M3105-Conception et Programmation Orientée Objet

Projets 2019 – 2020

Sujet 2 : Jeu de Pente

Java

Rando Mathias

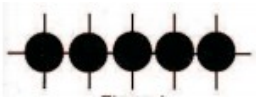
Sommaire

1) Introduction

Java est un langage orientée objet où on peut créer des applications graphiques grâce à Swing.
Le but du sujet est de programmer graphiquement un jeu créer par Gary Gabrel en 1973 : le Jeu de Pente.
Au premier abord, le sujet peut paraître simple mais devient plus complexe et intéressant si on se penche sur les règles du jeu original.

Le jeu de Pente tient ses similitudes du jeu de Go. On joue dans un tablier : le goban.
Le goban comme pour le jeu de go est d'une dimension de 19 X 19 soit 361 intersections où on peut placer des pions généralement de couleur blanche ou noire.
Ce jeu se joue seulement à deux et chacun possède des pions d'une couleur et chacun joue à son tour.
Mais là où dans le Jeu de Go, il faut contrôler le plan du jeu en « construisant » des territoires avec nos pions, les règles du jeu de Pente sont plus simples.
Pour remporter la partie, il existe 2 manières différentes, soit par l'alignement de pions, soit par la capture de pions.

La première manière est la plus rapide pour gagner la partie.
Il faut aligner diagonalement, verticalement ou horizontalement 5 pions de la même couleur que notre camp sans qu' il y est entre nos 5 pions, des espaces ou bien des pions adverses.



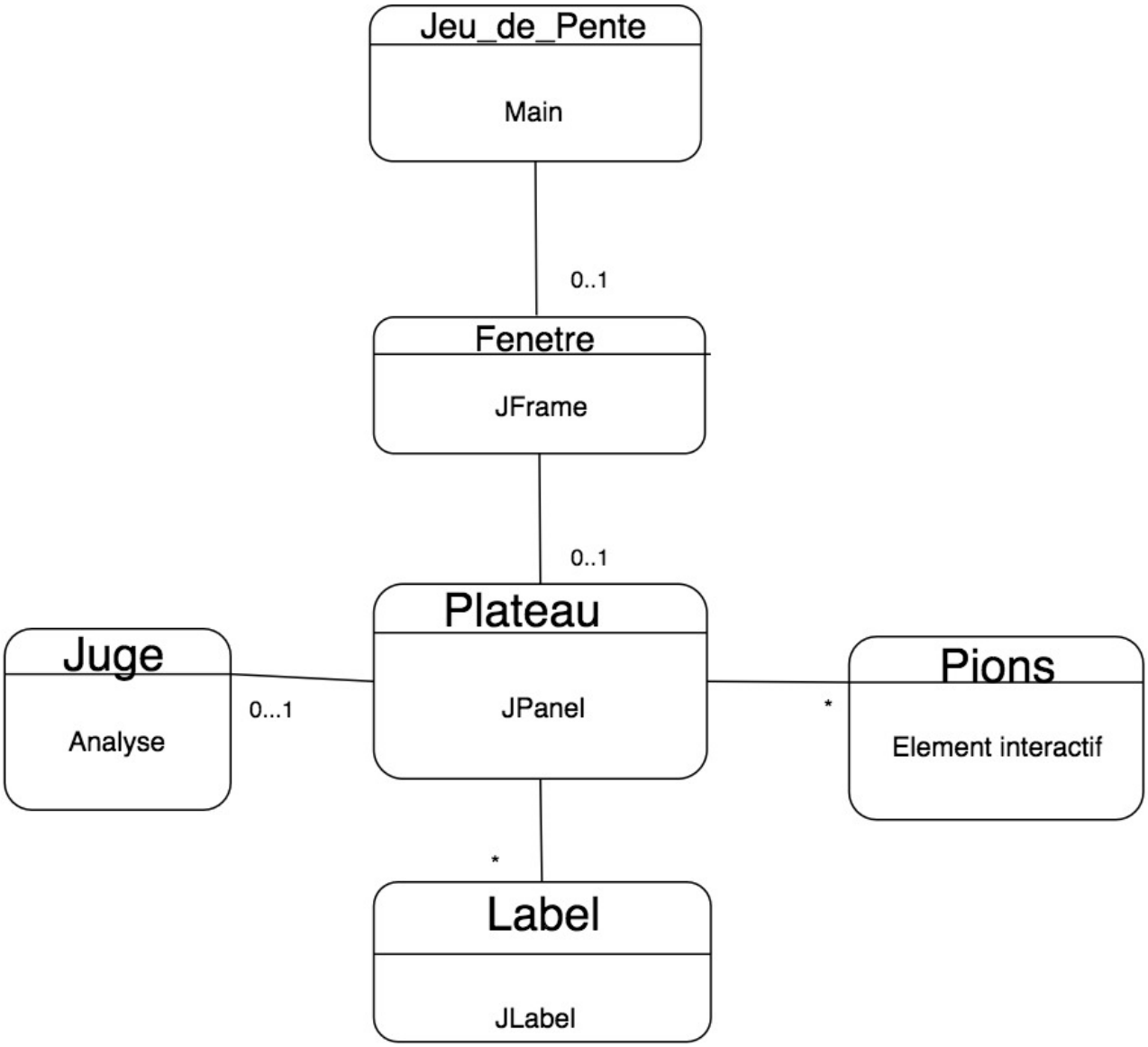
La deuxième est plus difficile, il faut capturer 5 fois une paire de pions de l'adversaire.
Pour effectuer une « capture », il suffit d'entourer les deux pions adverses avec un de nos pions à chaque extrémités.
Les pions capturés seront ensuite retirés du tablier.
Attention, si l'adverse place son pion entre deux extrémités déjà posé alors cela ne compte pas comme une capture.

