Rêve de Dragon

INFORMATIQUE ET SCIENCES DU NUMERIQUE

Lycée Murat - Issoire
Marguerite Sobkowicz, Cédric Mongiat, Jikael Larriven - Terminale S2 - 2016-2017

1. Introduction

Rêve de Dragon est un jeu de rôle français de Denis Gerfaud, dont la première version a été publiée en 1985, soit à une époque où l'informatique individuelle commençait à peine à se démocratiser. Ceci explique qu'aucun programme n'ai jamais été développé autour de ce jeu. Cette année, en spécialité Informatique et Sciences du Numérique (ISN), nous avons formé un groupe de trois, Marguerite Sobkowicz, Cédric Mongiat et Jikael Larriven, afin de réaliser un outil d'aide au jeu qui simplifierait les tâches de calcul et de notation et surtout accélérerait le déroulement du jeu en automatisant les tâches les plus fastidieuses.

L'UNIVERS DE REVE DE DRAGON

Les sages ont coutume de dire : "le monde est un rêve de dragon".

L'univers de Rêve de Dragon et complexe et virtuellement infini. L'aventure se passe dans une société à mi-chemin entre le moyen-âge et la renaissance. Le monde serait rêvé par des dragons, les joueurs étant des personnages du rêve.
Ces personnages sont des voyageurs qui vont de ville en ville, de rêve en rêve et suivent un scénario imaginé par le maître du jeu. Un jeu se décomposant en plusieurs parties dans des mondes différents où les personnages ont à affronter des situations qui nécessitent de disposer de talents spécifiques.
Un rêve correspond donc à une partie, les personnages étant créés en début de jeu avec des caractéristiques non modifiables, mais voyant leurs compétences modifiées à chaque partie ce qui leur permet de s'adapter à un nouvel environnement.

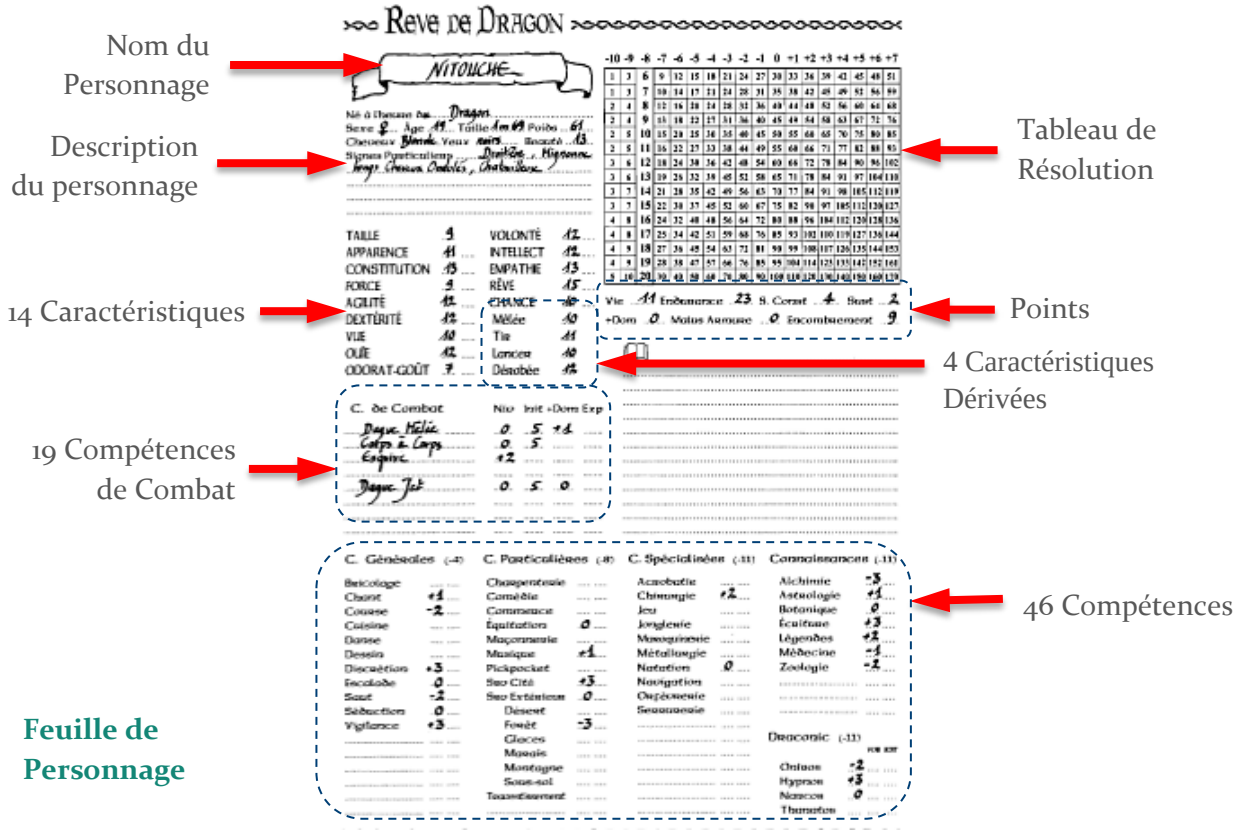
Un personnage est défini par 14 caractéristiques et 46 compétences.

- En début de jeu, le joueur dispose de 160 points à répartir entre ses caractéristiques en leur affectant un nombre de 0 à +20. Ce choix sera définitif pour tout le jeu.
- De même en début de partie, le joueur dispose de 3000 points à répartir entre ses compétences en leur affectant un nombre de -11 à +11. Certaines compétences ont des valeurs minimales de -8, -6 ou -4.
- Les joueurs de type « hauts-rêvant » ont des pouvoirs magiques qui vont leur permettre d'agir sur le cours du rêve. Ils disposent donc de 300 points parmi les 3000 à affecter dans les 4 compétences dont ne disposent pas les autres personnages.

Le maître du jeu introduit des situations, face auxquelles le joueur décide de faire une action. Une table de résolution liée à la fiche de personnage doit être consultée pour résoudre les actions. Elle donne une chance de succès en pourcentage en fonction d'une caractéristique et d'une compétence entrant dans la logique de l'action et choisies par le maître du jeu. Un lancer de deux dés à dix faces décide alors du succès ou non de cette action.

2. Cahier des charges

Nous sommes partis des règles du jeu telles que décrites dans l'ouvrage de Denis Gerfaud.
Les informations nécessaires à suivre un personnage au cours du jeu sont nombreuses et doivent être compilées sur 5 feuilles. Pour ne pas avoir un objectif irréalisable dans le temps dont nous disposions nous avons décidé de ne reproduire dans le programme que la feuille de personnage, qui est tout de même la plus complexe.



Le programme devra gérer un nombre illimité de joueurs, les créer, en sauvegarder et recharger les feuilles de personnage. Il devra permettre de saisir la description du personnage, les caractéristiques et les compétences. Le programme devra vérifier la cohérence des données par rapport aux règles et calculer automatiquement les caractéristiques dérivées.
On devra pouvoir décider de la réussite d'une action en jetant les dés et en utilisant le tableau de résolution ou un calcul équivalent.
On ajoutera une aide en ligne pour permettre aux joueurs néophytes de se familiariser avec les règles et l'utilisation du logiciel.

LICENCE ET DIFFUSION

Rêve de Dragon est une marque déposée par Denis Gerfaud.
Nous avons choisi la diffusion sous licence GPL V3.
L'intégralité du logiciel est téléchargeable sur la page Github du projet :
https://github.com/SirConfairance/Reve-de-Dragon

Pour que le logiciel fonctionne, l'ensemble des fichiers doit être installé dans le même dossier.
En annexe : Listing complet du programme.

3. Organisation du projet

Nous avons travaillé en Python 3.6 avec comme environnement de développement Python PyCharm Community Edition de Jetbrain, version gratuite.
Pour l'interface graphique nous avons utilisé tkinter.

ARCHITECTURE : MODELE – VUE - CONTROLEUR

L'architecture Modèle/Vue/Contrôleur (MVC) est une façon d'organiser une interface graphique introduite en 1978 et très utilisée dans les applications Web.
Elle consiste à distinguer trois entités distinctes qui sont, le modèle, la vue et le contrôleur ayant chacun un rôle précis dans l'interface. Les rôles des trois entités sont les suivants :

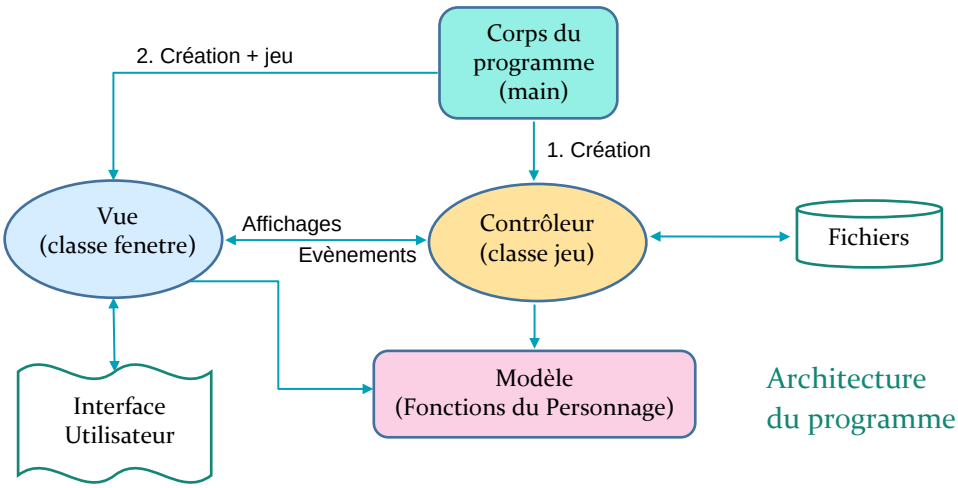
- Modèle : données (définition, accès et mise à jour)
- Vue : interface utilisateur (entrées et sorties)
- Contrôleur : logique des actions, contrôles et validation des commandes

Nous avons découpé notre programme selon cette architecture avec la répartition des fonctions suivante :

Modèle, **personnage.py**, qui contient les données manipulées par le programme.
Dans notre cas les données sont celles du personnage stockées dans un dictionnaire.
Le code du modèle inclus donc la création des structures, la vérification des données, le calcul des valeurs dérivées et tout ce qui permet l'accès aux données du personnage.

