Algorithmique & Programmation

Saute-mouton
Tic-Tac-Toe

yann.secq@univ-lille.fr

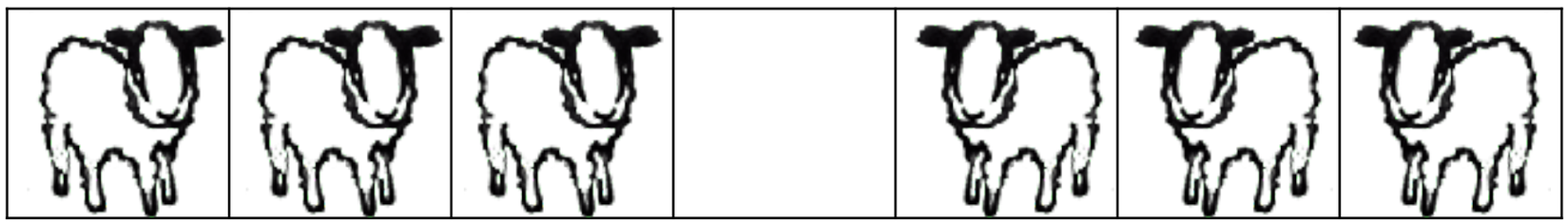
ABIDI Sofiene, ALMEIDA COCO Amadeu, BONEVA Iovka, CASTILLON Antoine, DELECROIX Fabien, LEPRETRE Éric, Timothé ROUZÉ, SANTANA MAIA Deise, SECQ Yann

Saison 2023-2024 / R1.01+R1.02 :
Initiation au développement

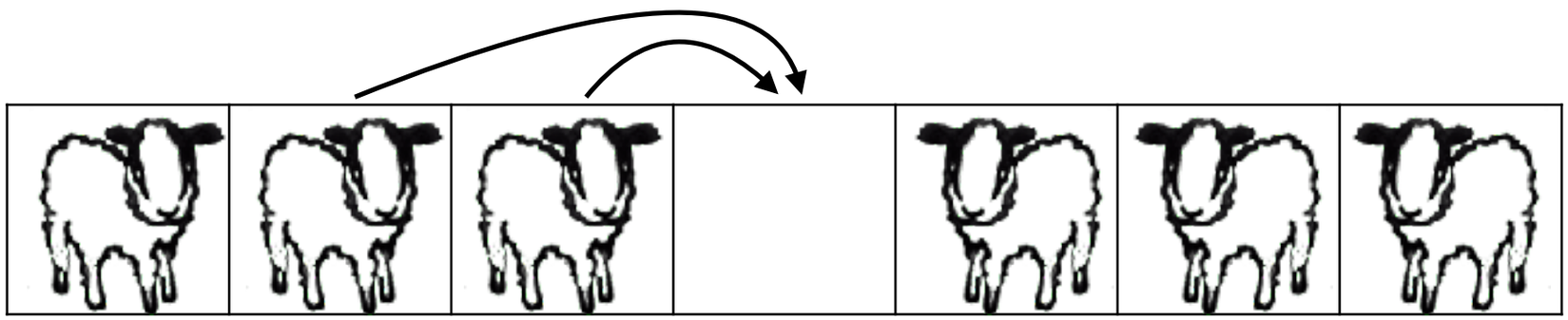
Cas d'étude

- Décomposer un problème complexe
    - Identifier les données puis les traitements
        - Choisir une structure de données judicieuse
        - Développer et … recommencer :)
- Retour sur **Saute-mouton** puis Tic-Tac-Toe

