Devine La Carte

Table des matières

Source :

- Le projet vient du dépôt : https://gitlab.com/sio-labo/devinelacarte.git
- Le dépôt du raport du projet : https://gitlab.com/alexandrepoirier69/devinelacarte.git

Thème

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

Jeu en mode console. Un début d'implémentation est proposé (`MainPlayConsole.kt` en interaction en mode console/terminal)

Prérequis

Niveau : Deuxième année de BTS SIO SLAM

- Bases de la programmation,
- IntelliJ et kotlin opérationnels sur votre machine de dev
- Avoir fait des premiers pas avec Kotlin (exercices d'initiation)
- Avoir eu une première introduction à la notion de Test Unitaire, distinction entre ***expected value*** et ***actual value***, dans une approche prédictive. En particulier avoir réalisé avec succès l'exercice Faites vos comptes

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)

Scénario typique d'une partie

1. (optionnel pour le joueur) paramétrage du jeu (par exemple choix du jeu de cartes, activation de l'aide à la recherche, …)
2. Lancement d'une partie (le jeu instancie un jeu de carte et tire une carte "au hasard"), que le joueur doit deviner en un nombre de proposition de cartes **optimal**
3. Le joueur propose une carte
4. Si ce n'est pas la bonne carte, alors si l'aide est activée, le joeur est informé si la carte qu'il a soumise est plus petite ou plus grande que celle à deviner. Retour en 3.

   Si c'est la bonne carte alors la partie se termine (passe à l'étape suivante). Le joueur peut choisir d'abandonner la partie.

5. Le jeu affiche des éléments d'analyse (nombre de fois où le joueur a soumis une carte, ses **qualités stratégiques**, …)
6. Fin de la partie.

Challenge-1 : Prise en main et tests unitaires

Pour le challenge-1 nous devions faire passer 4 tests sur 6 ce sont :

- `fabriqueDe52Cartes()`
- `fabriqueDe32Cartes()`
- `testGetCartes()`
- `compareCartesDeCouleurDifferenteMaisDeMemeValeur()`

Donc pour `fabriqueDe52Cartes()` nous avons fait un test qui vérifie si notre paquet de carte contient bien 52 cartes et vérifie si la première carte est bien un "DEUX" comme montrer ci-dessous :

```
    @Test
    fun fabriqueDe52Cartes() {
        val test = Paquet(createJeu52Cartes())
        assertEquals(52, test.cartes.size)
        assertEquals(NomCarte.DEUX, test.cartes[0].nom)
    }
```

Pour `fabriqueDe32Cartes()` nous avons fait un test qui vérifie si notre paquet de carte contient bien 32 et vérifie si la première carte est bien un "SEPT" cartes comme montrer ci-dessous :

```
    @Test
    fun fabriqueDe32Cartes() {
        val test = Paquet(createJeu32Cartes())
        assertEquals(32, test.cartes.size)
        assertEquals(NomCarte.SEPT,test.cartes[0].nom)
    }
```

Pour réaliser les tests de fabrication de carte nous avons réalisée une boucle dans la classe `FrabriqueJeuDeCartes.kt` lors de la fabrication du paquet de carte afin d'éviter de rajouter les cartes une à une dans le paquet en dur comme ci-dessous :

- `FrabriqueJeuDeCartes()`

```
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
}

// TODO Création d'un paquet de 52 cartes à implémenter (de 2 à AS)
fun createJeu52Cartes() : List<Carte> {
    var listeCartes: MutableList<Carte> = mutableListOf()
//  return listOf(
    for (couleur in CouleurCarte.values()) {
        for (nom in NomCarte.values()) {
            listeCartes.add(Carte(nom, couleur))
        }
    }
    return listeCartes
}
```

Pour `testGetCartes()` nous avons fait un test qui vérifie si le nom de la carte est bien correct vers laquelle on pointe comme montre ci-dessous:

```
    @Test
    fun testGetCartes() {
        val paquet2Cartes = Paquet(listOf(
            Carte(NomCarte.VALET, CouleurCarte.COEUR),
            Carte(NomCarte.DIX, CouleurCarte.TREFLE),
        ))
        var testCartes : List<Carte> = paquet2Cartes.cartes
        assertEquals(2,testCartes.size)
        assertEquals(NomCarte.VALET,testCartes[0].nom)
        assertEquals(CouleurCarte.COEUR,testCartes[0].couleur)
    }
```

