Bonjour, voici la seconde présentation du Tp3 d'OpenClassrooms

vu par Frédéric

Les raisons du rejet de mon projet peut se résumer en une phrase lourde de sens.

La massification de deux bloques (le main et le PartOfGames) constitués de 300 lignes chacun, fardés de 24 variables pour la première et 31 pour l'autre sont les conséquences de résumer les classes à de simples structures, sans méthode ni notion d'héritage qui sont pourtant les fondements même de la POO.

En conclusion, mon projet était rédigé en procédural.

Les remèdes…

Conceptualiser des objets intégrant leurs propres méthodes et propriétés engendrant une réduction drastique de celles-ci.
Utiliser la force de l'héritage simplifiera le code, le rendra plus stable et évolutif.

Sommaire

- Création d'un diagramme de Classes(revisité)
- Modification de la création des Classes
- Création successive des trois étapes
- Bonus
- Conclusions
- Démonstration

Diagramme de classes initial

CLASSES MERES:

- Soldat
- Joueur
- Equipe
- Partie

CLASSES FILLES de Soldat:

- Combattant
- Mage
- Colosse
- Nain
- Cavalier

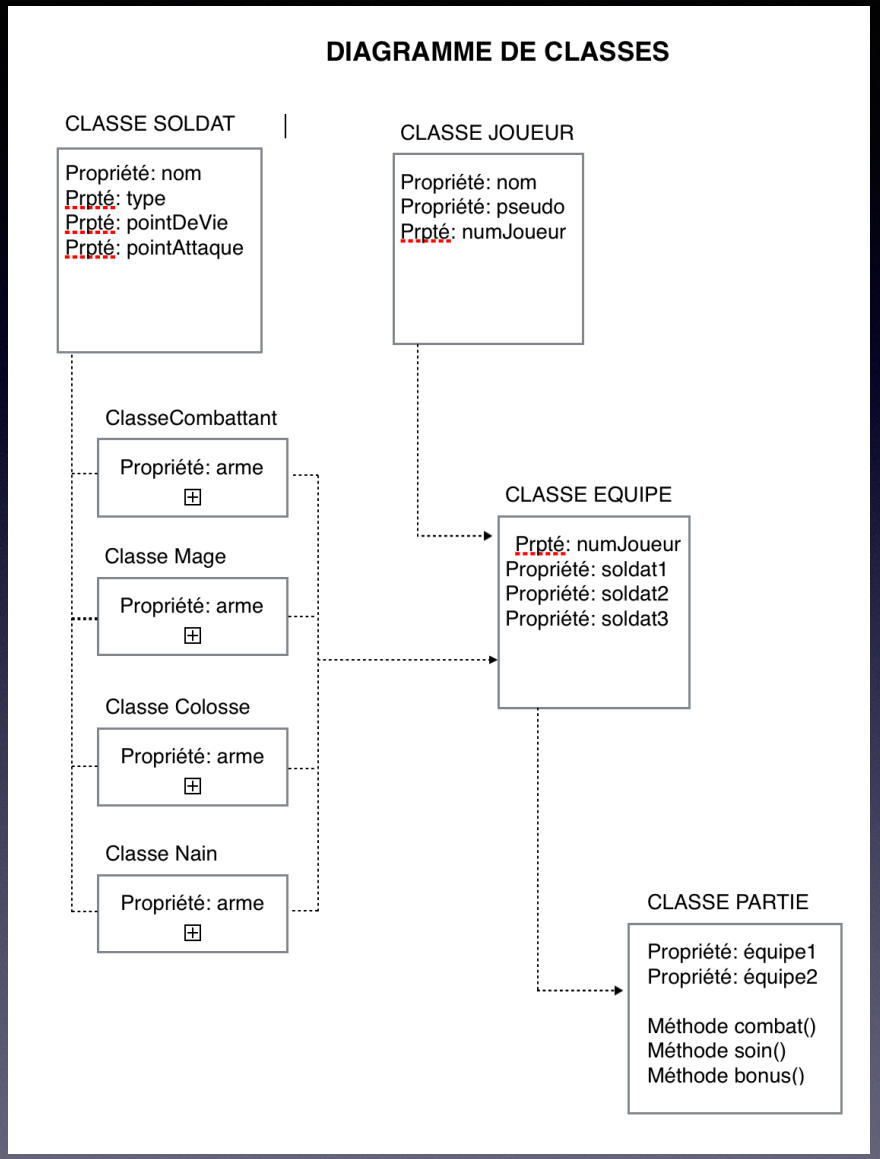
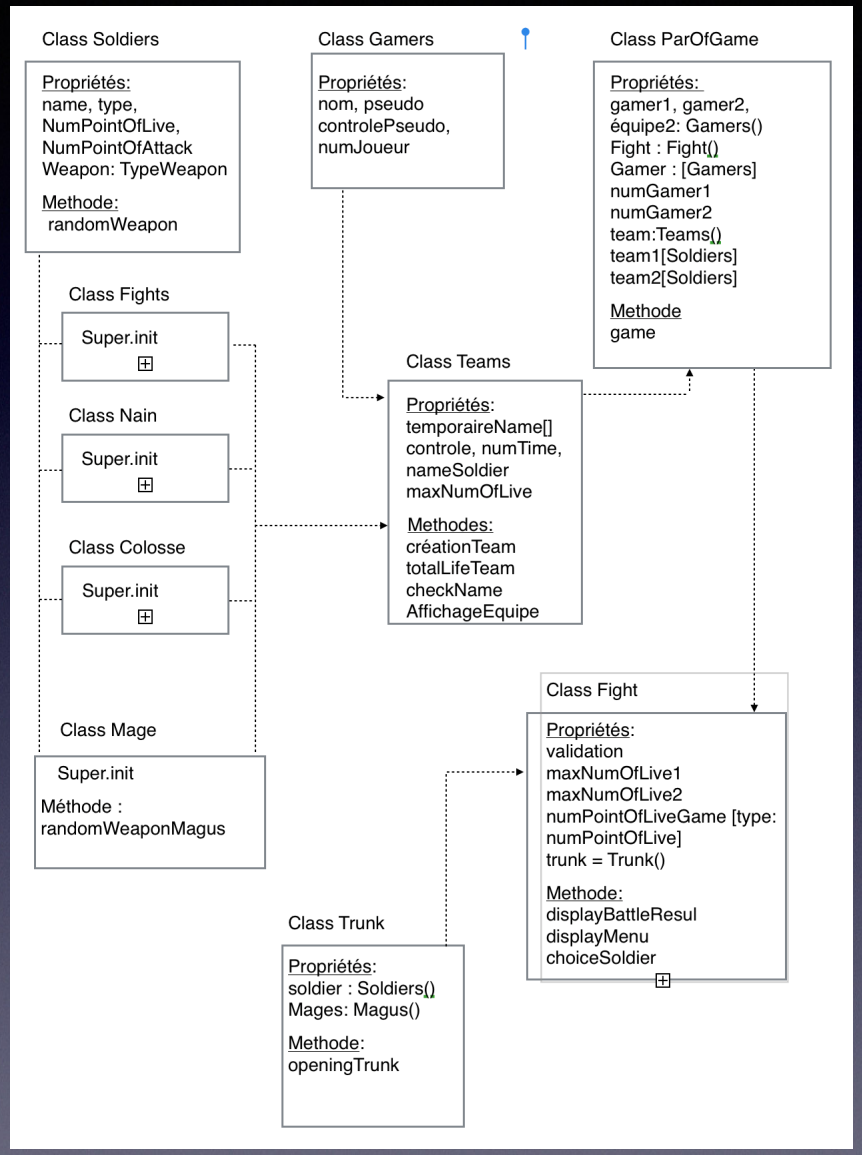


