Compte-rendu : Projet Kotlin Aventure

Sommaire :

Partie I : Présentation générale du projet et de son organisation

Pour mener à bien la réalisation de ce projet, nous nous sommes répartis. Ainsi, chaque tâche a été attribuée à membre de groupe et des deadlines avaient été établies.
Pour ce faire, nous avons utilisé le système de tableau de l'outil Trello pour mener à bien le projet.
Vous trouverez donc en pièce jointe les liens des tableaux pour le sprint I et II

Tableau sprint I :
https://trello.com/invite/b/6gL6r6p4/ATTIe1dcc22a5f9519c6806afec048cc9638A549AD1E/projetkotlinadventure-sprint-1

Tableau sprint II :
https://trello.com/invite/b/Q5YK3t1T/ATTIeb8932f2a2e88c6aec7f3b55dc5047f7E94D6EB9/projetkotlinadventure-sprint-2

Organisation :

Outre les différentes missions du projet à réaliser chacune des membres avait une tâche précise lui étant associé :

Océane : était la responsable du dépôt Git et devais s'assurer de tout bien push.

Lesly : était la responsable du Trello et devais s'assurer qu'il soit tout le temps bien mis à jour.

Anne-Julie : était responsable de la mise en commun de chaque mission et devais s'assurer que chaque membre du groupe avait bien compris et réaliser sa tâche.

Présentation du travail réalisé pour chaque mission :

Mission Sprint 1 :
Mission 1

Mission 1.1 : creationPerso() => Océane

```
55          fun creerPersonnage(): Personnage {
56              println("Créez votre personnage:")
57              println("saisissez votre nom") //L'utilisateur saisit le nom du personnage.
58              // TODO Mission 1.1
59              val nom: String = readln()
60              println("vous avez 40 points à répartir ")
61              var scoreAtk = 0 // attaque
62              var scoreVIT = 0 // vitesse
63              var scoreDef = 0 // défense
64              var scoreEND = 0 // endurance
65              var totalscore: Int
66
67              do {
68                  totalscore = 40
69                  println("Saisissez votre score d'attaque : ") // Saisit scores attaque
70                   scoreAtk= readln().toInt()
71                  totalscore -= scoreAtk // 40 est soustrait par le score
72
73                  println("Saisissez votre score de défense : ") //Saisit scores défense
74                   scoreDef = readln().toInt()
75                  totalscore -= scoreDef // 40 est soustrait par le score
76
77                          println("Saisissez votre score de vitesse : ") //Saisit scores vitesse
78                   scoreVIT = readln().toInt()
79                  totalscore -= scoreVIT // 40 est soustrait par le score
80
81                  println("Saisissez votre score d'endurance : ") //Saisit scores endurance
82                   scoreEND = readln().toInt()
83                  totalscore -= scoreEND
84      //              totalscore = scoreAtk + scoreDef + scoreEND + scoreEND
85              } while (totalscore < 0)
86              val pv= 50 +(10*scoreEND)
87
88              println("Choisissez la classe de votre personnage :") //demande de choisir la classe de son personnage
89              println("Tapez 0 pour Voleur, Tapez 1 pour mage ou Tapez 2 pour guerrier : ")
90
```

```
91              var choix_classe:Int = readln().toInt()
92              var hero = Personnage(nom, pv, pv, scoreAtk, scoreDef, scoreEND, scoreVIT,null,null)
93              if (choix_classe == 0){ //En fonction du choix de l'utilisateur, il faut instancier le bon type de personnage
94                  hero = Voleur(nom, pv, pv, scoreAtk, scoreDef, scoreEND, scoreVIT,null,null, mutableListOf())
95              }
96              else if (choix_classe == 1) {
97                  hero = Mage(nom, pv, pv, scoreAtk, scoreDef, scoreEND, scoreVIT,null,null, mutableListOf(),mutableListOf(projectionAcide,guerison,bouleDeFeu,missileMagique,armeMagique,armureMagique))
98
99                 // hero.armure=armure1
100             }
101             else {
102                 hero = Guerrier(nom, pv, pv, scoreAtk, scoreDef, scoreEND, scoreVIT,null,null,null,mutableListOf())
103             }
104
105
106
107
108             this.joueur = hero
109             return hero
110
111
112         }
113
114     }
```