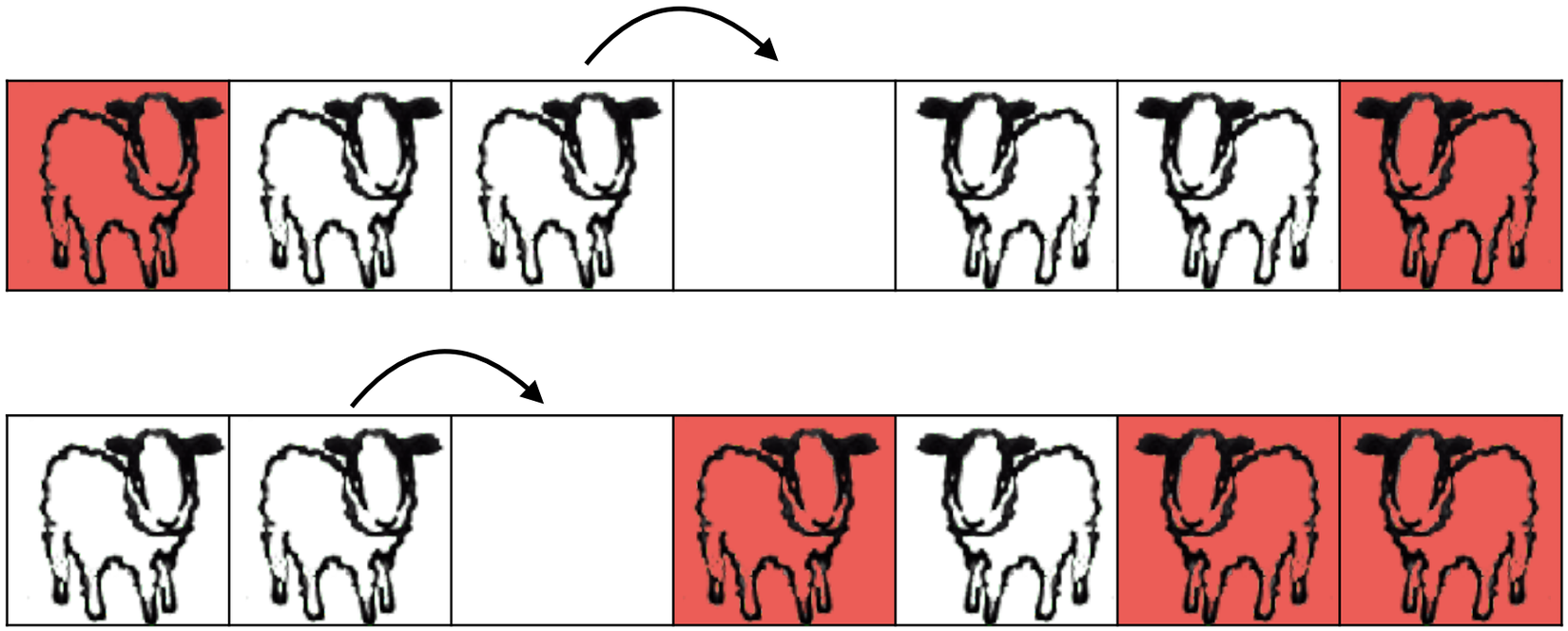
Jeu de Saute-mouton



Objectif : que les troupeaux se croisent, mais un mouton ne peut avancer que si il a une case libre devant lui ou en sautant au dessus d’un mouton se trouvant devant lui.