Vue, **fenetre.py**, qui fait l'interface avec l'utilisateur.
Sa première tâche est d'afficher le contenu de la fenêtre initiale à partir de libellés qu'elle a récupérés auprès du modèle.
Sa seconde tâche est de traiter les actions de l'utilisateur qui sont envoyés au contrôleur.
Enfin elle doit mettre à jour l'affichage suivant les données reçues du contrôleur.

Le Contrôleur, **jeu.py**, synchronise le modèle et la vue. Il répond aux actions effectuées sur la vue et modifie les données du modèle. C'est lui qui gère la sélection des personnages, le calcul de résolution des actions et la logique de tirage des Dés. Il est également responsable de l'archivage des parties.

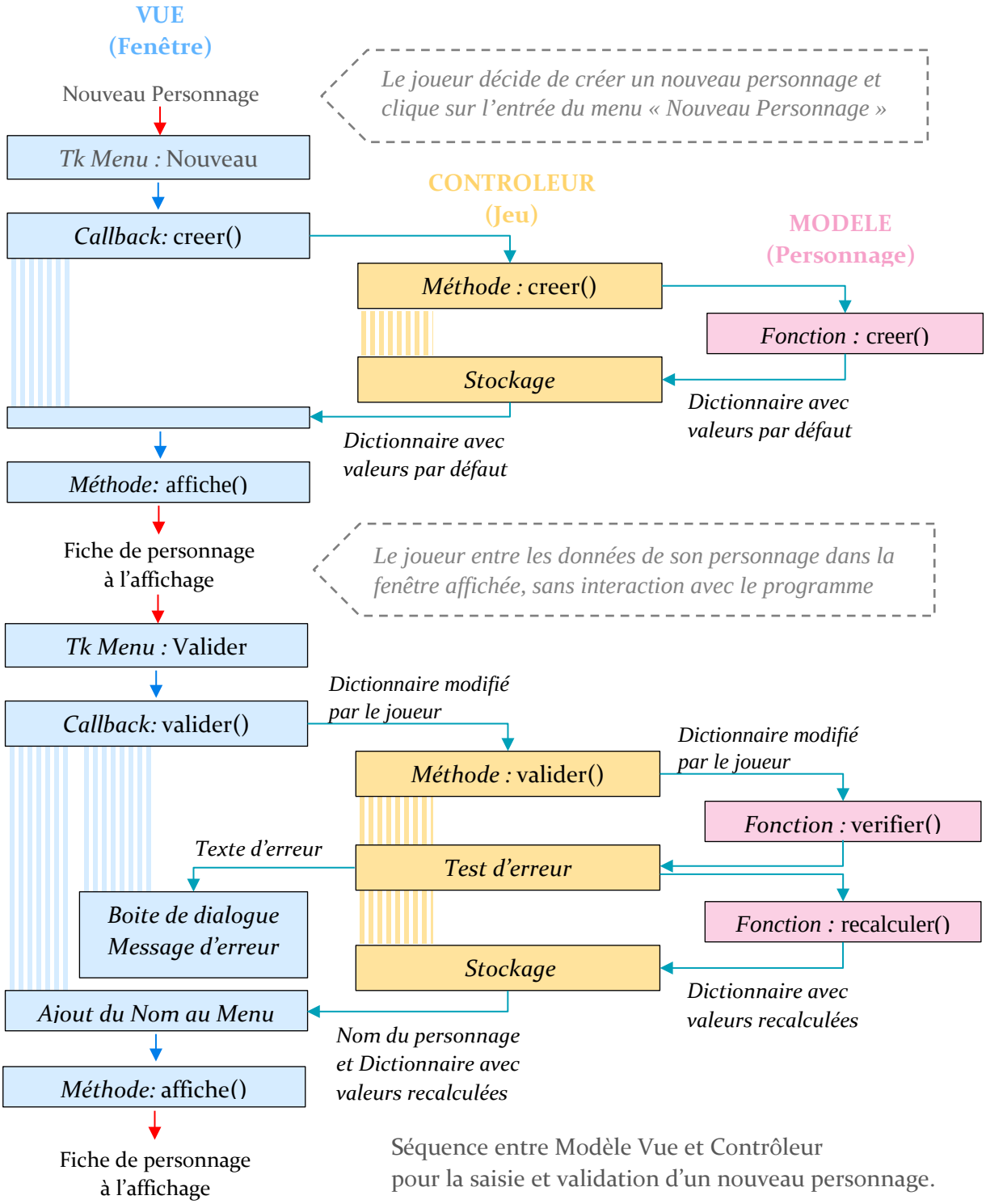


Architecture du programme

L'intérêt de cette architecture est qu'elle permet de modifier une des entités sans modifier les autres. On peut facilement ajouter des vues et rendre le programme multi utilisateurs.

EXECUTION EVENEMENTIELLE

Une fois la Vue initialisée, l'exécution commence par l'activation d'un évènement sur l'interface graphique qui active une des fonctions de la Vue.
Celle-ci appelle une fonction du Contrôleur en lui fournissant, si nécessaire, les paramètres correspondant à l'évènement ou des données de l'affichage.
La fonction du Contrôleur appelle éventuellement une ou plusieurs fonctions du Modèle pour traiter les données.
Elle renvoie finalement les données à la Vue, qui procède à l'affichage et se remet en attente.



Séquence entre Modèle Vue et Contrôleur
pour la saisie et validation d'un nouveau personnage.

CLASSES ET FONCTIONS DU PROGRAMME

Le contrôleur et la vue sont implémentés sous forme de classes. Pourquoi ce choix ?
Ces deux modules utilisent des variables locales qu'il est indispensable de conserver entre deux appels de fonctions.
Nous souhaitions développer de façon indépendante et ne pas avoir à utiliser de variables globales.

Le personnage est implémenté sous forme de fonctions.
Il ne dispose d'aucune variable locale et met simplement à disposition du Contrôleur des fonctions qui s'appliquent sur des personnages stockés par celui-ci.
Il n'était donc pas nécessaire d'en faire une classe.

REPARTITION DU TRAVAIL

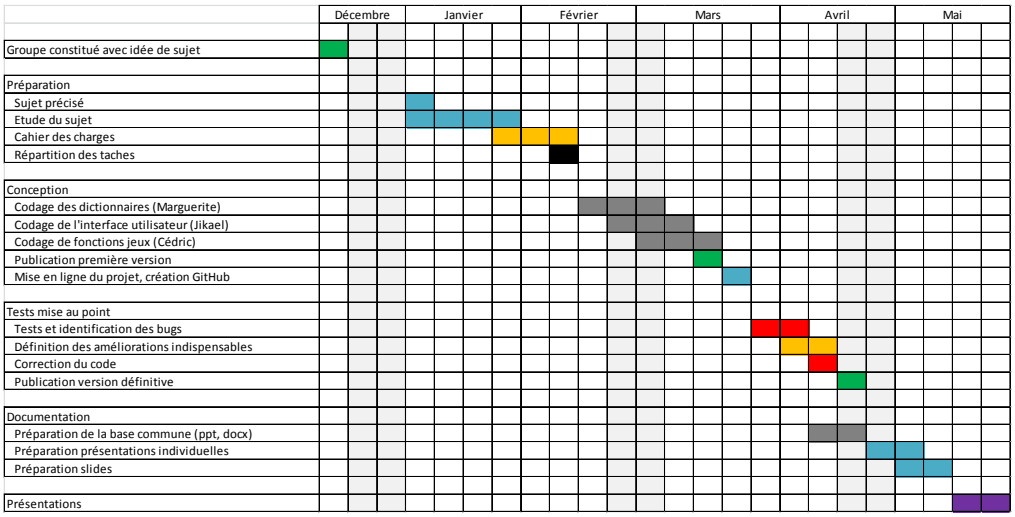
Pour réaliser ce projet, nous avons profité du découpage en blocs indépendants de l'architecture MVC et nous sommes partagés le travail de la façon suivante :

Marguerite : Modèle	Jikael : Vue	Cédric : Contrôleur
Cahier des Charges		
Personnage	Maquette de l'interface	Classe du jeu
Vérifications de points	Classe fenêtre	Gestion de fichiers
Recalcul des points	Intégration - GitHub	Tirage aléatoire
Tests		

Nous avons réfléchi au projet et rédigé le premier cahier des charges lors des séances d'ISN.
Dans la première phase du développement nous avons travaillé séparément et échangé notre code par clés USB lors des séances en classe.
Lorsque nous avons eu une première version de programme qui fonctionnait à peu près, nous avons créé un projet sur GitHub qui nous permettait de mieux partager le code en dehors des séances.