Mission 1.2 : tourDuJoueur() => Anne-Julie

Mission 1.3 : tourDuMonstre() => Lesly

```
// Méthode du tour de combat du monstre
fun tourDeMonstre() {
    var resultat = (1..100).random() //Séléctionne un nombre aléatoire
    val passe : String = "Le monstre ${monstre.nom} passe son tour"
    println("\u001B[31m---Tour de ${monstre.nom} (pv: ${monstre.pointDeVie}) ---") //Affiche le nom du monstre et son nombre de point de vie
    if (resultat <= 70){
        //Le monstre attaque le joueur
          this.monstre.attaque(this.jeu.joueur)
                println("Le monstre ${monstre.nom} a attaqué")
    }
```

Intermission 1 : Ajouter plus de monstres à la liste de monstres du jeu

```
val gnoll = Personnage(
    "le gnoll",
    pointDeVie = 20,
    pointDeVieMax = 25,
    attaque = 11,
    defense = 8,
    vitesse = 14,
    endurance = 11,
    armePrincipal = arme2,
    armure = null,
    inventaire = mutableListOf(bombe1),
)
```

```
val homme_lezard = Personnage(
    "l'homme-lèzard",
    pointDeVie = 20,
    pointDeVieMax = 22,
    attaque = 11,
    defense = 13,
    vitesse = 11,
    endurance = 11,
    armePrincipal = arme3,
    armure = null,
)
```

```
val armure_animee = Personnage(
    "l'armure-animée",
    pointDeVie = 20,
    pointDeVieMax = 33,
    attaque = 8,
    defense = 15,
    vitesse = 6,
    endurance = 8,
    armePrincipal = null,
    armure = armure2,
)
```

Mission 2

Mission 2.1 : les armes et types d'Armes => Lesly

```
package item


class TypeArme (
    val nombreDes: Int,
    val valeurDeMax: Int,
    val multiplicateurCritique: Int,
    val activationCritique: Int,
)
```

```
package item
import jeu.TirageDes
import personnage.Personnage

class Arme constructor(nom: String, description: String, val qualite: Qualite, val typeArme: TypeArme):Item(nom,description){


    fun calculerDegat():Int{//Ne prend pas de paramètres, retourne un entier.
            var desDegat = TirageDes(this.typeArme.nombreDes, this.typeArme.valeurDeMax).lance()
            var result = desDegat
            val desCritique = TirageDes(1, 20).lance()

        if(desCritique>=this.typeArme.activationCritique){

                var critique = result * this.typeArme.multiplictaeurCritique
                var degatsTotal = critique + this.qualite.bonusRarete
                return degatsTotal

        } else {
                var degatsTotal = result+this.qualite.bonusRarete //on ajoute le bonusRarete de la qualité de l'arme.
                return degatsTotal
        }

    }

    override fun utiliser(cible: Personnage)  { //Prend un paramètre cible de type Personnage et ne retourne rien.
        cible.equipe(this) //*la cible (paramètre) equipe l'arme en utilisant la méthode equipe() de la classe Personnage.//
    }
}
```

Mission 2.2 : Les armures et types d'armures => Océane

```
package item

class TypeArmure constructor(
    var name: String,
    val bonusType:Int,


)
```

```
package item

import personnage.Personnage

class Armure constructor( nom: String, description: String, val qualite: Qualite, val typeArmure: TypeArmure):Item(nom, description)
 {
    fun calculProtection(): Int {
        var protection = this.typeArmure.bonusType + this.qualite.bonusRarete // = Bonus de protection
        return protection
    }

     override fun utiliser(cible: Personnage) {
     }

}
```

Mission 2.3 : les potions et bombes => Anne-Julie

```
package item

import jeu.TirageDes
import personnage.Personnage

class Bombe (val nombreDeDes:Int, val maxDes:Int, nom:String, description:String):Item(nom, description){
    override fun utiliser(cible:Personnage){
        val des = TirageDes(this.nombreDeDes, this.maxDes) //création un objet de la classe TirageDes pour simuler le lancer de dés de dégâts
        val degats = des.lance()
        val protectionCible = cible.calculeDefense()

        val degatsPostProtection = maxOf(degats - protectionCible, 1) //soustraction du bonus de protection de la cible
        cible.pointDeVie -= degats //on retire au pv de la cible le nombre de points de dégâts.
        println("La bombe a fait $degatsPostProtection points de dégâts à la cible $cible.")
    }
```