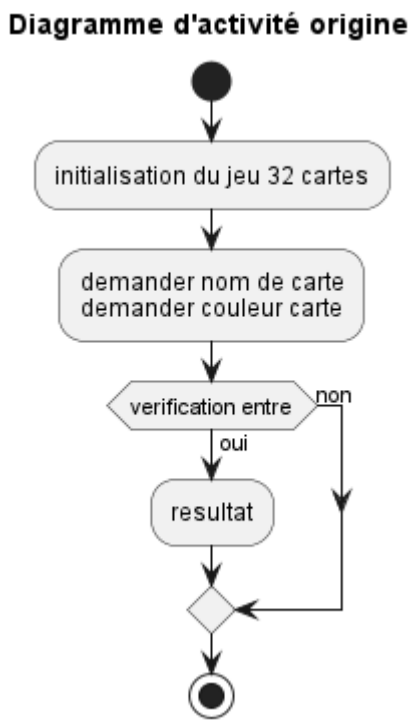
Pour `compareCartesDeCouleurDifferenteMaisDeMemeValeur()` nous avons fait un test qui vérifie deux cartes de couleur différente mais si elle est de même valeur comme montrer ci-dessous

```
@Test
fun compareCartesDeCouleurDifferenteMaisDeMemeValeur() {
    val roiDePique: Carte = Carte(NomCarte.ROI, CouleurCarte.PIQUE)
    val roiDeCoeur: Carte = Carte(NomCarte.ROI, CouleurCarte.COEUR)
    assertTrue(roiDePique.compareTo(roiDeCoeur) < 0)
    assertTrue(roiDeCoeur.compareTo((roiDePique)) > 0)
    assertTrue(roiDeCoeur.compareTo(roiDeCoeur) == 0)
}
```

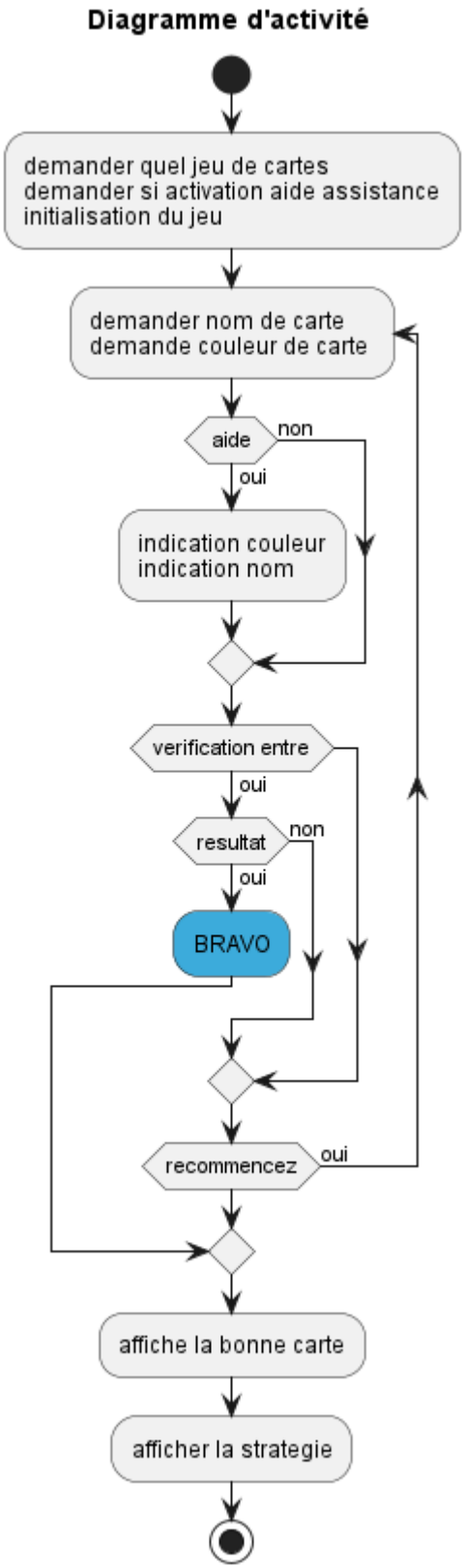
Challenge-2 : Jouer avec le jeu

Pour le challenge-2 nous devions faire 2 diagrammes plantuml afin d'expliquer comment le jeu de carte à deviner fonctionne qui sont :

