Travail pratique 3 et 4

Walid Boulabiar

Date de remise du TP3 (partie 1 de ce présent travail): le jeudi 22 juin 2017
TP3 noté sur 80 et compte pour 10% de la note finale
Date de remise du TP4 (partie 2 de ce présent travail): le jeudi 13 juillet 2017
TP4 noté sur 80 et compte pour 10% de la note finale

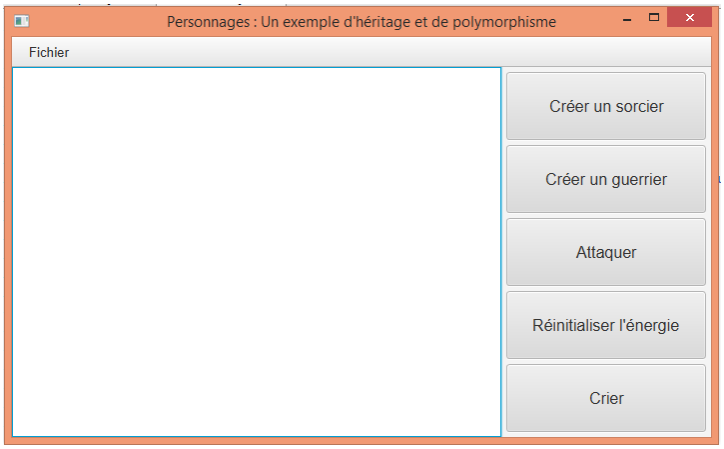
Travail en équipe de 2 pour les deux parties

1 Objectifs :

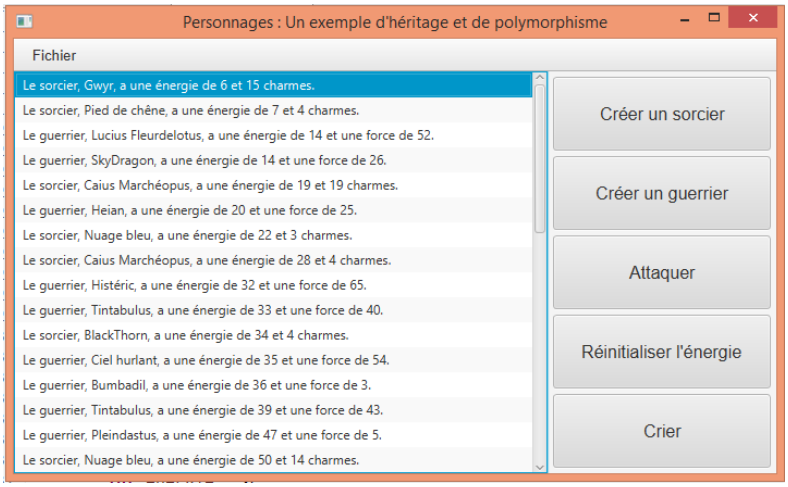
- Utiliser des classes
- Définir et utiliser des classes liées par le principe de l'héritage.
- Utiliser le principe de polymorphisme.
- Définir et utiliser des listes d'objets.
- Utiliser des composants graphiques.

2 Spécifications des programmes (jeu) :

Ce travail permet de mettre en pratique l'héritage de classes à travers une application très simple qui permet de créer et d'attaquer des personnages (Sorciers et Guerriers). Voici l'interface graphique de base lors de l'ouverture de l'application:



La même interface après la lecture de personnages à partir d'un fichier structuré :



Les actions disponibles à partir des options du menu « Fichier » sont:

Ouvrir, Enregistrer, Enregistrer Sous, Fermer et Quitter

Pour réaliser ce travail, vous devez compléter 6 classes à partir du diagramme de classe ici-bas. Les classes sont :

- La classe **GestionPersonnages** qui s'occupe de la partie gestion de l'application,
- La classe **Interface** qui crée la partie graphique. Les boutons et options de menu doivent faire appels aux méthodes de la classe GestionPersonnages
- Les classes **Personnage**, **Sorcier** et **Guerrier** qui représentent les données de l'application
- Et la classe **Util** qui renferme certaines méthodes très utiles pour la réalisation de votre application.

Vous devez compléter le projet fourni (une ébauche des scripts demandés est présente). On vous fournit aussi un modèle de classes UML (on vous offre l'image dans le zip pour pouvoir zoomer, au besoin, sur le contenu), le fichier fichierDePersonnages.txt pour importer les personnages dans votre interface grâce à l'option Ouvrir du menu Fichier et les explications suivantes :

Description des classes

Classe Personnage :

C'est une classe abstraite possédant 8 attributs, 1 constructeur et 14 méthodes

Les attributs sont:

energie_depart_defaut = 20
energie_depart_min = 1
energie_max = 100
longueur_nom_min = 3
longueur_nom_max = 30
nom = « »
energie_depart = 0
energie_courante = 0

Les méthodes sont :

Votre **constructeur** doit initialiser le nom, l'énergie de départ et l'énergie courante. À la création d'un objet personnage, l'énergie courante égale à l'énergie de départ.

Une méthode **reset_energie :** permet de remettre l'énergie courante du personnage à sa valeur de départ.

Une méthode **est_mort :** retourne vrai lorsque l'énergie du personnage est à 0.

Une méthode **valider_nom** : Un nom de personnage est valide lorsqu'il a la bonne longueur (entre min et max) bornes incluses.

Une méthode **valider_energie_depart** : L'énergie de départ est valide lorsqu'elle est entre energie_depart_min et energie_max. (bornes incluses). Il est à noter que ni l'énergie de départ ni l'énergie courante ne peuvent dépasser energie_max.