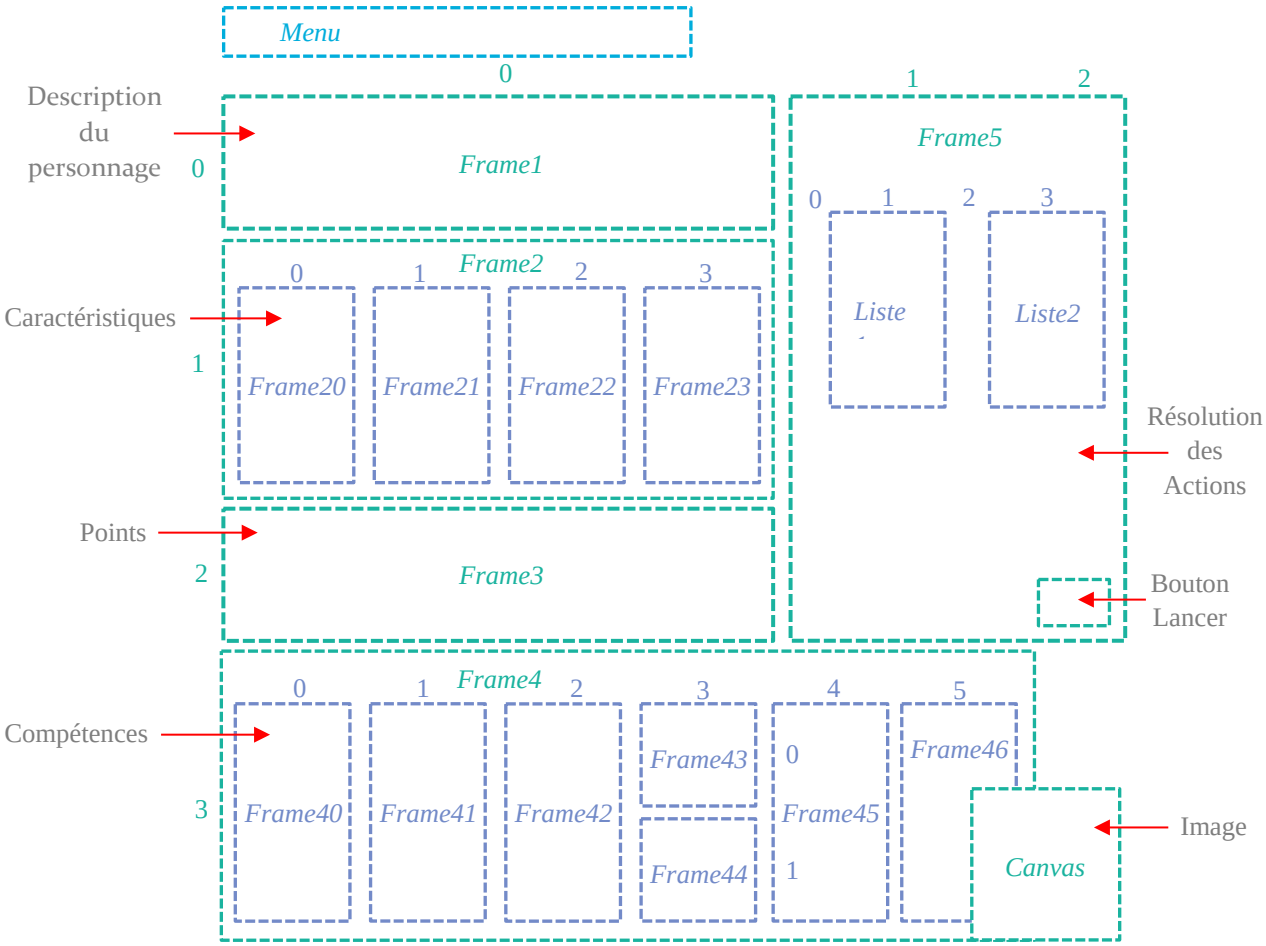
Planning

Nous avions réalisé un planning en début de projet.
Le début de projet a eu lieu en Décembre 2016. Nous avions pour objectif de terminer en Mai 2017.



4. Réalisation de l'interface graphique

Nous avons optés pour une représentation en lignes et colonnes en utilisant des frames.

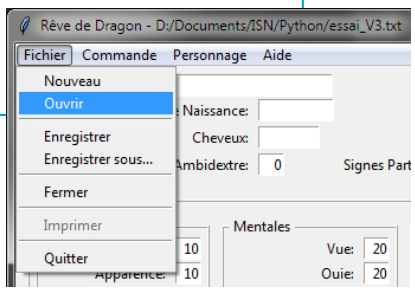


TECHNIQUE DE CREATION DES MENUS

Pour créer les menus de la fenêtre on utilise le widget *Menu*. On commence par créer la barre de menu (*menubar*) à laquelle on ajoute dans l'ordre : *Fichier*, *Commande*, *Personnage* et *Aide*.

```
self.menubar = Menu(root)
self.root.config(menu=self.menubar)

self.filemenu = Menu(self.menubar, tearoff=0)
self.menubar.add_cascade(label="Fichier", menu=self.filemenu)
self.filemenu.add_command(label="Nouveau", command=self.nouveau)
self.filemenu.add_command(label="Ouvrir", command=self.ouvrir)
self.filemenu.add_separator()
self.filemenu.add_command(label="Enregistrer",
command=self.jeu.enregistrer)
```



Chaque entrée de menu dirige soit :
- vers un sous-menu en cascade (*menu=self.xxxx*)
- vers une fonction à exécuter (*command=self.xxxx*)

Lorsque l'on sélectionne une entrée de menu la fonction indiquée est exécutée.

Dans le cas du menu personnages une difficulté supplémentaire est venue du nombre inconnu de personnages qui pouvaient être attachés au menu. Il fallait n'activer qu'une seule fonction en lui passant l'identification du personnage. Or il est interdit de passer des paramètres aux fonctions appelées par les menus. On a donc utilisé comme astuce, l'instruction lambda qui prend un nombre quelconque d'arguments et retourne une expression anonyme. Elle permet de créer une pseudo-fonction qui comporte le nom de la fonction avec son paramètre égal à l'index.
La syntaxe :

```
command=lambda index=numero: self.selectionner(index)
```

Lorsque `numero = 1` par exemple
est l'équivalent de :

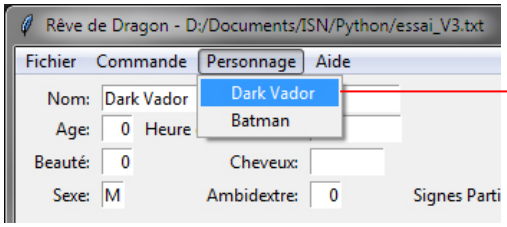
```
command= self.une_fonction_anonyme
```

avec :

```
def une_fonction_anonyme(self):
        self.selectionner(1)
```

Cette astuce permet de créer automatiquement une fonction par personnage, sans avoir à la déclarer explicitement.

Chaque personnage est visible dans le menu sous forme d'un nom, mais est sélectionné en utilisant l'index correspondant à sa position dans le menu.



Index= 0 pour Dark Vador, 1 pour Batman, etc.

```
def selectionner(self, code):
    self.pod = self.jeu.selectionner(code)
    self.affiche()
```

L'ordre dans le menu correspond à celui de la liste *data* de la classe Jeu, ce qui assure la cohérence entre affichage et données sélectionnées.

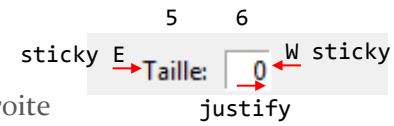
TECHNIQUE DE CREATION DES CHAMPS DE SAISIE

La création d'un champ de saisie comporte :

- La création d'une variable de type *IntVar* ou *StringVar* qui sert à recevoir le contenu du champ
- L'écriture d'un widget *Label*, texte statique qui précède le champ
- La saisie du champ par un widget *Entry*
- Le positionnement par le widget *grid* en ligne, colonne avec calage *sticky* et espacement *padx*

```
self.Entry_Taille = IntVar()
Label(frame1, text='Taille:').grid(row=1, column=5, sticky='E')
Entry(frame1, textvariable=self.Entry_Taille, justify='right', width=3)\
    .grid(row=1, column=6, sticky='W', padx="5")
```

Le code ci-dessus va créer le champ ci-contre, avec :
Le widget *Label* en colonne 5, cadré à droite (East)
Le widget *Entry* en colonne 6, cadré à gauche (West) et justifié à droite



5. Réalisation du Personnage

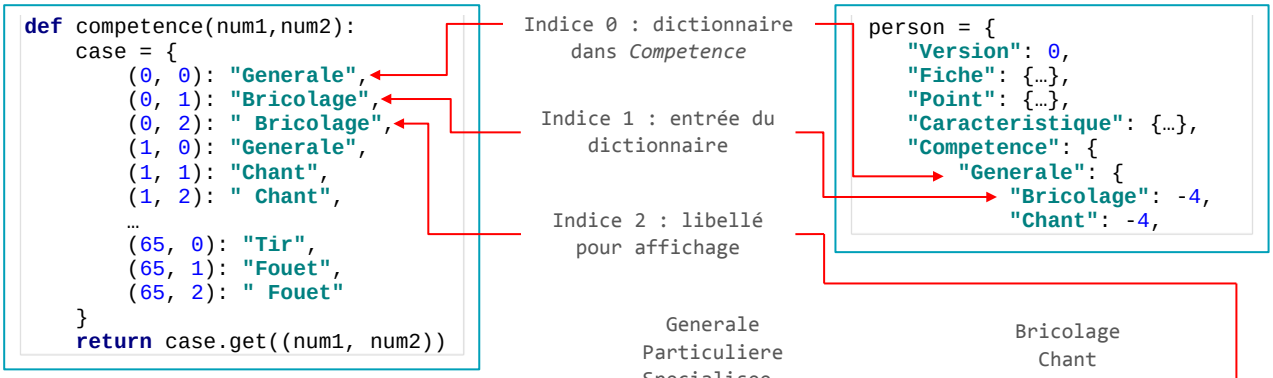
Le personnage est composé d'un ensemble de fonctions qui sont appelées soit par le *Jeu*, soit par la *Fenetre*.

- La première fonction est *creer* qui est appelée par le *Jeu*.
  Elle renvoie un dictionnaire initialisé avec des valeurs par défaut. Les éléments du dictionnaire sont accédés par des clés de type texte pour rendre le programme lisible.

- On trouve ensuite les fonctions *competence*, *caracteristique* et *point*. Ces fonctions utilisent des tuples pour renvoyer une valeur texte qui peut servir soit à l'affichage, soit à accéder aux données de compétence et caractéristiques du dictionnaire d'un personnage en utilisant des index numériques au lieu des clés. Ces fonctions sont utilisées par l'interface graphique et par les autres fonctions du personnage.
  Il est ainsi plus facile de faire des boucles sur les données d'un personnage.

