DAKHLI Sonia p1905223

TANG Kévin p1501263

**Rapport TP Troll**

Nous avons fait la distinction entre la construction du jeu en lui-même et l’implémentation de la stratégie prudente.

# Construction des sous-jeux

Nous avons déclaré des variables constantes indispensables au jeu :

* **NB\_CASES** : nombre de cases sur le plateau
* **NB\_PIERRESJ1** : nombre de pierres du joueur 1
* **NB\_PIERRESJ2** : nombre de pierres du joueur 2

Nous avons créé deux tableaux à 3 dimensions :

* **J**, dont les cases ont pour valeur le gain à l’indice x, y, t
* **SubGames**, dont les cases ont pour valeur les sous-jeux du jeu à l’indice x, y, t

Les indices correspondent à :

* **x** : le nombre de pierres restantes au joueur 1
* **y** : le nombre de pierres restantes au joueur 2
* **t** : la position du troll (variant de -NB\_CASES//2 à NB\_CASES//2)

Nous calculons toutes les valeurs possibles du jeu au démarrage du programme afin de ne plus avoir qu’à les consulter pendant l’exécution.

### Étape 1 : Initialisation des matrices V(0, y, t), V(x, 0, t) et V(x, x, 0)

La fonction **initJeu** permet d’initialiser la matrice **J** :

* si x == 0
  + si t se trouve sur le château du joueur 1 (t == -NB\_CASES//2), alors

le joueur 1 a perdu. J(0, y, t) = -1

* + si t se trouve sur le château du joueur 2 (t== NB\_CASES//2), alors

le joueur 1 a gagné. J(0, y, t) = 1

* si le nombre de pierres restantes au joueur 2 est égal à t (y == t), alors

le jeu est nul. J(0, y, t) = 0

* si le nombre de pierres restantes au joueur 2 est inférieur à t (y < t), cela signifie que le troll est plus proche du château du joueur 2, donc

le joueur 1 a gagné. J(0, y, t) = 1

* si le nombre de pierres restantes au joueur 2 est supérieur à t (y > t), cela signifie que le troll est plus proche du château du joueur 1, donc

le joueur 1 a perdu. J(0, y, t) = -1

* si y == 0
  + si t se trouve sur le château du joueur 1 (t == -NB\_CASES//2), alors

le joueur 1 a perdu. J(x, 0, t) = -1

* + si t se trouve sur le château du joueur 2 (t== NB\_CASES//2), alors

le joueur 1 a gagné. J(x, 0, t) = 1

* si le nombre de pierres restantes au joueur 1 est égal à -t (x == -t), alors

le jeu est nul. J(x, 0, t) = 0

* si le nombre de pierres restantes au joueur 1 est supérieur à t (y > t), cela signifie que le troll est plus proche du château du joueur 2, donc

le joueur 1 a gagné. J(x, 0, t) = 1

* si le nombre de pierres restantes au joueur 1 est inférieur à t (y < t), cela signifie que le troll est plus proche du château du joueur 1, donc

le joueur 1 a perdu. J(x, 0, t) = -1

* si t == 0 et x == y

J(x, y, 0) = 0

Une fois ces valeurs calculées, nous pouvons calculer les autres valeurs du jeu.

### Étape 2 : Calcul de tous les sous-jeux

Nous avons défini la fonction **fillMatrice** itérativement pour toutes les valeurs de 1 à NB\_PIERRES\_J1, de 1 à NB\_PIERRES\_J2 et de 0 à NB\_CASES.

Nous n’avons pas commencé à itérer à partir de 0, car pour 0 les cas ont déjà été traités.

Ensuite, nous avons construit tous les sous-jeux.

Pour a allant de 0 à la valeur de x actuelle et b allant de 0 à la valeur de y actuelle, nous avons calculé le sous-jeu de V(x, y, t). Nous avons adapté la position t en fonction du nombre de pierres jouées par les deux joueurs.

En construisant nos boucles de cette manière, toutes les valeurs sont parcourues dans l’ordre ; ainsi, il n’est pas nécessaire de refaire des calculs, les valeurs se trouvant déjà dans le tableau.

# Jeu du Troll

Le jeu repose sur 3 classes essentielles pour illustrer les différents éléments du jeu.

### Classe Joueur

Cette classe représente les deux joueurs.

Elle s’initialise à partir de 3 variables modifiables par l’utilisateur :

* **name** : le nom du joueur
* **nb\_pierres** : le nombre de pierres que le joueur peut lancer au début du jeu
* **position** : la position du château dans lequel il se trouve

En plus des méthodes permettant de récupérer les informations de la classe (getName, getNbPierres et getPosition)la méthode **jettePierres** permet au joueur de lancer ses pierres. La classe est également dotée d’une dernière méthode indiquant si le joueur n’a plus de pierres à lancer.

### Classe Troll

La principale fonctionnalité de cette classe est de déterminer la **position** du troll sur le plateau de jeu.

Lorsque les joueurs jettent leurs pierres, le troll se déplace en fonction du nombre de pierres lancées par ces derniers.

### Classe Game

C’est la classe la plus importante car elle est responsable de la gestion globale du jeu. Elle permet de coordonner les interactions entre les joueurs et le troll.

Elle s’initialise à partir de :

* **player1** : le joueur 1
* **troll** : le Troll
* **player2** : le joueur 2

La fonction **isOver** vérifie si l’état actuel du jeu amène ce dernier à s’arrêter.

Le jeu se termine dans 3 cas :

* lorsque le troll a atteint le château du joueur 1, ce qui entraîne la victoire du joueur 2
* lorsque le troll a atteint le château du joueur 2, ce qui entraîne la victoire du joueur 1
* lorsque les deux joueurs n’ont plus de pierres.

Dans le dernier cas, on vérifie la position du Troll ; le joueur ayant le château le plus proche de lui est le perdant.

La fonction **startRound** gère les évènements qui se déroulent durant un tour de jeu.

Elle commence par vérifier si un des deux joueurs n’a plus de pierres. Dans ce cas, l’autre joueur lance le reste de ses pierres et le troll se déplace d’autant de cases.

Si les deux joueurs possèdent encore des pierres, ils choisissent chacun un nombre de pierres et les lancent, le troll bouge en fonction des choix réalisés.

La fonction **stats** permet d’obtenir l’état actuel du jeu, c’est-à-dire le nombre de pierres du joueur 1, le nombre de pierres du joueur 2 et la position du troll.

### Main

Le main du jeu initialise tous ces éléments, affiche l’état du jeu à chaque tour et appelle les fonctions de la classe Game pour faire fonctionner le jeu et l’arrêter si besoin.