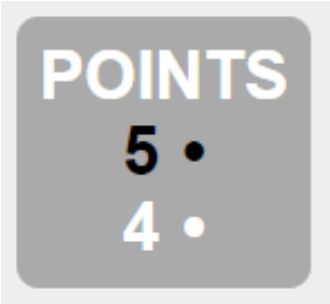
```
var pionNoir=0, pionBlanc=0;

for( var i = 0 ; i < NbrCases ; i++ ) {
    for( var j = 0 ; j < NbrCases ; j++ ) {
        if (theGame[i][j]==1) {
            pionNoir=pionNoir+1;
        }
        else if (theGame[i][j]==2) {
            pionBlanc=pionBlanc+1;
        }
    }
}
```



Dès que la boucle est terminée, les résultats sont affichés en HTML aux joueurs

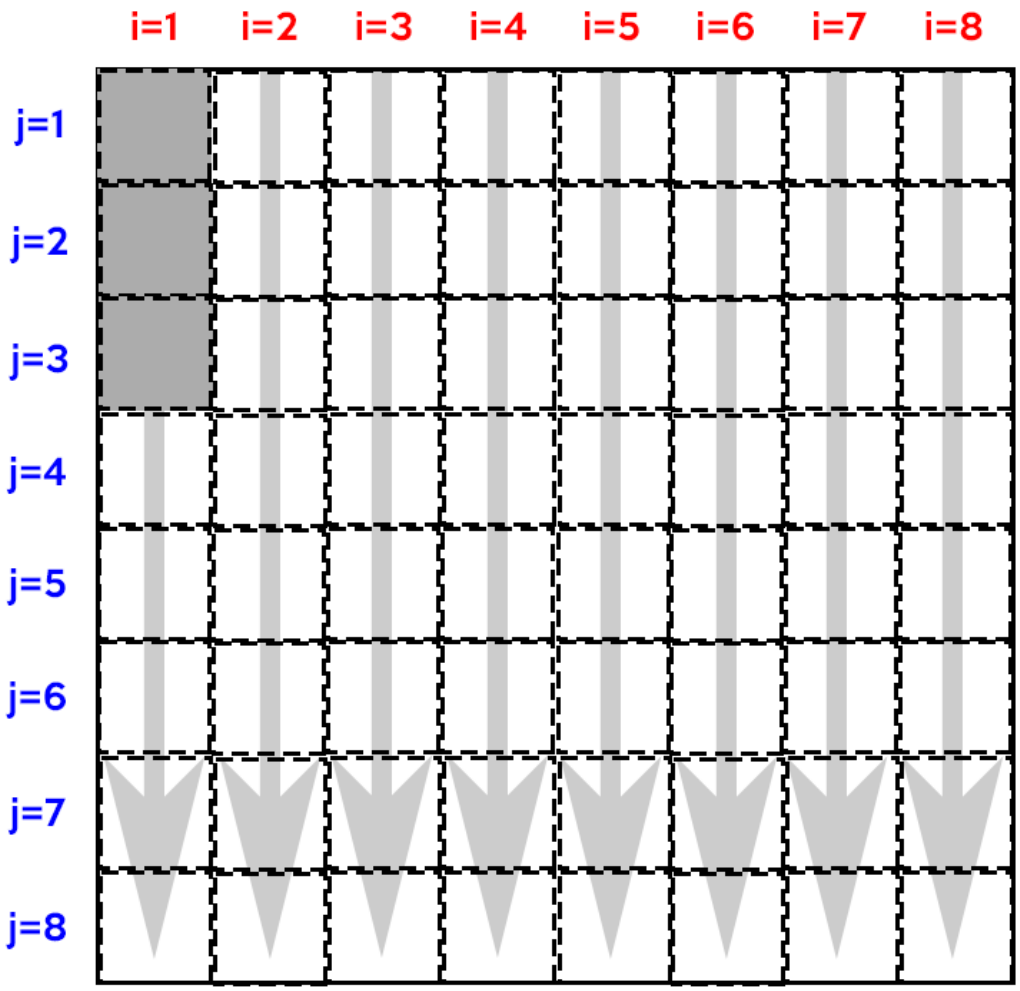
```
document.getElementById("pointsJ1").innerHTML = pionNoir;
document.getElementById("pointsJ2").innerHTML = pionBlanc;
```