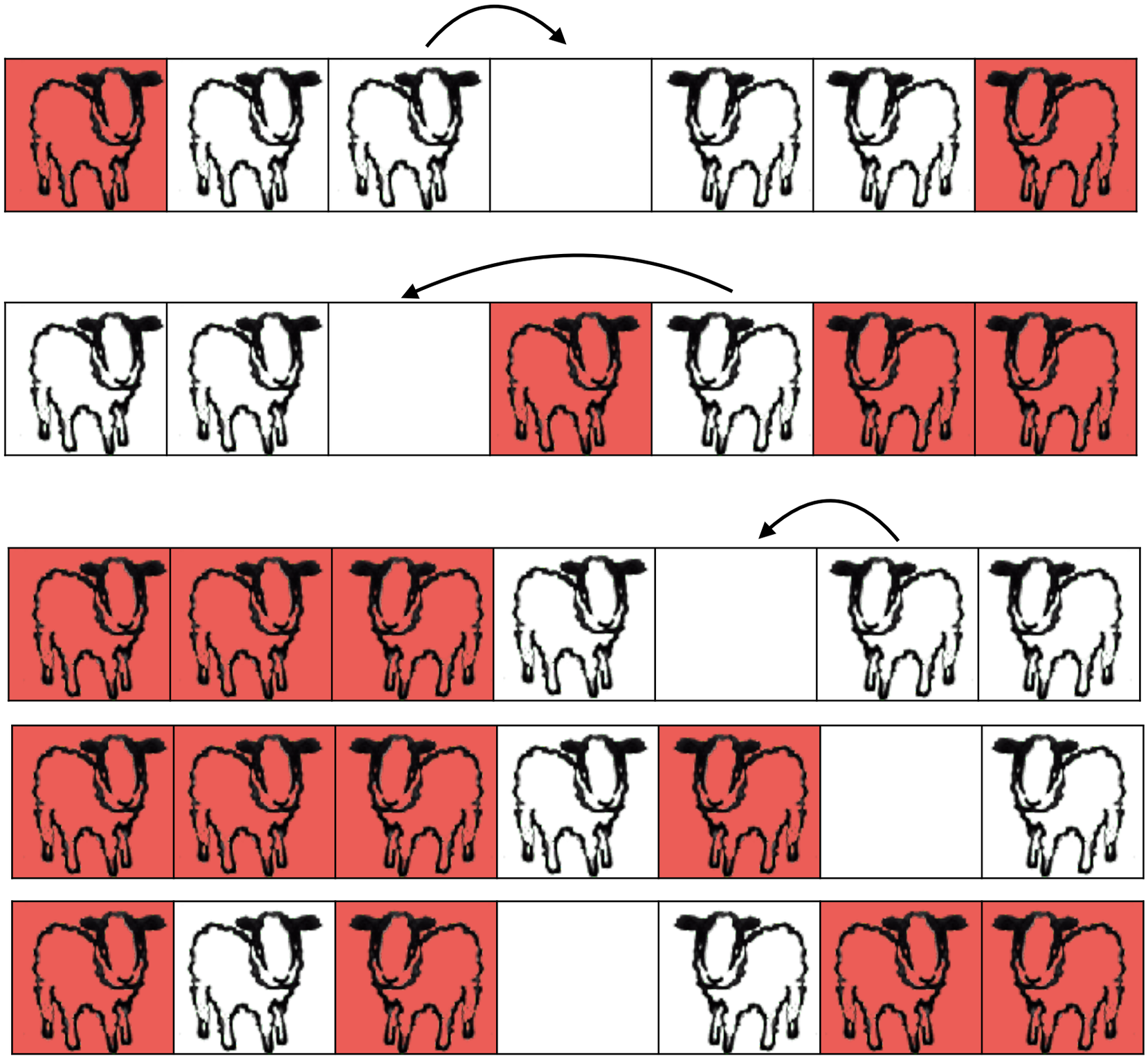
Exemple de partie



Hum, c’est déjà perdu …

BLOQUÉ !

Autre exemple de partie



Jeu de Saute-mouton

- **Traitements complexes**
  - Déterminer si un mouvement est possible (puis l'effectuer)
  - Détecter les situations de blocage
  - Déterminer si le joueur a gagné (facile)
- **Informations/données et structure de données**
  - L'orientation du mouton « droite » ou « gauche »
  - Un tableau de « moutons »

Jeu de Saute-mouton

- **Question de l'orientation du mouton :**
  - A quoi correspond l'orientation ?
  - Mouton « droite/gauche », se déplace vers la droite/gauche
- **Avec un tableau ?**

Jeu de Saute-mouton

- **Question de l'orientation du mouton :**
  - A quoi correspond l'orientation ?
  - Mouton « droite/gauche », se déplace vers la droite/gauche
    - **Avec un tableau, cela correspond à un décalage +/- 1**
- Coder les moutons par rapport au décalage à appliquer pour calculer plus facilement les déplacements possibles
- Comment se code « naturellement » la case vide ?

Structure de données

1	1	1	0	-1	-1	-1

```
final int DROITE = +1;
final int GAUCHE = -1;
final int VIDE   =  0;

int[] newPrairie(int nbMoutons) {
  final int TAILLE = nbMoutons*2 + 1;
  int[] prairie = new int[TAILLE];
  for (int idx=0; idx<TAILLE/2; idx++) {
    prairie[idx] = DROITE;
  }
  prairie[TAILLE/2] = VIDE;
  for (int idx = TAILLE/2+1; idx < TAILLE; idx++) {
    prairie[idx] = GAUCHE;
  }
  return prairie;
}
```

```
String toString(int[] prairie) {
  String res = "";
  String indices = "";
  for (int idx=0; idx<length(prairie); idx++) {
    if (prairie[idx] == DROITE) {
      res = res + ">";
    } else if (prairie[idx] == GAUCHE) {
      res = res + "<";
    } else {
      res = res + ".";
    }
    indices = indices + (idx+1);
  }
  return res + "\n" + indices;
}
```

Potentiellement un type `Prairie` avec le tableau d'entiers et l'indice de la case vide …

