Algorithmique & Programmation

Cas d'étude
Saute-mouton

yann.secq@univ-lille.fr

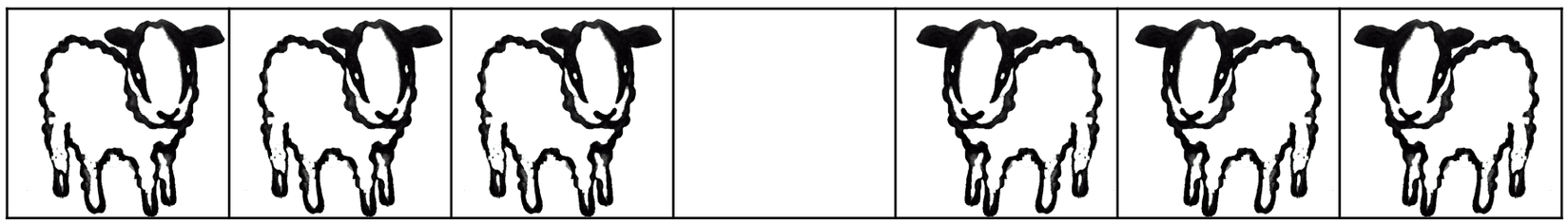
ABDELKADER Omar, BIRLOUEZ Martin, BONEVA Iovka, DELECROIX Fabien, LEQUINIOU Erwann, MARSHALL-BRETON Christopher, REKIK Yosra, SECQ Yann, SOW Younoussa, SUDHEENDRAN Megha, SU Yue