Une méthode **valider_energie_courante** : L'énergie courante doit être positive (0 inclus) et ne doit pas dépasser energie_max.

Une méthode **crier** : méthode abstraite (sans code) utile pour l'héritage, cela forcera la classe dérivée à surcharger la méthode (polymorphisme).

Une méthode **attaquer** : méthode abstraite. Vous devez écrire la définition de la méthode dans votre script mais sans contenu

Une méthode **to_string** qui retourne une chaîne du genre : nom du Personnage, a une énergie de valeur de l'énergie.

NB : n'oubliez pas d'ajouter les setter et getter pour les attributs nom, energie_depart et energie_courante (les autres attributs sont utilisés comme des constantes et leurs valeurs ne seront pas modifiées)

Classe Guerrier :

Un Guerrier est un personnage qui possède en plus une force. En plus des éléments de Personnage, cette classe possède 5 attribut, 1 constructeurs et 7 méthodes

Les attributs sont :

force_defaut = 20
force_max = 80
perte_force_defaut = 2
gain_force_defaut = 10
force = 0

La force d'un guerrier doit être positive (0 inclus) et ne doit pas dépasser force_max.

Les méthodes sont :

Votre **constructeur** doit initialiser le nom, l'énergie de départ, l'énergie courante et la force. NB : pensez à optimiser votre code et utiliser le constructeur de la classe parente.

Une méthode **crier** : retourne le cri du guerrier : "Vous allez goûter à la puissance de mon épée!"

Une méthode **attaquer** : Lorsqu'un guerrier se fait attaquer, son énergie est diminuée de la force de l'attaque. Si la force de l'attaque est plus grande que son énergie, l'énergie du guerrier devient 0 (il meurt). Lors d'une attaque, la force du guerrier est aussi modifiée. Elle est diminuée, à chaque attaque, de la valeur de perte_force_defaut jusqu'à concurrence de 0. Si le guerrier meurt pendant l'attaque, sa force est aussi mise à 0.

Une méthode **reset_energie :** permet de remettre l'énergie courante du guerrier à sa valeur de départ (héritage) et augmente sa force (la valeur de force) par la valeur de gain_force_defaut jusqu'à concurrence de la force maximale sans jamais la dépasser.

Une méthode **valider_force** : permet de valider si la force en paramètre est valide (entre 0 et force_max inclusivement). Cette méthode retourne un boolean.

Une méthode **to_string :** retourne une chaîne du genre :
Le guerrier, nom de Personnage, a une énergie de valeur de l'énergie et une force de valeur de la force.

NB : n'oubliez pas d'ajouter le setter et getter pour l'attribut force (les autres attributs sont utilisés comme des constantes et leurs valeurs ne seront pas modifiées)

Classe Sorcier :

Un Sorcier est un personnage qui possède en plus un nombre de charmes. En plus des éléments de Personnage, cette classe possède 3 attributs, 1 constructeur et 6 méthodes

Les attributs sont :

nbr_charmes_defaut = 20
nbr_charmes_max = 20
nbr_charmes = 0

Les méthodes sont :

Votre **constructeur** doit initialiser le nom, l'énergie de départ, l'énergie courante et le nombre de charmes. NB : pensez à optimiser votre code et utiliser le constructeur de la classe parente.

Une méthode **valider_nbr_charmes** : Le nombre de charmes doit être positif (0 inclus) et ne doit pas dépasser nbr_charmes_max. Cette méthode retourne un booléan.

Une méthode **crier** : retourne le cri du sorcier: "Je vais tous vous anéantir!"

Une méthode **attaquer** : lorsqu'un sorcier se fait attaquer son énergie est diminuée de la force de l'attaque. Si la force de l'attaque est plus grande que son énergie, l'énergie du sorcier devient 0 (il meurt).

Une méthode **to_string** retourne une chaîne du genre :
Le sorcier, nom de Personnage, a une énergie de, valeur de l'énergie et, valeur du nombre de charmes, charmes.

NB : n'oubliez pas d'ajouter le setter et getter pour l'attribut nbr_charmes (les autres attributs sont utilisés comme des constantes et leurs valeurs ne seront pas modifiées)

Classe GestionPersonnages :

Cette classe permet d'ajouter les personnages dans une liste et les gérer.

Voici comment doivent fonctionner chacun des boutons et chacune des options du menu Fichier.

Bouton **Créer un Sorcier** (de la classe interface): appel la méthode (**gestion_creer_sorcier**) pour créer un personnage sorcier si les informations du sorcier (méthode **saisir_et_creer_sorcier**) sont valides, on ajoute le sorcier à la liste (méthode **ajouter_personnage**) et on affiche le message approprié. Sinon, on affiche seulement que le sorcier n'a pas été ajouté.

La méthode **saisir_et_creer_sorcier** retourne un objet Sorcier valide. Chaque information du sorcier demandée doit être validée. L'annulation d'une info entraine automatiquement l'annulation des informations suivantes. Si toutes les informations sont valides, un sorcier est alors instancié.

Vous devez utiliser les méthodes fournies de la classe Util (saisir_objet_entier et saisir_string) pour effectuer toutes les saisies de votre application.

Bouton **Créer un Guerrier** (de la classe interface) : appel la méthode (**gestion_creer_guerrier**) pour créer un personnage guerrier si les informations du guerrier (méthode **saisir_et_creer_guerrier**) sont valides, on ajoute le guerrier à la liste (méthode **ajouter_personnage**) et on affiche le message approprié. Sinon, on affiche seulement que le guerrier n'a pas été ajouté.