2) Analyse

2-1) classes utilisées

Les principales classes utilisées sont :
- une classe « Jeu_de_Pente » qui correspond au main.
- une classe « Fenetre » qui hérite de JFrame, il créera la fenêtre de l'application.
- une classe « Plateau » qui hérite de Jpanel, il gérera tous les composants et les évènements.
- une classe « Pions » qui représente les pions du jeu ainsi que les sacs de pions.
- une classe « Juge » qui analysera le tablier et détectera d'éventuelles figures de pions.
- une classe « Label » qui représentera les 2 labels des captures et 1 pour désigner celui qui joue.

2-2) relation entre les classes

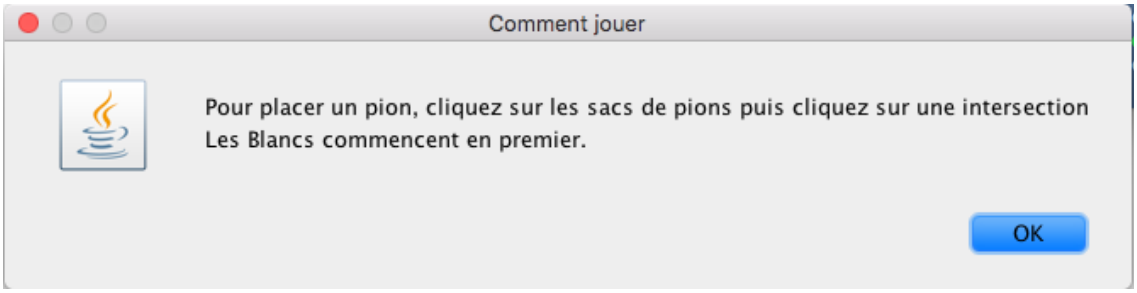


2-3) fonctionnement global

Lorsque on crée la fenêtre à partir de la fonction main, la classe Fenetre aura comme attribut privé, un objet Plateau qui sera instancié dans le constructeur de Fenetre.
C'est dans la classe Plateau que va se dérouler le fonctionnement global de l'application.
Dans le constructeur de Plateau, on va préparer l'interface du jeu qui seras un JPanel qui couvriras toute la fenêtre. On placera dans le JPanel :

- une image d'un palier du jeu de Pente.
- 361 pions « vide » sans image dans les intersections du palier, en attente d'interaction par l'utilisateur.
- 2 pions en état « attente » qui sont représentées par des images de sacs de pions qui varient en fonction de des couleurs, une pour les blancs et une pour les noirs.
- 2 Labels qui afficherons le nombre de paires capturées par les blancs ou par les noirs.
- 1 Label qui annoncera à qui ça sera le tour de jouer.

On activera dans le constructeur une fonction « preparer_tableaux_point » qui placeras les 361 pions vides dans le palier.
Et on exécutera une boite de dialogue avec un bouton « OK » qui expliquera comment jouer dans le logiciel avant de pouvoir lancer la fenêtre.



Ensuite lorsque toute l'interface a été mis en place, on pourras interagir avec le jeu grâce à la souris et surtout grâce aux objets de la classe Pions.