Existant :



Version final :



Et nous devions aussi faire les TODOs (A)

Celui qui demande de l'aide grâce à une condition qui prend en compte le choix de l'utilisateur avec la varible immuable `questionAide`

```
    println("Voulez-vous de l'aide ? (oui) ou (non): ")
    var aide = false
    val questionAide = readLine().toString().trim().lowercase()
    if (questionAide == "oui") {
        println("Tu as pris de l'aide")
        aide = true
    } else {
        println("Tu n'as pas pris d'aide")
    }
```

Celui qui demande le jeu de carte grâce à plusieurs conditions qui prennent en compte le choix de l'utilisateur avec la variable immuable `nbCartePaquet`

```
    var paquetDeCartes = Paquet(createJeu32Cartes())
    do {
        println("Choisir le type de jeu, 32 cartes ou 52 cartes: ")
        val nbCartePaquet = readLine()?.toInt()
        if (nbCartePaquet == 32) {
            paquetDeCartes = Paquet(createJeu32Cartes())
            println("Vous avez choisi 32 cartes")
        } else if (nbCartePaquet == 52) {
            paquetDeCartes = Paquet(createJeu52Cartes())
            println("Vous avez choisi 52 cartes")
        } else {
            println("Tu joues à quoi recommence: ")
        }
    } while (nbCartePaquet != 32 && nbCartePaquet != 52)
```

Celui qui permet d'écrire un nombre ou une chaine de caractère grâce à l'utilisation tableau grâce à la réponse de l'utilisateur avec la variable mutable `nomCarteUserStr`

```
        println("Entrez un nom de carte dans le jeu (exemples : Roi, sept, six, As...)
:")
        val nomCarteUserStr: String = readLine() + ""
        val nomCarteUser: NomCarte? =
            if (nomCarteUserStr in (2..10).toSet().toString()) {
                val words = arrayOf("deux", "trois", "quatre", "cinq", "six", "sept",
"huit", "neuf", "dix")
                val nomCarteUserInt = nomCarteUserStr.toInt()
                val result = words[nomCarteUserInt - 2]
                getNomCarteFromString(result.trim().uppercase())
            } else
                getNomCarteFromString(nomCarteUserStr.trim().uppercase())
```

Celui quand l'aide est activé indique la position de la carte (plus grande ou plus petite) par rapport à la réponse de l'utilisateur grâce à plusieurs conditions et appel de méthode `carteDuJoueur.valeur.compareTo` et `jeu.carteADeviner`

```
if (aide) {
                    // TODO: (A) si l'aide est activée, alors dire si la carte
proposée est
                    //  plus petite ou plus grande que la carte à deviner
                    if (carteDuJoueur.valeur.compareTo(jeu.carteADeviner.valeur) == 0)
{
                        println("la valeur est la meme")
                    }
                    if (carteDuJoueur.valeur.compareTo(jeu.carteADeviner.valeur) > 0)
{
                        println("la valeur est plus petite")
                    }
                    if (carteDuJoueur.valeur.compareTo(jeu.carteADeviner.valeur) < 0)
{
                        println("la valeur est plus grande")
                    }
                    if
(carteDuJoueur.valCouleur.compareTo(jeu.carteADeviner.valCouleur) == 0) {
                        println("la couleur est la meme")
                    }
                    if
(carteDuJoueur.valCouleur.compareTo(jeu.carteADeviner.valCouleur) > 0) {
                        println("la couleur est plus petite")
                    }
                    if
(carteDuJoueur.valCouleur.compareTo(jeu.carteADeviner.valCouleur) < 0) {
                        println("la couleur est plus grande")
                    }
```

```
                }
```