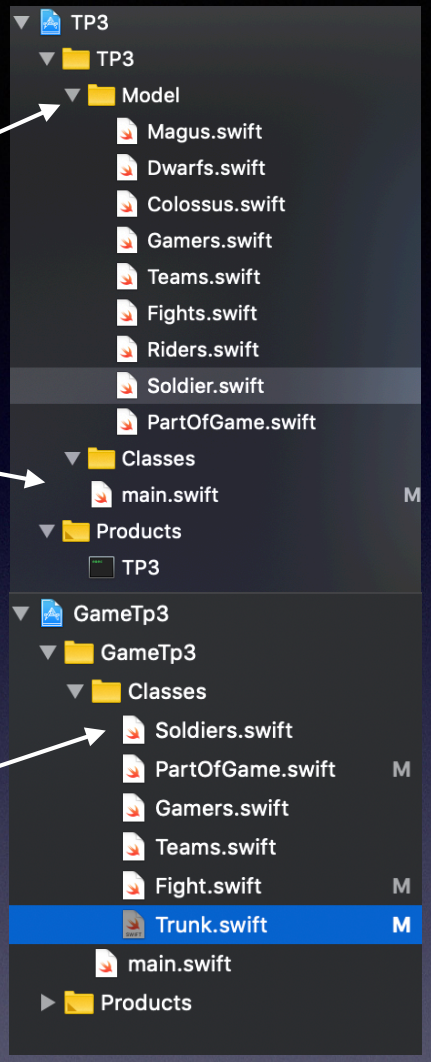
Diagramme de classes revisité

On remarque ici que les casses « main » et PartOfGames ont été allégées. On a restitué les méthodes et propriétés appropriées aux classes dédiées auxquelles on y a ajouté les classes combats et bonus.



Création des Classes

Rassemblement de toutes les classes dans le groupe Model..
Dans le groupe Classes..
que l'on aurait pu appeler Contrôleur.
On y trouve seulement le « main » qui assure les enregistrements et le contrôle des doublons tant pour les pseudos des joueurs que pour les noms des combattants



Modification des Classes

Dans le fichier Soldiers.. on a regroupé toutes les classes filles de la classe Soldiers et l'énumération des armes.
On a compléter avec les classes Fight et Trunk

Le fichier main.., continué de 5 lignes, sert seulement à lancer le code.

```
func gameMenu() {
    let partOfGame = PartOfGame()

    partOfGame.game()

}
gameMenu()
```

Créations successives des trois étapes.

- Etape 1: Les équipes
- Etape 2: Au combat !
- Etape 3: Changeons d’armes !
- Bonus

Etape 1: Les équipes

- Recueillir nom et pseudo du joueur
- Pour chaque équipe, par trois fois il faut:
  - sélectionner le soldat
  - recueillir son nom
  - enregistrer dans l'équipe dédiée

Recueillir nom et pseudo du joueur

Il semblait important d'intégrer un pseudo, car il paraissait plus aisé d'inciter le joueur à le modifier afin d'éviter les doublons, plutôt que de lui demander de changer son nom…
Est-ce une option inutile, ce n'est pas ce qui surchargera le code…

Cette méthode était domiciliée dans le main. Aujourd'hui elle est assumée par la classe Gamers en moins de 70 lignes…

```
var gamerName = ""

var controlePseudo = false
print("\nWhat is your name \(newPartOfGame.numberGamer) ? \n")
if let answerName = readLine() {
gamerName = answerName
}

repeat {
    controlePseudo = false
    print("Hello \(gamerName), what nickname did you choose to
        participate in this game ? ")

    if let answerPseudo = readLine() {
        if newPartOfGame.numberGamer == 1 {
            pseudoName = answerPseudo
            gamerPseudo1 = pseudoName
            gamer1 = Gamers(name: gamerName, pseudo: pseudoName,
                numberGamers: 1)
            print("Ok gamer \(newPartOfGame.numberGamer), you are
                called \(gamer1.name) and you will play under the
                nickname: \(gamer1.pseudo)")
        } else {
            if pseudoName == answerPseudo {
                print("This nickname is already used. Start
                    again...")
                controlePseudo = true
            } else {
                pseudoName = answerPseudo
                gamerPseudo2 = pseudoName
                gamer2 = Gamers(name: gamerName, pseudo:
                    pseudoName, numberGamers: 2)
                print("Ok gamer \(newPartOfGame.numberGamer), you
                    are called \(gamer2.name) and you will play
                    under the nickname: \(gamer2.pseudo)")
            }
        }
    }
} while controlePseudo
```

Sélectionner le soldats

La boucle « Repeat.. » assurera trois passages devant le menu afin d'assurer le recrutement des soldats devant former l'équipe.