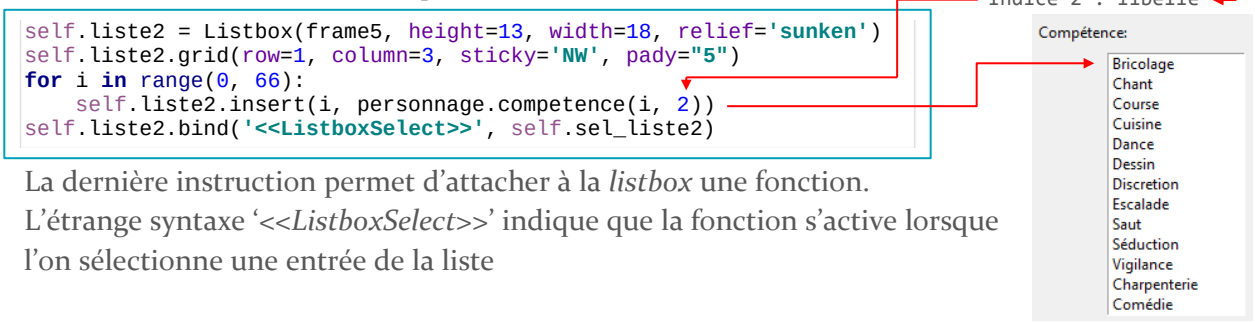
  Exemple avec les compétences du personnage :

  La fonction *competence* de *personnage* renvoie les index du dictionnaire et les libellés



  Elle permet d'initialiser les variables de saisie :

  Ou d'initialiser la *listbox* des compétences :



  La dernière instruction permet d'attacher à la *listbox* une fonction.
  L'étrange syntaxe '<<*ListboxSelect*>>' indique que la fonction s'active lorsque l'on sélectionne une entrée de la liste

- La fonction *recalculer* met à jour le dictionnaire en calculant les caractéristiques *Mélée*, *Tir*, *Lancer Dérobée* dont la valeur dépend des autres caractéristiques.
  Elle calcule également les points (*Vie*, *Endurance* etc.) en utilisant les caractéristiques.
  Cette fonction est appelée par le *Jeu* après la validation des données d'un personnage.

- La fonction *verifier*, vérifie les données du personnage.
  Une des principales taches de cette fonction est de vérifier la validité des données saisies.
  Elle contrôle les valeurs minimales et maximales sur chaque entrée du dictionnaire.
  Elle vérifie également que le nombre exact de points de compétences et de caractéristiques

ont été distribués. Le calcul est complexe pour les compétences car le coût en points augmente avec la valeur de la compétence.

Exemple avec la vérification des compétences spécialisées :

```
for i in range(26, 36):
    cs = person["Competence"]["Specialisee"][competence(i, 1)]
    v = 11 + cs                                  # Décalage pour faire une boucle à partir de 0
    scd = -11
    for j in range(v):                           # De 0 à cs+11
        scd += 1
        if scd > -11 and scd <= -8:              # De -11 à -8 : compte pour 5 points
            x -= 5
        if scd >= -7 and scd <= -4:              # De -7 à -4 : compte pour 10 points
            x -= 10
        if scd >= -3 and scd <= 0:               # De -3 à 0 : compte pour 15 points
            x -= 15
        if scd >= 1:                             # Au-dessus de 0 : compte pour 20 points
            x -= 20
```

6. Réalisation du Jeu

Jeu est une classe comme *Fenetre*. Il copie une partie de la structure de la classe *Fenetre*, les méthodes de *Fenetre* appelant directement leurs homonymes de *Jeu*.

Jeu conserve l'image de la partie dans une Liste de personnages: *data*.
La liste grandit à chaque création d'un nouveau personnage.
L'index dans la liste correspond à l'index dans le menu des personnages.

Jeu gère la sauvegarde et la relecture de *data* dans des fichiers de type texte.
On utilise un widget spécifique *filedialog* pour sélectionner les fichiers à ouvrir ou enregistrer.

Pour rendre les données lisibles dans un fichier texte nous les avons transformées en format JSON (JavaScript Object Notation), facilement lisible et modifiable. Pour éviter de charger des données non compatibles, nous avons ajouté un identificateur magique en tête de fichier, qui comporte également la version du programme. Si l'identificateur n'est pas trouvé les données sont refusées.

Exemple de début d'un fichier de sauvegarde :

```
MAGIC-ISN-V3.0[{"Version": 6, "Fiche": {"Nom": "Dark Vador", "Heure_Naissance": 11,
```

Identificateur Magique

Début du texte encodé en JSON

Début du dictionnaire du premier personnage

Une fonction spécifique de *Jeu* est le lancer de dés.
A partir des valeurs d'une compétence et d'une caractéristique on détermine un seuil de réussite.
Dans le jeu d'origine ce seuil était obtenu par le tableau de résolution. Nous avons réussi à coder le calcul qui donne les valeurs de ce tableau, ce qui simplifie grandement le programme.

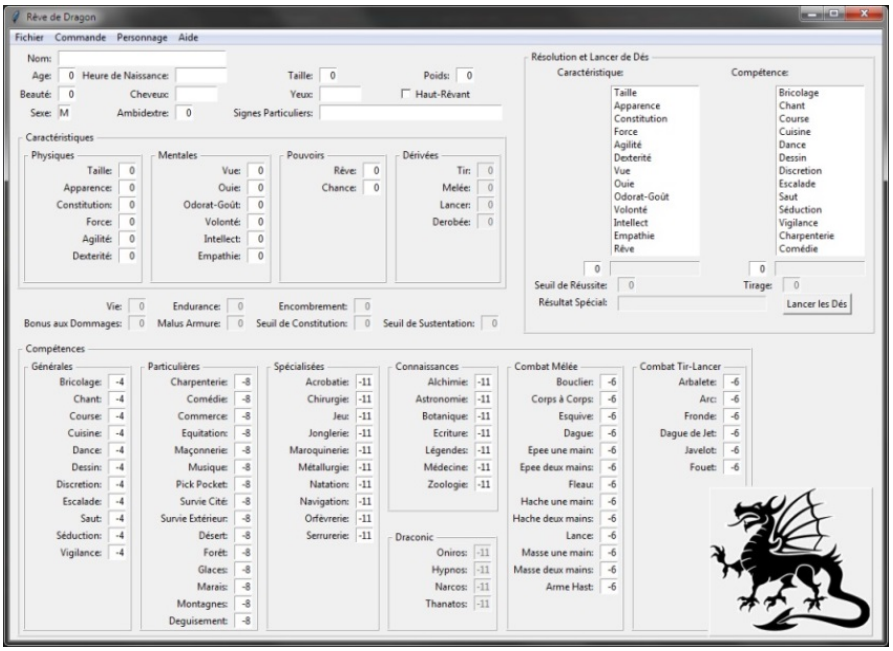
La formule est : `caractéristique * (1 + ½ (compétence + 8))`

Dans le jeu original on utilise deux dés à dix faces. On a reproduit le principe avec le tirage de 2 valeurs aléatoires de `0` à `9`. Quand on fait `00` cela donne `100`.

Jeu compare alors le score au seuil pour déterminer un jugement de l'action.
Cela peut être un échec ou une réussite.

7. Conclusion

ASPECT FINAL DU PROGRAMME



La création de ce jeu nous passionnés, le programme est complètement fonctionnel et est même mieux que ce que nous souhaitions.
Néanmoins si on souhaite l'utiliser pour jouer à Rêve de Dragon, il doit encore être amélioré.

L'enchainement des parties doit être développé. Cela implique de créer une nouvelle fenêtre pour saisir l'historique des personnages entre les parties : l'archétype. Cette fenêtre serait affichée lors du passage à une nouvelle partie et permettrait de décider quelles compétences sont appliquées au personnage dans la partie suivante.
Nous souhaitions pouvoir imprimer les feuilles de personnage. Malheureusement Python ne dispose pas de fonctions d'impression standard et portable. Il faudrait développer une fonction qui enregistre le personnage dans un fichier texte plus facile à imprimer.
Le calcul en temps réel des points de caractéristiques et compétences doit être ajouté lors de la saisie pour rendre le programme plus confortable. Cela nécessite une fonction associée à chaque champ qui appelle des fonctions supplémentaires de personnage qui renvoient les comptes des points.

Cette année d'ISN nous a permis d'acquérir des compétences en informatique, et en programmation. Nous avons beaucoup aimé faire cette spécialité et découvrir le langage Python qui est très utilisé pour les applications graphiques ou de Jeux. Blender par exemple est écrit en Python. C'est également un langage de script de Maya, 3DSMax, Rhino ou Nuke.

Nos sources d'inspiration :

Python : Gérard Swinnen, Apprendre à programmer avec Python 3
En version creative-common sur http://inforef.be/swi/python.htm

Tkinter : http://vincent.developpez.com
/cours-tutoriels/python/tkinter/apprendre-creer-interface-graphique-tkinter-python-3/