8. tracerDamier()

Cette fonction est appelée uniquement qu'au chargement de la page. Elle permet de tracer l'ensemble des cases du tableau. La fonction va appeler tracerCase(i,j,couleurTapis) ou i et j sont les coordonnées de theGame[i][j] pour chaque cases du tableau. CouleurTapis est une variable globale contenant un code couleur RGB correspondant à une teinte de gris.

```
for( var i = 0 ; i < NbrCases ; i++ ) {
    for( var j = 0 ; j < NbrCases ; j++ ) {
        tracerCase(i,j,CouleurTapis);
```

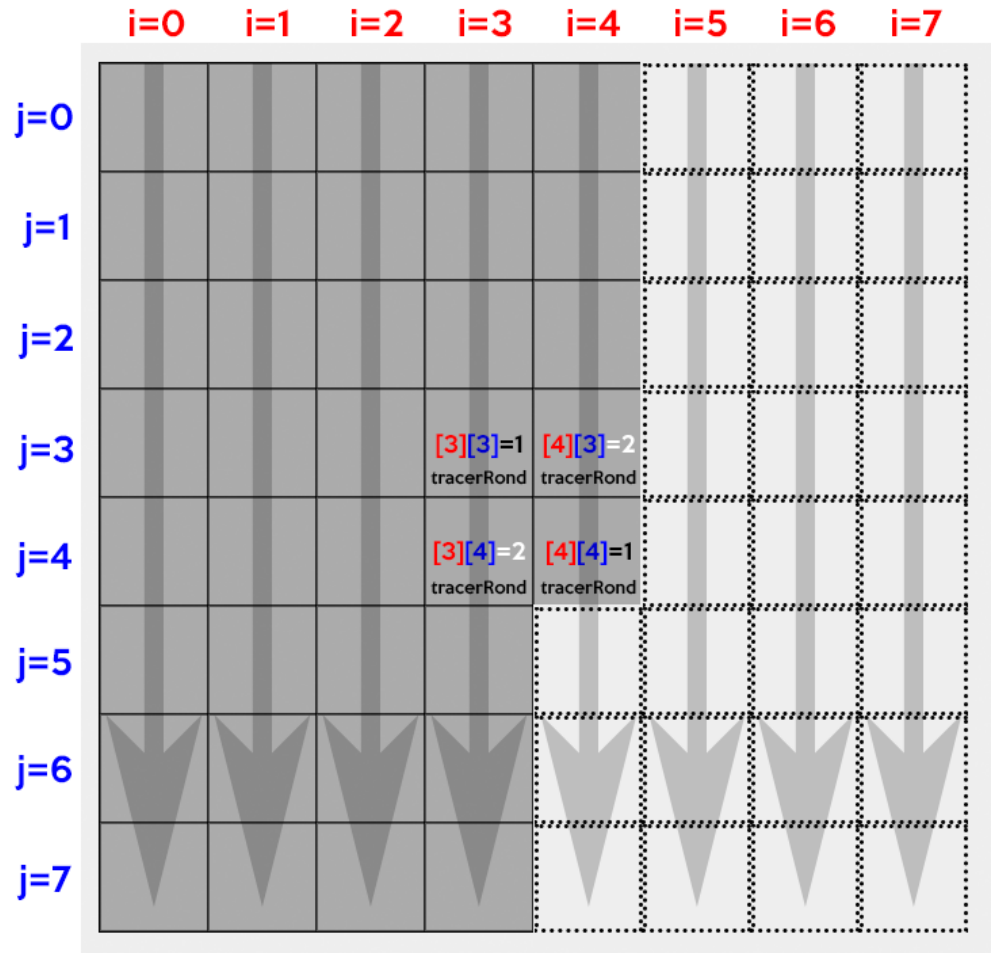


Lors de la création des cases, une verification est faite. Si la valeur de theGame[i][j] est égale à 1 ou 2, alors la variable coulPion prend la valeur de la variable globale associée.

