BONUS 1

Description fonctionnelle

J'ai ajouté d'un nouveau type de personnage (Assassin) et de son type d'arme (Knife).
Le but était d'ajouter un personnage avec une nouvelle aptitude, l'empoisonnement.

Mais finalement j'ai voulu aller un peu plus loin et ajouter de nouvelles aptitudes également à d'autres personnages.

J'ai finalement ajouter 3 nouvelles actions, qui sont spécifiques au type de personnage joué.
Il y a donc une nouvelle étape dans le jeu qui consiste à choisir l'action à effectuer.

En plus de pouvoir attaquer et soigner, il est alors possible d'assommer avec le Colosse, d'aveugler avec le Nain et d'empoisonner avec l'Assassin.

Ces états son cumulatifs, un personnage peut donc être par exemple être empoisonné et assommé. Mais il est aussi possible pour lui de s'en débarrasser. En attendant que l'effet cesse au bout de plusieurs tours, s'il s'agit d'un mal temporaire. Ou en recevant un soin du Mage, qui en plus de lui rendre des points de vie, le débarrassera de ces maux.

Les conséquences sur le personnage atteint sont les suivantes.
- Une fois assommé le personnage ne peut plus attaquer.
- S'il est aveuglé il réussira ou non, de manière aléatoire, à porter son attaque.
- En cas d'empoisonnement il perd de la vie chaque fois qu'il attaque.

Description technique

Pour ajouter des nouvelles attaques j'ai commencé par faire un enum WeaponEffect contenant les états que peuvent donner des armes.

```
76  /// Enum listing possible bad states that can be given to enemy
77  enum WeaponEffect {
78      case Stunned, Poisonned, Blinded
79
80      /// Return the name of the Action linked to the bad state
81      var linkedAction: String {
82          switch self {
83          case .Stunned:
84              return "Stun"
85          case .Poisonned:
86              return "Poison"
87          case .Blinded:
88              return "Blind"
89          }
90      }
91
92      /// Return how long is the bad state
93      var duration: Int {
94          switch self {
95          case .Stunned:
96              return 1
97          case .Blinded:
98              return 2
99          default:
100             return 0
101         }
102     }
```

J'ai ajouté 2 variables afin de donner un nom d'attaque correspondant et une durée (supérieur à 0 si l'effet est temporaire).

Puis j'ai fait le lien avec les armes grâce à un optionnel *canGiveStatus* présent dans la déclaration de chaque arme.

```
194     /// Return an optional containing or not the name of the special action wich can do the weapon
195     var canGiveStatus: WeaponEffect? {
196         return .Stunned
197     }
```

Les personnages sont destinés à subir ces états, je leur ai donc donné une variable *status*, qui est un dictionnaire qui prendra l'effet en clé et la *duration* correspondante en valeur. Ainsi, un combattant pourra être affecté par plusieurs effets en même temps.

```
28      /// The status of the Character
29      var status: [WeaponEffect: Int] = [:]
```

Une fois ces bases posées il restait à transférer l'effet d'une arme sur un adversaire.
C'est la fonction *specialAction(),* disponible sur chaque arme grâce à une *extension,* qui s'en occupe.

```
51      /// Use the special action of the weapon
52      func specialAction(_ characterPlaying: Character, _ characterTargeted: Character) -> () {
53          guard let newStatus = self.canGiveStatus  else {
54              print("But it has no effect")
55              return
56          }
57
58          let targetName = characterPlaying.name == characterTargeted.name ? "himself" : characterTargeted.name
59          print("\(characterPlaying.name) tries to \(newStatus.linkedAction) \(targetName)")
60
61          guard (characterTargeted.status[newStatus] == nil) else {
62              print("\(characterTargeted.name) is already \(newStatus)")
63              return
64          }
65
66          guard randomSuccess(limit: 2) else {
67              print("But \(characterPlaying.name) failed")
68              return
69          }
70
71          characterTargeted.status[newStatus] = newStatus.duration
72          print("\(characterTargeted.name) is \(newStatus)")
73      }
```

Je fais quelques vérifications pour savoir si je peux bien ajouter un status au personnage ciblé.

À noter l'utilisation de la fonction *randomSuccess* qui renvoie un booléen.
Dans cette fonction je tire au sort 2 chiffres entre 0 et 1. L'attaque ne réussi que si les 2 chiffres sont égaux. J'ai mis une limite très petite au random afin que les ratés soit plutôt rares.

