Étapes clés – Créez votre premier jeu vidéo avec Swift !

> ## Recommandations générales
> Lorsque l'on pense à un programme orienté objet (POO), on évite souvent de foncer tête baissée dans le code car, bien souvent, cela conduit à ne pas exploiter correctement les principes de la programmation orientée objet.
>
> N'hésitez pas à vous renseigner sur les 5 grand principes de la programmation orientée objet SOLID.

Étape 1 : Concevez une ébauche de la structure du jeu

5 % d'avancement

Avant de commencer à développer le jeu, il est important de réfléchir à la structure du code. La conception d'un programme développé avec un langage orienté objet passe souvent par l'analyse du cahier des charges pour déterminer les différentes classes.

Par exemple, la phrase suivante : *Le jeu est composé de deux joueurs* :

- permet de distinguer la classe Jeu ;
- possédera deux propriétés référençant une classe Joueur.

Une fois cette étape réalisée, vous aurez :

- Un ensemble de phrases décrivant le programme, qui vont vous permettre de concevoir le code.

Recommandations :

- Essayez de décomposer votre compréhension du jeu avec l'exemple de la phrase donnée ci-dessus ("*Le jeu est composé de deux joueurs*").
- Quand vous souhaitez référencer des attributs d'un élément, utilisez la phrase : Le Personnage **possède** des points de vie, un nom et une arme. 'Possède' ici sera le mot clé pour indiquer que l'élément

précédent (*personnage*) aura les propriétés mentionnées ensuite (*des points de vie, un nom et une arme*).

- Utilisez trois phrases pour décrire votre jeu :
    1. La classe A **est composée de #** classe B *(relation d'une classe à une autre)*.
    2. La classe C **possède** des propriétés nommées XXX, YYY et ZZZ.
    3. **Il y a plusieurs types de** classe C : classe D, classe E, classe F *(héritage)*.

Par exemple, pour la phrase numéro 1 :

- Le Jeu est composé de deux Joueurs.
- Chaque Joueur est composé de trois personnages.

À vous de continuer ensuite !

Points de vigilance :

- Il n'y a pas qu'une façon unique de concevoir un même logiciel ; néanmoins les principes SOLID doivent être respectés.
- Il n'est pas question dans cette étape d'écrire des lignes de code, uniquement du texte pour décomposer le programme.

Ressources :

- Vous pouvez utiliser des outils tels que Lucidchart pour vous aider à concevoir un plan de votre code.

Étape 2 : Transposez la structure en code

15 % d'avancement

Avec la description faite à l'étape 1, réaliser le code en Swift avec le guide suivant.

Une fois cette étape réalisée, vous aurez :

- Un code transposant la description du programme.

Recommandations :

- Allez-y étape par étape en prenant chaque phrase et en la transposant en code.
- Si vous avez respecté l'utilisation des trois types de phrases à l'étape 1, vous serez en mesure de transposer le code plus simplement.

Points de vigilance :

- À cette étape, Xcode vous indiquera probablement des erreurs et des warnings, ne vous en souciez pas.

Étape 3 : Réalisez la composition des équipes

35 % d'avancement

Maintenant que la structure du programme est fonctionnelle, il faut réaliser la première partie du jeu : la composition des équipes.

Une fois cette étape réalisée, vous aurez :

- Un jeu qui vous demande de composer deux équipes, et qui affiche les deux équipes avant de quitter le programme.

Recommandations :

- Utiliser les différentes classes conçues aux étapes 1 et 2 pour générer vos équipes.
- Utiliser les **init()** quand vous le pouvez plutôt que de créer des méthodes.
- Garder en tête qu'une classe ne doit avoir qu'une seule responsabilité. Cela vous permettra de réduire le volume de code dans les classes.
  Par exemple : la création d'équipe peut se faire dans la classe qui représente l'équipe, et non dans la classe représentant le jeu.
- Vous pouvez créer des fonctions qui simplifieront votre code.
  Par exemple, n'hésitez pas à créer une fonction **readInteger()** qui aura pour responsabilité d'utiliser la fonction **readLine()**, et de caster le résultat sous forme d'entier.
- Ne vous souciez pas de l'unicité du nom des personnages pour le moment.