Intermission 2 : Ajouter un attribut armePrincipale au personnage et un attribut armure au personnage.

```
open class Personnage(
    val nom: String,
    var pointDeVie: Int,
    val pointDeVieMax: Int,
    var attaque: Int,
    var defense: Int,
    var endurance: Int,
    var vitesse: Int,
    var armePrincipal: Arme?,
    var armure: Armure?,
```

Mission 3

Mission 3.1 : test unitaire pour calculerDegats() => Lesly

```
package item

import org.junit.jupiter.api.Assertions
import org.junit.jupiter.api.Test

import org.junit.jupiter.api.Assertions.*
import qualiteLegendaire
import typearme1

class ArmeTest {

    @Test
    fun calculerDegat() {
        var arme1 =Arme("Légendaire","Une dague légendaire en mithril ",qualiteLegendaire ,typearme1)
        var result:Int=arme1.calculerDegat();
        println(result)

        Assertions.assertTrue(result>=1)
        Assertions.assertTrue(result<=20)
    }
    }
```

Mission 3.2 : test unitaire pour calculProtection() => Océane

```
10  class ArmureTest {
11
12      @Test
13      fun calculProtection() {
14          var armure = Armure("cotte de maille", "Cotte de mailles plus lourde mais aussi plus solide",qualiteRare,typearmure6)
15      var result=armure.calculProtection();
16          Assertions.assertEquals(7,result)
17      }
18  }
```

Mission 3.3 : test unitaire pour la méthode utiliser() de la classe Bombe=> Anne-Julie

```
class BombeTest {

    @Test
    fun utiliser() {
        val homme_lezard = Personnage(
            "l'homme-lèzard",
            pointDeVie = 20,
            pointDeVieMax = 22,
            attaque = 11,
            defense = 13,
            vitesse = 11,
            endurance = 11,
            armePrincipal = null,
            armure = null,
        )
        val grenade = Bombe(
            5,
            6,
            "Grenade",
            "Une contraception qui explose une fois lancée",
        )
        grenade.utiliser(homme_lezard)
        var degat=20-homme_lezard.pointDeVie
        Assertions.assertTrue(degat>=1)
        Assertions.assertTrue(degat<=24)

    }
}
```

Intermission 3 : Créer une classe Item/ Dans la classe Personnage ajouter un attribut inventaire qui est une liste mutable d'item.

```
open class Item (val nom: String, val description: String){
    open fun utiliser(cible:Personnage){
    }
```

```
open class Personnage(
    val nom: String,
    var pointDeVie: Int,
    val pointDeVieMax: Int,
    var attaque: Int,
    var defense: Int,
    var endurance: Int,
    var vitesse: Int,
    var armePrincipal: Arme?,
    var armure: Armure?,
    var inventaire: MutableList<Item> = mutableListOf(),
) {
```

Mission 4

Mission 4.1 : Faire hériter la classe Arme de la classe Item => Lesly

```
package item
import jeu.TirageDes
import personnage.Personnage

class Arme constructor(nom: String, description: String, val qualite: Qualite, val typeArme: TypeArme):Item(nom,description)
```

Mission 4.2 : Faire hériter la classe Armure de la classe Item=> Océane

```
1   package item
2
3   import personnage.Personnage
4
5   class Armure constructor( nom: String, description: String, val qualite: Qualite, val typeArmure: TypeArmure):Item(nom, description)
6    {
```

Mission 4.3 : Faire hériter les classes Potion et Bombe de Item => Anne-Julie

```
class Bombe (val nombreDeDes:Int, val maxDes:Int, nom:String, description:String):Item(nom, description){
class Potion ( val soin:Int, nom:String, description:String):Item (nom, description){
```