Saison 2025-2026 / R1.01+R1.02
Initiation au développement

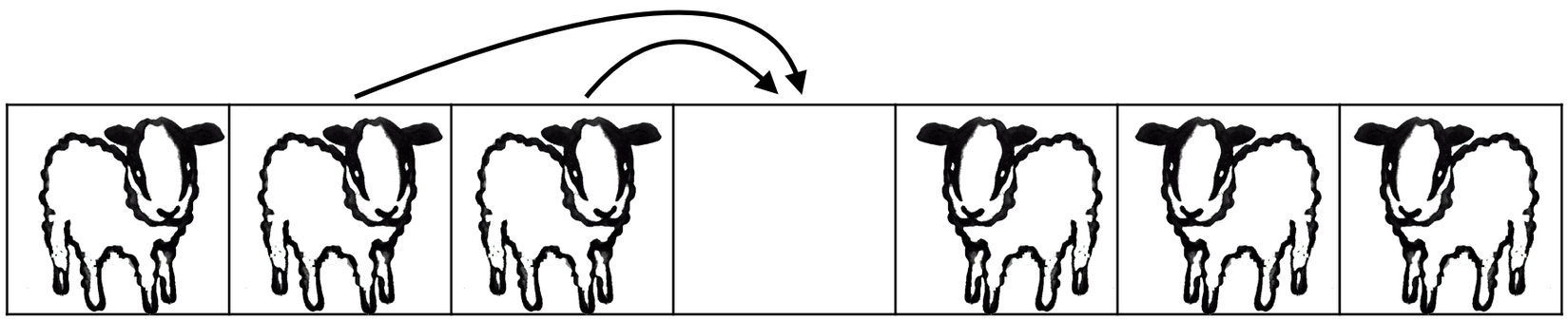
Cas d'étude

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

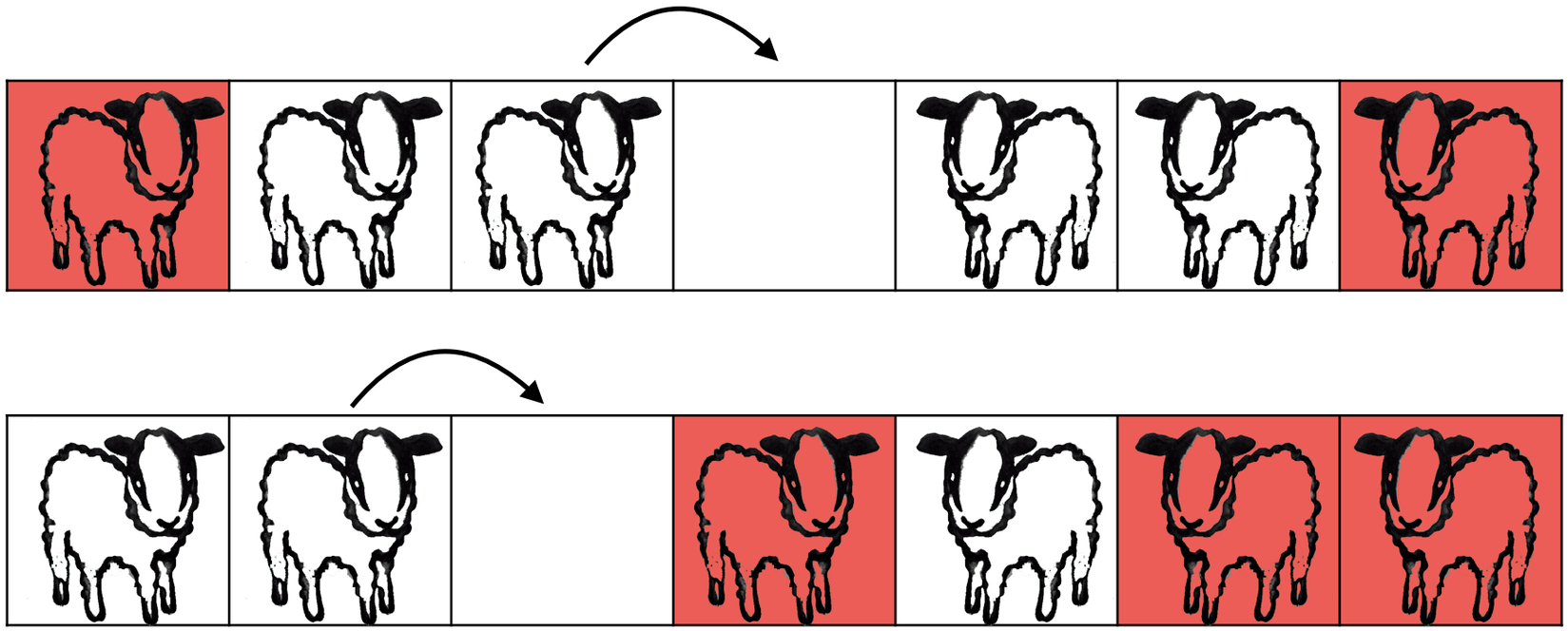
Jeu de Saute-mouton

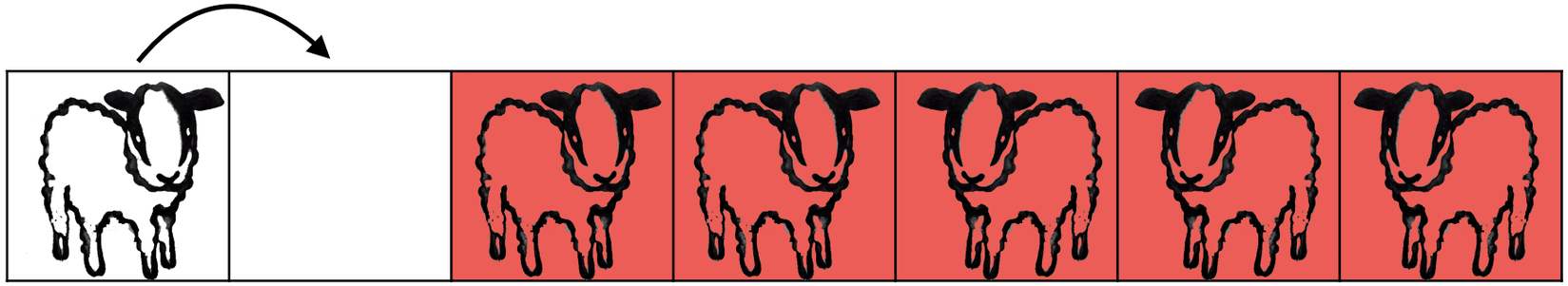


Objectif : que les troupeaux se croisent, mais un mouton ne peut avancer que si il a une case libre devant lui ou en sautant au dessus d’un mouton se trouvant devant lui.