Recueillir le nom

La méthode « readLine.. » affecte la saisie de l'opérateur dans un optionnel, si celui-ci n'est pas nul la variable « choice » est créée. Dans le if suivant le « String » est transformé en Int..

Cette méthode était domiciliée dans le main. Aujourd'hui elle est assumée par la classe Teams en moins de 90 lignes…

```
print("\nSELECT 3 SOLDIERS FROM THE ONES LISTED BELOW...")
if newPartOfGame.numberGamer == 2 {
    newPartOfGame.numberTime = 0
}
numberLoop = 0
repeat {
    print("\nWhat character number \(newPartOfGame.numberTime + 1)
        you want in your team :"
        + "\n-----------------------------------------------"
        + "\nStep : 1 to have a   fight  with 100 health and 10
            strength"
        + "\nStep : 2 to have a   magus  with  80 health and 10
            strength"
        + "\nStep : 3 to have a colossus with 120 health and  7
            strength"
        + "\nStep : 4 to have a   dwarf  with  60 health and 15
            strength"
        + "\nStep : 5 to have a   rider  with 150 health and 30
            strength")

    if let choice = readLine() {
        var choiceSoldierTeam = 0
        if Int(choice) != nil {
            choiceSoldierTeam = Int(choice)!
        }
        if choiceSoldierTeam <= 5 && choiceSoldierTeam > 0 {
            print("\nHow do you want to call it ?")

                if let name = readLine() {
                    check = false
                    warriorName = name
                    temporaryName.append(warriorName)
                    check = checkName(name: warriorName)
                    if check {
                        temporaryName.remove(at:
                            temporaryName.count-1)
                    }
                }
            switch choice {
```

Enregistrer dans l'équipe dédiée

Nous créons une instance de la classe fille seulement ici.. car nous avons tous les éléments nécessaires à l'initialisation de l'« init » hérité de la classe mère « Soldier ». Puis nous faisons appel à la fonction « recordName.. »

```
    }
switch choice {
case "1":
    if !check {

        let fight = Fights(nameSoldier: warriorName, type: "fight",
            numberPointsOfLive: 100, numberPointsOfAttack: 10)
        recordName(numberGamer: newPartOfGame.numberGamer,
            typeWarrior: fight, numberTime:
            newPartOfGame.numberTime, check: check)
        }
case "2":
    if !check {
        let magus = Magus(nameSoldier: warriorName, type: "magus",
            numberPointsOfLive: 80, numberPointsOfAttack: 10)
        recordName(numberGamer: newPartOfGame.numberGamer,
            typeWarrior: magus, numberTime:
            newPartOfGame.numberTime, check: check)
    }
case "3":
    if !check {
        let colossus = Colossus(nameSoldier: warriorName, type:
            "colossus", numberPointsOfLive: 120,
            numberPointsOfAttack: 7)
        recordName(numberGamer: newPartOfGame.numberGamer,
            typeWarrior: colossus, numberTime:
            newPartOfGame.numberTime, check: check)
        }
case "4":
    if !check {
        let dwarf = Dwarfs(nameSoldier: warriorName, type: "dwarf",
            numberPointsOfLive: 60, numberPointsOfAttack: 15)
        recordName(numberGamer: newPartOfGame.numberGamer,
            typeWarrior: dwarf, numberTime:
            newPartOfGame.numberTime, check: check)
```

Cette méthode était domiciliée dans le main. Elle n'est plus nécessaire puisqu'on l'instancie directement dans l'objet..

```
case "4":
    if !check {
        let dwarf = Dwarfs()
        dwarf.name = nameSoldier
        maxNumberOfLive += dwarf.numberPointsOfLive
        team.append(dwarf)
    }
```

La fonction « recordName »

```
//Recording function that selects the team in relation to the
    palyer's number
    func recordName (numberGamer: Int, typeWarrior: Soldiers,
        numberTime: Int, check: Bool) {
        let soldier = typeWarrior
        let gamerNumber = numberGamer
```

Cette méthode était domiciliée dans le main. Aujourd'hui elle n'est plus nécessaire puisque l'on instancie directement l'objet.

```
        if numberGamer == 2 && numberLoop == 3 {
            newPartOfGame.viewTeam(team: team2, gamerNumber:
                numberGamer)
        }
    }
```

Etape 2: Au combat !

