DevineLaCarte

Les differents challenge:

Fait par Durang Vincent et Theo Pinceel le 14/10/2022 Depot gitlab du projet: https://gitlab.com/theopcl/DevineLaC

Challenge 1

- Verification du code initiale avec plusieurs tests unitaires dans `CarteTest` et `PaquetTest`
- Dans `FrabriqueJeuDeCartes` ajout d'un paquet de 32 en créant une list des 32 cartes `createJeu32Cartes()` et un paquet de 52 cartes avec une list des 52 `createJeu52Cartes()`
- Conception de nouveau test pour comparer des cartes de meme couleur mais de valeur differentes `compareCartesDeMemeCouleurMaisDeValeurDifferente()` et de carte de meme valeur mais de couleur differentes `compareCartesDeCouleurDifferenteMaisDeMemeValeur()`
- Pour le bon fonctionnnement des tests unitaire, une modification de `compareTo`, est réalisée pour que la fonction return aussi la comparaison entre la couleur si deux valeurs sont identiques, or initialement return que la comparaison entre de valeurs

Voici le `compareTo`:

```
override fun compareTo(other: Carte): Int {
        return if (this.valeur.compareTo(other.valeur) != 0) {
            this.valeur.compareTo(other.valeur)
        } else {
            this.couleur.ordinal.compareTo(other.couleur.ordinal)
        }
    }
```

Les tests unitaire suivant reussi: `getNom()` :

```
 @Test
    fun getNom() {
        assertEquals("VALET", this.valetDeCoeur.nom.name)
        assertNotEquals("Valet", this.valetDeCoeur.nom.name)
    }
```

`getCouleur()` :

```
@Test
fun getCouleur() {
// test sur le nom (String)
assertEquals("COEUR", this.valetDeCoeur.couleur.name)

        // test sur la valeur énumérée
        assertEquals(CouleurCarte.COEUR, this.valetDeCoeur.couleur)

        // test sur mauvaise valeur énumérée
        assertNotEquals(CouleurCarte.PIQUE, this.valetDeCoeur.couleur)
    }
```

`valeurCartes()` :

```
@Test
fun valeurCartes() {
assertEquals(11, this.valetDeCoeur.valeur)

         val asDeCoeur : Carte =Carte(NomCarte.AS,CouleurCarte.COEUR)
         val roiDeCoeur : Carte = Carte(NomCarte.ROI, CouleurCarte.COEUR)
         val troisDePique : Carte = Carte(NomCarte.TROIS, CouleurCarte.PIQUE)

         assertEquals(14, asDeCoeur.valeur)
         assertEquals(13, roiDeCoeur.valeur)
         assertEquals(3, troisDePique.valeur)

         assertNotEquals(asDeCoeur.valeur,roiDeCoeur.valeur)
     }
```

`compareCartesDeMemeCouleurMaisDeValeurDifferente()`:

```
@Test
     fun compareCartesDeMemeCouleurMaisDeValeurDifferente()  {
         val asDeCoeur : Carte = Carte(NomCarte.AS, CouleurCarte.COEUR)
         val roiDeCoeur : Carte = Carte(NomCarte.ROI, CouleurCarte.COEUR)

         // test avec compareTo
         assertTrue(asDeCoeur.compareTo(roiDeCoeur) > 0 )

         // Finalement, si les objets sont *comparables*
         // alors les opérateurs binaires de comparaisons sont applicables
         // L'opérateur '>' appellera automatiquement la méthode compareTo (comme ci-
dessus)
         // voir https://kotlinlang.org/docs/collection-ordering.html" - ordre naturel
         assertTrue(asDeCoeur > roiDeCoeur)
```

`compareCartesDeCouleurDifferenteMaisDeMemeValeur()`:

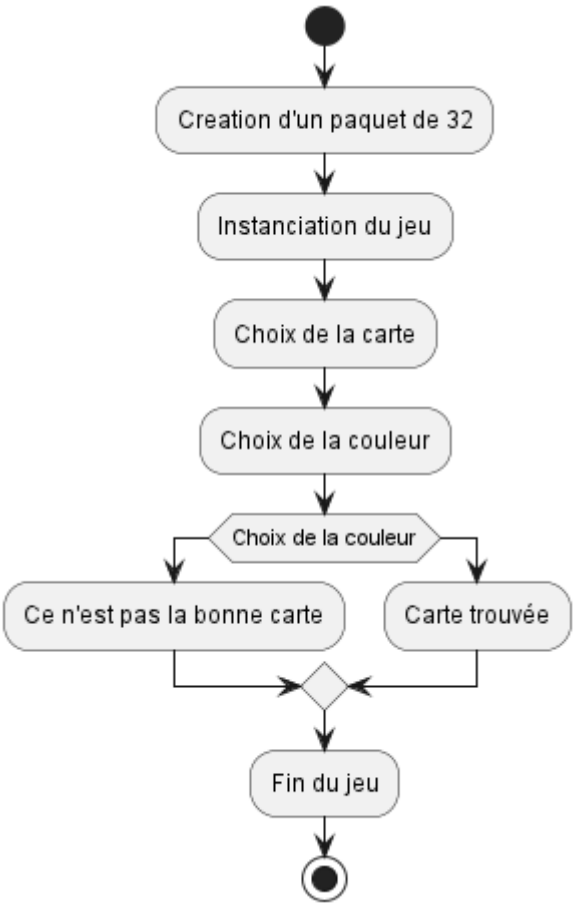
```
 @Test
    fun compareCartesDeCouleurDifferenteMaisDeMemeValeur()  {
        // TODO test à implémenter : Attention dépend de la hiérarchie des couleurs
retenue
        //   prévoir plusieurs méthodes (plusieurs cas)
        val asDeCoeur : Carte = Carte(NomCarte.AS, CouleurCarte.COEUR )
        val asDePique : Carte = Carte(NomCarte.AS, CouleurCarte.PIQUE)

        assertTrue ( asDeCoeur > asDePique)

    }
```