Exemple de partie

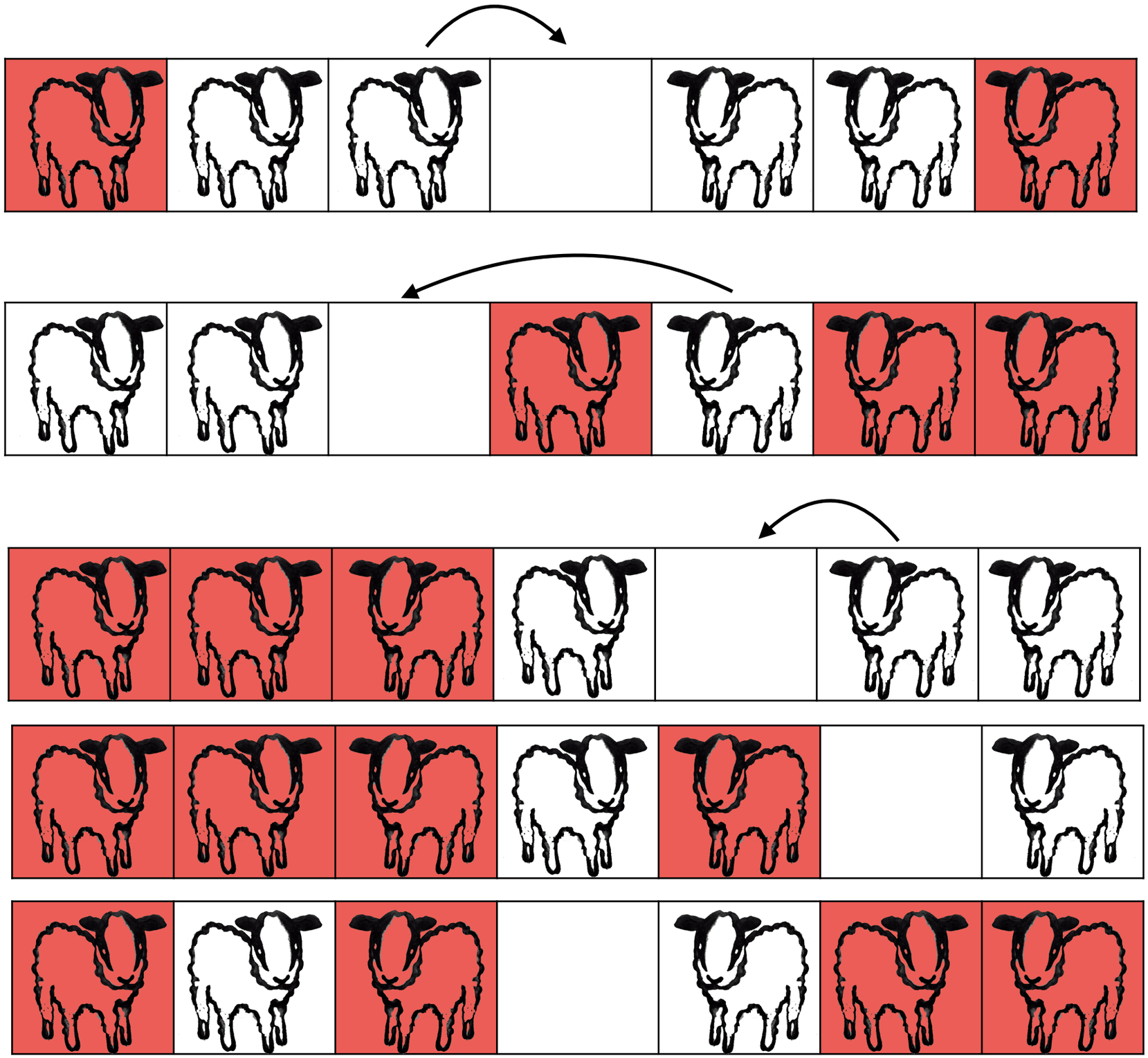




Hum, c’est déjà perdu …

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 ?

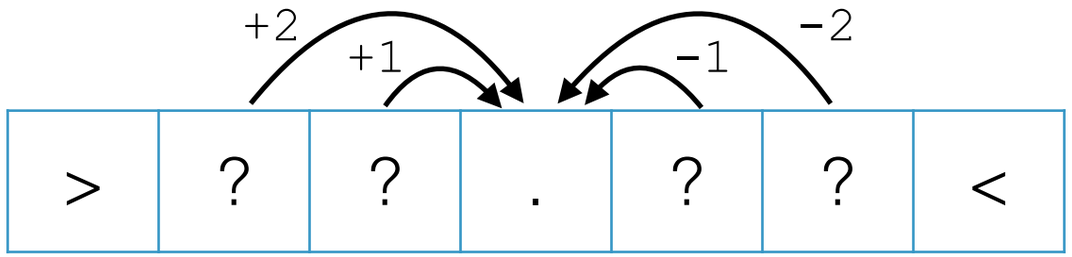
  - **Comment ne pas faire deux fois ce calcul** : mouvement possible et effectuer le mouvement ?

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 !



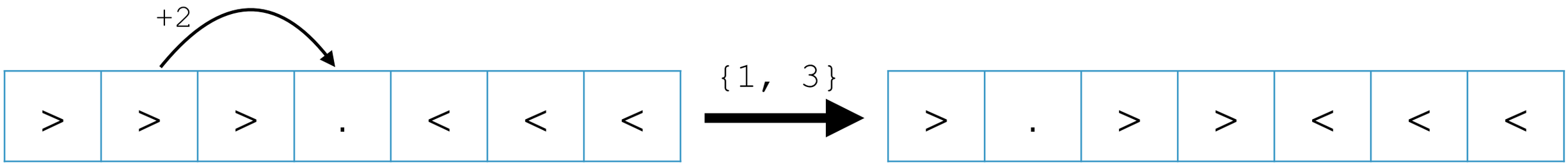
Toujours vérifier la validité des indices calculés avant d'accéder au tableau !

```
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