La classe Pions possède des attributs essentiels : la couleur du pion, le statut du pion, les coordonnées (x y) du pion par rapport à la fenêtre, l'image du pion, le label.
Tous ces attributs peuvent être affichés ou modifiée par d'autres classes.

Le pion peut avoir que 2 couleurs : BLANC et NOIR
Il peut avoir que 3 statuts : ATTENTE, VIDE et PLACE

- ATTENTE : correspond aux sacs de pions, si on clique dessus, cela permettra de placer un pion en fonction de la couleur du sac.
- VIDE : correspond aux pions sur les intersections du palier, ils attendent à ce que l'utilisateur clique avant sur un des sacs pour pouvoir être cliqués, si un pion vide est cliqué alors il changera son image en pion par rapport à la couleur du sac et passera du statut VIDE au statut PLACE.
- PLACE : ce sont des pions placés par l'utilisateur, ils sont stockés dans le tableau des pions et on ne pourras plus interagir avec eux.

Mais il reste l'attribut boolean « clique_par_souris » (faux par défaut) qui est essentiel pour les interactions faite par l'utilisateur car un objet pion possède un mouselistener dans son JLabel.
Cette variable détermine si l'élément pion a était cliqué par la souris.
En effet, la classe Pions implémente l'interface MouseListener et c'est dans la fonction « MouseClicked » que on fera basculer cette variable vers vrai en fonction du statut du pion.
Seuls les pions avec les statuts ATTENTE ou VIDE peuvent être cliqués.

De retour dans le JPanel, il possède également une interface MouseListener car dans la fonction « preparer_tableaux_point » et dans le constructeur du Plateau. On met un deuxième mouselistener dans le JLabel des pions vides et pour les pions « ATTENTE » (les sacs).

```
94                  tableau_pions_plateau[j][i].get_label().addMouseListener(this);
```

```
50          add(sac_blanc.get_label()); // on fait afficher le sac blanc
51          sac_blanc.get_label().addMouseListener(this); // le label du sac blanc réagit aux évènements
52          add(sac_noir.get_label()); // on fait afficher le sac noir
53          sac_noir.get_label().addMouseListener(this); // le label du sac noir réagit aux évènements
```

Lorsque on clique sur un des labels des objets Pions, on active le MouseClicked du JPanel qui permettra de définir de quelle couleur est le pion que tiendra la souris.
C'est la chaine de caractère « mouse_tient_couleur » qui stockera cette information.
En fonction du sac où on clique, cela changera cette variable soit en BLANC ou soit en NOIR.

```
231                     mouse_tient_couleur = "BLANC"; // on dit
232                     sac_blanc.set_false_clique_par_souris();
233                 }
```

Maintenant que on a définis cette variable, lorsque on cliquera sur un des 361 pions vides et non à nouveau sur les sacs, la fonction « placer_un_pion_dans_le_plateau » appelé dans le MouseClicked qui au premier clic ne faisait rien puisque « mouse_tient_couleur » n'a pas été définis, va placer un pion dans le palier.

```
221             placer_un_pion_dans_le_plateau(tableau_pions_plateau);
```

En réalité, la fonction va chercher parmi tous les pions vides, celui qui possède son « clique_par_souris » en true et va lui changer son statut et sa couleur (ce qui changera son image) par rapport au sac que on a cliqué avant.
La fonction changera au passage les trois Labels de l'interface après avoir préparer et laissé analyser le jeu du palier par l'objet de la classe Juge initialiser par le constructeur JPanel.
Pour analyser, la classe Juge a besoin d'une représentation du jeu dans la classe Plateau, pour cela, il a une copie du tableau à deux dimensions : tableau_pions_plateau chez lui.