La méthode **saisir_et_creer_guerrier** retourne un objet Guerrier valide. Chaque information du guerrier demandée doit être validée. L'annulation d'une information entraine automatiquement l'annulation des informations suivantes. Si toutes les infos sont valides, un guerrier est alors instancié.

Vous devez utiliser les méthodes de la classe Util (saisir_objet_entier et saisir_string) pour effectuer toutes les saisies de l'application.

Bouton **Attaquer** (de la classe interface): appel la méthode (**gestion_attaquer**)

Cette méthode reçoit l'indice du personnage sélectionné ou -1 si aucun personnage n'est sélectionné. Si le personnage sélectionné n'est pas mort, on saisit avec validation la force de l'attaque (> 0 et <= energie_max). Lorsque la force saisie est valide, on attaque le personnage sélectionné sinon on affiche un message adéquat. S'il n'y a aucun personnage sélectionné ou s'il est mort, un message est affiché.

Bouton **Réinitialiser l'énergie** (de la classe interface) : appel la méthode (**gestion_augmenter_energie**)

Cette méthode reçoit l'indice du personnage sélectionné ou -1 si aucun personnage n'est sélectionné. Si le personnage sélectionné n'est pas mort, réinitialiser son énergie. S'il n'y a aucun personnage sélectionné ou s'il est mort, un message personnalisé est affiché.

Bouton **Crier** (de la classe interface) : appel la méthode (**gestion_crier**)

Cette méthode reçoit l'indice du personnage sélectionné ou -1 si aucun personnage n'est sélectionné. Si le personnage sélectionné n'est pas mort, émettre son cri. S'il n'y a aucun personnage sélectionné ou s'il est mort, un message personnalisé est affiché.

Après l'exécution de chacun de ces boutons, il faut mettre à jour (méthode **mettre_a_jour_liste**) la Liste à partir de la liste triée (sort). Le tri doit se faire sur l'énergie courante.

Menu Fichier

L'option **Ouvrir…** : appel la méthode (**gestion_ouvrir**)

Permet de gérer l'ouverture et la lecture d'un fichier de personnages (un fichier .txt qui contient des informations sur des personnages, un personnage par ligne). Si la liste n'est pas vide, on demande à l'utilisateur s'il veut sauvegarder les données courantes et s'il répond oui, on fait appel à **gestion_enregistrer_sous**. Ensuite, on demande à l'utilisateur le nom du fichier à ouvrir. Si le fichier choisi n'est pas null, le fichier à ouvrir devient le fichier courant et si la lecture du fichier n'a pas bien fonctionné (voir méthode **lireFichierPersonnages** dans classe Util), un message d'erreur est affiché.

L'option **Enregistrer** : appel la méthode (**gestion_enregistrer**)

Permet de gérer l'enregistrement d'une liste de personnages dans le fichier courant. Si on a un fichier courant, on écrit les personnages de la liste dedans (voir méthode **ecrire_fichier_personnages** dans la classe Util) et on affiche un message approprié. Si l'enregistrement n'a pas fonctionné, un message d'erreur est affiché. Si on n'a pas de fichier courant, on enregistre dans un nouveau fichier en appelant la méthode (**gestion_enregistrer_sous**).

L'option **Enregistrer sous…** : appel la méthode (**gestion_enregistrer_sous**)

Permet de gérer l'enregistrement d'une liste de personnages dans un nouveau fichier. On demande un nom de fichier à l'utilisateur, on l'assigne au fichier courant et on écrit dedans les personnages (voir méthode **ecrire_fichier_personnages** dans la classe Util). Afficher un message personnalisé s'il y a erreur lors de la sauvegarde ou si la sauvegarde est ok.

L'option **Fermer** : appel la méthode (**gestion_fermer**)

Permet de fermer le fichier courant. Si la liste n'est pas vide et que l'utilisateur veut sauvegarder ses données, enregistrer les données de la liste dans le fichier courant (**gestion_enregistrer**) ou dans un nouveau fichier (**gestion_enregistrer_sous)** s'il n'y a pas de fichier courant. La liste est vidée et le fichier courant devient null.

L'option **Quitter** : appel la méthode (**gestion_quitter**)

Permet de quitter l'application après confirmation de l'utilisateur.

Classe Util:

Cette classe possède des 4 méthodes utilitaires dont 2 sont fournies et deux attributs.

Les méthodes **saisir_objet_entier** et **saisir_string** sont fournies et doivent être utilisées pour toutes les saisies de l'application.

La méthode **lire_fichier_personnages :** Permet de lire un fichier de personnages reçu en entrée et de remplir la liste de personnages. Le fichier est de format CSV et les séparateurs sont des points-virgules (;), pour plus d'information, consulter le lien suivant : https://fr.wikipedia.org/wiki/Comma-separated_values. Il faut donc ouvrir le fichier, y lire les informations de chaque personnage et ajouter chaque personnage instancié à la liste. L'ordre de lecture des données est le suivant :

- type de personnage.
- nom du personnage.
- énergie de départ.
- énergie courante
- et selon le type du personnage, la force ou les charmes.

La méthode retourne vrai si tout s'est bien passé. N'oubliez pas de fermer le fichier et de gérer les exceptions pouvant survenir lors de la lecture.