Intermission 4 : Créer des armes, armures, potions et bombes.

```
//création des types d'armes
val typearme1 = TypeArme(1,4,3,18)
val typearme2 = TypeArme(1,8,2,19)
val typearme3 = TypeArme(1,8,3,20)
val typearme4 = TypeArme(1,8,3,20)
val typearme5 = TypeArme(1,8,3,20)

//création des armes
val arme1 =Arme("Légendaire","Une dague légendaire en mithril ",qualiteLegendaire ,typearme1)
val arme2 = Arme("Epée longue du droit","Une épée en fer froid",qualiteRare,typearme2)
val arme3 = Arme("Lance du kobold","Une lance rudimentaire",qualiteCommun,typearme3)
val arme4 = Arme("Hache +2","Une hache tranchante",qualiteEpic,typearme4)
val arme5 = Arme("Tonnerre","Un marteau légendaire frappe comme la foudre",qualiteLegendaire ,typearme5)
```

```
//création des types d'armure
val typearmure1= TypeArmure("Rembourré",1)
val typearmure2= TypeArmure("Cuir",2)
val typearmure3= TypeArmure("Cuir clouté",3)
val typearmure4= TypeArmure("Chemise à chaîne",4)
val typearmure5= TypeArmure("Pectoral",5)
val typearmure6= TypeArmure("Cotte de mailles",6)

//création des armures
val armure1= Armure("cotte de maille", "Cotte de mailles plus lourde mais aussi plus solide",qualiteRare,typearmure6)
val armure2= Armure("Le manteau de la nuit","Armure en cuire obscure comme la nuit",qualiteEpic,typearmure2)
val armure3= Armure("Armure du gobelin","Armure en cuir rudimentaire",qualiteCommun,typearmure3)
```

```
val bombe1 = Bombe(4,6,"Feu grégeois","Une flasque qui contient un liquide inflammable.")
val bombe2 = Bombe(2,8,"Flasque dacide","Une flasque qui contient un liquide inflammable.")
val bombe3 = Bombe(5,6,"Grenade","Une flasque qui contient un liquide inflammable.")

//création des potions
val potion1 = Potion(20,"Potion de soins","Une potion qui contient un liquide rouge.")
val potion2 = Potion(30,"Grande potion de soins","Une contraception qui explose une fois lancée.")
```

Mission 5

Mission 5.1 :méthodes attaque() et equipe() =>Lesly

```
// Méthode pour attaquer un adversaire
open fun attaque(adversaire: Personnage) {
    //TODO Mission 4.1
    var degats = this.attaque /2 //calcul les degats de base
    if(armePrincipal != null) { //vérifie s'il a une arme équipée.
        degats += armePrincipal!!.calculerDegat() //Les dégâts infligés à l'adversaire sont calculés.
    }
    degats-= adversaire.calculeDefense()
    if (degats< 1){
        degats = 1
    }
    adversaire.pointDeVie -= degats //on retire des point de vie à l'adversaire
    println("$nom attaque ${adversaire.nom} avec ${this.armePrincipal!!} et inflige $degats points de dégâts.")
}
```

```
// Methode pour équiper le personnage d'une arme
open fun equipe(arme: Arme) { //prend en paramètre une arme.
    if (arme in inventaire) { //vérifie si l'arme (paramètre) est dans l'inventaire du personnage
        armePrincipal == arme //alors elle devient l'arme principale
    }
    println("${this.nom} equipe ${this.armePrincipal}")

}
```

```
override fun utiliser(cible: Personnage)  { //Prend un paramètre cible de type Personnage et ne retourne rien.
    cible.equipe(this) //*la cible (paramètre) equipe l'arme en utilisant la méthode equipe() de la classe Personnage.//
}
}
```


Mission 5.2 : méthodes calculTotalDefense() et equipe() => Océane

```
50      fun calculeDefense(): Int {
51          //TODO Mission 4.2
52          val result = this.defense / 2
53          if (this.armure != null){
54              defense += armure!!.calculProtection() //ajout bonus de l'armure en utilisant la méthode calculProtection()
55          }
56              return result;   // retourne resultat
57          }
```

```
33      // Methode pour équiper le personnage d'une armure
34      fun equipe(armure: Armure) {
35          if(armure in inventaire){
36              this.armure = armure
37          }
38  //          for (item in inventaire) {
39  //              if (item is Armure) {
40  //                  if (armure == item) {
41  //                      this.armure = armure
42  //                  }
43  //
44  //              }
45  //          }
46          println("${this.nom} equipe ${this.armure}")
47      }
```