Trucs et astuces : http://stackoverflow.com
Question-réponses avec de nombreux exemples en Python et tkinter.

```python
# Rêve de Dragon
# ISN Terminale S Lycée Murat Issoire 2016-2017
# Par Jikael Larriven, Marguerite Sobkowicz, Cédric Mongiat
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
#
# main.py
# Corps principal du programme
#
# Crée un objet de type jeu (contenant les règles et le moteur du jeu)
# Crée un objet de type fenetre (contenant l'interface graphique)
# Traite en boucle les évènements de l'interface graphique qui activent les fonctions du jeu
# Le jeu renvoie les valeurs d'affichage vers l'interface graphique

from tkinter import Tk

# Création du jeu
from jeu import Jeu
job = Jeu()

# Création de l'interface graphique

root = Tk()
from fenetre import Fenetre
```

```
36 wnd = Fenetre(root, job)
37
38 root.mainloop()
```

```
 1 # Rêve de Dragon
 2 # ISN Terminale S Lycée Murat Issoire 2016-2017
 3 # Par Jikael Larriven, Marguerite Sobkowicz, Cédric Mongiat
 4 #
 5 # This program is free software: you can redistribute it and/or modify
 6 # it under the terms of the GNU General Public License as published by
 7 # the Free Software Foundation, either version 3 of the License, or
 8 # (at your option) any later version.
 9 #
10 # This program is distributed in the hope that it will be useful,
11 # but WITHOUT ANY WARRANTY; without even the implied warranty of
12 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13 # GNU General Public License for more details.
14 #
15 # You should have received a copy of the GNU General Public License
16 # along with this program.  If not, see <http://www.gnu.org/licenses/>.
17 #
18 # personnage.py
19 # Dictionnaire des personnages
20 #
21 # Comporte les définitions du dictionnaire contenant chaque personnage
22 # Applique les règles du jeu relatives aux personnages
23 # Les fonctions permettent les traitements sur le dictionnaire :
24 # - creer : création d'un personnage vide
25 # - caracteristiques, competence, point : renvoient les textes correspondant aux index
26 # - verifier : vérification de la cohérence des données du personnage par rapport aux règles d'attribution des points
27 # - calculer : calcul des caractéristiques du personnage à partir des valeurs saisies ou modifiées
28 #
29
30 from math import *
31
32 # La fonction creer retourne un dictionnaire de personnage avec des données de base
33 def creer():
34
35     person = {
```

```
        "Version": 0,
        "Fiche": {
            "Nom": "",
            "Heure_Naissance": 0,
            "Sexe": "M",
            "Age": 0,
            "Taille": 0,
            "Poids": 0,
            "Cheveux": "",
            "Yeux": "",
            "Beaute": 10,
            "Signes_Particulier": "",
            "Ambidextre": 0,
            "Haut_Revant": 0
        },
        "Point": {
            "Vie": 0,
            "Endurance": 0,
            "Seuil_Constitution": 0,
            "Sustain": 0,
            "PlusDmg": 0,
            "Malus_Armor": 0,
            "Encombrement": 0
        },
        "Caracteristique": {
            "Taille": 0,
            "Apparence": 0,
            "Constitution": 0,
            "Force": 0,
            "Agilite": 0,
            "Dexterite": 0,
            "Vue": 0,
            "Ouie": 0,
            "Odorat_Gout": 0,
            "Volonte": 0,
```

```
71              "Intellect": 0,
72              "Empathie": 0,
73              "Reve": 0,
74              "Chance": 0,
75              "Tir": 0,
76              "Melee": 0,
77              "Lancer": 0,
78              "Derobee": 0
79          },
80          "Competence": {
81              "Generale": {
82                  "Bricolage": -4,
83                  "Chant": -4,
84                  "Course": -4,
85                  "Cuisine": -4,
86                  "Dance": -4,
87                  "Dessin": -4,
88                  "Discretion": -4,
89                  "Escalade": -4,
90                  "Saut": -4,
91                  "Seduction": -4,
92                  "Vigilance": -4
93              },
94              "Particuliere": {
95                  "Charpenterie": -8,
96                  "Comedie": -8,
97                  "Commerce": -8,
98                  "Equitation": -8,
99                  "Maconnerie": -8,
100                 "Musique": -8,
101                 "Pick_Pocket": -8,
102                 "Survie_Cite": -8,
103                 "Survie_Exterieur": -8,
104                 "Desert": -8,
105                 "Foret": -8,
```

```
106                 "Glaces": -8,
107                 "Marais": -8,
108                 "Montagnes": -8,
109                 "Deguisement": -8
110             },
111             "Specialisee": {
112                 "Acrobatie": -11,
113                 "Chirurgie": -11,
114                 "Jeu": -11,
115                 "Jonglerie": -11,
116                 "Maroquinerie": -11,
117                 "Metallurgie": -11,
118                 "Natation": -11,
119                 "Navigation": -11,
120                 "Orfevrerie": -11,
121                 "Serrurerie": -11
122             },
123             "Connaissances": {
124                 "Alchimie": -11,
125                 "Astronomie": -11,
126                 "Botanique": -11,
127                 "Ecriture": -11,
128                 "Legendes": -11,
129                 "Medecine": -11,
130                 "Zoologie": -11
131             },
132             "Draconic": {
133                 "Oniros": -11,
134                 "Hypnos": -11,
135                 "Narcos": -11,
136                 "Thanatos": -11
137             },
138             "CMelee": {
139                 "Bouclier": -6,
140                 "CorpsaCorps": -6,
```

```
141                 "Esquive": -6,
142                 "Dague": -6,
143                 "Epee1Main": -6,
144                 "Epee2Main": -6,
145                 "Fleau": -6,
146                 "Hache1Main": -6,
147                 "Hache2Main": -6,
148                 "Lance": -6,
149                 "Masse1Main": -6,
150                 "Masse2Main": -6,
151                 "ArmeHast": -6
152             },
153             "Tir": {
154                 "Arbalete": -6,
155                 "Arc": -6,
156                 "Fronde": -6,
157                 "DagueJet": -6,
158                 "Javelot": -6,
159                 "Fouet": -6,
160             },
161         }
162     }
163     return person
164 
165 # Détermine la clé et le texte d'affichage associés à un index de caractéristiques
166 def caracteristique(num1,num2):
167     case = {
168         (0, 0): "Taille",
169         (0, 1): " Taille",
170         (1, 0): "Apparence",
171         (1, 1): " Apparence",
172         (2, 0): "Constitution",
173         (2, 1): " Constitution",
174         (3, 0): "Force",
175         (3, 1): " Force",
```

```
176         (4, 0): "Agilite",
177         (4, 1): " Agilité",
178         (5, 0): "Dexterite",
179         (5, 1): " Dexterité",
180         (6, 0): "Vue",
181         (6, 1): " Vue",
182         (7, 0): "Ouie",
183         (7, 1): " Ouie",
184         (8, 0): "Odorat_Gout",
185         (8, 1): " Odorat-Goût",
186         (9, 0): "Volonte",
187         (9, 1): " Volonté",
188         (10, 0): "Intellect",
189         (10, 1): " Intellect",
190         (11, 0): "Empathie",
191         (11, 1): " Empathie",
192         (12, 0): "Reve",
193         (12, 1): " Rêve",
194         (13, 0): "Chance",
195         (13, 1): " Chance",
196         (14, 0): "Tir",
197         (14, 1): " Tir",
198         (15, 0): "Melee",
199         (15, 1): " Melée",
200         (16, 0): "Lancer",
201         (16, 1): " Lancer",
202         (17, 0): "Derobee",
203         (17, 1): " Derobée"
204     }
205     return case.get((num1,num2))
206 
207 # Détermine les clés et le texte associées à un index de compétence
208 def competence(num1,num2):
209     case = {
210         (0, 0): "Generale",
```

```
211         (0, 1): "Bricolage",
212         (0, 2): " Bricolage",
213         (1, 0): "Generale",
214         (1, 1): "Chant",
215         (1, 2): " Chant",
216         (2, 0): "Generale",
217         (2, 1): "Course",
218         (2, 2): " Course",
219         (3, 0): "Generale",
220         (3, 1): "Cuisine",
221         (3, 2): " Cuisine",
222         (4, 0): "Generale",
223         (4, 1): "Dance",
224         (4, 2): " Dance",
225         (5, 0): "Generale",
226         (5, 1): "Dessin",
227         (5, 2): " Dessin",
228         (6, 0): "Generale",
229         (6, 1): "Discretion",
230         (6, 2): " Discretion",
231         (7, 0): "Generale",
232         (7, 1): "Escalade",
233         (7, 2): " Escalade",
234         (8, 0): "Generale",
235         (8, 1): "Saut",
236         (8, 2): " Saut",
237         (9, 0): "Generale",
238         (9, 1): "Seduction",
239         (9, 2): " Séduction",
240         (10, 0): "Generale",
241         (10, 1): "Vigilance",
242         (10, 2): " Vigilance",
243         (11, 0): "Particuliere",
244         (11, 1): "Charpenterie",
245         (11, 2): " Charpenterie",
```

```
246         (12, 0): "Particuliere",
247         (12, 1): "Comedie",
248         (12, 2): " Comédie",
249         (13, 0): "Particuliere",
250         (13, 1): "Commerce",
251         (13, 2): " Commerce",
252         (14, 0): "Particuliere",
253         (14, 1): "Equitation",
254         (14, 2): " Equitation",
255         (15, 0): "Particuliere",
256         (15, 1): "Maconnerie",
257         (15, 2): " Maçonnerie",
258         (16, 0): "Particuliere",
259         (16, 1): "Musique",
260         (16, 2): " Musique",
261         (17, 0): "Particuliere",
262         (17, 1): "Pick_Pocket",
263         (17, 2): " Pick Pocket",
264         (18, 0): "Particuliere",
265         (18, 1): "Survie_Cite",
266         (18, 2): " Survie Cité",
267         (19, 0): "Particuliere",
268         (19, 1): "Survie_Exterieur",
269         (19, 2): " Survie Extérieur",
270         (20, 0): "Particuliere",
271         (20, 1): "Desert",
272         (20, 2): " Désert",
273         (21, 0): "Particuliere",
274         (21, 1): "Foret",
275         (21, 2): " Forêt",
276         (22, 0): "Particuliere",
277         (22, 1): "Glaces",
278         (22, 2): " Glaces",
279         (23, 0): "Particuliere",
280         (23, 1): "Marais",
```

```
281         (23, 2): " Marais",
282         (24, 0): "Particuliere",
283         (24, 1): "Montagnes",
284         (24, 2): " Montagnes",
