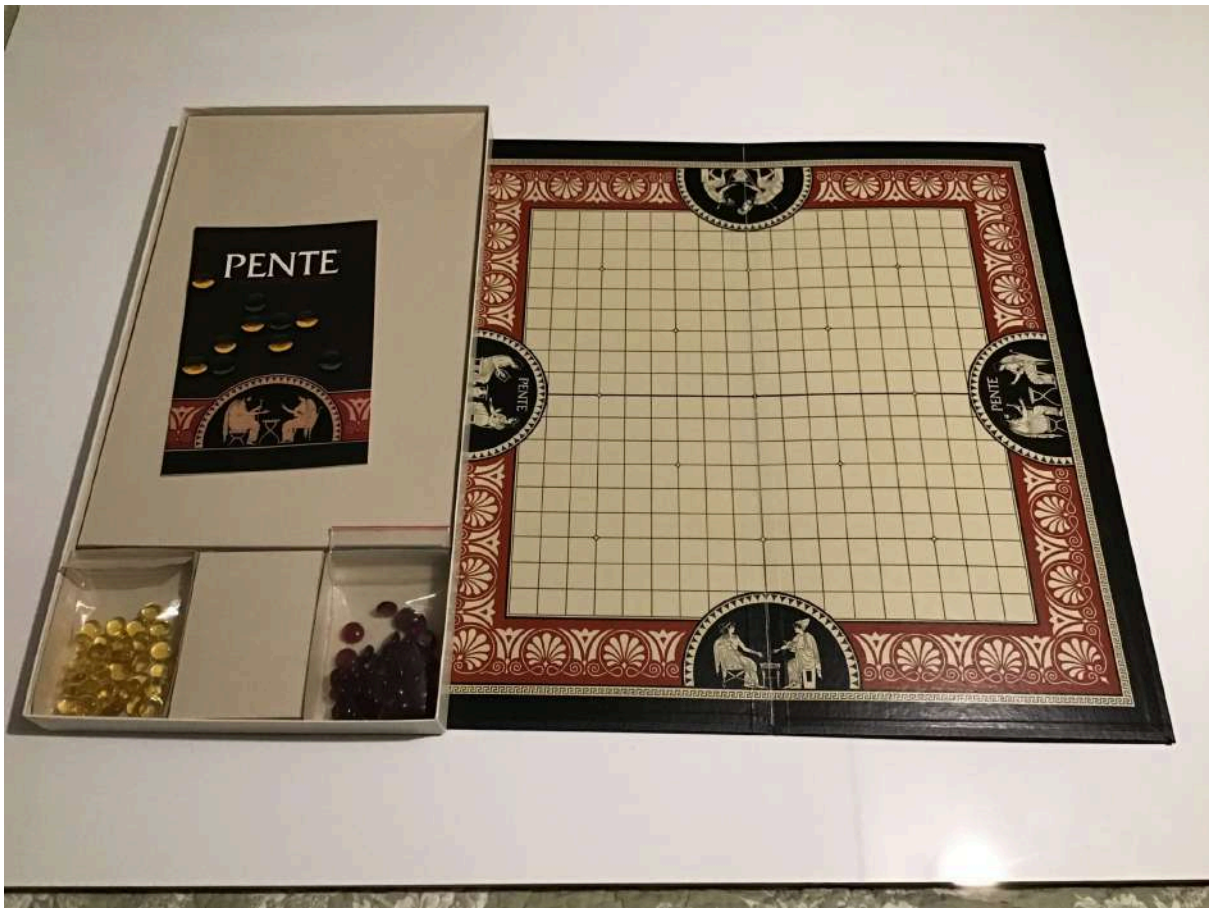
ALGORITHMIQUE ET PROGRAMMATION

Projet Penté

Encadrant : ***Thibault Liétard***

Réaliser par : ***AMINE NAKROU*** & ***YOUSSEF MNIF***

IS3



Université de Lille

POLYTECH LILLE

Cti Commission des titres d'ingénieur

Sommaire

1. Introduction

Le Penté est un jeu de plateau à forte composante stratégique, généralement pratiqué à deux joueurs. Chacun dispose de pions, habituellement marqués « X » ou « O », qu'il place à tour de rôle sur une grille de 19×19 cases. Le jeu offre deux conditions de victoire :

1. **Aligner au moins cinq de ses pions** (horizontalement, verticalement ou en diagonale),
2. **Capturer un total de dix pions adverses** conformément à la règle de prise caractéristique du Penté (que nous détaillerons plus loin).

Dans le cadre de ce projet, nous proposons **l'implémentation complète** des règles du Penté au moyen d'un programme en langage C. Concrètement, l'application permettra de :

- Gérer l'initialisation et l'affichage d'un **plateau** 19×19,
- **Surveiller** l'état du jeu en temps réel (positions des pions, coordonnées, compteurs de captures),
- **Détecter** automatiquement les alignements, les prises et l'abandon,
- **Déclarer** un joueur vainqueur dès qu'il remplit l'une des deux conditions de victoire.