Mission 5.3 : méthodes avoirPotion(), avoirBombe(),boirePotion() => Anne-Julie

```
// Méthode pour avoir une bombe
fun avoirBombe(): Boolean {
    for (item in inventaire) {
        if (item is Bombe) {
            return true
        }
    }
    return false
}
```

```
// Méthode pour avoir une potion
fun avoirPotion(): Boolean {
    for (item in inventaire) {
        if (item is Potion) {
            return true
        }
    }
    return false
}
```

```
// Méthode pour boire une potion
fun boirePotion(potion: Potion? = null) {
    var laPotion: Potion? = null
    if (avoirPotion() == true) { //Appelle la fonction avoirPotion
        if (potion == null) {
            for (item in inventaire) {
                if (item is Potion) {
                    laPotion = item
                    break //Arrête la boucle à la première potion
                }
            }
        } else {
            laPotion = potion
        }
        var boire = laPotion!!.soin //Le personnage récupère le montant de soin de la potion
        pointDeVie += boire
        //Le personnage met à jour ses points de vie en ajoutant le montant de soin de la potion. Si
        //les points de vie dépassent le maximum, ils sont ajustés au maximum
        if (pointDeVie > pointDeVieMax) {
            pointDeVie = pointDeVieMax
        }
        inventaire.remove(laPotion) //Le personnage retire la potion de son inventaire
        println("Vous avez bu la potion ${laPotion.nom} et avez récupéré $boire points de vie.")
    } else {
        println("Vous n'avez pas de potion dans votre inventaire.")
    }
}
```

Mission 6

Mission 6.1 : méthodes afficheInventaire() et loot() => Océane

```
123         fun afficheInventaire(){
124             println("inventaire de ${this.nom}")//Afficher chaque item de l'inventaire du personnage.
125              for (item in inventaire) {
126                  println(" ${inventaire.indexOf(item)} => ${item.nom}")//Afficher l'index de chaque élément de l'inventaire.
127              }
128         }
129
130         fun loot(cible:Personnage){
131             if (cible.pointDeVie <= 0){ //On vérifie que la cible a des pv inférieure ou égale à 0.
132                 this.inventaire += cible.inventaire //on transfère son inventaire a l'objet courant
133                 cible.inventaire= mutableListOf()
134                 cible.armePrincipal = null //on remplace l'arme principale de la cible par null
135                 cible.armure = null//et son armure par null
136             }
137         }
```

Mission 6.2 : modification de la méthode tourJoueur() => Lesly

```
else if( choix == 1){ // permet au joueur de boire une potion
    this.jeu.joueur.boirePotion() //appelle de la fonction boirepotion
    println("\u001b[0m") //
}
```

Mission 6.3 : modification de la méthode tourMonstre() => Anne-Julie

```
//Si le monstre a une potion dans son inventaire et que ses pv sont plus bas que la moitié de ses pvmax
else if (monstre.pointDeVieMax / 2 < monstre.pointDeVie && resultat <= 80){ //alors il a une chance assez faible (10%) de boire une potion.
    //Le monstre boit la potion
    monstre.boirePotion()
}
else {
    println("$passe")
}
```

Intermission 6 : Ajouter dans la méthode tourJoueur() de la classe Combat le fait de pouvoir choisir un objet de son inventaire et de l'utiliser.

```
//Le joueur est invité à choisir une action parmi les options affichées
println("tape 0 pour attaquer, 1 pour boire une potion , 2 pour voler , 3 pour lancer un sort ou 4 pour utiliser un objet de l'inventaire : ")
```

Mission sprint II
Mission7

Mission 7.1 : Création de la classe Guerrier => Lesly

```
class Guerrier constructor(nom:String,
                           pointDeVie:Int,
                           pointDeVieMax:Int,
                           attaque:Int, defense:Int,
                           endurance:Int, vitesse:Int,
                           armePrincipal:Arme?,
                           var armeSecondaire:Arme?,
                           armure:Armure?,
                           inventaire:MutableList<Item>)
    :Personnage(nom, pointDeVie, pointDeVieMax, attaque, defense, endurance, vitesse, armePrincipal, armure, inventaire){
    override fun toString(): String {
        return super.toString()
    }
```

Mission 7.2 : Création de la classe Voleur => Anne-Julie

```
package personnage

import ...

class Voleur (nom:String, pointDeVie:Int, pointDeVieMax:Int, attaque:Int, defense:Int, endurance:Int, vitesse:Int, armePrincipal:Arme?, armure:Armure?,
              inventaire:MutableList<Item>): Personnage(nom, pointDeVie, pointDeVieMax, attaque, defense, endurance, vitesse, armePrincipal, armure, inventaire){
    override fun toString(): String {
        return super.toString()
    }
```