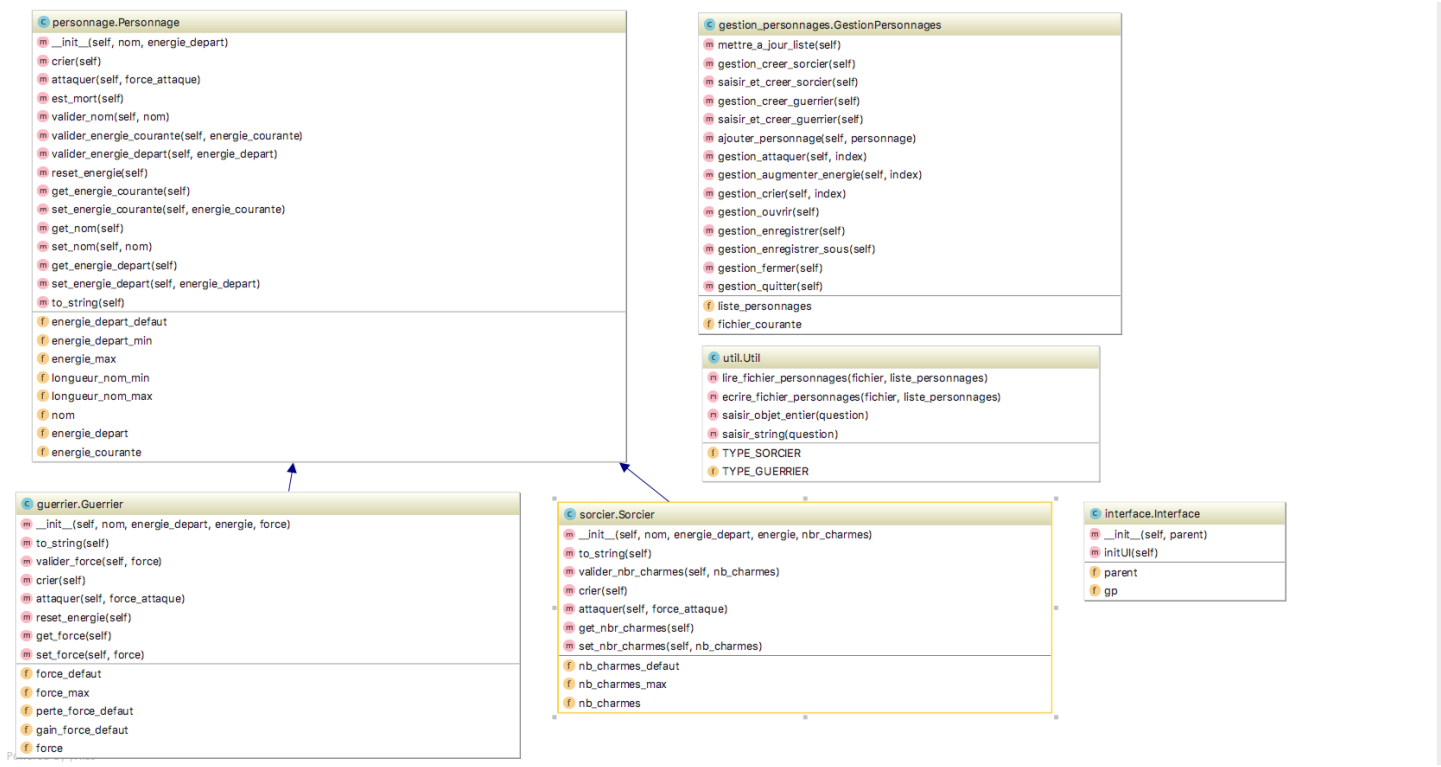
La méthode **ecrire_fichier_personnages :** Permet d'écrire la liste de personnages reçue en paramètre dans le fichier de personnages aussi reçu en entrée. Il faut donc ouvrir le fichier, y écrire chacun des personnages de la liste. Pour chacun des personnages écrire son type ("Guerrier" ou "Sorcier") au début de chaque enregistrement et ensuite ses informations. L'ordre d'écriture des données est le suivant :

- type de personnage.
- nom du personnage.
- énergie de départ.
- énergie courante
- et selon le type du personnage, la force ou les charmes.

Le fichier doit avoir le même format que le fichier « fichierDePersonnages.txt » fourni avec le travail. La méthode retourne vrai si tout s'est bien passé. N'oubliez pas de fermer le fichier et de gérer les exceptions pouvant survenir lors de l'écriture.

Diagramme de classes

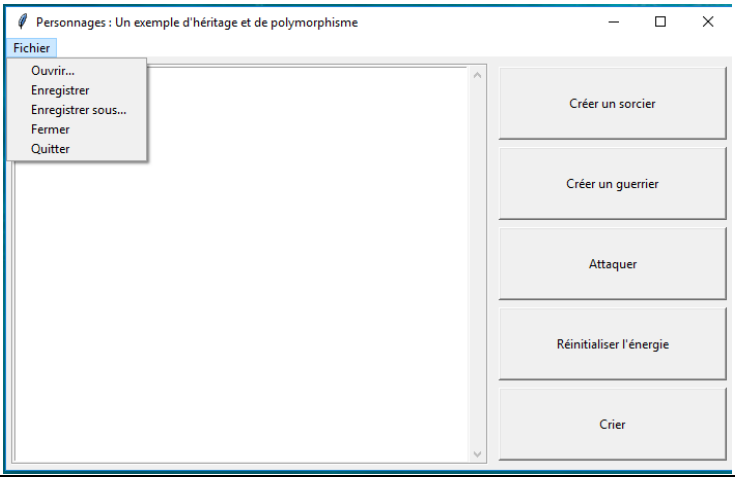
IMPORTANT : Démarrez avec les classes fournies.