Afin de couvrir l'ensemble des aspects techniques et conceptuels liés à cette réalisation, le présent rapport s'organise comme suit :

1. Nous explicitons **les règles** du Penté retenues dans ce projet, afin de poser un cadre clair sur les mécanismes de victoire et de capture.
2. Nous présentons **le cahier des charges fonctionnel**, décrivant les contraintes et les attentes pour notre programme (interface utilisateur, saisie des coups, gestion des erreurs, etc.).
3. Nous abordons **l'analyse de la solution**, où nous expliquons en détail les algorithmes de vérification des alignements, de détection des captures et de gestion de l'abandon.
4. Nous détaillons **la conception**, en précisant les structures de données adoptées, la logique de stockage et de mise à jour de l'état du jeu, ainsi que certains algorithmes .
5. Nous fournissons **le code source** en C, enrichi de commentaires et d'explications pour une meilleure compréhension et une maintenance aisée.
6. Enfin, nous concluons sur **les perspectives de développement**, en évoquant des améliorations potentielles pour prolonger ce projet (par exemple, l'implémentation d'un bot adverse ou la conception du code en hybride).

Au fil de ce document, nous mettrons l'accent sur la robustesse des algorithmes, la clarté des explications et la précision des implémentations, dans le but d'offrir une solution complète, fiable et évolutive.

2. Règles de Penté (rappel)

- **Plateau** :
  Le jeu se déroule sur une grille de 19×19 cases, ce qui offre une grande liberté de placement et permet des stratégies variées. Chaque intersection est éligible pour la pose d'un pion, et aucune case ne peut être occupée par plus d'un pion à la fois.
- **Joueurs** :
  Deux joueurs s'affrontent, chacun étant identifié par un symbole distinct, généralement « X » pour le premier et « O » pour le second. L'ordre de jeu est déterminé en début de partie et s'alterne tour après tour.
- **Placement** :
  Au début de son tour, le joueur actif place un pion portant sa marque (« X » ou « O ») sur une case inoccupée. Il ne peut pas déplacer ou retirer de pions déjà placés, sauf en cas de capture (voir plus bas). Cette simplicité dans la pose des pions rend le jeu rapide à apprendre tout en offrant une profondeur stratégique.
- **Alignement** :
  Un joueur remporte immédiatement la partie s'il parvient à aligner au moins cinq de ses propres pions de manière consécutive. Cet alignement peut être horizontal, vertical ou diagonal. Il est donc essentiel de surveiller l'ensemble du plateau pour empêcher son adversaire de constituer une ligne victorieuse.
- **Captures** :
  Le Penté se distingue par sa **règle de capture**, qui donne lieu à des retournements de situation. Si un joueur place son pion de sorte à encadrer deux pions adverses consécutifs entre deux de ses propres pions alors les deux pions adverses au milieu sont immédiatement retirés du plateau. Chaque paire ainsi capturée est comptabilisée dans un compteur de captures.
- **Victoire par capture** :
  En plus de la condition d'alignement, un joueur peut gagner s'il parvient à **capturer un total de dix pions adverses**, soit l'équivalent de cinq paires. Dès qu'un joueur atteint ce seuil, il est déclaré vainqueur, même si aucun alignement de cinq pions n'est encore formé.
- **Abandon** :
  Un joueur peut décider d'abandonner la partie à tout moment en saisissant les coordonnées **0,0** (ou toute autre convention prévue par l'implémentation). Cet abandon confère la victoire immédiate à l'autre joueur.

Grâce à ces règles relativement simples, le Penté parvient à marier accessibilité et subtilité stratégique. Les joueurs doivent en permanence équilibrer la constitution de leurs alignements et la prévention des captures adverses, tout en cherchant à capturer eux-mêmes les pions adverses pour se rapprocher de la victoire.

3.Cahier des charges :

Le présent projet consiste à **programmer en C** une application permettant à deux joueurs humains de s'affronter en respectant les règles du Penté.

1. Objectifs

1. **Modéliser le jeu** :
    - Gérer un plateau de 19×19 cases.
    - Permettre de stocker l'état de chaque case (vide, pion O, pion X).
2. **Gérer le déroulement d'une partie** :
    - Alterner les tours entre le joueur O et le joueur X.
    - Permettre la saisie des coups (lignes, colonnes).
    - Vérifier la validité de chaque coup (cases hors limites, cases déjà occupées).
    - Gérer la possibilité d'abandon (entrée de la coordonnée 0,0).
    - Détecter la victoire :
        - Par alignement d'au moins cinq pions,
        - Ou par capture de dix pions adverses (en prenant par paires).
3. **Assurer la robustesse** :
    - L'application doit résister aux erreurs de saisie (ex. format invalide).
    - Un affichage clair du plateau et des informations (prises, coordonnées) doit être fourni après chaque coup.
4. **Proposer une conception claire du code** :
    - Séparer le plus possible la logique du jeu (structures, règles) et l'interface utilisateur (affichage console, saisie clavier).
    - Fournir un pseudo-code pour les fonctionnalités majeures (alignement, capture, etc.).
    - Faciliter la maintenance et l'évolutivité du programme.