285         (25, 0): "Particuliere",
286         (25, 1): "Deguisement",
287         (25, 2): " Deguisement",
288         (26, 0): "Specialisee",
289         (26, 1): "Acrobatie",
290         (26, 2): " Acrobatie",
291         (27, 0): "Specialisee",
292         (27, 1): "Chirurgie",
293         (27, 2): " Chirurgie",
294         (28, 0): "Specialisee",
295         (28, 1): "Jeu",
296         (28, 2): " Jeu",
297         (29, 0): "Specialisee",
298         (29, 1): "Jonglerie",
299         (29, 2): " Jonglerie",
300         (30, 0): "Specialisee",
301         (30, 1): "Maroquinerie",
302         (30, 2): " Maroquinerie",
303         (31, 0): "Specialisee",
304         (31, 1): "Metallurgie",
305         (31, 2): " Métallurgie",
306         (32, 0): "Specialisee",
307         (32, 1): "Natation",
308         (32, 2): " Natation",
309         (33, 0): "Specialisee",
310         (33, 1): "Navigation",
311         (33, 2): " Navigation",
312         (34, 0): "Specialisee",
313         (34, 1): "Orfevrerie",
314         (34, 2): " Orfèvrerie",
315         (35, 0): "Specialisee",
```

```
316         (35, 1): "Serrurerie",
317         (35, 2): " Serrurerie",
318         (36, 0): "Connaissances",
319         (36, 1): "Alchimie",
320         (36, 2): " Alchimie",
321         (37, 0): "Connaissances",
322         (37, 1): "Astronomie",
323         (37, 2): " Astronomie",
324         (38, 0): "Connaissances",
325         (38, 1): "Botanique",
326         (38, 2): " Botanique",
327         (39, 0): "Connaissances",
328         (39, 1): "Ecriture",
329         (39, 2): " Ecriture",
330         (40, 0): "Connaissances",
331         (40, 1): "Legendes",
332         (40, 2): " Légendes",
333         (41, 0): "Connaissances",
334         (41, 1): "Medecine",
335         (41, 2): " Médecine",
336         (42, 0): "Connaissances",
337         (42, 1): "Zoologie",
338         (42, 2): " Zoologie",
339         (43, 0): "Draconic",
340         (43, 1): "Oniros",
341         (43, 2): " Oniros",
342         (44, 0): "Draconic",
343         (44, 1): "Hypnos",
344         (44, 2): " Hypnos",
345         (45, 0): "Draconic",
346         (45, 1): "Narcos",
347         (45, 2): " Narcos",
348         (46, 0): "Draconic",
349         (46, 1): "Thanatos",
350         (46, 2): " Thanatos",
```

```
351         (47, 0): "CMelee",
352         (47, 1): "Bouclier",
353         (47, 2): " Bouclier",
354         (48, 0): "CMelee",
355         (48, 1): "CorpsaCorps",
356         (48, 2): " Corps à Corps",
357         (49, 0): "CMelee",
358         (49, 1): "Esquive",
359         (49, 2): " Esquive",
360         (50, 0): "CMelee",
361         (50, 1): "Dague",
362         (50, 2): " Dague",
363         (51, 0): "CMelee",
364         (51, 1): "Epee1Main",
365         (51, 2): " Epee une main",
366         (52, 0): "CMelee",
367         (52, 1): "Epee2Main",
368         (52, 2): " Epee deux mains",
369         (53, 0): "CMelee",
370         (53, 1): "Fleau",
371         (53, 2): " Fleau",
372         (54, 0): "CMelee",
373         (54, 1): "Hache1Main",
374         (54, 2): " Hache une main",
375         (55, 0): "CMelee",
376         (55, 1): "Hache2Main",
377         (55, 2): " Hache deux mains",
378         (56, 0): "CMelee",
379         (56, 1): "Lance",
380         (56, 2): " Lance",
381         (57, 0): "CMelee",
382         (57, 1): "Masse1Main",
383         (57, 2): " Masse une main",
384         (58, 0): "CMelee",
385         (58, 1): "Masse2Main",
```

```
386         (58, 2): " Masse deux mains",
387         (59, 0): "CMelee",
388         (59, 1): "ArmeHast",
389         (59, 2): " Arme Hast",
390         (60, 0): "Tir",
391         (60, 1): "Arbalete",
392         (60, 2): " Arbalete",
393         (61, 0): "Tir",
394         (61, 1): "Arc",
395         (61, 2): " Arc",
396         (62, 0): "Tir",
397         (62, 1): "Fronde",
398         (62, 2): " Fronde",
399         (63, 0): "Tir",
400         (63, 1): "DagueJet",
401         (63, 2): " Dague de Jet",
402         (64, 0): "Tir",
403         (64, 1): "Javelot",
404         (64, 2): " Javelot",
405         (65, 0): "Tir",
406         (65, 1): "Fouet",
407         (65, 2): " Fouet"
408     }
409     return case.get((num1, num2))
410 
411 # Détermine la clé et le texte d'affichage associés à un index de points
412 def point(num1,num2):
413     case = {
414         (0, 0): "Vie",
415         (0, 1): "Vie",
416         (1, 0): "Endurance",
417         (1, 1): "Endurance",
418         (2, 0): "Encombrement",
419         (2, 1): "Encombrement",
420         (3, 0): "PlusDmg",
```

```
421         (3, 1): "Bonus aux Dommages",
422         (4, 0): "Malus_Armor",
423         (4, 1): "  Malus Armure",
424         (5, 0): "Seuil_Constitution",
425         (5, 1): "  Seuil de Constitution",
426         (6, 0): "Sustain",
427         (6, 1): "  Seuil de Sustentation"
428     }
429     return case.get((num1, num2))
430 
431 # La fonction verifier assure que les données du personnage sont valides
432 # Elle est utilisée après les saisies
433 # Elle retourne un message d'erreur texte ou None
434 def verifier (person):
435 
436     # Vérification de l'identification
437     if len(person["Fiche"]["Nom"]) < 3:
438         return "Le Nom du personnage\ndoit être au moins de 3 caractères"
439 
440     # Est-ce la création du personnage
441     # Si oui, version est zéro
442     if person["Version"] == 0:
443         creation = True
444     else:
445         creation = False
446 
447     # Taille  de 6 à 15
448     ht = person["Fiche"]["Taille"]
449     if ht < 6 or ht > 15:
450         return "La Taille doit être entre 6 et 15"
451 
452     # Poids de 31 à 110, à vérifier selon la taille
453     pds = person["Fiche"]["Poids"]
454     if pds < 31 or pds > 110:
455         return "Le Poids doit être entre 31 et 110"
```

```
456
457     # Age minimum de 10
458     hn = person["Fiche"]["Age"]
459     if hn < 10:
460         return "L'Age minimum est de 10"
461
462     # Heure de naissance de 1 à 12
463     hn = person["Fiche"]["Heure_Naissance"]
464     if hn < 1 or hn > 12:
465         return "L'Heure de Naissance doit être entre 1 et 12"
466
467     # Beauté 0 à 16, les points au dessus de 10 sont retranchés des caractéristiques
468     bte = person["Fiche"]["Beaute"]
469     if bte < 1 or bte > 16:
470         return "La Beaute doit être entre 1 et 16"
471     if bte > 10:
472         cpt = bte - 10
473     else:
474         cpt = 0
475
476     # Vérification des valeurs limites de caractéristiques à la création du personnage
477     # On utilise Version qui vaut 0 à la création et s'incrémente à chaque recalcul des points
478     # On ne compte l'affectation des 160 points que lors de la première saisie
479     for i in range(0, 14):
480         ca = person["Caracteristique"][caracteristique(i, 0)]
481         if ca < 0:
482             return "La valeur minimale pour la caractéristique\n" + caracteristique(i, 1) + " est de 0"
483         if ca > 20:
484             return "La valeur maximale pour la caractéristique\n" + caracteristique(i, 1) + " est de 20"
485         cpt += ca
486     if person["Version"] < 1:
487         if cpt < 160:
488             return "Vous n'avez affecté que " + str(cpt) + " points aux caractéristiques\n Vous disposez de 160 points"
489         if cpt > 160:
490             return "Vous avez affecté " + str(cpt) + " points aux caractéristiques\n Vous ne disposez que de 160 points"
```

```
491
492     # Si le personnage est Haut-Rêvant on doit réserver 300 points aux compétences Draconic
493     if person["Fiche"]["Haut_Revant"] > 0:
494         xd = 300
495         draconic = True
496     else:
497         xd = 0
498         draconic = False
499
500     # Vérification du nombre de points de compétences à distribuer
501     # (selon le livre de règles page 22)
502     x = 3120 - xd
503
504     # Competences générales
505     for i in range(0, 11):
506
507         # Les compétences générales ne peuvent pas être inférieures à -4
508         # A la création aucune ne peut être supérieure à +3
509         cg = person["Competence"]["Generale"][competence(i, 1)]
510         if cg < -4:
511             return "La valeur minimale pour la compétence\n" + competence(i, 2) + " est de -4"
512         if creation and cg > 3:
513             return "La valeur maximale pour la compétence\n" + competence(i, 2) + " est de 3 à la création"
514
515         # Decalage de 4 pour avoir des indices partant de 0
516         v = 4 + cg
517         g = -4
518
519         # On compte les points utilisés par tranches de niveaux
520         for j in range(v):
521             # de -3 à 0 : 15 points
522             g += 1
523             if g <=0 :
524                 x -= 15
525             # de +1 à +20 : 20 points
```

```
526             if g >= 1:
527                 x -= 20
528 
529 
530     # Competences particulières
531     for i in range(11, 26):
532 
533         # Les compétences particulières ne peuvent pas être inférieures à -8
534         # A la création aucune ne peut être supérieure à +3
535         cp = person["Competence"]["Particuliere"][competence(i, 1)]
536         if cp < -8:
537             return "La valeur minimale pour la compétence\n" + competence(i, 2) + " est de -8"
538         if creation and cp > 3:
539             return "La valeur maximale pour la compétence\n" + competence(i, 2) + " est de 3 à la création"
540 
541         # Decalage de 8 pour avoir des indices partant de 0
542         v = 8 + cp
543         p = -8
544 
545         # On compte les points utilisés par tranches de niveaux
546         for j in range(v):
547             p += 1
548             # de -7 à -4 : 10 points
549             if p <= -4:
550                 x -= 10
551             # de -3 à 0 : 15 points
552             if p >= -3 and p <= 0:
553                 x -= 15
554             # de +1 à +20 : 20 points
555             if p >= 1:
556                 x -= 20
557 
558     # Competences spécialisées
559     for i in range(26, 36):
560 
```

```
561         # Les compétences spécialisées ne peuvent pas être inférieures à -11
562         # A la création aucune ne peut être supérieure à +3
563         cs = person["Competence"]["Specialisee"][competence(i, 1)]
564         if cs < -11:
565             return "La valeur minimale pour la compétence\n" + competence(i, 2) + " est de -11"
566         if creation and cs > 3:
567             return "La valeur maximale pour la compétence\n" + competence(i, 2) + " est de 3 à la création"