Mission 7.3 : Création de la classe Mage et Sort => Océane

```
package personnage

import ...

class Mage constructor( nom: String, pointDeVie: Int, pointDeVieMax: Int, attaque: Int, defense: Int, endurance: Int,
                        vitesse: Int, armePrincipal: Arme?, armure: Armure?, inventaire: MutableList<Item> = mutableListOf(),var grimoire:MutableList<Sort> = mutableListOf())
    : Personnage(nom, pointDeVie, pointDeVieMax, attaque, defense, endurance, vitesse, armePrincipal, armure, inventaire) {
```

```
package personnage

import jeu.TirageDes


class Sort constructor(val nom: String,val effet:(Personnage,Personnage)->Unit ){

}
```

Intermission 7 : Lors de la création du personnage (joueur), on lui demande de choisir la classe de son personnage, en fonction du choix de l'utilisateur, il faut instancier le bon type de personnage (Mage, Voleur, Guerrier).

```
var choix_classe:Int = readln().toInt()
var hero = Personnage(nom, pv, pv, scoreAtk, scoreDef, scoreEND, scoreVIT,null,null)
if (choix_classe == 0){ //En fonction du choix de l'utilisateur, il faut instancier le bon type de personnage
    hero = Voleur(nom, pv, pv, scoreAtk, scoreDef, scoreEND, scoreVIT,null,null, mutableListOf())
}
else if (choix_classe == 1) {
    hero = Mage(nom, pv, pv, scoreAtk, scoreDef, scoreEND, scoreVIT,null,null, mutableListOf(),mutableListOf(projectionAcide,guerison,bouleDeFeu,missileMagique,armeMagique,armureMagique)

   // hero.armure=armure1
}
else {
    hero = Guerrier(nom, pv, pv, scoreAtk, scoreDef, scoreEND, scoreVIT,null,null,null,mutableListOf())
}
```

Mission 8

Mission 8.1 : Méthodes toString() equipe() attaque() => Lesly

```
override fun toString(): String {
    return super.toString()
}

override fun equipe(arme: Arme){ //permet de choisir l'emplacement de l'arme
    println("choisissez l'emplacement de l'arme ")
    println("Tapez 0 pour l'avoir en arme principal ")
    println("Ou Tapez 1 pour l'avoir en arme secondaire ")

    val choix_arme :Int = readln().toInt()
     if (choix_arme == 0) {
          super.equipe(arme) //lance la fonction equipe() de la classe Personnage
    }
    else {
         /**vérifie si l'arme (paramètre) est dans l'inventaire du personnage
          * si c'est le cas, alors elle devient l'arme secondaire
          */
         if (arme in inventaire) {
             armeSecondaire = arme
            println("${this.armeSecondaire}est devenue votre arme secondaire")
         }
    }
}
```

```
override fun attaque(adversaire: Personnage){
    var degat = this.attaque/2

    super.attaque(adversaire) //lance la fonction attaque() de la classe Personnage

    if (armeSecondaire!= null){
        /**vérifie si on a une arme secondaire
        **si c'est le cas on calcul les dégats causé par celle-ci*/
        degat+= this.armeSecondaire!!.calculerDegat()
    }
    degat = degat - adversaire.calculeDefense()

    if (degat<1){
        degat=1 //un score de degat sera quand même infligé même si celui-ci est inférieur à 1
    }
    println("$nom attaque ${adversaire.nom} avec ${this.armeSecondaire!!} et inflige $degat points de dégâts.")

}
}
```

Mission 8.2 : Méthode toString() et voler() => Anne-Julie

```
override fun toString(): String {
    return super.toString()
}

fun voler(cible:Personnage){
    var objectVolee:Item? = null
    if (cible.inventaire.size > 0){
        for (item in inventaire){
            if(item == armePrincipal){
                objectVolee = item
                this.inventaire+=objectVolee
                inventaire.remove(objectVolee)
                cible.armePrincipal = null
                break
                println("Vous avez volé(e) ${objectVolee} de ${cible}")
            }
            else if(item == armure){
                break
                objectVolee = item
                this.inventaire+=objectVolee
                inventaire.remove(objectVolee)
                cible.armure = null
                println("Vous avez volé(e) ${objectVolee} de ${cible}")
            }
            else {
                break
                objectVolee = item
                this.inventaire+=objectVolee
                inventaire.remove(objectVolee)
                println("Vous avez volé(e) ${objectVolee} de ${cible}")
            }
        }
    }
    else{
        println("L'inventaire de ${cible} est vide.")
    }
}
```