2. Fonctionnalités attendues

1 Initialisation du plateau

- Mettre toutes les cases à vide ('.').
- Initialiser à zéro le nombre de captures pour chaque joueur.

2 Affichage du plateau

- Afficher les numéros de ligne et de colonne (1 à 19).
- Afficher le contenu de chaque case ('.', 'O', 'X').
- Afficher le nombre de pions capturés pour chacun des deux joueurs.

3 Saisie des coups

- Lire un coup au format ligne,colonne (ex. 5,10).
- Gérer la possibilité de saisies invalides et redemander la saisie le cas échéant.

- Gérer l'abandon si le joueur saisit 0,0.

4 Placement des pions

- Vérifier la validité du placement (position dans le plateau, case libre).
- Placer le pion O ou X dans la case correspondante.

5 Détection des captures

- Détecter et retirer du plateau les pions adverses capturés lorsque le motif (joueur)(adversaire)(adversaire)(joueur) est formé dans l'une des huit directions.
- Incrémenter le compteur de prises du joueur ayant capturé.

6 Détection de la victoire

- Vérifier après chaque coup si un alignement d'au moins cinq pions a été créé (4 directions principales).
- Vérifier si le nombre de pions capturés par un joueur atteint 10.
- En cas de victoire (par alignement ou par 10 captures), afficher le vainqueur et arrêter la partie.

7 Fin de la partie

- Un joueur peut abandonner en saisissant 0,0, ce qui déclare immédiatement l'autre joueur vainqueur.
- Si l'une des conditions de victoire est remplie (alignement ou nombre de captures), la partie s'arrête.

3. Contraintes et ressources

1. **Langage** : C (compilateur standard, par exemple gcc).
2. **Plateforme** : Programme console (terminal), pas de dépendances spécifiques en dehors de la bibliothèque standard.
3. **Mémoire** : Utilisation de structures simples (tableau statique 19×19) pour stocker le plateau.
4. **Interface** : Affichage texte, saisie depuis stdin (clavier ou redirection depuis fichier).
5. **Testing** :
    - Rédiger des jeux de test dans des fichiers de saisie (pente1.txt, pente2.txt) simulant différents scénarios : victoire par alignement, victoire par captures, abandon d'un joueur, etc.
    - Les tests doivent couvrir les cas limites (coordonnées hors-limites, case déjà occupée, saisie invalide…).

4. Livrables attendus

1. **Code source complet** en C (fichier .c), documenté et commenté :
    - Structures de données (plateau, pions).

   - Fonctions (initialisation, affichage, placement de pion, détection de captures et de victoire, etc.).
   - Procédure de jeu (main() ou fonction jouer()).
2. **Rapport de projet** (format PDF ou équivalent), incluant :
   - Introduction et conclusion.
   - Cahier des charges succinct (celui-ci) rappelant les objectifs et les exigences.
   - Analyse du problème et description de la méthode de résolution (diagrammes ou pseudo-code).
   - Déroulement global du programme (logique d'enchaînement des tours, gestion des entrées).
   - Description et justification des structures de données.
   - Description de chaque fonction (hors affichage) : paramètres, traitements effectués, rôle de chaque variable.
   - Détails d'implémentation sur la prise de pions (algorithme de détection).
   - Tests effectués, problèmes rencontrés, améliorations possibles.
3. **Fichiers de test** :
   - Fichiers texte (pente1.txt, pente2.txt, etc.) permettant d'automatiser des tests via redirection d'entrée.
   - Scénarios variés (victoire par alignement, par captures, abandon…).
4. **Archive compressée** :
   - Une archive .tgz ou .zip regroupant l'ensemble du code et du rapport.
   - Livraison sur la plateforme de cours (Moodle ou autre) avant la date limite fixée.

5. Organisation et calendrier

- **Étapes de développement**
  - Analyse du projet et rédaction du cahier des charges (séance 1).
  - Implémentation progressive des fonctionnalités (séances 2 à N−1).
  - Finalisation : tests, corrections, documentation (dernière séance).
  - Remise du projet avant le **23 janvier 2025** à 18h (date indicative tirée de l'énoncé).
- **Dépôt des documents**
  - Après chaque séance, déposer l'avancement dans la rubrique Projet - Rendu.
  - L'archive doit contenir le code source (à jour), la documentation complémentaire et un court compte-rendu d'avancement.

6. Critères de validation

1. **Fonctionnalités minimales** :
   - Le programme doit compiler sans erreur et lancer une partie jouable.
   - Possibilité de jouer en alternant O et X, de vérifier les coups et d'enchaîner jusqu'à la fin de la partie.
2. **Robustesse de la saisie** :
   - Le programme gère les entrées invalides (mauvais format, coordonnées hors-limites, case occupée).
3. **Détection correcte de la victoire** :

- Alignement horizontal, vertical, diagonal d'au moins 5 pions.
- 10 pions capturés.
- Abandon.