- Sélectionner l’action.
- Opter pour un personnage.
- Choisir un combattant de l’équipe adverse.

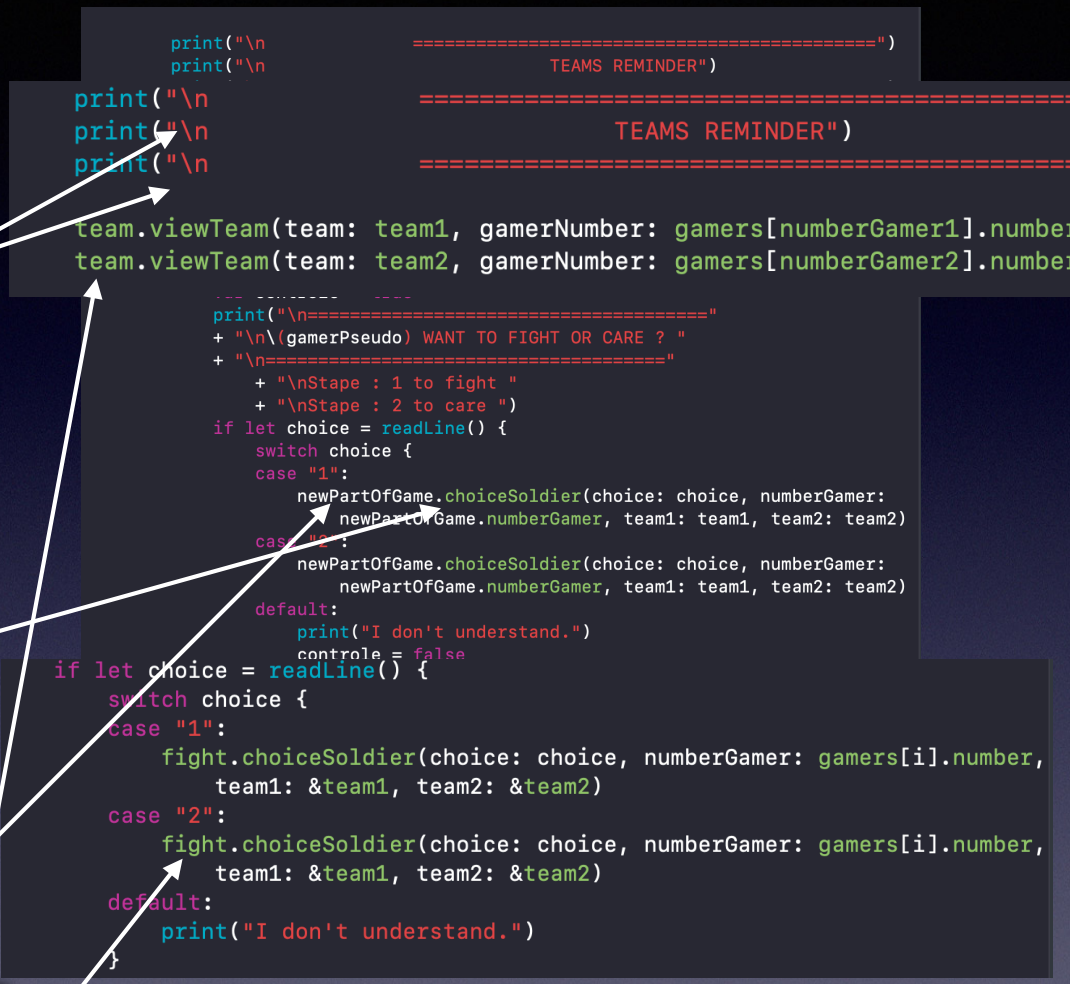
Sélectionner l'action

Ici on rappelle la constitution des deux équipes par la méthode « viewTeam.. » puis nous interrogeons le joueur sur l'action qu'il veut mener.

L'option combat.

On confie son choix à la méthode « choiceSoldier.. ». C'est deux méthodes sont positionnées dans la Classe « PartOfGame ». Nous les atteignions par son instance nommée « newPartOfGame.. »

Ces méthodes étaient domiciliées dans le main. Aujourd'hui pour la première elle intègre la classe Teams.. et la seconde appelé de PartOfGames sollicite choiseSoldier de l'instance de la classe fille Fights.