Mission 8.3 : Méthode toString(), afficherGrimoire() et choisirEtLancerSort() => Océane

```
10        override fun toString(): String {
11            return super.toString()
12        }
13        fun afficheGrimoire() {
14            println("Grimoire ${this.nom}") //affiche tous les sorts présents dans le grimoire
15            for(unSort in this.grimoire) {
16                println("${grimoire.indexOf(unSort)} => ${unSort.nom}")// affiche l'index de chaque sort
17            }
18        }
19        fun choisirEtLancersort(monstre:Personnage) { //Invoquer la méthode afficherGrimoire() puis demander à l'utilisateur de choisir un sort.
20            this.afficheGrimoire()
21            println("choisis une cible: Tape 1 Pour joueur; 2 pour monstre")//l'utilisateur choisis la cible du sort
22            var choixsort = readln().toInt()
23            println("choix du sort: Tape 0 = Boule de feu; 1 = Sort de soins") //l'utilisateur choisis un sort.
24            this.grimoire[choixsort].effet(this,monstre)
25            for (choixSort in this.grimoire) {
26            }
27            println("${afficheGrimoire()} choisir ${this.grimoire}")
28        }
29    }
```

Intermission 8 : Modifier la méthode tourJoueur() (dans la classe Combat) en ajoutant comme actions : voler et lancer un sort.

```
//Le joueur est invité à choisir une action parmi les options affichées
println("tape 0 pour attaquer, 1 pour boire une potion ,2 pour voler , 3 pour lancer un sort ou 4 pour utiliser un objet de l'inventaire : ")
```

```
else if( choix == 2){ // permet au joueur de voler la cible
    var leVoleur= this.jeu.joueur as Voleur //fonctionne que si le joueur est un voleur
    leVoleur.voler(monstre)//appelle de la fonction voler
    println("\u001b[0m") //
}
else if (choix == 3) { //permet au joueur de lancer un sort
    var leMage= this.jeu.joueur as Mage //fonctionne que si le joueur est un mage
    leMage.choisirEtLancersort(monstre)//appelle de la fonction choisirEtLancersort
    println("\u001b[0m") //
}
```

Mission 9

Mission 9.1 : Création de sorts : Boule de feu, Missile magique => Lesly

```
//création du sort de boule de feu
val bouleDeFeu = Sort("Sort de boule de feu"){ joueur, cible-> Unit
    run{
        var degatCaster = joueur.attaque/3
        val des = TirageDes(1,6)//crétion de l'objet TirageDes
        var degats = des.lance()//appelle de la méthode lance()
        degats+=degatCaster
        degats = maxOf(1, degats - cible.calculeDefense())//calcul les dégats
        cible.pointDeVie -= degats//réduit les points de vie de la cible
        println("Le sort de boule de feu inflige $degats points de dégats à ${cible.nom}")
    }
}

// sort de lancement de missile
val missileMagique = Sort("Sort de missile magique"){ joueur, cible-> Unit
    run{
        val des = TirageDes(1,6)//crétion de l'objet TirageDes
        var compteur = joueur.attaque/2 // calcul le nombre de tour de la boucle

        /**Le joueur tire des missiles magiques en boucle et le nombre de tour
         * correspond à son score d'attaque divisé par deux
         */
        repeat(compteur) {
            var degats = des.lance()//appelle de la méthode lance()
            degats = maxOf(1, degats - cible.calculeDefense())//calcul les dégats
            cible.pointDeVie -= degats//réduit les points de vie de la cible
            println("Le sort de missile magique inflige $degats points de dégats à ${cible.nom}")
        }
    }
}
```

Mission 9.2 : Création de sorts : Invocation d'une arme magique, Invocation d'une armure magique => Océane

```
33          // Methode pour équiper le personnage d'une armure
34          fun equipe(armure: Armure) {
35              if(armure in inventaire){
36                  this.armure = armure
37              }
38          //      for (item in inventaire) {
39          //          if (item is Armure) {
40          //              if (armure == item) {
41          //                  this.armure = armure
42          //              }
43          //
44          //          }
45          //      }
46              println("${this.nom} equipe ${this.armure}")
47          }
```