4. **Documentation** :
   - Présence d'un rapport clair et structuré (conforme à la section 3. « Travail à rendre » de l'énoncé).
   - Commentaires et algo cohérents avec l'implémentation.
5. **Respect des consignes de rendu** :
   - Remise du projet complet (code, rapport, fichiers de tests) avant la date imposée.

4. Analyse du problème

1. Modéliser le plateau

- **Structure de données**
  Pour gérer un plateau de 19×19, le choix le plus intuitif consiste à utiliser un **tableau à deux dimensions**. Chaque cellule de ce tableau stocke l'état d'une case :
  - '.' pour une case vide,
  - 'X' pour un pion du premier joueur,
  - 'O' pour un pion du second joueur (ou inversement),

---

2. Repérer et valider la saisie de l'utilisateur

- **Format de saisie**
  Les coups sont saisis au format ligne,colonne (par exemple, 5,10). Le programme doit :
  1. Lire la chaîne de caractères saisie par l'utilisateur,
  2. Séparer la partie « ligne » et la partie « colonne » via la virgule,
  3. Convertir ces valeurs en entiers.
- **Gestion de l'abandon**
  Si le joueur saisit 0,0, le programme interprète cette entrée comme un **abandon**. Dès lors, l'autre joueur est **déclaré vainqueur** et la partie s'arrête.
- **Contrôle des limites et de la disponibilité**
  Après conversion, le programme doit vérifier :
  1. Que ligne et colonne sont bien compris entre 1 et 19,
  2. Que la case correspondante (dans le tableau 2D) est **libre**.
     Si ces conditions ne sont pas remplies, la saisie est invalide et le programme doit redemander un coup.

---

3. Vérifier les alignements de 5 (ou plus)

- **Principe**
  Pour déterminer si un joueur à aligner au moins cinq pions, il suffit de contrôler le pion tout juste placé dans **quatre directions principales** :
  1. Horizontale (gauche et droite),
  2. Verticale (haut et bas),
  3. Diagonale montante (↗ et ↙),
  4. Diagonale descendante (↘ et ↖).
- **Méthode de comptage**
  L'idée est de **compter le nombre de pions identiques** de part et d'autre de la case où le pion vient d'être posé. Concrètement :
  1. On part de la position nouvellement occupée,
  2. On se déplace dans une direction (ex. vers la droite), en incrémentant un compteur tant que les pions sont identiques,
  3. On refait la même opération dans le sens opposé (ex. vers la gauche),
  4. On additionne les deux compteurs (en incluant le pion de départ).
     Si cette somme atteint **5 ou plus**, le joueur actif gagne.

---

4. Vérifier les prises

- **Motif à détecter**
  Le Penté autorise la capture de deux pions adverses consécutifs lorsqu'ils sont encadrés par deux pions du joueur actif.
  Dès qu'un tel motif est repéré dans l'une des **huits directions possibles** (horizontal, vertical et les deux diagonales, dans les deux sens), les **deux pions centraux** sont retirés du plateau.
- **Mise à jour des compteurs**
  Chaque capture réussie **incrémente le compteur** de pions capturés du joueur actif (l'équivalent d'une paire capturée). Le joueur doit donc maintenir un total des pions capturés, afin de déterminer si la **victoire par capture** (10 pions, soit 5 paires) est atteinte.

---

5. Déterminer si un joueur gagne

Le programme doit **déclarer un vainqueur** dès que l'une des conditions suivantes est remplie :

1. **Alignement ≥ 5**
   Lorsqu'un joueur forme une ligne continue de cinq pions ou plus dans l'une des quatre directions principales, il gagne immédiatement.
2. **10 pions capturés**
   Si le compteur de pions capturés d'un joueur atteint **10**, ce joueur remporte la partie

sans autre forme de vérification, même si aucun alignement de 5 n'est présent sur le plateau.

3. **Abandon de l'adversaire**
   En cas de saisie de 0,0 par l'adversaire, la victoire est immédiatement accordée au joueur restant.

---

6. contrôle après chaque coup

Pour intégrer ces vérifications de manière efficace, on procède ainsi à chaque nouveau coup:

1. **Valider la saisie**
   - Vérifier que la case indiquée est dans les limites (1..19) et n'est pas déjà occupée.
   - Vérifier si le coup correspond à un abandon (0,0).
2. **Placer le pion**
   - Mettre à jour le plateau en insérant le symbole (« X » ou « O ») au bon emplacement.
3. **Détecter et résoudre les prises**
   - Identifier les éventuelles configurations (joueur)(adversaire)(adversaire)(joueur) dans les huit directions.
   - Retirer les pions capturés et mettre à jour le **compteur de captures** du joueur actif.
4. **Vérifier l'alignement**
   - Contrôler les quatre directions depuis la case nouvellement occupée.
   - Si l'on recense au moins cinq pions alignés, déclarer le joueur actif vainqueur.
5. **Vérifier le total de captures**
   - Si le joueur actif atteint **10 pions capturés**, la partie s'achève immédiatement.
6. **Poursuivre la partie**
   - Si aucune condition de victoire n'est atteinte, on passe au tour du joueur suivant.