Ensuite, la fonction vérifie si coulPion existe (donc n'est pas « null »). Si c'est le cas, alors la fonction appelle tracerRond(i,j,coulPion) ou i et j sont les coordonnées de theGame[i][j] et coulPion la couleur du pion définie précédemment.

```
switch ( theGame[i][j] ) {
    case 0 : coulPion = null ; break ;
    case 1 : coulPion = CouleurPionNoir ; break ;
    case 2 : coulPion = CouleurPionBlanc  ; break ;
}
if (coulPion) {
    tracerRond(i,j,coulPion);
}
```

Les 4 pions tracer seront ceux définit par initGame() au chargement de la page



9. tracerCase(x,y,color)

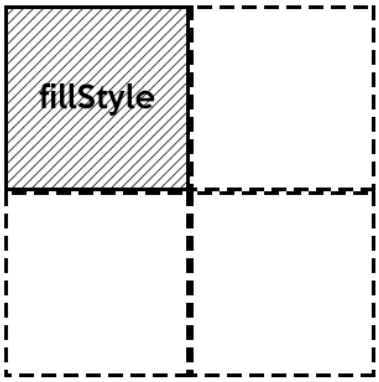
La fonction va tracer une case aux coordonnées reçus en paramètres x et y et appliquer une couleur avec color.

beginPath() et closePath() correspondent au début et à la fin du dessin de la case.

```
context.beginPath();
```
```
context.closePath();
```

Entre ces deux instructions, la fonction va appliquer un « fillStyle » avec la couleur reçu en paramètre (le gris). Cette couleur va etre attribué au tracé. Fill() va executer le remplissage.

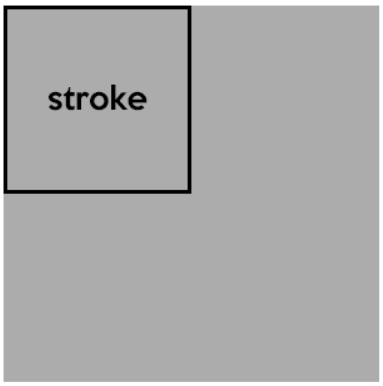
```
context.fillStyle = color;
```




```
context.fill();
```

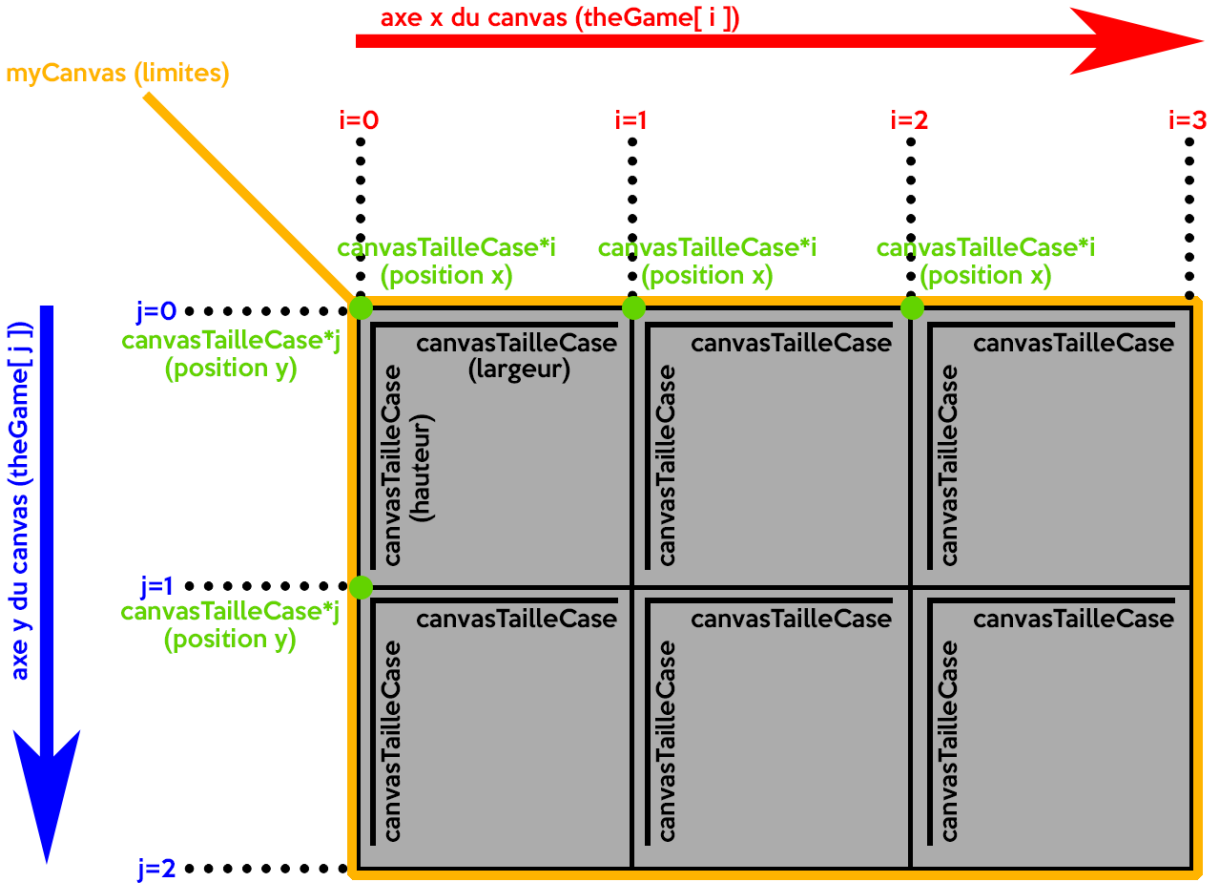
Stroke va ajouter une bordure à notre tracé.

```
context.stroke();
```