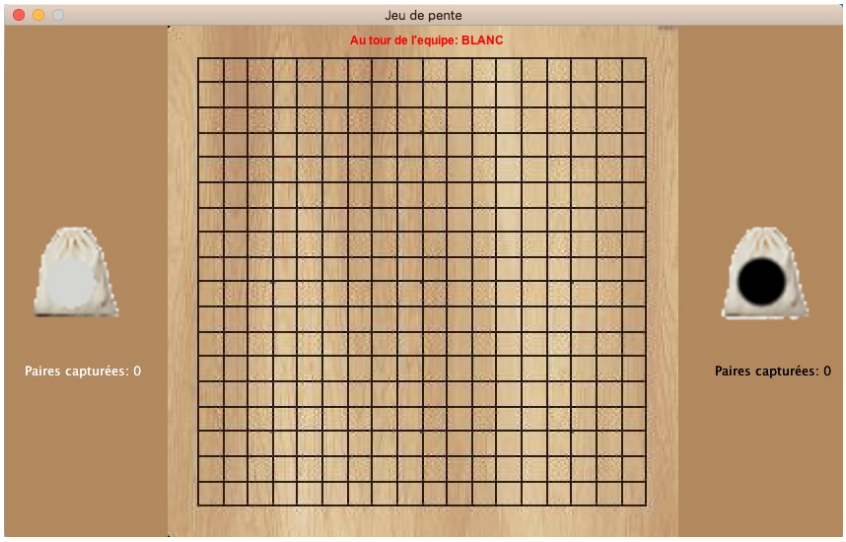
La classe Juge à pour seul but d'analyser l'ensemble du palier à la recherche de figures de pion pour pouvoir définir un vainqueur.
Dans la fonction « placer_un_pion_dans_le_plateau », après avoir placé le nouveau pion dans le palier et modifié la case du tableau de pions du JPanel.
Il va d'abord mettre à jour son coup dans le tableau de pions de l'objet Juge avec le pion en paramètre.
Ensuite on lance la fonction de l'objet Juge « analyse_plateau_pion » avec en paramètre les coordonnées (x y) du pion dans le tableau de la classe Plateau et quelque paramètres secondaires.

« analyse_plateau_pion » va activer à partir des coordonnées du pion deux algorithmes, un pour détecter un alignement de pion et un autre pour détecter et compter une capture de pion.
Si un des deux algorithmes détecte soit un alignement ou bien une capture qui se résultera à 5 captures pour une équipe, alors le jeu affichera une boite de dialogue pour prévenir que la partie est finie et qu'elle est l' équipe qui a gagné.

3) Réalisation

3-1) Choix des techniques

Pour des raisons de style, j'ai choisi de mettre comme interface, une image de palier avec un fond de couleur marron qui va avec.
Cela explique aussi l'emplacement fixe des pions vides dans les intersections du palier.

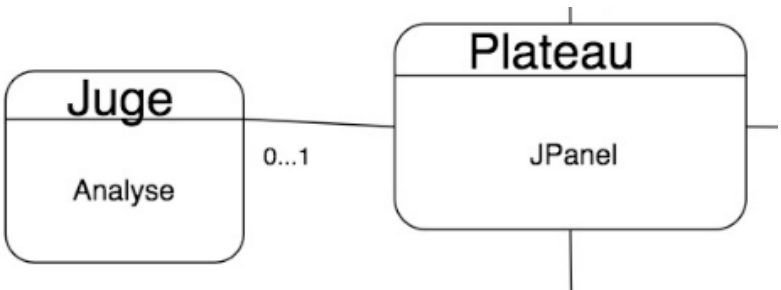


Parlons des interactions, on utilise deux mouselistener sur les JLabels des pions car c'est le seul moyen de faire fonctionner l'application.

Si on avait choisi de mettre un seul mouselistener sur le JLabel des pions dans la classe Pions, comment pouvait t'on alors communiquer avec la classe Plateau ?
Et Si on avait choisi de mettre un mouselistener que du coté du JPanel ?
Alors on aurait eu aucun moyen de faire comprendre au logiciel quel élément exact l'utilisateur a cliqué.
On pouvait savoir que les données de Pions sans que les pions aient la capacitée de réagir à un clic du coté de la classe Pions.

La raison de l'utilisation d'un mouselistener pour les JLabels des pion dans les deux classes Plateau et Pions est la suivante :
Si un pion peut stocker l'information de l'évènement qu'il a reçu dans un boolean(clique_par_souris).
Alors du coté du JPanel, on peut alors récupérer cette information du pion et l'utiliser pour déterminer quel pion a été cliqué.

Ensuite parlons du choix des classes, particulièrement pour la classe juge, si on désire analyser le jeu du palier c'est à dire du coté du tableau de pion. Pourquoi utiliser un nouvelle classe qui copiera exactement le tableaux de pions de la classe Plateau, chaque fois que on place un nouveau pion.
Cela risque d'utiliser des ressources inutilement et de faire ralentir l'application ainsi que l'expérience de l'utilisateur.
La raison est simple : c'est par soucis de propreté du code, si on avait mit les algorithmes de détections et les autres fonctions dans la même classe.
Alors cela aurait pus être encore plus dur à lire et moins beau à regarder.
De plus d'un point de vue logique, la tâche du Plateau est de faire afficher des composants pas d'analyser le jeu, c'est le travail d'une autre entité.