Sélectionner l’action

L’option Soigner.

Une distinction importante réside dans ce choix en effet, la sélection du premier personnage est sous-jacente puisque seul le « Mage.. » peut dispenser des soins.
On aurait pu penser unir la suite des actions « combat » et « soin » malheureusement, seule la capture de la valeur initiale est en commun puisque là où on soustrait des points dans la phase du combat on les ajoute.. dans l’option soin.

Cette méthode était domiciliée dans PartOfGame. Aujourd’hui elle intégre la classe Fight qui simplifie la lecture…

```
if choice == "2" {
    validation = true
    // Cette boucle a pour mission de controler la présence de Mage dans l'équipe
        et si celui-ci est encore en vie

    for i in 0...2 {
        if teamSwap1.team[i].type == "magus" {
            choiseInInt1 = i
        }
    }

    if teamSwap1.team[choiseInInt1].numberPointsOfLive == 0 { //if !validChoiceMage
        {
        print("You don't have a magus in your team able to act ")
        validation = false
    } else {

        // Here we capture the initial value of the attack of the soldier
        initialForce = teamSwap1.team[choiseInInt1].numberPointsOfAttack
        teamSwap1 = openingTrunk(soldier: teamSwap1, choice: choice)

        print("\nChoose the person to look after ")

        viewTeam(team: teamSwap1, gamerNumber: numberGamerSwap1)

        soldierSwap2 = displayMenu(team: teamSwap1, gamerNumber: numberGamerSwap1,
            oneTime: 2)

        for (typeSoldier, numberPointOfLive) in numberPointOfLiveGame {
            if typeSoldier == soldierSwap2.team[choiseInInt2].type {
            // Recovery of the maximum number of life points of the soldier type
                maxNumberOfLive = numberPointOfLive
            }
        }
        if soldierSwap2.team[choiseInInt2].numberPointsOfLive == maxNumberOfLive {
            print("No need to treat him he is in great shape ! ")
            validation = false
        }
        if soldierSwap2.team[choiseInInt2].numberPointsOfLive <= 0 {
            print("You arrive too late, your soldier is dead \n")
            validation = false
        } else {
            soldierSwap2.team[choiseInInt2].numberPointsOfLive +=
                teamSwap1.team[choiseInInt1].numberPointsOfAttack
} else {
    soldierSwap2.numberPointsOfLive += soldierSwap1.numberPointsOfAttack
}
```

Opter pour un personnage

Dans la méthode « choiceSoldier » nous créons des propriétés locales.. permettant le swap entre les équipes en fonction du numéro du joueur qui est reçu dans les paramètres attendus. Puis, nous capturons la valeur initiale.. de sa force d'attaque car celle-ci risque d'être modifiée plus tard dans le jeu et ainsi pouvoir la lui restituer.

```
if numberGamer == 2 {
    teamSwap1 = team2
    teamSwap2 = team1
    soldierSwap1 = soldier2
    soldierSwap2 = soldier1
    maxPointOfLiveTeamSwap1 = maxPointOfLiveTeam1
    numberGamerSwap1 = numberGamer
    numberGamerSwap2 = 2
}
        if choice == "1" {
            var validationMage = true
            var numberMage = 0
            // This function is ot count the number of mages in the team
            for i in 0...2 {
                if teamSwap1.team[i].type == "magus" {
                    numberMage += 1
                }
            }
            if numberMage == 3 {
                print("You only have mages in your team, you can't attack ! ")
                validationMage = false
            } else {

                repeat {
                    print("\nCHOOSE A MEMBER OF YOUR TEAM... ")
                    viewTeam(team: teamSwap1, gamerNumber: numberGamerSwap1)

                    soldierSwap1 = displayMenu(team: teamSwap1, gamerNumber:
                        numberGamerSwap1, oneTime: 1)

                    //Here we capture the initial value of the attack of the
                        soldier
                    initialForce =
                        soldierSwap1.team[choiseInInt1].numberPointsOfAttack

                    if soldierSwap1.team[choiseInInt1].numberPointsOfLive <= 0 {
                        print("Can't select it is dead")
                        validation = false
                    }
                    if soldierSwap1.team[choiseInInt1].type == "magus" {
                        print("CAN NOT SELECT HIM TO FIGHT, HE CAN ONLY CARE \n")
                        validationMage = false
                    }
                } while !validation
```

Choisir un combattant de l'équipe adverse

Cette partie du code contrôle l'état de santé du soldat.., affiche les deux acteurs du combat.. et réactualise la valeur totale.. du nombre de point de vie de l'équipe. Là aussi le swap évite la répétition des lignes de code. Le second screenshot.. informe du résultat. Ici il donne le nombre de point ajouté du mage..