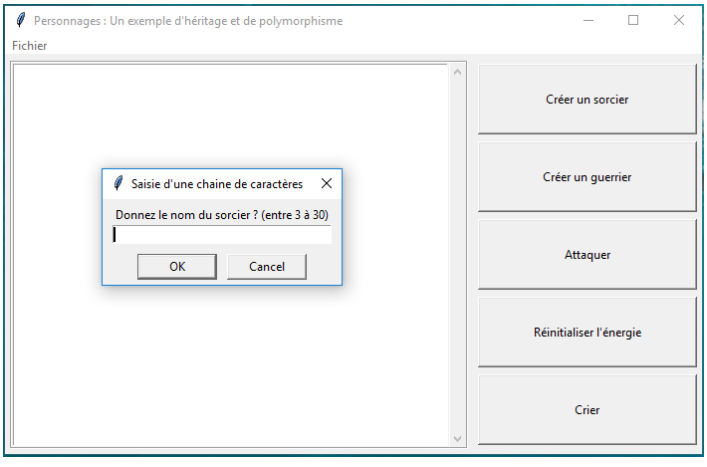
Imprimes écrans

IMPORTANT : Les messages ainsi que les détails présents dans les imprimes écrans font partie de la description de l'application.

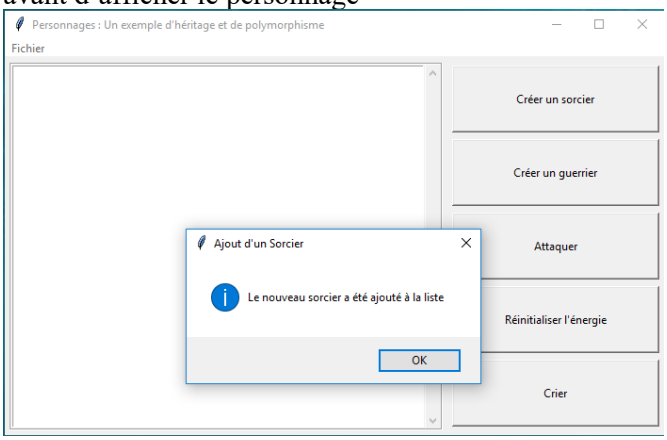
Le menu Fichier



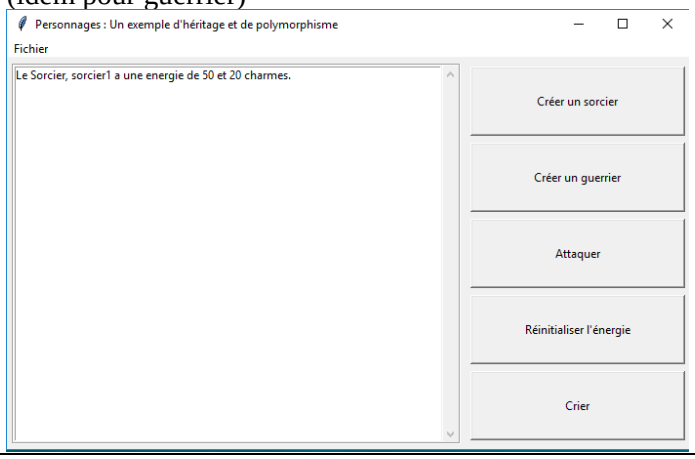
La création d'un Sourcier



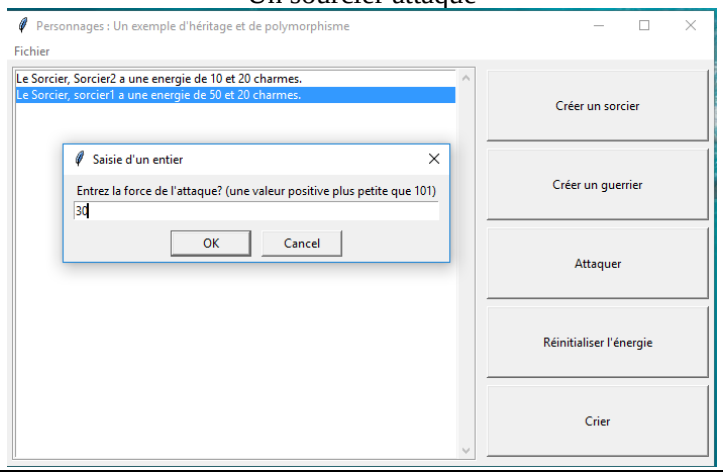
Fenêtre de validation si tous les informations sont valides avant d'afficher le personnage



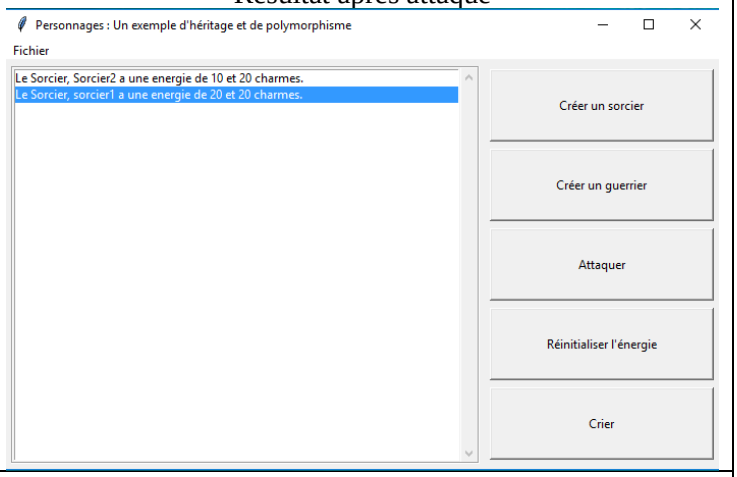
Le personnage Sourcier est affiché après le clique sur le OK (idem pour guerrier)



