TP DiceGame

En suivant le cahier des charges suivant, vous devez réaliser un jeu de dès.

Diagramme

Copiez le diagramme suivant sur www.planttext.com

```
@startuml

skinparam classAttributeIconSize 0

class De {
  +valeur
  +lancer()
}

De "0..*" --o "1" Gobelet

class Gobelet {
  +valeur
  lancer()
  afficherScore()
}

class Joueur {
  +nom
  +score
  jouer(Gobelet)
  afficherScore()
```

```
}

Joueur "0..*"  --o "1" Partie
Partie "1" o-- "1" Gobelet

class Partie {
  nbTours: int
  initialiserPartie()
  lancerPartie()
  afficherGagnant()
}

@enduml
```

Règles

Nous avons un gobelet par partie. Les joueurs se partagent le gobelet. Le gobelet contient tous les dès de la partie. Il y aura toujours un dés par joueur. On veut en revanche pouvoir décider du nombre de manches par partie. Si le nombre de joueurs est pair, alors le nombre de manche doit être impair et inversement. Et ce afin d'éviter qu'il ne puisse y avoir des égalités.

Le joueur avec le plus haut score gagne la manche. Le joueur qui a remporté le plus de manches gagne la partie !

La partie elle même se déroulera automatiquement, et enverra les informations suivants en console :

- Pour chaque tour, le jet réalisé par chaque joueur
- Le vainqueur de la manche
- Une fois toutes les manches jouées, on veut voir en console :
    - Le nombre de manches gagnées pour chaque joueur,
    - le vainqueur de la partie.

Aides du poussin

Logique technique

Vous constaterez qu'il manquera certains attributs sur le diagramme. N'oubliez pas que vous êtes libre d'ajouter autant de méthodes et d'attributs que vous souhaitez au

classe ! L'essentiel étant que les méthodes présentes sur le diagramme puissent être appelées et aient l'effet esconté !

Agrégation

N'oubliez pas de regarder les relations d'agrégation présentes, et portez une attention particulière aux cardinalités : Elles impliquent souvent que la classe ait un atribut qui ne pourra contenir qu'un certain type appartenant à la classe associée. N'oubliez pas que cet attribut peut être une liste ! Pensez à utiliser le REST param si nécessaire, et si vous vous en servez, pensez à insérer son contenu individuellement dans l'attribut associé.

Ordre

Vous remarquerez qu'il y a beaucoup de relations d'agrégation. Il sera pour vous important de déterminer dans quel ordre vous commencerez à créer les classes. Vous devez donc partir des classes qui sont le plus autonomes.

Aides du Poussin

Méthodes

Pensez à bien découper la logique de votre application en plusieurs méthodes distinctes. Vous aurez certainement besoin de procéder ainsi, notamment dans les méthodes de la classe Partie. N'hésitez donc pas à créer les méthodes suivantes :

- Préparer partie : pour instancier le nombre de dès nécessaires
- Jouer le tour : pour lancher les dès de chaque joueur et calculer le score
- Définit vainqueur : pour déterminer qui a gagné la manche

La liste peut être longue en fonction de comment vous avez prévu de gérer le déroulement d'un tour.

Connaitre le vainqueur

Voici le bloc de code permettant d'identifier le plus haut score d'un joueur :

```
let winner: Player = this._players[0];
this._players.forEach(player => {
    if (player.gameScore > winner.gameScore) {
        winner = player;
    }
```

On affecte le premier joueur de la liste à la variable vainqueur, puis on parcourt la liste de joueurs en comparant le score de chaque joueur avec le score du joueur vainqueur. On remplace le vainqueur si le score du joueur actuel est supérieur au score du joueur vainqueur.

Maintenant, pensez à ce qu'il faudrait faire en cas d'agalité ? Il faudra tout simplement reroll pour les deux joueurs ex-aequo. La fonction peut donc se rappeler elle même pour relancer le résultat.

Roll

Voici le code permettant d'obtenir une valeur aléatoire sur un dès :

```
roll(): number {
        this._value = Math.floor(Math.random() * this._sides) + 1;
        return this._value;
    }
```