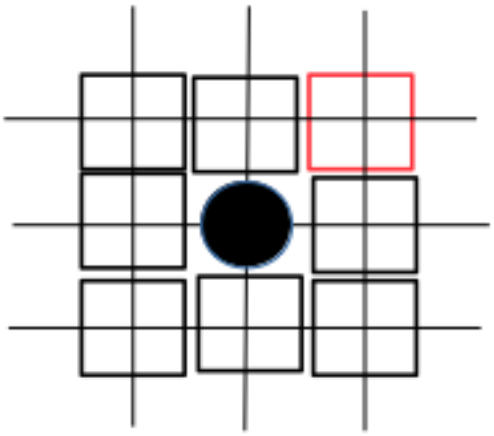
3-2) Les algorithmes d'analyse de jeu

Comme dit dans la partie « fonctionnement global », lorsque on place un nouveau pion, on active une fonction d'un objet de type Juge pour analyser et chercher dans le jeu d'éventuelles figures de pion pour désigner un vainqueur ou non.

La fonction « analyse_plateau_pion » utilise deux algorithmes, un pour l'alignement des pions et un pour la capture des pions

Les deux algorithmes se basent et utilisent le même algorithme :
l'algorithme pour rechercher un pion précis autour du notre pion

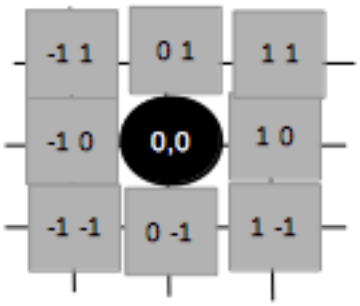
il permet à partir d'un pion précis de chercher autour de lui si il y a un pion aux coordonne et de couleur précis.



```
public boolean algorithme_pion_precis(int x_param, int y_param, int x_a_detect, int y_a_detect, String couleur)
	{
	boolean trouve = false; // determine si la fonction a trouve le pion ciblé
	for (int i = -1; i < 2 ; i++)
	{
		for (int j = -1 ; j < 2 ; j++)
		{
			int x_autour = (x_param + i);
			int y_autour = (y_param + j);

			if ( (x_autour >= 0 && x_autour < 19) && (y_autour >= 0 && y_autour < 19)
			&& (x_autour != x_param || y_autour != y_param))
			{
				if (tableau_pions_analyse[x_autour][y_autour].get_statut_pion() == "PLACE"
				&& tableau_pions_analyse[x_autour][y_autour].get_couleur_pion() == couleur
				&& (x_autour == x_a_detect && y_autour == y_a_detect))
				{
					trouve = true;
				}
			}
		}
	}
```

Au début on définie une variable locale booléenne et on entre dans une double boucle for allant de -1 vers 1. les valeurs i et j des boucles for vont être utiles dans l'algorithme.
Elles vont permettre de faire le tour des pions proche de notre pion initiale.

On crée des coordonnées x_autour et y_autour temporaires en additionnant j avec le x du pion initial et i avec le y. cela permet d'avoir à la fin des coordonnées de tableaux autour du pion initiale.
Par exemple pour un pion de coordonnée 0, 0 avec j = 1 et i = 0
on peut avoir par x_autour et y_autour, le pion de coordonne 1 0.



Après avoir créer ces coordonnées temporaires, nous allons vérifier en premier si les coordonnées ne dépassent pas 0 ni 19 et également si ils sont pas exactement comme les coordonnées du pion initial.
Si on passe la première condition, la deuxième est plus sélectif.
En effet, il faut que le pion sélectionné par les coordonnées temporaires ait le statut PLACE et aussi qu'il possède la même couleur que en paramètre. Pour finir il faut que les coordonnées temporaires x_autour et y_autour correspondent exactement aux coordonnées du pion rechercher définie par x_a_detect et y_a_detect.

Si c'est le cas alors on fait passer la variable trouve faux par défaut à vrai sinon rien.
Et on continue avec les autres valeurs de j et i.