Jeu de Saute-mouton

- **Comment déterminer si la joueuse a gagné ?**
  - Prairie finale : `"<<<.>>>"`
  - On pourrait tester l'égalité de deux tableaux …
  - Ou entre chaînes via les `toString`(…) …
  - **Solution plus courte avec le codage choisi ?**

Jeu de Saute-mouton

- **Comment déterminer si la joueuse a gagné ?**
  - Prairie finale : `"<<<.>>>"`
  - On pourrait tester l'égalité de deux tableaux …
  - Ou entre chaînes via les `toString`(…) …
  - Solution plus courte avec le codage choisi ?
- **Rappel : Mouton droite = -1**

Jeu de Saute-mouton

- Déterminer si le joueur a gagné ?
  - Prairie finale = "<<<.>>>"
  - Rappel : Mouton gauche = -1

```
boolean gagne(int[] prairie) {
  int sum = 0;
  for (int idx=0; idx<=length(prairie)/2; idx++) {
    sum = sum + prairie[idx];
  }
  return (sum == -length(prairie)/2);
}
```

Jeu de Saute-mouton

- **Déterminer si un mouvement est possible et effectuer ce mouvement**

  - Contrôle de saisie : dans l'idéal seulement un mouton pouvant se déplacer …

  - Vérifier lors de la saisie ? Puis re-déterminer si le mouton doit avancer ou sauter ?

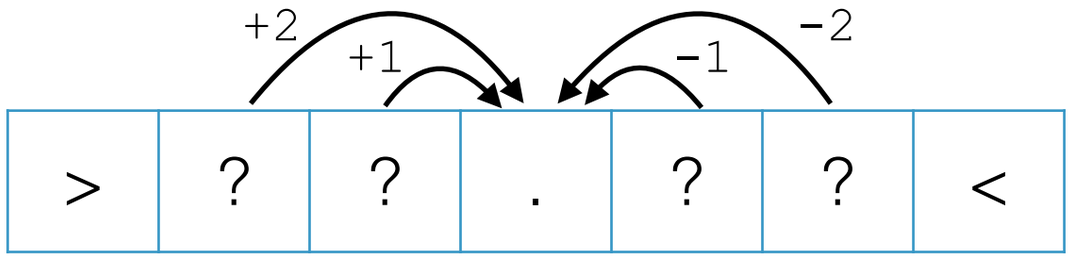
  - **Comment ne pas faire deux fois ce calcul** : mouvement possible et effectuer le mouvement ?

Jeu de Saute-mouton

- **Piste : calculer les mouvement possibles à chaque tour !**
- Puis, vérifier que le mouvement proposé appartient aux mouvements possibles
- **Comment représenter un mouvement ?**
  - Un indice de départ (qui doit être un mouton pouvant se déplacer) + un indice d'arrivée (qui doit être la case vide)
  - Un tableau de deux entiers : `{idxMouton, idxVide}` **(type?)**
  - Tous les mouvements : un tableau d'entiers à 2 dimensions : `{{idxM1, idxV}, {idxM2, idxV},…}`

Jeu de Saute-mouton

- Les mouvements ne peuvent se produire qu'autour de la case vide … tester les 4 cas possibles !



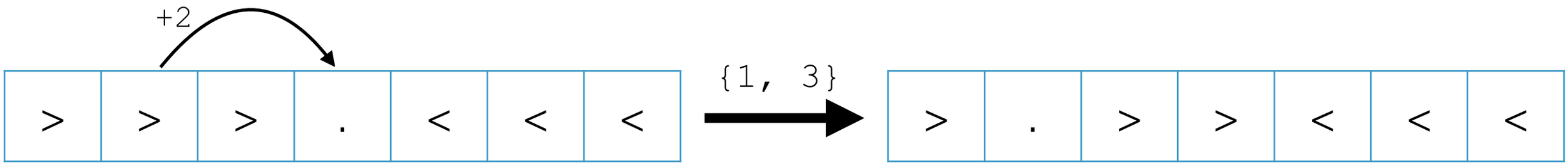
Toujours vérifier la validité des indices calculés avant d'accéder au tableau !

```
int deplacements(int[] p, int idxV, int[][] dep) {
  int nbDep = 0;
  if (idxV-1 >= 0         && p[idxV-1] == DROITE) { dep[nbDep] = new int[]{idxV-1, idxV}; nbDep++; }
  if (idxV-2 >= 0         && p[idxV-2] == DROITE) { dep[nbDep] = new int[]{idxV-2, idxV}; nbDep++; }
  if (idxV+1 < length(p) && p[idxV+1] == GAUCHE) { dep[nbDep] = new int[]{idxV+1, idxV}; nbDep++; }
  if (idxV+2 < length(p) && p[idxV+2] == GAUCHE) { dep[nbDep] = new int[]{idxV+2, idxV}; nbDep++; }
  return nbDep;
}
```

A quoi correspond le type de retour ?