Celui qui permet de retenter sa chance sans quitter la partie nous avons réaliser un do while à partir de la demande de carte et couleur et déclarer/initialiser une variable immuable qui demande la réponse de l'utilisateur puis nous prenons en compte sont choix dans la condition du do while

```
    do {
        println("Entrez un nom de carte dans le jeu (exemples : Roi, sept, six, As...)
:")
        // TODO (optionnel) permettre de saisir un chiffre au lieu d'une chaine : 7 au
lieu de Sept...
        val nomCarteUserStr: String = readLine() + ""
        val nomCarteUser: NomCarte? =
            if (nomCarteUserStr in (2..10).toSet().toString()) {
                val words = arrayOf("deux", "trois", "quatre", "cinq", "six", "sept",
"huit", "neuf", "dix")
                val nomCarteUserInt = nomCarteUserStr.toInt()
                val result = words[nomCarteUserInt - 2]
                getNomCarteFromString(result.trim().uppercase())
            } else
                getNomCarteFromString(nomCarteUserStr.trim().uppercase())

        println("Entrez un nom de \"couleur\" de carte parmi : Pique, Trefle, Coeur,
Carreau : ")
        val couleurCarteUserStr: String = readLine() + ""
        val couleurCarteUser: CouleurCarte? =
getCouleurCarteFromString(couleurCarteUserStr.trim().uppercase())
        essaie++
        if (nomCarteUser != null && couleurCarteUser != null) {
            // prise en compte de la carte du joueur
            val carteDuJoueur: Carte = Carte(nomCarteUser, couleurCarteUser)

            if (jeu.isMatch(carteDuJoueur)) {
                println("Bravo, vous avez trouvé la bonne carte !")
                recommencer = false
                win = true
            } else {
                println("Ce n'est pas la bonne carte !")
                println("votre proposition  : $carteDuJoueur")
                if (avecAide) {
                    // TODO: (A) si l'aide est activée, alors dire si la carte
proposée est
                    //  plus petite ou plus grande que la carte à deviner
                    if (carteDuJoueur.valeur.compareTo(jeu.carteADeviner.valeur) == 0)
{
                        println("la valeur est la meme")
                    }
                    if (carteDuJoueur.valeur.compareTo(jeu.carteADeviner.valeur) > 0)
{
                        println("la valeur est plus petite")
```

```
                    }
                    if (carteDuJoueur.valeur.compareTo(jeu.carteADeviner.valeur) < 0)
{
                        println("la valeur est plus grande")
                    }
                    if
(carteDuJoueur.valCouleur.compareTo(jeu.carteADeviner.valCouleur) == 0) {
                        println("la couleur est la meme")
                    }
                    if
(carteDuJoueur.valCouleur.compareTo(jeu.carteADeviner.valCouleur) > 0) {
                        println("la couleur est plus petite")
                    }
                    if
(carteDuJoueur.valCouleur.compareTo(jeu.carteADeviner.valCouleur) < 0) {
                        println("la couleur est plus grande")
                    }
                }
            }
        } else {
            // utilisateur a mal renseigné sa carte
            val nomCarte = if (nomCarteUserStr == "") "?" else nomCarteUserStr
            val couleurCarte = if (couleurCarteUserStr == "") "?" else
couleurCarteUserStr

            println("Désolé, mauvaise définition de carte ! (${nomCarte} de
${couleurCarte})")
        }
        if (!win) {
            println("recommencez ? : (oui/ non)")
            val reponse = readLine()
            if (reponse == "non") {
                recommencer = false
            }
        }
    } while (recommencer)
```

Celui de la carte à deviner nous avons appellés la méthode `jeu.carteADeviner` comme ci-dessous

```
    println("La carte à deviner ${jeu.carteADeviner}")
```

Challenge-3 : Rebattre les cartes

Pour le challenge 3 nous avons réalisé deux méthodes et deux tests qui sont:

Méthodes :

- `getCarteADeviner()`
- `melange()`

Tests :