5. Description et justification des structures de données utilisées

5.1 Structure Pion

```c
typedef struct {
    char symbole; // 'X', 'O', ou '.' pour une case vide
} Pion;
```

- **Rôle :** représenter le contenu d'une **case** du plateau.
- **Justification :**
    - On a besoin de mémoriser un caractère (symbole) indiquant si la case contient un pion 'X', un pion 'O' ou est vide ('.').
    - Une structure minimaliste suffit pour ce stockage.

5.2 Structure Plateau

```c
typedef struct {
    Pion plateau[TAILLE_PLATEAU][TAILLE_PLATEAU];
    int prises_joueur_X;
    int prises_joueur_O;
} Plateau;
```

Rôle : représenter l'**ensemble** de l'état du jeu.
Justification :

- Le tableau bidimensionnel plateau[19][19] (ou [TAILLE_PLATEAU][TAILLE_PLATEAU]) stocke tous les pions sur le jeu.
- Les compteurs prises_joueur_X et prises_joueur_O mémorisent le nombre de pions adverses capturés par chaque joueur (pour la victoire par prises).
- Cette structure permet de manipuler l'état du jeu (pions + compteurs) de façon groupée et homogène.

6. déclaration des paramètres, explications des traitements et rôle des variables

Nous listons ci-dessous les **sous-programmes** exclus de l'affichage. Pour chacun, nous précisons :

1. **La déclaration (signature)**,

2. **Les paramètres** (nom, type, rôle),
3. **Le traitement effectué** (description courte),
4. **Les variables locales**(si présentes) et leur utilité.

6.1 position_valide(ligne, colonne)

```
bool position_valide(int ligne, int colonne) {
```

Paramètres :

- ligne (int) : index de la ligne à tester.
- colonne (int) : index de la colonne à tester.

Traitement :

- Vérifie si (ligne, colonne) appartient à la plage [0..18].
- Retourne true si c'est le cas, false sinon.

Variables Locales :

- Aucune variable locale particulière (on utilise directement ligne et colonne).

6.2 initialiser_plateau(p)

```
void initialiser_plateau(Plateau *p)
```

Paramètres :

- p (pointeur vers Plateau) : la structure contenant le plateau et les compteurs de prises.

Traitement :

- Parcourt le tableau plateau[19][19] et place '.' dans chaque case (toutes les positions sont vides au départ).
- Réinitialise les compteurs de prises (prises_joueur_X et prises_joueur_O) à 0.

Variables Locales :

- i, j (entiers) : indices de boucle pour parcourir les lignes et colonnes.

6.3 placer_pion(p, ligne, colonne, symbole)

```
bool placer_pion(Plateau *p, int ligne, int colonne, char symbole)
```

Paramètres :

- p (pointeur vers Plateau) : le plateau.
- ligne, colonne (int) : position où poser le pion.
- symbole (char) : 'X' ou 'O'.

Traitement :

- Vérifie d'abord si (ligne, colonne) est **valide** (dans le plateau) et si la case est **vide** ('.').
- Si oui, place symbole dans la case ; sinon, retourne false pour indiquer un placement invalide.

Variables Locales :

- Aucune variable Locale, hormis l'utilisation du test position_valide(...).

6.4 verifier_alignement(p, ligne, colonne, symbole)

```
bool verifier_alignement(const Plateau *p, int ligne, int colonne, char symbole)
```

Paramètres :

- p (pointeur constant vers Plateau) : le plateau (non modifié ici).
- ligne, colonne (int) : la dernière position jouée.
- symbole (char) : 'X' ou 'O' (le pion joué).

Traitement :

- Vérifie, dans 4 directions (horizontal, vertical, diagonales « \ » et « / »), s'il existe un alignement d'au moins 5 pions identiques.
- Calcule pour chaque direction la longueur de la "suite" de pions en partant dans un sens, puis dans le sens opposé (total = suite + pion joué + suite).
- Retourne true si compteur >= 5, false sinon.

Variables Locales :

- directions[4][2] : tableau contenant les vecteurs de direction (ex. (0,1), (1,0), (1,1), (1,-1)).
- d : entier, indice pour parcourir le tableau de directions.
- compteur : entier, compte le nombre de pions consécutifs.
- x, y, dx, dy : entiers permettant de se déplacer le long d'une direction.

6.5 verifier_prise(p, ligne, colonne, symbole)

```
void verifier_prise(Plateau *p, int ligne, int colonne, char symbole)
```

Paramètres :

- p (pointeur vers Plateau) : le plateau à modifier si des pions sont capturés.
- ligne, colonne (int) : la position du dernier pion joué.
- symbole (char) : 'X' ou 'O' (joueur qui vient de jouer).

Traitement :

- Parcourt les 8 directions (haut, bas, gauche, droite, et les 4 diagonales).