Jeu de Saute-mouton

- Effectuer un déplacement revient à appliquer la copie du mouton à l'indice de départ et mettre du vide là où il était avant son déplacement




```
void deplacer(int[] prairie, int[] deplacement) {
  prairie[deplacement[1]] = prairie[deplacement[0]];
  prairie[deplacement[0]] = VIDE;
}
```

Jeu de Saute-mouton

- Contrôler la saisie revient à vérifier que l'indice choisit par l'utilisa·teur·trice est présent dans la liste des déplacements possibles

- Parcourir le tableau des déplacements possibles en recherchant l'indice saisit dans la première coordonnée de chaque déplacement

On quitte violemment la boucle et la fonction dès que l'on a trouvé ! (`while` normalement …)

```
int[] chercher(int[][] deps, int indice) {
  for (int idx=0; idx<length(deps, 1); idx++) {
    if (deps[idx][0] == indice-1) {
      return deps[idx];
    }
  }
  return new int[]{};
}
```

Jeu de Saute-mouton

- Contrôler la saisie revient à vérifier que l'indice choisit par l'utilisa·teur·trice est présent dans la liste des déplacements possibles

```
int[] readIndice(int[] prairie, int[][] deplacementsPossibles) {
  int[] coup;
  do {
    print("Entrez la position du mouton à déplacer : ");
    coup = chercher(deplacementsPossibles, readInt());
  } while (length(coup) == 0);
  return coup;
}
```

Algorithme principal

```
int[][] newDeplacementsVides(int idxVide) {
  return new int[][]{{-1, idxVide}, {-1, idxVide},
                     {-1, idxVide}, {-1, idxVide}};
}
void algorithm() {
  int[] prairie = newPrairie(3);
  int idxVide   = length(prairie)/2;
  int[][] dep   = newDeplacementsVides(idxVide);
  while (deplacements(prairie, idxVide, dep) > 0) {
    println(toString(prairie));
    int[] coup = readIndice(prairie, dep);
    deplacer(prairie, coup);
    idxVide = coup[0];
    dep = newDeplacementsVides(idxVide);
  }
  if (gagne(prairie)) {
    println("Bravo, vous avez réussi !");
  } else {
    println("Dommage, vous êtes bloqué ...");
  }
}
```

Cas d'étude

- Décomposer un problème complexe
  - Identifier les données puis les traitements
    - Choisir une structure de données judicieuse
    - Développer et … recommencer :)
- Retour sur Saute-mouton puis **Tic-Tac-Toe**

Tic-Tac-Toe

1. **Quelles informations ?**
2. **Quels traitements ?**
3. **Quels types et structure de données ?**
4. **Ébauche d'algorithme principal**

```
X . .
. X .
O . .
```

Tic-Tac-Toe

1. **Quelles informations ?**
   1. **`Symbole` -> type (enum)**
   2. **`Joueur` -> type**
   3. **`Grille` -> tableau de Symboles**
2. **Quels traitements ?**
3. **Quels types et structure de données ?**
4. **Ébauche d'algorithme principal**

```
X . .
. X .
O . .
```

Tic-Tac-Toe

1. **Quelles informations ?**
   1. **Symboles -> enum**
   2. **Joueurs -> type**
   3. **Grille -> tableau de symboles**
2. **Quels traitements ?**
3. **Quels types et structure de données ?**
4. **Ébauche d'algorithme principal**

X . .

. X .

O . .

```
enum Symbole {
  X, O, V;
}
```

```
class Joueur {
  Symbole symbole;
  int victoire = 0
}
```

```
Symbole[][] grille = new
Symbole[][] {{V,V, V},
{V,V, V}, {V,V, V}};
```

Est-ce vraiment utile ?