Enfin, le placement de la case est géré par rect(x,y,h,w). Cette propriété attend une position x et y ainsi qu'une auteur h et une largeur w.

```
context.rect(x*canvasTailleCase, y*canvasTailleCase, canvasTailleCase, canvasTailleCase);
```



Sur le schéma :

X et Y sont représentés par les informations vertes

H et W sont représentés par les informations noires dans les cases

Grace aux coordonnées i et j de theGame, la fonction peut tracée un cadrillage de cases.

10. tracerRond(x,y,color)

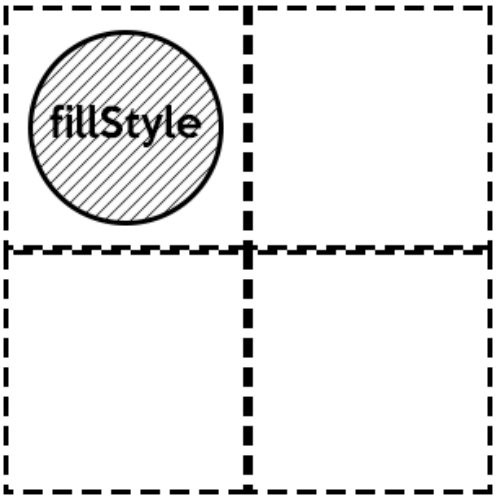
La fonction va tracer un pion aux coordonnées reçus en paramètres x et y et appliquer une couleur avec color.

beginPath() et closePath() correspondent au début et à la fin du dessin du pion.

```
context.beginPath();
context.closePath();
```

Entre ces deux instructions, la fonction va appliquer un « fillStyle » avec la couleur reçu en paramètre (soit noir ou blanc). Cette couleur va etre attribué au tracé. Fill() va executer le remplissage.

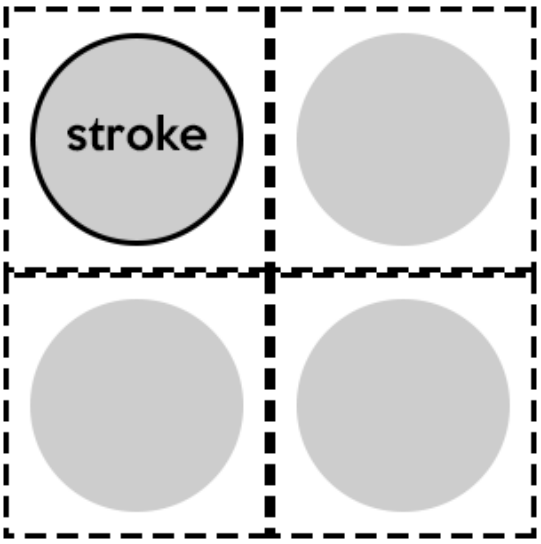
```
context.fillStyle = color;
```




```
context.fill();
```