Challenge 2

- Representation de l'algorithme initial avec un `planuml`:



- Objectif implementer les differents TODO Ajouter un choix aide : Initialisation d'une valeur aide de type boolean Creation d'une boucle qui demande au joueur d'active l'option aide si le joueur choisi "oui" la valeur aide est vrai, sinon le joueur choisi "non" la valeur est fausse et la boucle prend fin or si le joueur entre rien ou pas les réponses proposer imprime "erreur" la boucle prends pas fin

```
var aide = false

    do {
        println("Voulez-option de l'aide. (oui ou non)")
        var reponse = readln().trim().uppercase()
        if (reponse == "OUI") {
            aide = true
        } else if (reponse == "NON") {
            aide = false
        } else {
            print("erreur\n")
        }
    } while (reponse != "OUI" && reponse != "NON")
```

Ajouter un choix paquet : Le joueur choisi entre un paquet de 32 (entre "32" stocker dans la valeur paquet) ou un paquet de 52 (entre "52" stocker dans la valeur paquet) si paquet egale à 32, la valeur "PaquetDeCarte" prends la list des 32 cartes sinon la list de 52 cartes mis par defaut. Le code est dans une boucle do s'arretant quand le joueur rentre readln 32 ou 52 sinon une erreur est imprimé, la boucle recommence les conditions ne sont pas remplis.

```
do {
println("Choix du paquet : Paquet de 32 (entrez 32) ou Paquet de 52 (entrez 52)")
       val paquet = readln().toInt()
           if (paquet === 32) {
              println("Création d'un paquet de 32 cartes")
              paquetDeCartes = Paquet(createJeu32Cartes())
           } else if (paquet === 52) {
              println("Création d'un paquet de 52 cartes")
           } else {
              print("erreur\n")
        }
      } while (paquet != 32 && paquet != 52)
```

Carte moins predictive : Changement de la fonction `getCarteADeviner()` dans `Paquet`, ajout d'un système Ramdom.

```
fun getCarteADeviner(): Carte {

fun rand(start: Int, end: Int): Int {
require(!(start > end || end - start + 1 > Int.MAX_VALUE)) { "Illegal Argument" }
return Random(System.nanoTime()).nextInt(end - start + 1) + start
}
var start = 0
var end =0

        if ( this.cartes == createJeu32Cartes()){
            start = 0
            end = 31
            return this.cartes[rand(start, end)]
        } else {
            start = 0
            end = 51
            return this.cartes[rand(start, end)]
        }
    }
```

Aide active : si l'aide est vrai, montre si la carte est plus grande ou plus petite, utilisation de `compareTo` nécessaire.

```
if (aide == true) {
                    carteADeviner.compareTo(carteDuJoueur)
                    if (carteDuJoueur < carteADeviner)
                        println("La carte a deviner est plus grande")

                    if (carteDuJoueur > carteADeviner)
                        println("La carte a deviner est plus petite")

                    // TODO: (A) si l'aide est activée, alors dire si la carte
proposée est
                    //  plus petite ou plus grande que la carte à deviner

                }
```