Algorithme de recherche d'alignement de pion.

```
boolean detection_alignement_pions(int x_param, int y_param, String couleur ,int nombre_pions_aligne)

    for (int i = -1; i < 2 ; i++)
    {
        for (int j = -1 ; j < 2 ; j++)
        {
            int x_autour = (x_param + i);
            int y_autour = (y_param + j);


            if ( (x_autour >= 0 && x_autour < 19)
            && (y_autour >= 0 && y_autour < 19)
            && (x_autour != x_param || y_autour != y_param))
            {
                if (tableau_pions_analyse[x_autour][y_autour].get_statut_pion() == "PLACE"
                        && tableau_pions_analyse[x_autour][y_autour].get_couleur_pion() == couleur)
                {

                    PARTIE 2.
                }
            }
        }
    }
    return false
```

PARTIE 1 :

Le début de l'algorithme ressemble à celui de l'algorithme de base avec quelques exception.
Dans le deuxième if, on ne cherche pas des coordonnées particuliers et on met à la fin du premier for, un return false. Ce dernier se déclenche lorsque on a terminé le tour des pions autour du pion en paramètre et que on ne trouve toujours pas de figures alors on retourne un faux.

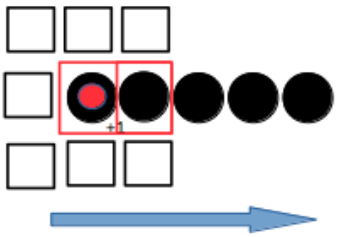
PARTIE 2 :

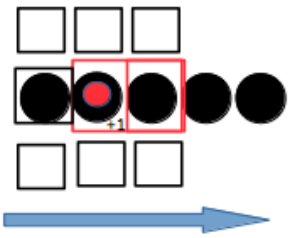
```
boolean pion_trouve = false;
int compteur_pion_aligne = 1;
for (int b = 0 ; b < nombre_pions_aligne-1 ; b++)
{
        pion_trouve = detection_pion_precis_autour((x_param + (b*i)), (y_param + (b*j)),(x_param + ((b+1)*i)),(y_param + ((b+1)*j)),
        couleur);
        if (pion_trouve == true){
                compteur_pion_aligne++;
        }
        else if (pion_trouve == false) {
                break;
        }
}
for (int n = 0 ; n < nombre_pions_aligne-1 ; n++)
{
        if (compteur_pion_aligne < nombre_pions_aligne){
                pion_trouve = detection_pion_precis_autour((x_param + (n*(-i))), (y_param + (n*(-j))), (x_param +
                ((n+1)*(-i))), (y_param + ((n+1)*(-j))), couleur);

                if (pion_trouve == true){
                        compteur_pion_aligne++;
                }
        }
        else {
                break;
        }
}
if (compteur_pion_aligne == nombre_pions_aligne){
        return true;
}
```

L'algorithme fonctionne de cette manière : lorsque il trouve un pion au alentour, il va garder en tête les premiers itérateurs i et j .
Il va vérifier à la chaine si le nouveau pion temporaire est bien celui que on cherche et est de la même couleur en utilisant l'algorithme de base.
Si l'algorithme de base trouve alors on compte cela comme un pion aligné en plus dans le compteur sinon on quitte la boucle for.
Et on ré-effectue l'algorithme de base en changeant les coordonnées du pion de base avec le pion temporaire et on prend comme pion temporaire à vérifier le pion suivant de la chaine de pion grâce à b et n. ce sont des itérateurs représentant le nombre de vérification pour un alignement de n pions
Par exemple : pour un alignement de 5 pions, il faut seulement 4 vérifications de paires.

1 + 1

(i j) *(b = 0)
(i j) * (b = 1)





2 + 1

(i j) * (b = 1)
(i j) *(b = 2)

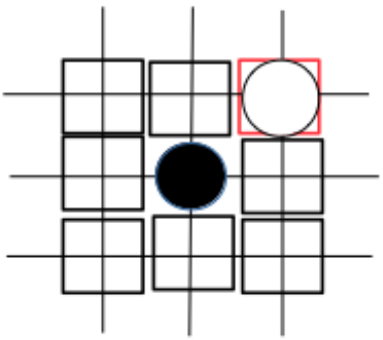
Si jamais dans le sens, on n'arrive pas à trouver un alignement de n pions car on a placer notre pion au milieu de 2 paires de pions. Alors dans une autre boucle for, on effectue la même opération mais dans le sens inverse avec (-i) et (-j) ! Si jamais le compteur du nombre de pion aligne correspond à nombre_pion_aligne alors on fait retourner vrai.