Un avantage indéniable de la POO, elle rend un code épuré et plus accessible…

```
repeat {
    if validationMage {
        print("\nCHOOSE A MEMBER OF THE OTHER TEAM... ")
        viewTeam(team: teamSwap2, gamerNumber: numberGamerSwap2)

        soldierSwap2 = displayMenu(team: teamSwap2, gamerNumber:
            numberGamerSwap2, oneTime: 2)
        if soldierSwap2.team[choiseInInt2].numberPointsOfLive <= 0 {
            print("Can't select it is dead")
            validation = false
        }
    }
} while !validation

if validationMage {
    print("=======================================================")
    print("You just selected \(soldierSwap1.team[choiseInInt1].nameSoldier)
        with numberPointsOfAttack property \
        (soldierSwap1.team[choiseInInt1].numberPointsOfAttack)")
    print("You just selected \(soldierSwap2.team[choiseInInt2].nameSoldier)
        with numberPointsOfLife property\
        (soldierSwap2.team[choiseInInt2].numberPointsOfLive)")
    print("=======================================================\n")
}
if validationMage {
    if soldierSwap1.team[choiseInInt1].numberPointsOfAttack >
        soldierSwap2.team[choiseInInt2].numberPointsOfLive {
        maxPointOfLiveTeamSwap1 -=
            soldierSwap2.team[choiseInInt2].numberPointsOfLive
        soldierSwap2.team[choiseInInt2].numberPointsOfLive = 0
    }
    else {
        soldierSwap2.team[choiseInInt2].numberPointsOfLive -=
            soldierSwap1.team[choiseInInt1].numberPointsOfAttack
        maxPointOfLiveTeamSwap1 -=
            soldierSwap1.team[choiseInInt1].numberPointsOfAttack
    }
}
```

```
//Return of the initial strike force
if validation {
    print("Here is the result of the treatment \
        (teamSwap1.team[choiseInInt1].type) he added \
        (teamSwap1.team[choiseInInt1].numberPointsOfAttack) life")
    displayBattleResul()
}

// Retunr of the initial strike
teamSwap1.team[choiseInInt1].numberPointsOfAttack = initialForce
}
```

```
if soldierSwap1.numberPointsOfAttack > soldierSwap2.numberPointsOfLive {
    maxNumberOfLive -= soldierSwap2.numberPointsOfLive
    soldierSwap2.numberPointsOfLive = 0

} else {
    soldierSwap2.numberPointsOfLive -= soldierSwap1.numberPointsOfAttack
    maxNumberOfLive -= soldierSwap1.numberPointsOfAttack
}
```

Etape 3: Changeons d’armes

Le choix aléatoire

L'ouverture du coffre devait se manifester par un changement d'arme. L'option retenue a été le doublement de la capacité de frapper ou de soigner. Comme ce doublement doit-être éphémère, au préalable sa capacité initiale a été capturée afin de pouvoir lui restituer après ce passage et l'action qui doit en découler.
La méthode random extrait du tableau de « chest.. » une valeur de manière aléatoire. Celle-ci peut ou pas affecter la force du combattant selon les prédispositions du soldat.. à y être sensible.

