DevineLaCarte

Table des matières

Noms : Tonino Falzone Alvarez, Benjamin Nauguet, Damien Caye et Yoan Delbois

Date : 14/12/2022

Thème

Développer une logique de jeu mettant en œuvre de la conception objet et des tests unitaires.

Objectif

- Conception et mise au point d'une logique applicative avec Kotlin et JUnit
- Structure de données, recherche d'un élément dans une liste
- Analyse des actions du joueur (fonction du nombre de cartes, aides à la décision)

Challenge 1 : Prises en mains et tests unitaires.

Tout d'abord, nous faisons les différents tests afin de faire les différents tests unitaires

Ensuite, nous avons pris en main le projet sur nos IDE et de surcroit créé les différentes fabriques de paquets de cartes. D'abord celle du paquet de 32 cartes et ensuite celle du paquet de 52 cartes.

Vous trouverez ci-dessous les codes qui ont permis de créer ces différents jeux de cartes :

Fabrique paquet de 32 cartes :

```
/*** Création d'un paquet de 32 cartes ***/
fun createJeu32Cartes() : List<Carte>
{
var listeCartes: MutableList<Carte> = mutableListOf()
for (couleur in CouleurCarte.values())
{
for (nom in NomCarte.values())
{
when (nom.toString()) {
"DEUX", "TROIS", "QUATRE", "CINQ", "SIX" -> continue
else -> listeCartes.add(Carte(nom, couleur))
}
}
}
return listeCartes
}
```

Fabrique paquet de 52 cartes :

```
/*** Création d'un paquet de 52 cartes ***/
fun createJeu52Cartes() : List<Carte>
{
    var listeCartes: MutableList<Carte> = mutableListOf()
    for (couleur in CouleurCarte.values())
    {
        for (nom in NomCarte.values())
        {
            listeCartes.add(Carte(nom, couleur))
        }
    }
    return listeCartes
}
```

Challenge 2 : représentation de l'algorithme initial de la fonction main.

Pour ce challenge, le travail demandé était de faire l'implémentation des différents tests afin de créer différentes fonctions du jeu de cartes. Celles-ci doivent ; donner une aide au joueur et l'aider à trouver la carte proposée par le jeu en lui indiquant si la valeur est supérieure ou inférieure à la proposition du joueur ou si la couleur est la bonne. Ensuite grâce à des diagrammes, expliquer le fonctionnement du programme avant et après modifications.

Diagrammes avant modifications

```
@startuml
!pragma useVerticalIf on
title Diagramme d'activité 1

start

:Création du paquet;
:Instanciation du jeu;
:Donner un nom;
:Donner une couleur;

if (Définition couleur et nom) then (Pas Bon)
    :Erreur et affichage de
          la carte donnée;

else (Bon)
    if (Carte devinée) then (oui)
        :affichage de la carte
                à deviner;
    else (non)
endif
endif
:fin de la partie;
stop
@enduml
```

Diagrammes Post modifications

```
@startuml
!pragma useVerticalIf on
title Diagramme d'activité 2

start

:Activation ou désactivation de l'aide;
:Création du paquet;
:Instanciation du jeu;
:Donner un nom;
:Donner une couleur;

repeat :Définition couleur et nom;
if (Bonne carte) then (Oui)
:Affichage de la carte à deviner;
:Affichage de la statégie;
break
else (Non)
```

```
if (Aide) then (Oui)
:Affichage de l'aide;
else (Non)
:Pas d'affichage d'aide;
endif
endif
repeat while (Recommencer) is (Oui)

:Fin de la partie;

stop
@enduml
```

Challenge 3 : Rebattre les cartes

Le concept vise à rebattre le paquet de cartes qui est initialement dans un ordre précis. Pour cela nous utilisons la methode Shuffle qui permettra au programme de choisir une carte au hasard dans le paquet.

```
    fun rebattre() {
        return shuffle(cartes)
    }
```

Challenge 4

Ce challenge consiste à aider le joueur à élaborer une stratégie pour mieux trouver la carte à deviner du programme.

```
fun strategiePartie(nbEssais : Int): String {
            if(avecAide){
                val iaTry : Double = log2(paquet.cartes.size.toDouble())
                if(nbEssais.toDouble() >= iaTry1.80){
                    return "Stratégie dichotomique peu précise, vous avez fais
$nbEssais essais"
                }
                else if (nbEssais.toDouble() >= iaTry +1 && nbEssais.toDouble() <
iaTry 1.80){
                    return "Stratégie dichotomique assez précise, vous avez fais
$nbEssais essais"
                }
                else if (nbEssais == iaTry.toInt()) {
                    return "Stratégie dichotomique très précise, vous avez fais
$nbEssais essais"
                }
                else{
                    return "Peu de stratégie sûrement de la chance, vous avez fais
$nbEssais essais"
```

```
                }
            }
            else{
                val pourcentChance : Double = (nbEssais.toDouble() /
paquet.cartes.size.toDouble())*100.0
                return if (nbEssais / paquet.cartes.size <= 0.25){
                    "Stratégie linéaire, vous aviez ${pourcentChance.toInt()}% de
chance de trouver, vous avez $nbEssais essais"
                }
                else{
                    "Stratégie linéaire, vous aviez ${pourcentChance.toInt()}% de
chance de trouver, vous avez $nbEssais essais"
                }
            }
            return "Erreur"
    }
```