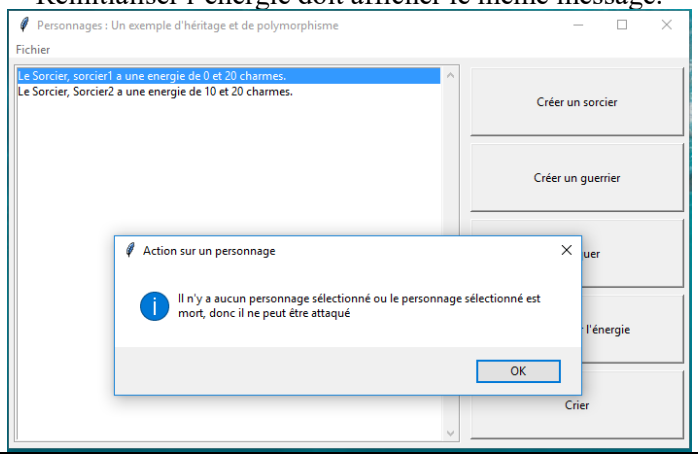
Un sourcier attaque



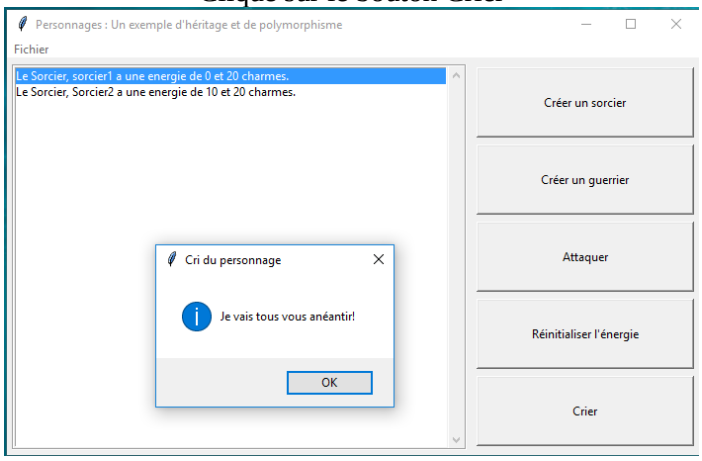
Résultat après attaque



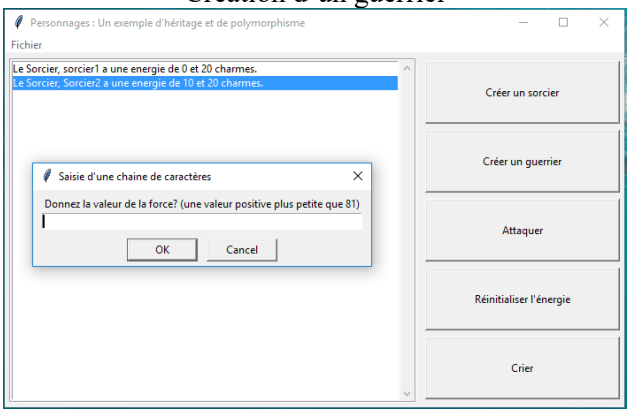
Après une attaque de sourcier1 d'une valeur de 40 (>= à son énergie), il meut mais reste affiché. Si on essaie d'attaquer de nouveau, une fenêtre comme suit doit s'afficher. Le bouton Réinitialiser l'énergie doit afficher le même message.