Initialement domicilié dans PartOfGame cette méthode est rendu à la class Trunk qui la simplifie avec seulement 30 lignes…

```
func openingTrunk(soldier: Teams, choice: String) -> Teams {
    // Step 3 : We change the strength of the weapon in this function
    var chest = ["Club","Crossbow","Drugs","Injection","Doping"]
    let randomIndex = Int(arc4random_uniform(UInt32(chest.count)))
    let newWeapon = chest[randomIndex]
    var controle = 1
    print("\n=============================================================")
    print("                          WATCH THE CHEST OPEN !!!!!!!")
    print("=============================================================")
    print("\nHere is what came out of the chest: \(newWeapon).")
    switch soldier.team[choiseInInt1].type {

    case "magus":
        if choice == "2" {
            if newWeapon == "Drugs" || newWeapon == "Injection" {
                soldier.team[choiseInInt1].numberPointsOfAttack *= 2
            }
        }
        else {
            controle += 1
        }
    case "rider":
        if newWeapon == "Doping" {
            soldier.team[choiseInInt1].numberPointsOfAttack *= 2
        }
    case "fight":
        if newWeapon == "Club" || newWeapon == "Crossbow" {
            soldier.team[choiseInInt1].numberPointsOfAttack *= 2
        }
    case "Colossus":
        if newWeapon == "Club" || newWeapon == "Crossbow" {
            soldier.team[choiseInInt1].numberPointsOfAttack *= 2
```

```
func openingTrunk(soldier: Soldiers, choice: String) -> Soldiers {
    let newWeapon: TypeWeapon
    // Step 3 : We change the strength of the weapon in this function
    let soldier = soldier
    if soldier.type == "magus" {
        newWeapon = magus.randomWeaponMagus()
    } else {
        newWeapon = soldier.randomWeapon()
    }

    let controle = 1
    soldier.numberPointsOfAttack = newWeapon.rawValue
    print("\n=============================================================================")
    print("                    WATCH THE CHEST OPEN !!!!!!!")
    print("=============================================================================")
    print("Here's what came out of the chest: \(newWeapon) with the strong value of: \
        (newWeapon.rawValue)")

    if controle == 1 {
        print("\nHere the new strike force of \(soldier.type): \(soldier.numberPointsOfAttack)"
            + "\n-----------------------------------")
    }
    return soldier
}
```

Le Bonus

Le Bonus

- J'ai intégré un combattant super puissant appelé « Rider » malheureusement, celui-ci est moins chanceux dans la phase d'ouverture du coffre. En effet, à l'inverse des autres personnages, il n'est affecté que par un seul des éléments du tableau.

Le fait d'avoir intégré deux randoms l'approche précédente est caduc.

- J'ai intégré un compteur informant en fin de partie le nombre de loops.

- J'ai souhaité aussi améliorer le graphisme en confectionnant des bannières d'agréments.

Conclusions

- Les difficultés rencontrées
- Les éléments positifs
- Les améliorations

Les difficultés rencontrées

- Ne pas confondre hâte et précipitation. Il semble que cette dernière se soit manifestée suffisamment pour évincer la partie cruciale de l'énoncé. Bien qu'il soit très clair, celui-ci n'a pas su générer une attention suffisante pour éviter la refonte presque totale du code. Pourtant, il est précisé dans la mise en situation que nous devions démontrer notre compréhension du concept « POO » et notre connaissance de « Swift ». De plus c'est la voie que j'ai choisi….

J'ai effectué une seconde fois la révision profonde du code, c'est dommage que l'on ne m'ai pas alerté que je faisait fausse route.

Les éléments positifs

- Indéniablement, la transformation du code de la phase procédurale à la POO a été pour moi un vrai défi. L'usage du Swap a des avantages, mais révèle des difficultés dans le recueil de l'indice de positionnement dans les tableaux. De plus, la veille de le présenter, le programme s'est corrompu et m'a obligé à le reprendre au niveau que j'avais sauvegardé sur GitHub. Après près de 11 heures, celui-ci devenait opérationnel. C'est cool…

Les Améliorations

- Dans les classes filles, j'ai laissé une propriété « weapon », elle pourrait avoir pour usage de collecter les différents bonus et ainsi au bout d'un certain nombre, permettre une augmentation pérenne de sa capacité ou un doublement pendant quelques loops.

La propriété weapon n'a pas eu besoin d'attendre pour faire partie intégrante du programme par l'initialisation d'une énumération d'armes et de deux méthodes random une dans la classe Soldier et une dans l'une de ses filles la classe Magus.

Merci pour votre attention.

J’espère que cette présentation vous a plu.

Que le programme correspondra aux attendus .

DEMONSTRATION