Points de vigilance :

- À cette étape, Xcode vous indiquera probablement des erreurs et des warnings, ne vous en souciez pas.

Ressources :

- Pour réaliser votre fonction basée sur **readLine()**, n'hésitez pas à utiliser la programmation récursive.

Étape 4 : Gérez l’unicité du noms des personnages

55 % d’avancement

Maintenant que votre jeu vous permet de composer vos équipes, vous allez vous assurer que les noms donnés à chaque personnage sont uniques.

Une fois cette étape réalisée, vous aurez :

- Un jeu composé d'équipes de personnages avec des noms uniques.

Recommandations :

- Ne vous embêtez pas à fournir en paramètre la liste des noms ou la liste des personnages de l’équipe 1 lors de la création de l’équipe 2.
- Privilégiez l’utilisation de variables statiques pour sauvegarder la liste des noms déjà utilisés.
    - Par exemple, sur la classe représentant les personnages, créez une variable statique nommée *names*, que vous lirez après chaque saisie de nom pour vous assurer qu’il n’est pas déjà donné ; le cas échéant, vous ajouterez ce nouveau nom dans la liste.

Points de vigilance :

- L’utilisation de variables globales est à proscrire, utilisez plutôt des variables statiques.

Étape 5 : Réalisez le combat

80 % d’avancement

Avant l’algorithme de combat, n'hésitez pas à écrire sur papier le déroulement d’un combat tour par tour.

Une fois cette étape réalisée, vous aurez :

- Un jeu qui s'exécute jusqu’au bout (combat), mais qui n’affiche pas le résultat final et qui n’enregistre pas encore toutes les statistiques nécessaires.

Recommandations :

- Commencer par une méthode **fight()** comme point d'entrée de votre classe représentant le jeu.
- L'exécution d'un jeu se fait souvent au travers d'une boucle d'exécution *(while)* qui sera basée sur les conditions de fin du jeu (l'équipe A ou B n'a plus de personnage vivant).
- N'oubliez pas de déléguer la responsabilité de déterminer quelque chose aux classes en question.
  Par exemple, vous pouvez créer une fonction **teamIsAlive()-> Bool** dans la classe qui représente votre joueur ou votre équipe, et qui sera utilisée par la fonction **fight()** pour savoir si le jeu continue de s'exécuter.
  Autre exemple, le fait qu'un personnage attaque un autre personnage peut être délégué à la classe qui représente le personnage, avec une fonction du genre **attack(otherCharacter)**.

Points de vigilance :

- Le code de votre fonction **fight** doit être assez concis et simple. Beaucoup de règles doivent être déléguées aux classes sous-jacentes (en règle générale, une fonction fait rarement plus de 20 lignes).

Étape 6 : Enregistrez les statistiques dans la boucle de jeu

90 % d'avancement

Maintenant que votre jeu s'exécute, il vous faut enregistrer les éléments qui vous permettront d'avoir les statistiques de votre jeu.

Une fois cette étape réalisée, vous aurez :

- Un jeu qui s'exécute jusqu'au bout, qui n'affiche aucun résultat, mais qui a enregistré toutes les statistiques nécessaires.

Recommandations :

- Pour le compteur de tours, incrémentez une variable à chaque itération de la boucle *while*.

- Pour un compteur de dégâts, vous pouvez remonter l'information des points de dégâts subis lors de l'action.
- Pour un compteur de soins, incrémentez une variable à chaque fois que l'action soin est choisie.

Points de vigilance :

- Attention à ne pas repenser complètement votre code pour les statistiques. Le code d'enregistrement des statistiques doit s'intégrer directement dans le code existant sans avoir besoin de le modifier.

Étape 7 : Affichez le résultat

100 % d'avancement

Il ne reste plus qu'à afficher le joueur qui a gagné ainsi que les statistiques recensées.

Une fois cette étape réalisée, vous aurez :

- Le jeu complètement terminé.

Recommandations :

- Réaliser une fonction spécifique dans la classe représentant votre jeu pour réaliser l'affichage des statistiques. Cela évitera de surcharger la fonction **fight()**.

Projet terminé !