- Cherche le motif (joueur) (adversaire) (adversaire) (joueur).
- Si trouvé, remplace les deux pions adverses par '.' et incrémente le compteur de prises du joueur (prises_joueur_X ou prises_joueur_O).
- Peut réaliser cette détection dans plusieurs directions pour capturer plusieurs paires.

Variables Locales :

- adversaire (char) : déterminé en fonction du symbole.
- directions[8][2] : liste des directions explorées.
- x1, y1, x2, y2, x3, y3 : positions intermédiaires pour vérifier le motif.
- i : compteur de boucle pour parcourir les 8 directions.

6.6 tour_de_jeu(plateau, joueur)

```
int tour_de_jeu(Plateau *plateau, char joueur)
```

Paramètres :

- plateau (pointeur vers Plateau) : l'état du jeu.
- joueur (char) : 'X' ou 'O' (indique quel joueur doit jouer).

Traitement :

1. (Optionnel, mais souvent fait) Affiche l'état du plateau.
2. Demande au joueur de saisir (ligne, colonne). Si (0,0), c'est un abandon ⇒ retourne -1.
3. Convertit (ligne, colonne) de [1..19] à [0..18].
4. Valide et place le pion avec placer_pion(...). Si invalide, renvoie 0 (coup raté, on recommence).
5. Vérifie la capture (verifier_prise) et l'alignement (verifier_alignement).
6. Si la capture fait atteindre 10 pions capturés pour le joueur, ou si l'alignement ≥ 5 est détecté, retourne 1 (victoire).
7. Sinon, retourne 0 (la partie continue).

Variables Locales :

- ligne, colonne : récupérés via la saisie utilisateur.
- ch :permet de vider le buffer

6.7 est_plein(const Plateau *p)

```
bool est_plein(const Plateau *p) {
```

Paramètres :

- plateau (pointeur vers Plateau) : l'état du jeu.

Traitement :

retourne true si toutes les cases du plateau sont remplies

6.8 Fonction principale (main) : déroulement du jeu

1- Déclaration du plateau et initialisation :

```
Plateau plateau;
initialiser_plateau(&plateau);
```

On crée une variable locale plateau de type Plateau.
La fonction initialiser_plateau est appelée pour remplir toutes les cases du tableau avec '.' (vide) et réinitialiser les compteurs de prises (prises_joueur_X, prises_joueur_O) à 0.
Ainsi, le plateau est prêt à accueillir le début de la partie.

2-Choix du joueur qui commence :

```
char joueur = 'O';
```

Ici, on décide que le joueur « O » débutera. (On pourrait choisir « X » selon les préférences, mais dans ce code c'est « O ».)

3-Variable de contrôle de fin de partie :

```
bool partieFinie = false;
```

- Un booléen qui indique si la partie est terminée. Il est initialisé à false : la partie n'est pas finie au lancement.

4-Boucle de jeu :

```
while (!partieFinie) {
    int resultat = tour_de_jeu(&plateau, joueur);
    ...
}
```

Tant que partieFinie est faux, on continue à demander au joueur courant de jouer.
On appelle la fonction tour_de_jeu, qui effectue :

- L'affichage du plateau,
- La saisie et le placement d'un pion,
- La vérification d'éventuelles captures,
- La vérification d'alignements ou d'une victoire par 10 prises.

tour_de_jeu renvoie un **entier** :

- -1 si le joueur abandonne,
- 1 s'il gagne,

- 0 si le coup s'est déroulé normalement et la partie continue.

5-Gestion du retour de tour_de_jeu :

```
        if (resultat == -1) {            // Abandon···
        else if (resultat == 1) {            // Le joueur courant gagne···
        else {
            // Si personne n'a gagné et que le plateau est plein => match nul
            if (est_plein(&plateau)) {···
            else { // Sinon, on continue le jeu en changeant de joueur···
        }
    }
}
```

- **Cas -1 (abandon)** :

```
printf("Joueur %c abandonne! ", joueur);
printf("Le joueur %c est donc vainqueur!\n", (joueur == 'X') ? 'O' : 'X');
partieFinie = true;
```

Si le joueur courant abandonne, on affiche un message et on déclare immédiatement l'autre joueur vainqueur, puis on met partieFinie à true pour sortir de la boucle.

- **Cas 1 (victoire)** :

```
afficher_plateau(&plateau);
printf("Joueur %c gagne la partie!\n", joueur);
partieFinie = true;
```

Le joueur qui vient de jouer a rempli une des conditions de victoire (alignement de 5 pions ou 10 captures). On réaffiche le plateau, on annonce le gagnant et on met fin à la partie.

- **Cas 0 (match nul ou partie continue)** :

si le plateau est plein:

```
if (est_plein(&plateau)) {
    afficher_plateau(&plateau);
    printf("Match nul! Le plateau est plein et aucun joueur n'a pu remplir les conditions de victoire.\n");
    partieFinie = true;
```

on affiche le plateau et est message de match nul .
La boucle prend true la partie est finie sans vainqueur
sinon:

```
joueur = (joueur == 'X') ? 'O' : 'X';
```

Aucune victoire ni abandon. On alterne simplement le joueur courant (de X à O ou de O à X) pour le tour suivant.