Si cette dernière condition est vrai, j'ajoute l'effet dans le dictionnaire *status* de la cible.

Le bonus étant prêt il fallait désormais l'incorporer au déroulement de base du jeu.
J'ai donc ajouté une étape entre la sélection du personnage à jouer, et la sélection du personnage à attaquer ou soigner.

Grace à la fonction *chooseAction()*, il est demandé au joueur l'action qu'il veut faire avec son personnage. La fonction renvoie en réponse le résultat, sur lequel je switch pour lancer l'action adéquate.

```swift
    /// Perform the battle sequence
    func runBattle() {
        repeat{
            // choose the character that will play
            let characterPlaying = chooseCharacter(fromTeam: teams[0])
            if characterPlaying.weaponUpdated == false {
                searchForBonus(character: characterPlaying)
            }

            // choose an action
            let action = chooseAction(character: characterPlaying)

            // choose the character to target
            let characterTargeted = chooseTarget(depending: characterPlaying)

            // does the playing character can make his move ?
            if characterCanMove(characterPlaying) {
                // if so, he does his action
                switch action {
                case .normal:
                    performNormalAction(characterPlaying, characterTargeted)
                case .special:
                    characterPlaying.weapon.specialAction(characterPlaying, characterTargeted)
                }
            } else {
                // if not we check if he's not dead
                if (teams[0].isDefeated) {
                    winner = teams[1]
                    return
                }
            }
```

Mais avant cela j'ai une autre vérification à faire. Il se peut que le personnage choisie pour jouer, soit lui même sous l'effet d'un ou plusieurs *status.* Je m'assure donc qu'il puisse agir grâce à la fonction *characterCanMove().*

Dans celle-ci je teste la présence de chaque status. Par exemple, si le personnage est empoisonné. J'applique l'effet du poison en enlevant 5 points de vie. Puis je teste si le personnage n'en meurt pas. Si c'est le cas il ne peut pas lancer son attaque, je sors de la fonction en renvoyant *false,* et le joueur passe son tour.

```swift
        if (character.status[.Poisonned] != nil) {
            print("\n\(character.name) is \(WeaponEffect.Poisonned)")
            character.life -= 5
            print("\(character.name) looses 5 PV")
            if character.life <= 0 {
                character.life = 0
                character.isDead = true
                print("\(character.name) is dead")
                character.status.removeValue(forKey: .Poisonned)
                teams[0].bringDeadToCemetery()
                if teams[0].lastCharacterIsMage {
                    giveAttackWeaponToMage(mage: teams[0].characters[0])
                }
                return false
            }
        }
```

BONUS 2

Description fonctionnelle

Afin d'éviter la composition d'une équipe de Mages qui ne gagnera jamais, ou pire, un combat de Mages qui ne finira jamais non plus. j'ai ajouté une restriction d'un seul Mage par équipe.

Lors de la création de la partie, une fois qu'un Mage a été sélectionné, il n'apparait plus par la suite dans la liste des personnages à ajouter dans l'équipe.

Description technique

Pour cela j'ai ajouté une variable *hasMage* de valeur *false* dans ma fonction *createTeam()* qui crée une équipe. Cette variable indique qu'il n'y a pas encore de Mage dans l'équipe.

```
88      /// Return a Team with characterics chosen by the player
89      func createTeam(index: Int) -> Team {
90          var playerName = ""
91          var charactersOfTheTeam: [Character] = []
92          var hasMage = false
93
94          repeat {
95              print("\nPlayer\(index) enter your name:")
96
97              // Check if the player entered a unique name
98              if let name = readLine()?.uppercased(), !playersNames.contains(name) {
99                  playerName = name
100             } else {
101                 print("This name has already been taken")
102             }
103         } while (playerName.isEmpty)
104
105         /// Add the name to the list of players names
106         playersNames.append(playerName)
107
108         // Create each character of the team
109         for i in 1...Game.numberOfCharactersByTeam {
110             print("\n\(playerName) - choose the character \(i) of your team:")
111
112             let newCharacter = createCharacter(typeMageForbidden: hasMage)
113             charactersOfTheTeam.append(newCharacter)
114             print("\(newCharacter.description()) joins your team")
115
116             if newCharacter.type == .Mage {
117                 hasMage = true
118             }
119         }
120
121         return Team(playerName: playerName, characters: charactersOfTheTeam)
122     }
```