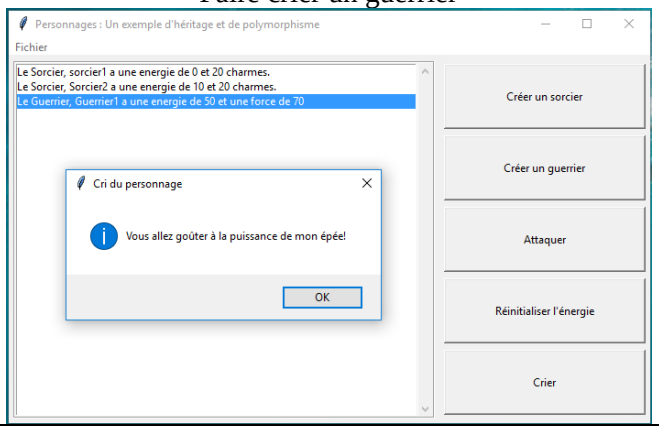
Clique sur le bouton Crier



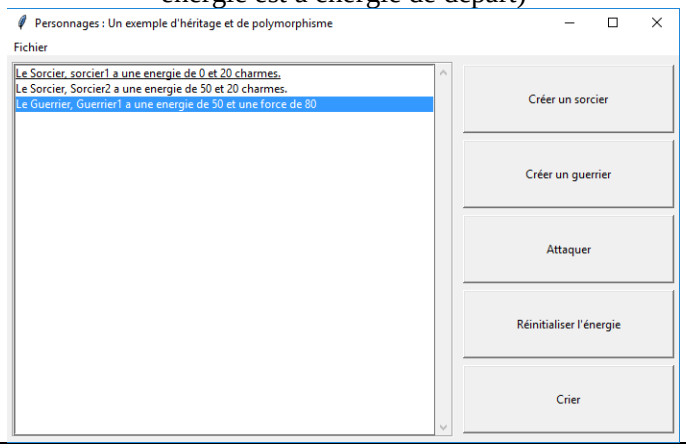
Création d'un guerrier



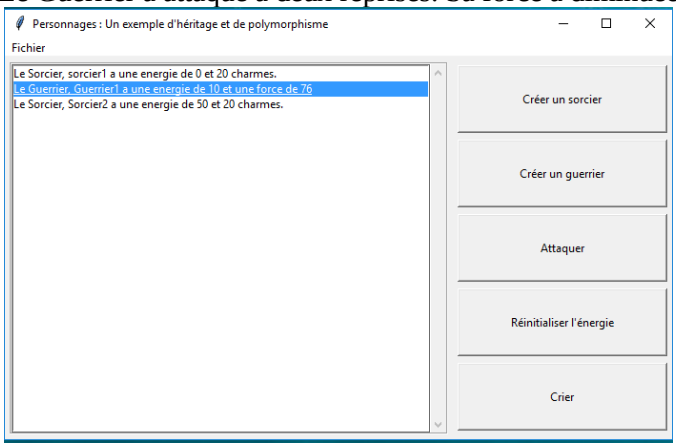
Faire crier un guerrier



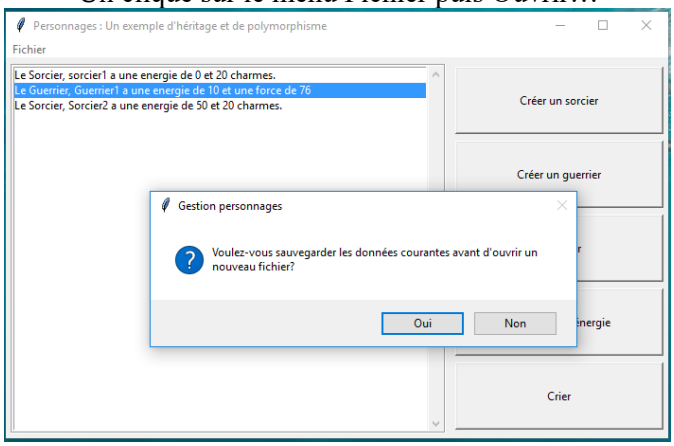
Suite au clique sur le bouton Réinitialiser l'énergie sur le Guerrier1 (sa force est rendu à 80, le max) et Sourcier2 (son énergie est à énergie de départ)



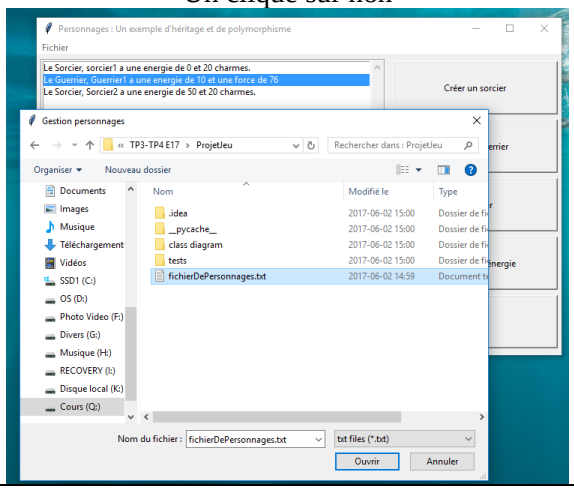
Le Guerrier a attaqué à deux reprises. Sa force a diminuée



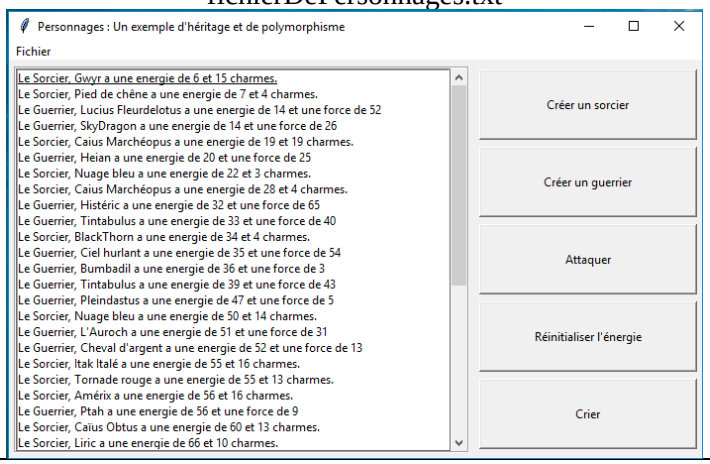
Un clique sur le menu Fichier puis Ouvrir…



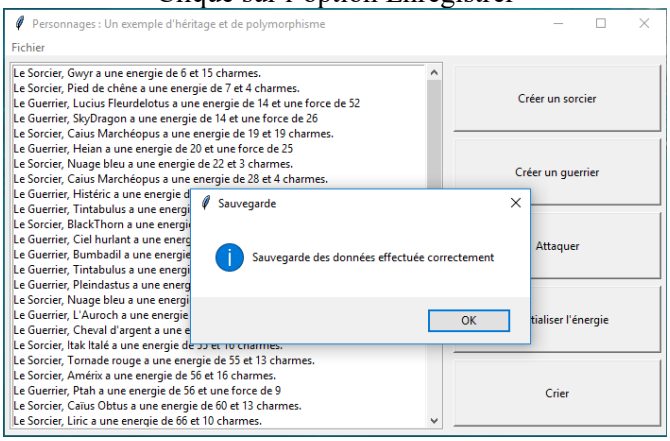
Un clique sur non



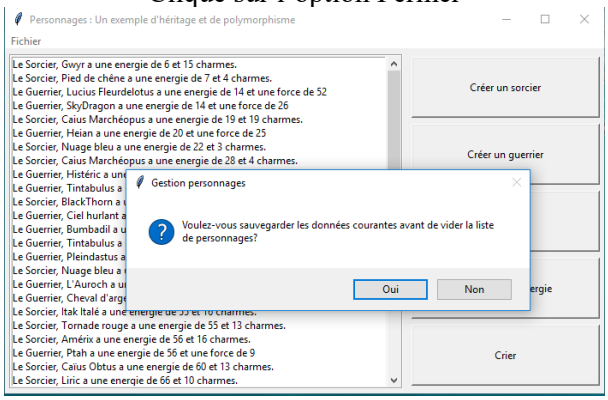
Affichage du contenu de mon fichier fichierDePersonnages.txt