568 
569         # Decalage de 11 pour avoir des indices partant de 0
570         v = 11 + cs
571         scd = -11
572 
573         # On compte les points utilisés par tranches de niveaux
574         for j in range(v):
575             scd += 1
576             # de -11 à -8 : 5 points
577             if scd >= -11 and scd <= -8:
578                 x -= 5
579             # de -7 à -4 : 10 points
580             if scd >= -7 and scd <= -4:
581                 x -= 10
582             # de -3 à 0 : 15 points
583             if scd >= -3 and scd <= 0:
584                 x -= 15
585             # de +1 à +20 : 20 points
586             if scd >= 1:
587                 x -= 20
588 
589     # Competences connaissances
590     for i in range(36, 43):
591 
592         # Les compétences connaissances ne peuvent pas être inférieures à -11
593         # A la création aucune ne peut être supérieure à +3
594         cc = person["Competence"]["Connaissances"][competence(i, 1)]
595         if cc < -11:
```

```
596             return "La valeur minimale pour la compétence\n" + competence(i, 2) + " est de -11"
597         if creation and cc > 3:
598             return "La valeur maximale pour la compétence\n" + competence(i, 2) + " est de 3 à la création"
599 
600         # Decalage de 11 pour avoir des indices partant de 0
601         v = 11 + cc
602         scd = -11
603 
604         # On compte les points utilisés par tranches de niveaux
605         for j in range(v):
606             scd += 1
607             # de -11 à -8 : 5 points
608             if scd >= -11 and scd <= -8:
609                 x -= 5
610             # de -7 à -4 : 10 points
611             if scd >= -7 and scd <= -4:
612                 x -= 10
613             # de -3 à 0 : 15 points
614             if scd >= -3 and scd <= 0:
615                 x -= 15
616             # de +1 à +20 : 20 points
617             if scd >= 1:
618                 x -= 20
619 
620     # Competences draconic
621     if draconic:
622 
623         # Sauf Thanatos
624         for i in range(43, 46):
625 
626             # Les compétences draconic ne peuvent pas être inférieures à -11
627             # A la création aucune ne peut être supérieure à +3
628             cd = person["Competence"]["Draconic"][competence(i, 1)]
629             if cd < -11:
630                 return "La valeur minimale pour la compétence\n" + competence(i, 2) + " est de -11"
```

```
631             if creation and cd > 3:
632                 return "La valeur maximale pour la compétence\n" + competence(i, 2) + " est de 3 à la création"
633 
634             # Decalage de 11 pour avoir des indices partant de 0
635             v = 11 + cd
636             scd = -11
637 
638             # On compte les points utilisés par tranches de niveaux
639             for j in range(v):
640                 scd += 1
641                 # de -11 à -8 : 5 points
642                 if scd >= -11 and scd <= -8:
643                     xd -= 5
644                 # de -7 à -4 : 10 points
645                 if scd >= -7 and scd <= -4:
646                     xd -= 10
647                 # de -3 à 0 : 15 points
648                 if scd >= -3 and scd <= 0:
649                     xd -= 15
650                 # de +1 à +20 : 20 points
651                 if scd >= 1:
652                     xd -= 20
653 
654         # Competences draconic Thanatos
655         # Les compétences draconic ne peuvent pas être inférieures à -11
656         # A la création aucune ne peut être supérieure à +3
657         ct = person["Competence"]["Draconic"]["Thanatos"]
658         if ct < -11:
659             return "La valeur minimale pour la compétence\nThanatos est de -11"
660         if creation and ct > 3:
661             return "La valeur maximale pour la compétence\nThanatos\n est de 3 à la création"
662 
663         # Decalage de 11 pour avoir des indices partant de 0
664         v = 11 + ct
665         scd = -11
```

```
666
667         # On compte les points utilisés par tranches de niveaux
668         for j in range(v):
669             scd += 1
670             # de -11 à -8 : 10 points
671             if scd >= -11 and scd <= -8:
672                 xd -= 10
673             # de -7 à -4 : 20 points
674             if scd >= -7 and scd <= -4:
675                 xd -= 20
676             # de -3 à 0 : 30 points
677             if scd >= -3 and scd <= 0:
678                 xd -= 30
679             # de +1 à +20 : 40 points
680             if scd >= 1:
681                 xd -= 40
682
683     # Compétences de combat :  mélée
684     for i in range(47, 60):
685
686         # Ces compétences de combat ne peuvent pas être inférieures à -6
687         # A la création aucune ne peut être supérieure à +3
688         ccm = person["Competence"]["CMelee"][competence(i, 1)]
689         if ccm < -6:
690             return "La valeur minimale pour la compétence\n" + competence(i, 2) + " est de -6"
691         if creation and ccm > 3:
692             return "La valeur maximale pour la compétence\n" + competence(i, 2) + " est de 3 à la création"
693
694         # Decalage de 6 pour avoir des indices partant de 0
695         v = 6 + ccm
696         cm = -6
697
698         for j in range(v):
699             cm += 1
700             # de -6 à -4 : 10 points
```

```
701             if (cm >= -6) and (cm <= -4):
702                 x -= 10
703             # de -3 à 0 : 15 points
704             if (cm >= -3) and (cm <= 0):
705                 x -= 15
706             # de +1 à +20 : 20 points
707             if cm >= 1:
708                 x -= 20
709 
710     # Compétences de combat : Tir et Lancer
711     for i in range(60, 66):
712 
713         # Ces compétences de combat ne peuvent pas être inférieures à -8
714         # A la création aucune ne peut être supérieure à +3
715         ctl = person["Competence"]["Tir"][competence(i, 1)]
716         if ctl < -8:
717             return "La valeur minimale pour la compétence\n" + competence(i, 2) + " est de -8"
718         if creation and ctl > 3:
719             return "La valeur maximale pour la compétence\n" + competence(i, 2) + " est de 3 à la création"
720 
721         # Decalage de 6 pour avoir des indices partant de 0
722         v = 8 + ctl
723         tl = -8
724         for l in range(v):
725             tl += 1
726             # de -11 à -8 : 5 points
727             if tl >= -11 and tl <= -8:
728                 x -= 5
729             # de -7 à -4 : 10 points
730             if tl >= -7 and tl <= -4:
731                 x -= 10
732             # de -3 à 0 : 15 points
733             if tl >= -3 and tl <= 0:
734                 x -= 15
735             # de +1 à +20 : 20 points
```

```
736             if tl >= 1:
737                 x -= 20
738 
739     # On vérifie que tous les points ont été utilisés
740     if x+xd > 0:
741         return "Il reste encore " + str(x+xd) + " points"
742     elif x+xd < 0:
743         return "Attention le seuil de point est dépassé de " + str(-(x+xd)) + " points"
744     return None
745 
746 # La fonction calculer met à jour le personnage à partir des données saisie
747 # Elle recalcule les valeur qui dépendent d'autres valeurs
748 # Elle retourne le dictionnaire personnage modifié
749 def recalculer (person):
750 
751     TabConst = [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6]
752     TabTaille = [0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5]
753     Tabdmg = [-1, -1, -1, -1, -1, -1, 0, 0, 0, 0, 1, 1, 2, 2]
754 
755     # calcul des caractéristiques (selon le livre de règles page 11)
756     person["Caracteristique"]["Melee"] = (person["Caracteristique"]["Force"] + person["Caracteristique"]["Agilite"]) // 2
757     person["Caracteristique"]["Tir"] = (person["Caracteristique"]["Vue"] + person["Caracteristique"]["Dexterite"]) // 2
758     person["Caracteristique"]["Lancer"] = (person["Caracteristique"]["Tir"] + person["Caracteristique"]["Force"]) // 2
759     person["Caracteristique"]["Derobee"] = (person["Caracteristique"]["Agilite"] + 21 - person["Caracteristique"]["Taille"
    ]) // 2
760 
761     # calcul des points (formules du livre page 22)
762 
763     person["Point"]["Vie"] = (person["Caracteristique"]["Taille"] + person["Caracteristique"]["Constitution"]) / 2
764     person["Point"]["Endurance"] = person["Caracteristique"]["Taille"] + person["Caracteristique"]["Constitution"]
765     person["Point"]["Seuil_Constitution"] = TabConst[person["Caracteristique"]["Constitution"]]
766     person["Point"]["Sustain"] = TabTaille[person["Caracteristique"]["Taille"]]
767     person["Point"]["PlusDmg"] = Tabdmg [(person["Caracteristique"]["Taille"] + person["Caracteristique"]["Force"] )// 2]
768     person["Point"]["Encombrement"] = (person["Caracteristique"]["Taille"] + person["Caracteristique"]["Force"]) / 2
769 
```

```
770     # Incrément de la version du personnage
771     person["Version"] += 1
772
773     return person
774
```

```
 1 # Rêve de Dragon
 2 # ISN Terminale S Lycée Murat Issoire 2016-2017
 3 # Par Jikael Larriven, Marguerite Sobkowicz, Cédric Mongiat
 4 #
 5 # This program is free software: you can redistribute it and/or modify
 6 # it under the terms of the GNU General Public License as published by
 7 # the Free Software Foundation, either version 3 of the License, or
 8 # (at your option) any later version.
 9 #
10 # This program is distributed in the hope that it will be useful,
11 # but WITHOUT ANY WARRANTY; without even the implied warranty of
12 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13 # GNU General Public License for more details.
14 #
15 # You should have received a copy of the GNU General Public License
16 # along with this program.  If not, see <http://www.gnu.org/licenses/>.
17 #
18 # fenetre.py
19 # Classe de gestion de la fenetre
20 # La classe gère tous les objets affichés dans la fenêtre du programme:
21 # - Menus, permettant d'activer les commandes
22 # - Frames, contenant les différents widgets d'affichage
23 # A sa création, la classe reçoit en paramètre l'objet job de classe Jeu
24 #
25 # Utilisation:
26 #  from fenetre import Fenetre
27 #  wnd = Fenetre(job)
28 #
29
30 from tkinter import Tk, StringVar, IntVar, Label, Entry, BooleanVar, Checkbutton, Listbox, Menu, Frame, LabelFrame, Canvas,
    \
31     PhotoImage, Button, Text, Scrollbar, Toplevel
32 from tkinter.messagebox import *
33 import personnage