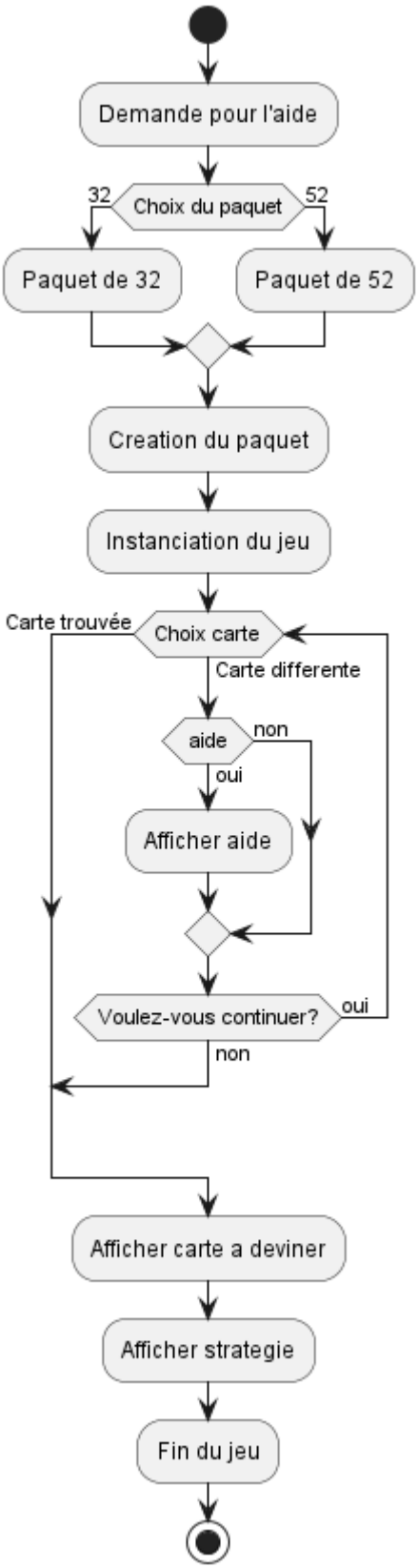
Ajouter un choix continuer ou abandonner : Dans le `MainPlayConsole`, une valeur abandonner est initialisé en dehors d'une boucle. Afin, que le joueur rejoue une carte une boucle "DO WHILE" (sorti de la boucle quand "recommencer" est different de 0) est crée, apres que le joueur est choisis une carte, le choix de continuer est proposé quand le joueur choisis de pas continuer (entre dans le readln "non") la valeur initialiser précedemment appeler "recommencer" prend 1 cequi fait sorti de la boucle si le joueur choisis de continuer (entre dans le readln "oui") la boucle reprends au choix de la carte.

```
println("Voulez-continuer?(oui ou non)")
                val restart = readln().trim().uppercase()
                if (restart == "OUI") {
                    abandonner = true
                } else if (restart == "NON") {
                    recommencer++
                    abandonner = false
                    println(" ==== Fin prématurée de la partie ====")
                } else {
                    println("erreur")
                }
```

Presentation de la carte à deviner Imprime la valeur `carteAdeviner` à la fin de la partie.

```
println("La carte a deviner etais la ${carteADeviner}")
```

- Representation de l'algorithme après changement avec un `planuml`:



Challenge 3

*Creation d’une fonction "rebattreCarte()" qui utilise la methode shuffle. Shuffle permet de melanger aleatoirement les valeurs d’une liste.

```
fun rebattreCarte() {
        return shuffle(this.cartes)
    }
```

*Creation d’un test unitaire "testMelangeCarte()" qui verifie que les cartes soient bien melangées en comparant la place des cartes avant et apres le passage de la fonction rebattreCarte().

```
fun testMelangeCarte() {
        val paquet2Cartes = Paquet(listOf(
            Carte(NomCarte.VALET, CouleurCarte.COEUR),
            Carte(NomCarte.DIX, CouleurCarte.TREFLE),
            Carte(NomCarte.DAME, CouleurCarte.PIQUE),
            Carte(NomCarte.NEUF, CouleurCarte.TREFLE),
        ))
        var desCartes: List<Carte> = paquet2Cartes.cartes
        shuffle(desCartes)

        assertNotEquals(NomCarte.DIX,desCartes[1].nom)
    }
```

Challenge 4

*Creation d'une fonction `strategiePartie()` qui vérifie si le nombre d'essais (paremetre compteur) du joueur est convenable par rapport au paquet choisi. L'utilisation de `log2()` permet de savoir si le nombre d'essais etait convenable ou invenable par rapport au paquet choisis. Puis on calcule le pourcentage de chance d'avoir trouvé la bonne carte en fonction du nombre d'essais et du nombre de cartes du paquet. Avec ce pourcentage de chance on peut affirmer ou non si le joueur a eu de la chance.

```
fun strategiePartie(abandonner : Boolean , compteur: Int): String? {

var reponse: String
var paquet2 = paquet.cardinal().toDouble()
val pChance = (compteur.toDouble() / paquet.cartes.size.toDouble())*100
//var chance: String

    if (abandonner == false) {
        return("Vous avez abandonné (nb d'essai(s) $compteur)")
    } else {
         if (compteur > log2(paquet2)) {
            reponse = "Vous avez trouvé la carte avec un nb d'essais inconvenable (nb
d'essai(s) $compteur)"
            if (pChance < 30.0){
                println("Vous avez de la chance, vous n'aviez que $pChance% de chance
de trouver la carte")
             } else {
            println ("Vous n'avez pas beaucoup de chance vous aviez $pChance% de
chance de trouver la bonne carte")

          }
           return reponse
         } else if (compteur < log2(paquet2)) {
            reponse = "Vous avez trouvé la carte avec un nb d'essais convenable (nb
d'essai(s) $compteur)"
            if (pChance < 30.0){
                println("Vous avez de la chance, vous n'aviez que $pChance% de chance
de trouver la carte")
            }else{
                println ("Vous n'avez pas beaucoup de chance vous aviez $pChance% de
chance de trouver la bonne carte")

            }
            return reponse
        }

}
        return null
    }
```