Tic-Tac-Toe

1. **Quelles informations ?**
2. **Quels traitements ?**
    1. **Afficher la grille**
    2. **Grille remplie ?**
    3. **Alignement présent ?**
    4. **Changer de joueur**
3. **Quels types et structure de données ?**
4. **Ébauche d'algorithme principal**

```
X . .
. X .
O . .
```

Tic-Tac-Toe

```
X . .

. X .

O . .
```

1. **Quelles informations ?**
2. **Quels traitements ?**
    1. **Afficher la grille**
    2. **Grille remplie ?**
    3. **Alignement présent ?**
    4. **Changer de joueur**
3. **Quels types et structure de données ?**
4. **Ébauche d'algorithme principal**

```
String toString(Symbole s)
String toString(Symbole[][] s)
boolean remplie(Symbole[][] s)
boolean alignement(Symbole[][] s)
Symbole changer(Symbole c)
```

Tic-Tac-Toe

```
X . .
. X .
O . .
```

1. **Quelles informations ?**
2. **Quels traitements ?**
3. **Quels types et structure de données ?**
4. **Ébauche d'algorithme principal**

```
void algorithm() {
   Symbole[][] grille = initialiser();
   int nbTours = 0;
   Symbole joueurCourant = Symbole.O;
   while (!fini(grille, nbTours)) {
       joueurCourant = changer(joueurCourant);
       println(toString(grille));
       Coup coup = saisirCoup(grille, joueurCourant);
       appliquer(grille, coup);
       nbTours++;
   }
   afficherGagnant(joueurCourant)
}
```

Tic-Tac-Toe

```
X . .
. X .
O . .
```

1. **Quelles informations ?**
2. **Quels traitements ?**
3. **Quels types et structure de données ?**
4. **Ébauche d'algorithme principal**

```
void algorithm() {
    Symbole[][] grille = initialiser();
    int nbTours = 0;
    Symbole joueurCourant = Symbole.O;
    while (!fini(grille, nbTours)) {
        joueurCourant = changer(joueurCourant);
        println(toString(grille));
        Coup coup = saisirCoup(grille, joueurCourant);
        appliquer(grille, coup);
        nbTours++;
    }
    afficherGagnant(joueurCourant)
}
```

Qu'est-ce qu'un coup ?

Tic-Tac-Toe

```
X . .
. X .
O . .
```

1. **Quelles informations ?**
2. **Quels traitements ?**
3. **Quels types et structure de données ?**
4. **Ébauche d'algorithme principal**

```
void algorithm() {
   Symbole[][] grille = initialiser();
   int nbTours = 0;
   Symbole joueurCourant = Symbole.O;
   while (!fini(grille, nbTours)) {
       joueurCourant = changer(joueurCourant);
       println(toString(grille));
       Coup coup = saisirCoup(grille, joueurCourant);
       appliquer(grille, coup);
       nbTours++;
   }
   afficherGagnant(joueurCourant)
}
```

```
class Coup {
  int l, c;
  Symbole s;
}
```

Conclusion

- Décomposer un problème complexe
  - Identifier les informations pertinentes
    - Choisir une structure de données (judicieuse !)
    - Identifier les traitements difficiles
    - Recommencer :)
- Toujours plusieurs solutions possibles …

```
Mot secret (nombre d'erreur = 0) : **********
Veuillez entrer une lettre minuscule de l'alphabet : a
Mot secret (nombre d'erreur = 0) : ******a****
Veuillez entrer une lettre minuscule de l'alphabet : i
Mot secret (nombre d'erreur = 0) : i*****a*i**
Veuillez entrer une lettre minuscule de l'alphabet : e
Mot secret (nombre d'erreur = 1) : i*****a*i**
Veuillez entrer une lettre minuscule de l'alphabet : r
Mot secret (nombre d'erreur = 1) : i***r*a*i**
Veuillez entrer une lettre minuscule de l'alphabet : s
Mot secret (nombre d'erreur = 2) : i***r*a*i**
Veuillez entrer une lettre minuscule de l'alphabet : n
Mot secret (nombre d'erreur = 2) : in**r*a*i*n
Veuillez entrer une lettre minuscule de l'alphabet : o
Mot secret (nombre d'erreur = 2) : in*or*a*ion
Veuillez entrer une lettre minuscule de l'alphabet : f
Mot secret (nombre d'erreur = 2) : infor*a*ion
Veuillez entrer une lettre minuscule de l'alphabet : m
Mot secret (nombre d'erreur = 2) : informa*ion
Veuillez entrer une lettre minuscule de l'alphabet : t
Bravo, vous avez gagné !
```