- `testCarteADeviner()`
- `melangeDeCartes()`

Donc pour `getCarteADeviner()` nous avons mis en place une méthode random pour que l'ordinateur choisisse une carte au hasard comme montrer ci-dessous :

```
    fun getCarteADeviner(): Carte {
        return this.cartes[cartes.indices.random()]
    }
```

Pour `melange()` nous avons mis en place une méthode qui permet de modifier l'ordre des cartes pour qu'elle soit ainsi mélanger comme montrer ci-dessous :

```
fun melange(): Unit = shuffle(cartes)
```

Pour `testCarteADeviner()` nous avons mis en place un test qui vérifie si les cartes choisis dans notre condition est bien vrais comme montrer ci-dessous :

```
    @Test
    fun testCarteADeviner() {
        val paquet2Cartes = Paquet(
            listOf(
                Carte(NomCarte.VALET, CouleurCarte.COEUR),
                Carte(NomCarte.DIX, CouleurCarte.TREFLE),
            )
        )
        var test = false
        val rand = paquet2Cartes.getCarteADeviner().nom
        if (rand == NomCarte.VALET || rand == NomCarte.DIX) {
            test = true
        }
        assertTrue(test)
    }
```

Pour `melangeDeCartes()` nous avons mis en place un test qui vérifie si la première carte est bien un "DEUX" dans notre paquet non mélanger puis après on vérifie après avoir mélanger le paquet de carte si la première carte n'est plus un "DEUX" comme montrer ci-dessous :

```
    @Test
    fun melangeDeCartes(){
        val test = Paquet(createJeu52Cartes())
        assertEquals("DEUX",test.cartes[0].nom.toString())

        test.melange()
        assertNotEquals("DEUX",test.cartes[0].nom)
    }
```

Challenge-4 : La stratégie du joueur

Pour le challenge 4 nous avons réalisé une méthode et une condition qui prend en compte plusieurs variables booléennes :

Méthode :

Donc pour la méthode `strategiePartie(nbEssais: Int)` nous avons fait en sorte que notre condition prend en compte le choix de l'utilisateur pour l'aide et si celle-ci est active notre méthode prend alors en compte une recherche dichotomique sinon elle prend en compte une simple recherche linéaire comme montré ci-dessous:

```
fun strategiePartie(nbEssais: Int): String {
        if (avecAide) {
            val testBot: Double = log2(paquet.cartes.size.toDouble())
            if (nbEssais.toDouble() >= testBot * 1.80) {
                return "Stratégie dichotomique peu précise, vous avez fais $nbEssais
essais"
            }
            if (nbEssais.toDouble() == 1.0){
                return "C'est vous dieux ? pour être aussi fort"
            }
            if (nbEssais.toDouble() <= testBot + 1 && nbEssais.toDouble() < testBot *
1.80) {
                return "Stratégie dichotomique assez précise, vous avez fais $nbEssais
essais"
            }
            if (nbEssais == testBot.toInt()) {
                return "Stratégie dichotomique très précise, vous avez fais $nbEssais
essais"
            } else {
                return "Stratégie dichotomique peu précise, vous avez fais $nbEssais
essais"
            }
        } else {
            val pourcentChance: Double = (nbEssais.toDouble() /
paquet.cartes.size.toDouble()) * 100.0
            return if (nbEssais / paquet.cartes.size <= 0.10) {
                "Bravo avec votre nombre d'essaie vous avez ${pourcentChance.toInt()}%
de chance, vous l'avez fait en $nbEssais essais et vous etes chanceux de l'avoir fait
en moins de 10% de chance"
            }
            else {
                "Bravo, mais avec votre nombre d'essaie vous avez
${pourcentChance.toInt()}% de chance, vous l'avez fait en $nbEssais essais soit plus
de 10% pour etre chanceux"
            }
        }
        return "Erreur"
```

```
    }
```

Condition :

Pour la condition nous pris en compte le choix de l'utilisateur si celui-ci n'a pas gagné et souhaite continuer avec un compteur qui calcule son nombre d'essais comme montré ci-dessous:

```
        if (!win) {
            println("recommencez ? : (oui/ non)")
            val reponse = readLine()
            if (reponse == "non") {
                recommencer = false
            }
        }
```