Mission 9.3 : Création de sort : Sort de soins => Anne-Julie

```
//création du sort de guérison
val guerison = Sort("Sort de guérison"){ joueur, cible-> Unit
    run{
        val des = TirageDes(1,6) //crétion de l'objet TirageDes
        var soins = des.lance()//appelle de la méthode lance()
        soins = maxOf(1, soins + joueur.attaque / 2)
        joueur.pointDeVie += soins //application des soins à la cible
        if(joueur.pointDeVie > joueur.pointDeVieMax){
            joueur.pointDeVie = joueur.pointDeVieMax
        }
        println("Le sort de guérison donne $soins points de soin à ${joueur.nom}")
    }
}
```

Intermission 9 : Modifier la méthode creationPersonnage() en ajoutant des sorts dans le grimoire du mage

```
else if (choix_classe == 1) {
    hero = Mage(nom, pv, pv, scoreAtk, scoreDef, scoreEND, scoreVIT, armePrincipal: null, armure: null,
            mutableListOf() mutableListOf(projectionAcide,guerison,bouleDeFeu,missileMagique,armeMagique,armureMagique))
```

La mission 10 était facultative et consistait à faire les différents tests unitaires des sorts précédemment créées.

En voici un exemple :

```
@Test
fun testBouleDeFeu(){
    val gobelin = Personnage(
        "le gobelin",
        pointDeVie = 20,
        pointDeVieMax = 20,
        attaque = 5,
        defense = 4,
        vitesse = 11,
        endurance = 6,
        armePrincipal = null,
        armure = null,
    )
    val mage= Mage(
        "Le mage",
        pointDeVie = 20,
        pointDeVieMax = 20,
        attaque = 10,
        defense = 10,
        vitesse = 10,
        endurance = 10,
        armePrincipal = null,
        armure = null,
        inventaire = mutableListOf(),
        grimoire = mutableListOf(),
    )

    println(mage.inventaire.size)
    armureMagique.effet(mage,mage)
    println(mage.inventaire.size)

}
```

Partie II : Présentation des différentes difficultés rencontrées et leurs solutions

Sprint I et II

Missions Sprint I	Problème rencontrés	Solutions	Modification code
Mission 1.1: creationPerso()	Répartition des points : faire en sorte que le jeu démarre uniquement quand les 40 points sont correctement répartis. Le jeu continuait de se mettre en route même quand les 40 points n'étaient pas correctement attribués.	Faire une boucle "Do, while" avec une condition : while (totalscore < 0) val pv= 50 +(10*scoreEND)	La répartition des points se faisait en commençant par 0 mais nous avons modifier le code pour qu'une soustraction se fasse à la place de l'addition de base.
	Anne Julie était plus rapide que Lesly et Océane, elle ne pouvait pas avancer plus sur certaine mission car il fallait que l'on termine nos missions respectives ou crée des class/méthode qu'elle devait exploiter pour avancer.	Quand l'une de nous ne pouvait plus avancer (car trop d'avance ou bloquer) on s'aidait l'unes et l'autres pour pouvoir avancer	
Mission 6.1: méthodes afficheInventaire() et loot() Mission 5.1: methodes attaque() et equipe()	Nous avons rencontré des soucis pour vérifier les armes de l'inventaire ainsi qu'afficher toutes les Items de l'inventaire	Code pour vérifier une arme If (arme in inventaire) Code pour afficher tous les items de l'inventaire : println("inventaire de ${this.nom}")	
Mission Sprint II	Problèmes rencontrés	Solutions	Modification code
Mission 8.1/8.2/8.3	Gestion du temps pour crée chaque methodes	La syntaxe des méthodes est toujours pareil, il fallait faire beaucoup de copier/coller.	

Partie III : Conclusion

En conclusion, ce projet a été une expérience enrichissante qui a considérablement renforcé notre compréhension et notre expertise en Kotlin. Notre enthousiasme pour ce projet passionnant a été récompensé par l'acquisition de compétences techniques précieuses, mais également par une croissance significative sur le plan personnel. Il a mis en lumière notre aptitude au travail d'équipe, notre capacité d'organisation, notre volonté de partager des connaissances et notre solidarité face aux obstacles rencontrés tout au long du projet Kotlin-Adventure, principalement axé sur le développement de code.

Si nous avions l'opportunité de recommencer ce projet, nous ne changerions rien à notre approche, car la répartition des tâches a été optimale, et notre autonomie et cohésion de groupe étaient exemplaires. Cependant, nous envisageons d'ajouter une page de connexion au lancement du jeu pour recueillir les

données du joueur, même si la base de données n'a pas encore été créée, afin d'améliorer encore davantage l'expérience de jeu et faciliter la collecte de données à l'avenir.