Jeu du pendu

```
Mot secret (nombre d'erreur = 0) : **********
Veuillez entrer une lettre minuscule de l'alphabet : a
Mot secret (nombre d'erreur = 0) : a*********
Veuillez entrer une lettre minuscule de l'alphabet : e
Mot secret (nombre d'erreur = 0) : a********e
Veuillez entrer une lettre minuscule de l'alphabet : i
Mot secret (nombre d'erreur = 0) : a****i***e
Veuillez entrer une lettre minuscule de l'alphabet : o
Mot secret (nombre d'erreur = 0) : a**o*i***e
Veuillez entrer une lettre minuscule de l'alphabet : t
Mot secret (nombre d'erreur = 0) : a**o*it**e
Veuillez entrer une lettre minuscule de l'alphabet : f
Mot secret (nombre d'erreur = 1) : a**o*it**e
Veuillez entrer une lettre minuscule de l'alphabet : e
Mot secret (nombre d'erreur = 2) : a**o*it**e
Veuillez entrer une lettre minuscule de l'alphabet : e
Mot secret (nombre d'erreur = 3) : a**o*it**e
Veuillez entrer une lettre minuscule de l'alphabet : e
Mot secret (nombre d'erreur = 4) : a**o*it**e
Veuillez entrer une lettre minuscule de l'alphabet : e
Désolé, vous avez perdu ...
```

```
Mot secret (nombre d'erreur = 0) : ***********
Veuillez entrer une lettre minuscule de l'alphabet : a
Mot secret (nombre d'erreur = 0) : ******a****
Veuillez entrer une lettre minuscule de l'alphabet : i
Mot secret (nombre d'erreur = 0) : i*****a*i**
Veuillez entrer une lettre minuscule de l'alphabet : e
Mot secret (nombre d'erreur = 1) : i*****a*i**
Veuillez entrer une lettre minuscule de l'alphabet : r
Mot secret (nombre d'erreur = 1) : i***r*a*i**
Veuillez entrer une lettre minuscule de l'alphabet : s
Mot secret (nombre d'erreur = 2) : i***r*a*i**
Veuillez entrer une lettre minuscule de l'alphabet : n
Mot secret (nombre d'erreur = 2) : in**r*a*i*n
Veuillez entrer une lettre minuscule de l'alphabet : o
Mot secret (nombre d'erreur = 2) : in*or*a*ion
Veuillez entrer une lettre minuscule de l'alphabet : f
Mot secret (nombre d'erreur = 2) : infor*a*ion
Veuillez entrer une lettre minuscule de l'alphabet : m
Mot secret (nombre d'erreur = 2) : informa*ion
Veuillez entrer une lettre minuscule de l'alphabet : t
Bravo, vous avez gagné !
```

Jeu du pendu

1. **Quelles informations ?**
2. **Quels traitements ?**
3. **Quels types et structure de données ?**

```
Mot secret (nombre d'erreur = 0) : **********
Veuillez entrer une lettre minuscule de l'alphabet : a
Mot secret (nombre d'erreur = 0) : a*********
Veuillez entrer une lettre minuscule de l'alphabet : e
Mot secret (nombre d'erreur = 0) : a********e
Veuillez entrer une lettre minuscule de l'alphabet : i
Mot secret (nombre d'erreur = 0) : a****i***e
Veuillez entrer une lettre minuscule de l'alphabet : o
Mot secret (nombre d'erreur = 0) : a**o*i***e
Veuillez entrer une lettre minuscule de l'alphabet : t
Mot secret (nombre d'erreur = 0) : a**o*it**e
Veuillez entrer une lettre minuscule de l'alphabet : f
Mot secret (nombre d'erreur = 1) : a**o*it**e
Veuillez entrer une lettre minuscule de l'alphabet : e
Mot secret (nombre d'erreur = 2) : a**o*it**e
Veuillez entrer une lettre minuscule de l'alphabet : e
Mot secret (nombre d'erreur = 3) : a**o*it**e
Veuillez entrer une lettre minuscule de l'alphabet : e
Mot secret (nombre d'erreur = 4) : a**o*it**e
Veuillez entrer une lettre minuscule de l'alphabet : e
Désolé, vous avez perdu ...
```

Jeu du Pendu

- **Informations** : un mot secret, une visibilité associée à chaque lettre du mot secret

- **Traitements** (complexes)
  - Mettre à jour le mot en fonction de la lettre
  - Déterminer si le joueur a gagné

- **Types et structure de données**
  - **`Lettre`** = caractère + visible ou pas (nouveau type)
  - **Mot** = un tableau de lettres