Ensuite, dans une boucle permettant de créer le nombre adéquate de personnages, je passe la variable *hasMage* dans la fonction *createCharacter().*

```
122     /// Return a Character named and typed by player
123     func createCharacter(typeMageForbidden: Bool) -> Character {
124         var availableChoice = false
125         var typeOfTheCharacter =  CharacterType.Fighter
126         var nameOfTheCharacter = ""
127
128         repeat {
129             // Show all types of character available
130             for i in 0...CharacterType.count{
131                 if let type = CharacterType(rawValue: i) {
132                     if type == .Mage && typeMageForbidden {
133                         continue
134                     } else {
135                         print(String(type.rawValue + 1) + ". \(type)")
136                     }
137                 }
138             }
139
140             // Player choose the type of character he wants
141             let typeChosen = input()
142
143             // Check if the player choose an available option
144             if (typeChosen == 0 || typeChosen > CharacterType.count ||
145                 (typeChosen == (CharacterType.Mage.rawValue + 1) && typeMageForbidden))
146             {
147                 print(unavailableChoice())
148             } else {
149                 availableChoice = true
150                 typeOfTheCharacter = CharacterType(rawValue: typeChosen - 1)!
151             }
152
153         } while (!availableChoice)
```

Dans *createCharacter(),* le paramètre *typeMageForbidden* reçoit la valeur de *hasMage.*
S'il est à *true,* il empêchera d'afficher le Mage dans la liste des personnages proposés.

Enfin, je vérifie quand même que le joueur ne tape pas dans la console le numéro correspondant au Mage, et si c'est le cas je l'informe qu'il a fait un mauvais choix.

Une fois le personnage créé, je stocke l'objet dans la variable *newCharacter* et je vérifie s'il s'agit d'un Mage. Si c'est le cas je passe *hasMage* à true. Créer un Mage dans la même équipe ne sera plus possible.

BONUS 3

Description fonctionnelle

Un autre bonus spécifique au Mage. Si celui-si se retrouve seul dans son équipe, plutôt que d'attendre la fin en encaissant et se soignant à chaque tour, il récupère aléatoirement l'arme d'un de ses partenaires mort au combat afin de se battre. Cela lui laisse une petite chance de remporter la partie.

Description technique

Chaque fois qu'un personnage meurt, je vérifie si on ne rentre pas dans le cas suivant:
« Il reste un seul personnage dans l'équipe et ce personnage est un Mage ». J'ai donné cette mission à la propriété calculée *lastCharacterIsMage* qui renvoie un booléen. Quand elle répond *true,* je lance la fonction *giveAttackWeaponToMage().*

```
366     /// Perform basic action character can do
367     func performNormalAction(_ characterPlaying : Character, _ characterTargeted: Character) {
368         if (characterPlaying.weapon is Ring) {
369             characterTargeted.getHealed(by: characterPlaying)
370         } else {
371             characterTargeted.receiveDamage(from: characterPlaying)
372             if characterTargeted.isDead {
373                 teams[1].bringDeadToCemetery()
374                 if teams[1].lastCharacterIsMage {
375                     giveAttackWeaponToMage(mage: teams[1].characters[0])
376                 }
377             }
378         }
```

Cette fonction sélectionne aléatoirement une arme parmi celles portées par les partenaires décédés, puis l'attribue au Mage. Elle passe la variable *weaponUpdated* à true, pour éviter que le Mage ne reçoive une arme bonus. Cela lui redonnerai une arme plus puissante mais de type soin. Et on retomberai dans le cas du Mage qui ne peut pas se défendre.

```
381     /// Give an attack weapon to the Mage if he is the last Character of the Team
382     func giveAttackWeaponToMage(mage: Character) {
383         let randomNumber =  randomInt(max: teams[1].cemetery.count)
384         let randomPartner = teams[1].cemetery[randomNumber]
385         let newWeapon = randomPartner.weapon
386 
387         mage.weapon = newWeapon
388         mage.weaponUpdated = true
389         print("\nBut \(mage.name) doesn't intend to surrender")
390         print("He grabs the \(randomPartner.name)'s weapon")
391     }
```