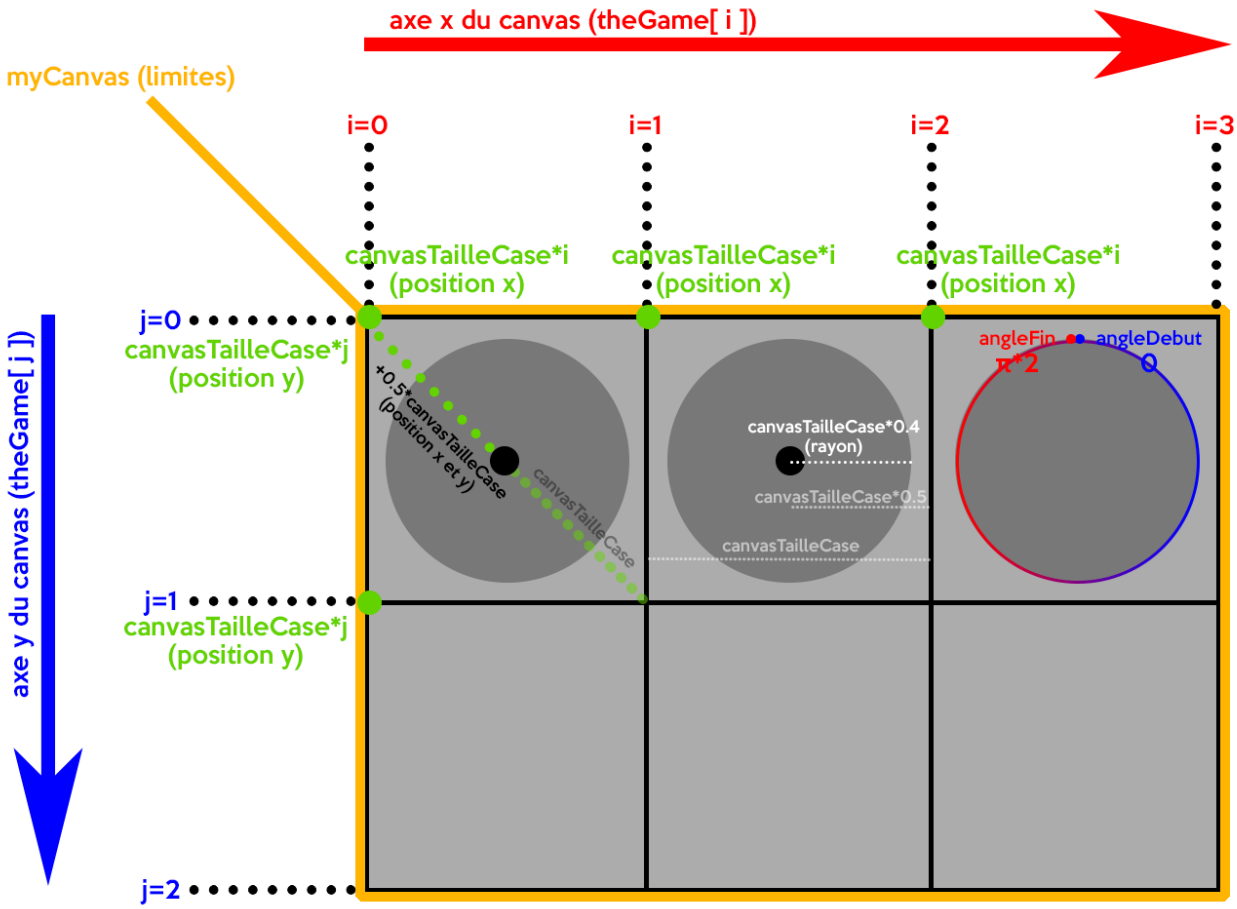
Stroke va ajouter une bordure à notre tracé.

```
context.stroke();
```



Enfin, le placement du pion est géré par arc(x,y,r,angleD,angleF). Cette propriété attend une position x et y, un rayon ainsi qu'un angle de départ angleD et un angle de fin angleF.

```
context.arc(x*canvasTailleCase+canvasTailleCase*0.5,y*canvasTailleCase+canvasTailleCase*0.5,
            canvasTailleCase*0.4,0, Math.PI*2);
```



Sur le schéma :

X et Y sont représentés par les informations vertes sur le premier pion

R est représenté par les informations blanches sur le deuxième pion

angleD et angleF sont représentés par les informations rouge et bleu sur le troisième pion

Grace aux coordonnées i et j de theGame, la fonction peut tracée un pion au centre de chaque case.