Jeu du Pendu

- **Informations** : un mot secret, une visibilité associée à chaque lettre du mot secret

- **Traitements** (complexes)
  - Mettre à jour le mot en fonction de la lettre
  - Déterminer si le joueur a gagné

- **Types et structure de données**
  - **`Lettre`** = caractère + visible ou pas (nouveau type)
  - **Mot** = un tableau de lettres

Type `Lettre`

```
class Lettre {
  char caractere;
  boolean visible = false;
}
```

- **Type** : un champs pour stocker un caractère + un booléen pour indiquer si le caractère a été découvert ou pas (initialisé à `false` car aucune mettre n'est visible au début du jeu)

- **Traitements**
  - Créer une nouvelle valeur de type `Lettre`
  - Afficher une valeur de type `Lettre`
  - Créer un tableau de `Lettre` à partir d'une chaîne
  - Afficher un tableau de `Lettre`

```
Lettre newLettre(char caractere)

String toString(Lettre l)

Lettre[] convertir(String mot)

String toString(Lettre[] mot)
```

Type Lettre

```
class Lettre {
  char caractere;
  boolean visible = false;
}
```

```
Lettre newLettre(char caractere) {
  Lettre l = new Lettre();
  l.caractere = caractere;
  return l;
}

String toString(Lettre l) {
  if (l.visible) {
    return "" + l.caractere;
  }
  return "*";
}

Lettre[] convertir(String mot) {
  Lettre[] lettres = new Lettre[length(mot)];
  for (int idx=0; idx < length(mot); idx++) {
    lettres[idx] = newLettre(charAt(mot, idx));
  }
  return lettres;
}

String toString(Lettre[] mot) {
  String res = "";
  for (int idx=0; idx < length(mot,1); idx++) {
    res = res + toString(mot[idx]);
  }
  return res;
}
```

DS2 passé !
On s'autorise ce type de raccourci maintenant :)

Jeu du Pendu

- **Traitements complexes**

  - **Déterminer si le joueur a gagné**

    - Si une lettre non visible encore présente, la partie n’est pas gagnée, sinon c’est le cas

  - **Mettre à jour le mot en fonction de la lettre**

    - Rendre visible les occurrences de la lettre présente et savoir si au moins une lettre a été modifiée

Jeu du Pendu

- **Déterminer si le joueur a gagné**
  - Si une lettre non visible est encore présente, la partie n'est pas gagnée, sinon c'est le cas

DS2 passé ! On s'autorise ce type de raccourci maintenant :)

```
boolean decouvert(Lettre[] mot) {
  for (int idx=0; idx < length(mot); idx++) {
    if (!mot[idx].visible) {
      return false;
    }
  }
  return true;
}
```

Jeu du Pendu

- **Mettre à jour le mot en fonction de la lettre**
  - Rendre visible les occurrences de la lettre présente et savoir si au moins une lettre a été modifiée

```
boolean miseAJour(Lettre[] mot, char lettre) {
  boolean changement = false;
  for (int idx=0; idx < length(mot); idx++) {
    if (!mot[idx].visible && mot[idx].caractere == lettre) {
      mot[idx].visible = true;
      changement = true;
    }
  }
  return changement;
}
```

Jeu du Pendu

Algorithme principal

- Tableau de mots et choix aléatoire d'un des mots
- Conversion de la chaîne en mot (`Lettre[]`)
- Tant que mot non découvert et `nbErreurs` acceptable
  - Saisir une nouvelle lettre
  - Mettre à jour le mot secret et incrémenter le `nbErreurs` si aucune nouvelle lettre découverte
- Indiquer si la partie est gagnée ou pas

Algorithme principal

```
void algorithm() {
  final int MAX_ERREURS = 5;
  final String[] MOTS = new String[]{"algorithme", "machine",
                                     "information", "langage", "programme"};
  Lettre[] secret = convertir(motAuHasard(MOTS));
  int nbErreurs = 0;
  while (nbErreurs < MAX_ERREURS && !decouvert(secret)) {
    println("Mot secret (nb erreur = "+nbErreurs+"):"+ toString(secret));
    char lettre = readChar(); // contrôle de saisie ?
    if (!miseAJour(secret, lettre)) {
      nbErreurs++;
    }
  }
  if (nbErreurs == MAX_ERREURS) {
    println("Désolé, vous avez perdu ...");
  } else {
    println("Bravo, vous avez gagné !");
  }
}
```