6-Fin du programme :

```
return 0;
```

Lorsque la boucle se termine (partieFinie devient true), le programme atteint la fin de la fonction main et retourne 0 pour indiquer que tout s'est bien déroulé.

7. Algorithme des sous-programme pour la prise des pions

```
action verifier_prise(p , ligne , colonne , symbole )
                    données : p : (pointeur vers Plateau)
                              ligne : entier
                              colonne :entier
                              symbole:chaine de caractère représentant le joueur ('O' ou 'X')

                    Variables locales :
                              adversaire       : chaine de caractere ('O' si symbole == 'X', sinon 'X')
                              directions[8][2] : tableau des 8 directions à tester
                              x1,y1,x2,y2,x3,y3 : entier (indices pour repérer les pions)

                    // 1. Déterminer l'adversaire
                    si symbole = 'X' alors
                         adversaire ← 'O'
                    sinon
                         adversaire ← 'X'
                    finsi

                    // 2. Définir les 8 directions à explorer
                    directions ← { (1, 0), (-1, 0), (0, 1), (0, -1),(1, 1), (1, -1), (-1, 1), (-1, -1) }

                   // 3. Pour chaque direction, vérifier le motif
                    pour i  de 0 à 7 faire
                         dx ← directions[i][0]
                         dy ← directions[i][1]

                              x1 ← ligne + dx
                              y1 ← colonne + dy
                              x2 ← x1 + dx
                              y2 ← y1 + dy
                              x3 ← x2 + dx
                              y3 ← y2 + dy

                              si position_valide(x1, y1) ET position_valide(x2, y2) ET position_valide(x3, y3) alors

                                        // verifier le motif (joueur, adversaire, adversaire, joueur)
                                        si (p->plateau[x1][y1].symbole = adversaire) ET (p->plateau[x2][y2].symbole = adversaire) ET (p->plateau[x3][y3].symbole = symbole)
alors

                                                  // 4. Capturer les deux pions adverses
                                                  p->plateau[x1][y1].symbole ← '.'
                                                  p->plateau[x2][y2].symbole ← '.'

                                                                            // 5. Incrémenter le compteur de prises
                                                                            si symbole = 'X' alors
                                                                                  p->prises_joueur_X ← p->prises_joueur_X + 2
                                                                            sinon
                                                                                  p->prises_joueur_O ← p->prises_joueur_O + 2
                                                                            finsi
                                                            finsi
                                        finsi
                    finpour

fin action
```

L'action verifier_prise sert à vérifier, immédiatement après qu'un pion ait été posé par le joueur (symbole), s'il est possible de capturer deux pions adverses dans l'une des huit directions autour du pion joué. Concrètement, on commence par déterminer qui est l'adversaire : si le joueur qui vient de jouer est 'X', l'adversaire est 'O', et inversement. Ensuite, on définit un tableau de huit vecteurs (dx, dy) qui correspondent aux directions haut, bas, gauche, droite et aux quatre diagonales. Pour chaque direction, on calcule trois positions successives (x1, y1, puis x2, y2, puis x3, y3) en avançant pas à pas dans cette direction à partir de la case du pion posé. On teste d'abord que ces trois positions restent valides (c'est-à-dire à l'intérieur du plateau) grâce à la fonction position_valide. Si elles sont valides, on regarde alors si elles forment le motif « (pion_joueur) (adversaire) (adversaire) (pion_joueur) », c'est-à-dire : la première case contient l'adversaire, la deuxième contient aussi l'adversaire, et la troisième case contient à nouveau le pion du joueur actuel. Si le motif est trouvé, on « capture » immédiatement les deux pions adverses, en remplaçant leurs symboles par '.' dans le tableau, et on incrémente le compteur de prises (prises_joueur_X ou prises_joueur_O) de deux. De cette manière, la procédure verifier_prise gère toutes les captures possibles découlant du dernier coup joué, et met à jour l'état du plateau en conséquence.

8. Avancement du projet et problèmes rencontrés

Dans cette section, nous détaillons davantage :