Algorithme de recherche de capture de pion :

PARTIE 1 :

```
public void detection_capture_pions(int x_param, int y_param, String couleur ,int nombre_pions_aligne)
{
	String couleur_adverse = "";
	if (couleur == "NOIR")
		couleur_adverse = "BLANC";
	else if (couleur == "BLANC")
		couleur_adverse = "NOIR";

	for (int i = -1; i < 2 ; i++)
	{
		for (int j = -1 ; j < 2 ; j++)
		{
			int x_autour = (x_param + i);
			int y_autour = (y_param + j);

			if ( (x_autour >= 0 && x_autour < 19) && (y_autour >= 0 && y_autour < 19) &&
			(x_autour != x_param || y_autour != y_param))
			{
				if (tableau_pions_analyse[x_autour][y_autour].get_statut_pion() == "PLACE" &&
				tableau_pions_analyse[x_autour][y_autour].get_couleur_pion() == couleur_adverse)
				{
					PARTIE 2.
				}
			}
		}
	}
}
```

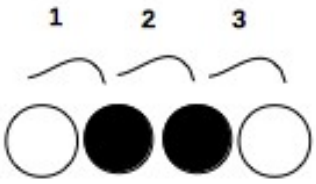
Rien de spécial à part le fait que on a besoin d'avoir comme variable la couleur inverse en fonction du paramètre « couleur » et du fait que dans le deuxième if, on ne cherche pas à vérifier si le pion est de la même couleur que nous mais belle et bien de la couleur adverse.



```
boolean pion_capture = true;
for (int b = 0 ; b < nombre_pions_capture+1 ; b++)
{
	if (b < nombre_pions_capture) {
		pion_capture = detection_pion_precis_autour((x_param + (b*i)), (y_param + (b*j)), (x_param + ((b+1)*i)), (y_param ((b+1)*j)),
		couleur_adverse);

		liste_pion_capture_a_enlever.addElement(new Pions(tableau_pions_analyse[x_param + ((b+1)*i)]
		[y_param + ((b+1)*j)].get_couleur_pion(),tableau_pions_analyse[x_param + ((b+1)*i)] [y_param +
		((b+1)*j)].get_statut_pion(),x_param + ((b+1)*i),y_param + ((b+1)*j) ) );
	}
	else{
		pion_capture = detection_pion_precis_autour((x_param + (b*i)), (y_param + (b*j)), (x_param + ((b+1)*i)),
		(y_param + ((b+1)*j)), couleur);
	}

	if (pion_capture == false)
		break;

}

if (pion_capture == true){
	if (couleur == "BLANC")
		compteur_capture_équipe_blanc++;
	else if (couleur == "NOIR")
		compteur_capture_équipe_noir++;
}
else{
	clear_liste_pion_capture_a_enlever();
}
```

PARTIE 2 :

Dans la boucle for, on va faire plus d'itérations que la valeur « nombre_pions_capture » car cette variable représente combien de pion doit t'on enfermer par les extrémités pour que cela soit considéré comme une capture.
Par exemple pour une capture d'une paire de pions adverses soit 2 pions, il faut vérifier la figure 3 fois par l'algorithme de base. Donc on met (nombre_pions_capture + 1)



On va à la fois chercher pour des pions de la couleur adverse et aussi de notre couleur. Pour comprendre le if, il faut voir comment va se comporter la lecture d'une capture dans un sens.
Dans la figure à droite, si on lit cette chaine de gauche à droite, on remarquera que dans les vérifications 1 et 2, peut importe soit la couleur du pion du centre, on cherchera un pion de la couleur adverse même si dans la vérification 1 le pion du centre est de couleur blanche et noir dans la vérification 2.
C'est seulement dans la dernière vérification (le 3) que on va chercher un pion de la même couleur que le notre puisque ce pion en question enferme la paire de noir.
Ainsi la dernière vérification sera l'équivalent du (nombre_pions_capture + 1).

Ainsi pour = 0 tant que b restera inférieur à nombre_pion_capture, on cherche des pions de couleur adverses et on les enregistre dans un vecteur pour pouvoir les enlever du palier si jamais la figure est complète.Et lorsque b atteindra nombre_pion_capture alors on est à la dernière vérification et on cherche un pion de la même couleur que le notre.
Si jamais pion_capture devient faux alors on arrête par un break.
Lorsque « pion_capture » devient et reste vrai alors on va incrémente le compteur de capture de pion en fonction de la couleur en paramètre.
Si la couleur est BLANC alors on incrémentera le compteur des blancs et si la couleur est NOIR alors on incrémentera le compteur des noirs.
Sinon on vide le vecteur « liste_pions_capture » , la liste de pions noirs que on comptait enlever palier de JPanel si jamais « pion_capture » restait vrai.