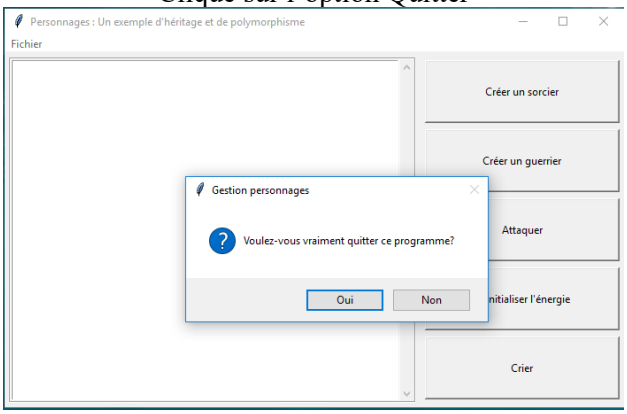
Clique sur l'option Enregistrer



Clique sur l'option Fermer



Clique sur l'option Quitter



Échéancier pour les remises

Modalités d'évaluation

Notez qu'un programme qui n'est pas fonctionnel (qui ne s'exécute pas ou qui plante à l'exécution) pourrait recevoir une note de 0.

TP3 : L'arbre d'héritage Personnage, Sorcier et Guerrier (80 points)

- À partir des classes **Personnage**, **Sorcier** et **Guerrier** fournies, coder toutes les méthodes de ces classes.
- Faire les tests unitaires des classes **Personnage**, **Sorcier** et **Guerrier**. Attention aux tests réalisés! Toutes les méthodes des classes qui ont été codées doivent être testées ainsi que tous les cas possibles de validation.

  Testez dans Personnage ce qui est commun aux 2 types de personnages et testez ce qui est propre à chaque type de personnage dans leur classe de test respective.

À remettre :

- Votre projet doit contenir les fichiers nécessaires au nombre de 3 en Python (extension.py) avec les tests unitaires et le fichier CorrectionTP3-TP4.xlsx fourni avec l'énoncé, que vous devez compléter en y indiquant votre nom ou les noms des deux co-équipiers, votre NI, et le nombre d'heures que vous avez consacré à compléter votre TP (pour chaque membre de l'équipe). **IMPORTANT** : si vous faites le travail en équipe, **une seule personne** doit faire le dépôt sur ENA.

- Vous devez compresser votre projet dans une archive Zip (fichier avec extension .zip). Assurez-vous que vous remettez le bon fichier. Nous ne pourrons pas donner de points à votre travail si vous remettez le mauvais fichier (la procédure du dépôt est la responsabilité de l'étudiant).

TP4 : Les classes Interface, GestionPersonnages et Util. (80 points)

- À partir des classes fournies, coder toutes les méthodes de ces classes.

À remettre :

- Votre projet doit contenir les fichiers nécessaires au nombre de 6 en Python (extension.py) et le fichier CorrectionTP3-TP4.xlsx fourni avec l'énoncé, que vous devez compléter en y indiquant votre nom ou les noms des deux co-équipiers, votre NI, et le nombre d'heures que vous avez consacré à compléter votre TP (pour chaque membre de l'équipe). **IMPORTANT** : si vous faites le travail en équipe, **une seule personne** doit faire le dépôt sur ENA.

- Vous devez compresser votre projet dans une archive Zip (fichier avec extension .zip). Assurez-vous que vous remettez le bon fichier. Nous ne pourrons pas donner de points à votre travail si vous remettez le mauvais fichier (la procédure du dépôt est la responsabilité de l'étudiant).

Remarques

Plagiat: Tel que décrit dans le plan de cours, le plagiat est strictement interdit. Ne partagez pas votre code source à quiconque. Une politique stricte de tolérance zéro est appliquée en tout temps et sous toute circonstance. Tous les cas détectés seront référés à la direction de la faculté. Des logiciels comparant chaque paire de TPs pourraient être utilisés pour détecter les cas de plagiat.
Retards : Tout travail remis en retard peut être envoyé par courriel à l'enseignant si le portail des cours n'accepte pas la remise. Une pénalité de 25% sera appliquée pour un travail remis le lendemain de la remise. Une note de 0 sera attribuée pour un plus long retard. Remises multiples: Il vous est possible de remettre votre TP plusieurs fois sur le portail des cours. Seulement la dernière version sera considérée pour la correction. La correction des TPs ne commence qu'après la date limite du dépôt du TP.
Respect des normes de programmation: Nous vous demandons de prêter attention au respect des normes de programmation établies pour le langage Python, notamment de nommer vos variables et fonctions en utilisant la convention suivante: ma_variable, fichier_entree, etc. Utiliser PyCharm s'avère être une très bonne idée ici, car celui-ci nous donne des indications sur la qualité de notre code (en indentation, marge à droite, et souligné).

Bon travail!