1. Étapes de développement

- **Étape 1 :** Mise en place de la structure de données
    1. Création des structures Pion et Plateau.
    2. Choix de constants (TAILLE_PLATEAU = 19).
    3. Simplification de la représentation d'une case par un unique char symbole.
- **Étape 2 :** Écriture des fonctions basiques
    1. position_valide(...) pour gérer les bornes du plateau.
    2. initialiser_plateau(...) pour remplir toutes les cases de '.' et remettre les compteurs de prises à zéro.
- **Étape 3 :** Gestion de la saisie utilisateur
    1. Mise en place d'une boucle pour récupérer (ligne, colonne) selon le format scanf("%d,%d", &ligne, &colonne).
    2. Vérification du format : si invalide, on signale l'erreur et on continue (ou on vide le buffer).
    3. Conversion (1..19) en (0..18) dans le code pour placer le pion correctement dans le tableau.
- **Étape 4 :** Vérification des coups et placement
    1. Écriture de placer_pion(...), qui refuse un coup si la case est déjà occupée ou si (ligne, colonne) est hors-limite.
- **Étape 5 :** Détection de la victoire par alignement
    1. Création de verifier_alignement(...) qui teste 4 directions autour du pion joué et calcule la longueur de l'alignement.
- **Étape 6 :** Implémentation de la capture
    1. Introduction de verifier_prise(...), gestion des 8 directions, vérification du motif (joueur)(adversaire)(adversaire)(joueur).
    2. Test de la validité des positions autour du pion.
    3. Incrémentation des compteurs de prises pour le joueur qui vient de jouer.
- **Étape 7 :** Condition de victoire par 10 prises
    1. Vérification à chaque coup si prises_joueur_X >= 10 ou prises_joueur_O >= 10.
    2. Si oui, fin de partie et affichage du vainqueur.
- **Étape 8 :** Gestion de l'abandon
    1. Si (ligne, colonne) = (0,0), on décrète l'abandon du joueur courant, l'autre est vainqueur.
- **Étape 9 :** Finalisation et tests
    1. Tests avec différents scénarios :
        - Aligner 5 pions pour X.

        - Aligner plus de 5 pions (ex. 6 ou 7 à la suite).
        - Capturer par multiples paires en un coup.
        - Tester l'abandon.
    2. Vérification qu'un joueur qui atteint 10 pions capturés l'emporte effectivement.

2. Problèmes et difficultés rencontrés

- **Indices et conversions** : La différence entre les indices internes ([0..18]) et la saisie utilisateur ([1..19]) a nécessité une attention particulière pour éviter les placements décalés ou hors-limite.
- **Multiples captures simultanées** : Le jeu Penté autorise de capturer plusieurs paires d'affilée dans un même coup, potentiellement dans plusieurs directions. Il a fallu s'assurer que la fonction verifier_prise(...) parcourt **toutes** les directions et effectue la suppression des pions adverses dès qu'un motif valide est détecté, sans en "oublier".
- **Gestion des entrées invalides** :
    1. Utiliser scanf("%d,%d", &ligne, &colonne) peut entraîner des soucis si l'utilisateur ne respecte pas la virgule, ou entre autre chose qu'un nombre.
    2. Nous avons résolu cela en affichant un message d'erreur et en vidant le buffer, puis le joueur peut retenter un coup.
- **Conflit de captures vs. alignement** : L'ordre d'exécution (placer → capturer → vérifier alignement) doit être strict. Par exemple, on ne veut pas compter un alignement qui inclut des pions adverses destinés à être capturés dans le même coup. La séquence finale est donc :
    1. Placer le pion,
    2. Capturer les pions adverses,
    3. Vérifier l'alignement avec la configuration finale.

3. Tests effectués

- **Cas de victoire de X par alignement**.
- **Cas de victoire de O par alignement**
- **Cas de victoire de X par capture**
- **Cas de victoire de X par capture**
- **Abandon** : entrée (0,0) pour voir si l'autre joueur est bien déclaré vainqueur.
- **cas de match Nul :** toutes les cases sont remplis aucune condition de victoire remplie

Globalement, chaque étape a été validée par des tests simples, puis combinée avec des scénarios plus complexes. Les erreurs détectées ont principalement concerné la validation des positions, la gestion de la saisie mais parfois de la mise en place d'un fichier teste qui marche.

9. Conclusion

En conclusion, le jeu Penté développé en C **répond aux exigences** du cahier des charges :

- Nous **modélisons le jeu** à l'aide de structures simples (Plateau, Pion).
- Le programme **gère** :
    1. L'affichage du plateau,
    2. La saisie des coups dans le format imposé,
    3. La détection de **l'alignement d'au moins 5 pions**,
    4. La **capture** de pions par paires ((joueur)(adversaire)(adversaire)(joueur)),
    5. Les **compteurs** de prises (victoire possible à 10 captures),
    6. L'**abandon** (coup 0,0).
- Plusieurs **tests** (simples et avancés) ont été menés pour vérifier la validité de ces fonctionnalités. Les retours confirment que le programme se comporte correctement dans les scénarios exigés (y compris ceux fournis comme pente1.txt et pente2.txt).

9.1 Perspectives et améliorations possibles

1. **Interface graphique** :
    - Une version **graphiquement plus conviviale** pourrait être développé en utilisant du code C# et en liant les calculs avec une DLL(direct link library) .
2. **IA / Mode Solo** :
    - Ajouter une **intelligence artificielle** capable de jouer contre l'utilisateur, avec différentes stratégies (ex. algorithmes minimax, heuristiques de placement, etc.).
3. **Sauvegarde / Chargement** :
    - Permettre de **sauvegarder** l'état du plateau pour reprendre la partie plus tard.