4) Utilisation

4-1) Configuration requise

Vous avez besoin de pas grands choses pour exécuter ce jeu si ce n'est que l'environnement java.
Vous pouvez le télécharger depuis le site officiel : https://www.java.com/fr/download/

Pour lancer le programme, il faut aller dans le répertoire où se trouve toute les classes via le Terminal.
ensuite vous exécuter la commande suivante : java Jeu_de_Pente.

4-2) mode d'emploi

Les 2 joueurs utiliseront le même programme sur un seul ordinateur pour jouer.

Lorsque vous lancer le programme, une boite de message apparaitra pour vous expliquer comment jouer dans le programme.

Dans le jeu, on désire placer un pion dans le palier, pour cela vous avez 1 sac dans chaque coté avec un cercle noir ou blanc dessiné dessus.
Le programme ne placera pas de pion tant que vous n'avez pas cliqué dans un de ses sacs.
Lorsque vous cliquez sur un sac, votre souris au prochain clic dans le palier placera un pion de la couleur du logo du sac.

Si vous avez cliqué sur le sac blanc à gauche, vous placerez un pion blanc.
Si vous avez cliqué sur le sac noir à droite, vous placerez un pion noir.

Attention : lorsque vous cliquez sur un des sacs, c'est irréversible, vous ne pourrez pas cliquer par erreur sur le palier. Cela placera automatiquement le pion.

Au début de la partie, c'est au blanc de jouer, cela sera marqué en rouge au milieu en haut.
Du coup vous ne pourrez pas placer de pion d'une équipe alors que le jeu attribue le tour à l'autre équipe. Ni même changer la couleur des pions en cliquant sur le sac de la couleur de la mauvaise équipe.

Si c'est au tour des blancs, vous ne pourrez pas placer de pions noirs ni même cliquer sur le sac noir et inversement au tour des noirs

Bon jeu.

5) Conclusion

5-1) Bilan

Le sujet n’a pas de but précis technique, scientifique, il a pour vocation le défi.

Mais une chose est sûr c’est que on le veuille ou non, la technologie est encore un domaine récent y compris les ordinateurs.
Le sujet montre que la transmission des éléments du monde réel vers l’informatique est très complexe car les éléments simples et acquis de la vie réel sont difficiles à traduire en informatique.

En comparaison du Jeu de Pente, il est beaucoup plus simple d’aller acheté en magasin voire même dessiner le palier et bricoler avec un peu d’imagination les pions.
Tant dis que retranscrire ça en informatique sur un ordinateur est beaucoup plus complexe car il faut aller creuser la logique derrière les règles du jeu, la préparation du palier et des pions, les gestes simples comme poser ou bien enlever des pions, la fin de la partie et bien plus encore.
Il faut aussi séparer les différents comportements durant la partie comme analyser, jouer, compter en différents entités en informatique. Par exemple : lorsque on joue des pions, on analyse en même temps et on détecte directement par réflexe des captures ou bien des alignements.
En informatique, il faut séparer cela et donner des tâches précises, des fonctions, des algorithmes qui s’occupent de reproduire les éléments de la réalités.

C’est le cas du réseau qui s’inspire de la réalité pour perfectionner ce domaine.

5-2) optimisations possibles

Comme je l’ai dit dans la partie Choix des techniques si on veut améliorer la performance du logiciel, on peut mettre toute les méthodes de la classe Juge directement sur la classe Plateau qui est le JPanel en modifiant son tableau de Pion, en dépit de la propreté du code.
À noter que la différence de performance est drastique.

5-3) extensions possibles

On peut imaginer des extensions comme :

-des petites animations comme lorsque on place les pions dans le palier

-lorsque on clique dans le sac, on peut voir le pion de la couleur du sac qui suit la position de la souris jusqu’à ce que on le place dans le palier.

- avoir un menu d’accueil pour le joueur avec la possibilité de choisir différents modes de jeu, modifier les règles (combien de pion a aligner, combien de pion a capturer)

- pouvoir jouer contre l’ordinateur, un joueur humain qui joue contre l’ordinateur au jeu de Pente