34
```

```
35 class Fenetre:
36
37     def __init__(self, root, job):
38
39         # Récupération de l'objet Jeu
40         self.jeu = job
41         self.saved = True
42
43         # Création de la fenêtre principale
44         self.root = root
45         self.root.title('Rêve de Dragon')
46         self.root.resizable(True, True)
47
48         # Création des menus
49         # On a 4 menu principaux : filemenu, cmdmenu, viewmenu et helpmenu
50         self.menubar = Menu(root)
51         self.root.config(menu=self.menubar)
52
53         # filemenu: menu de manipulation des fichiers contenant les personnages
54         self.filemenu = Menu(self.menubar, tearoff=0)
55         self.menubar.add_cascade(label="Fichier", menu=self.filemenu)
56         self.filemenu.add_command(label="Nouveau", command=self.nouveau)
57         self.filemenu.add_command(label="Ouvrir", command=self.ouvrir)
58         self.filemenu.add_separator()
59         self.filemenu.add_command(label="Enregistrer", command=self.jeu.enregistrer)
60         self.filemenu.add_command(label="Enregistrer sous...", command=self.jeu.enregistrer_sous)
61         self.filemenu.add_separator()
62         self.filemenu.add_command(label="Fermer", command=self.fermer)
63         self.filemenu.add_separator()
64         self.filemenu.add_command(label="Imprimer", command=self.void, state='disabled')
65         self.filemenu.add_separator()
66         self.filemenu.add_command(label="Quitter", command=self.quitter)
67
68         # cmdmenu: menu des commandes sur les personnages
69         self.cmdmenu = Menu(self.menubar, tearoff=0)
```

```
70          self.menubar.add_cascade(label="Commande", menu=self.cmdmenu)
71          self.cmdmenu.add_command(label="Nouvelle Partie", command=self.partie)
72          self.cmdmenu.add_separator()
73          self.cmdmenu.add_command(label="Nouveau Personnage", command=self.creer)
74          self.cmdmenu.add_separator()
75          self.cmdmenu.add_command(label="Valider le Personnage", command=self.valider)
76
77          # viewmenu: menu de sélection du personnage à l'affichage
78          # Ce menu est vide en l'absence de personnage
79          # Il est rempli au chargement ou à la création d'un personnage
80          self.viewmenu = Menu(self.menubar, tearoff=0)
81          self.menubar.add_cascade(label="Personnage", menu=self.viewmenu)
82
83          # helpmenu: menu d'aide
84          self.helpmenu = Menu(self.menubar, tearoff=0)
85          self.menubar.add_cascade(label="Aide", menu=self.helpmenu)
86          self.helpmenu.add_command(label="Règles du Jeu", command=self.regles)
87          self.helpmenu.add_command(label="Utilisation du Programme", command=self.utilise)
88          self.helpmenu.add_command(label="A Propos...", command=self.a_propos)
89
90          # frame1 : Fiche du personnage
91          frame1 = Frame(root, borderwidth=0, relief='flat', height=200, width=600)
92          frame1.grid(row=0, column=0, sticky='NW', padx="10", pady="5")
93
94          # Nom
95          self.Entry_Nom = StringVar()
96          self.Old_Nom = ""
97          Label(frame1, text='Nom:').grid(row=0, column=0, columnspan=2, sticky='E')
98          Entry(frame1, textvariable=self.Entry_Nom, justify='left', width=34)\
99              .grid(row=0, column=2, columnspan=4, sticky='W', padx="5")
100
101         # Age
102         self.Entry_Age = IntVar()
103         Label(frame1, text='Age:').grid(row=1, column=0, columnspan=2, sticky='E')
104         Entry(frame1, textvariable=self.Entry_Age, justify='right', width=3)\
```

```
105             .grid(row=1, column=2, sticky='W', padx="5")
106
107         # Heure de naissance (pour hauts-rêvants)
108         self.Entry_Heure = IntVar()
109         Label(frame1, text='Heure de Naissance:').grid(row=1, column=3, sticky='E')
110         Entry(frame1, textvariable=self.Entry_Heure, justify='right', width=3) \
111             .grid(row=1, column=4, sticky='W', padx="5")
112
113         # Taille
114         self.Entry_Taille = IntVar()
115         Label(frame1, text='Taille:').grid(row=1, column=5, sticky='E')
116         Entry(frame1, textvariable=self.Entry_Taille, justify='right', width=3)\
117             .grid(row=1, column=6, sticky='W', padx="5")
118
119         # Poids
120         self.Entry_Poids = IntVar()
121         Label(frame1, text='Poids:').grid(row=1, column=7, sticky='E')
122         Entry(frame1, textvariable=self.Entry_Poids, justify='right', width=3)\
123             .grid(row=1, column=8, sticky='W', padx="5")
124
125         # Beauté
126         self.Entry_Beaute = IntVar()
127         Label(frame1, text='Beauté:').grid(row=2, column=0, columnspan=2, sticky='E')
128         Entry(frame1, textvariable=self.Entry_Beaute, justify='right', width=3) \
129             .grid(row=2, column=2, sticky='W', padx="5")
130
131         # Cheveux
132         self.Entry_Cheveux = StringVar()
133         Label(frame1, text='Cheveux:').grid(row=2, column=3, sticky='E')
134         Entry(frame1, textvariable=self.Entry_Cheveux, justify='left', width=8)\
135             .grid(row=2, column=4, sticky='W', padx="5")
136
137         # Yeux
138         self.Entry_Yeux = StringVar()
139         Label(frame1, text='Yeux:').grid(row=2, column=5, sticky='E')
```

```
140         Entry(frame1, textvariable=self.Entry_Yeux, justify='left', width=8)\
141             .grid(row=2, column=6, sticky='W', padx="5")
142 
143         # Haut rêvant
144         self.Entry_HRevant = IntVar()
145         Checkbutton(frame1, text="Haut-Rêvant", variable=self.Entry_HRevant, command=self.sel_revant) \
146             .grid(row=2, column=7, columnspan=2, sticky='W', padx="5")
147 
148         # Sexe
149         self.Entry_Sexe = StringVar()
150         Label(frame1, text='Sexe:').grid(row=3, column=0, columnspan=2, sticky='E')
151         Entry(frame1, textvariable=self.Entry_Sexe, justify='left', width=2)\
152             .grid(row=3, column=2, sticky='W', padx="5")
153 
154         # Ambidextre
155         self.Entry_Ambidextre = IntVar()
156         Label(frame1, text='Ambidextre:').grid(row=3, column=3, sticky='E')
157         Entry(frame1, textvariable=self.Entry_Ambidextre, justify='right', width=3)\
158             .grid(row=3, column=4, sticky='W', padx="5")
159 
160         # Signes Particuliers
161         self.Entry_SignesP = StringVar()
162         Label(frame1, text='Signes Particuliers:').grid(row=3, column=5, sticky='E')
163         Entry(frame1, textvariable=self.Entry_SignesP, justify='left', width=37)\
164             .grid(row=3, column=6, columnspan=3, sticky='W', padx="5")
165 
166         # Frame 2 : Caractéristiques
167         frame2 = LabelFrame(root, text=" Caractéristiques ", borderwidth=2, relief='ridge', height=200, width=600)
168         frame2.grid(row=1, column=0, sticky='NW', padx="10", pady="5")
169         frame20 = LabelFrame(frame2, text=' Physiques ', borderwidth=2, relief='ridge', height=200, width=200)
170         frame20.grid(row=0, column=0, sticky='NW', padx="5", pady="5")
171         frame21 = LabelFrame(frame2, text=' Mentales ', borderwidth=2, relief='ridge', height=200, width=200)
172         frame21.grid(row=0, column=1, sticky='NW', padx="5", pady="5")
173         frame22 = LabelFrame(frame2, text=' Pouvoirs ', borderwidth=2, relief='ridge', height=200, width=200)
174         frame22.grid(row=0, column=2, sticky='NW', padx="5", pady="5")
```

```
175         frame23 = LabelFrame(frame2, text=' Dérivées ', borderwidth=2, relief='ridge', height=200, width=200)
176         frame23.grid(row=0, column=3, sticky='NW', padx="5", pady="5")
177         self.Entry_C = []
178 
179         # Colonne 0 de taille à Dextérité
180         for i in range(0, 6):
181             self.Entry_C.append(IntVar())
182             Label(frame20, text="           "+personnage.caracteristique(i, 1)+':')\
183                 .grid(row=i, column=0, sticky='E')
184             Entry(frame20, textvariable=self.Entry_C[i], justify='right', width=3)\
185                 .grid(row=i, column=1, sticky='W', padx="5")
186         Label(frame20, text=' ').grid(row=6, column=0, sticky='E')
187 
188         # Colonne 1 de Vue à Empathie
189         for i in range(6, 12):
190             self.Entry_C.append(IntVar())
191             Label(frame21, text="           "+personnage.caracteristique(i, 1) + ':')\
192                 .grid(row=i-6, column=0, sticky='E')
193             Entry(frame21, textvariable=self.Entry_C[i], justify='right', width=3)\
194                 .grid(row=i-6, column=1, sticky='W', padx="5")
195         Label(frame21, text=' ').grid(row=6, column=0, sticky='E')
196 
197         # Colonne 2 de Rêve à Chance
198         for i in range(12, 14):
199             self.Entry_C.append(IntVar())
200             Label(frame22, text="               "+personnage.caracteristique(i, 1) + ':')\
201                 .grid(row=i-12, column=0, sticky='E')
202             Entry(frame22, textvariable=self.Entry_C[i], justify='right', width=3)\
203                 .grid(row=i-12, column=1, sticky='W', padx="5")
204         for i in range(2,7):
205             Label(frame22, text=' ').grid(row=i, column=0, sticky='E')
206 
207         # Colonne 3 de Tir à Dérobée (ne peuvent être saisies)
208         for i in range(14, 18):
209             self.Entry_C.append(IntVar())
```

```
210             Label(frame23, text="              "+personnage.caracteristique(i, 1) + ':')\
211                 .grid(row=i-14, column=0, sticky='E')
212             Entry(frame23, textvariable=self.Entry_C[i], justify='right', width=3, state='disabled')\
213                 .grid(row=i-14, column=1, sticky='W', padx="5")
214         for i in range(4,7):
215             Label(frame23, text=' ').grid(row=i, column=0, sticky='E')
216 
217         # frame 3 : Points et Seuils (ne peuvent être saisis)
218         frame3 = Frame(root, borderwidth=0, relief='flat', height=200, width=600, padx="5", pady="5")
219         frame3.grid(row=2, column=0, sticky='NW', padx="10")
220         self.Entry_P = []
221 
222         # Vie - Endurance - Encombrement
223         for i in range(0, 3):
224             self.Entry_P.append(IntVar())
225             Label(frame3, text=personnage.point(i, 1) + ':').grid(row=0, column=2*i, sticky='E')
226             Entry(frame3, textvariable=self.Entry_P[i], justify='right', width=3, state='disabled')\
227                 .grid(row=0, column=2*i+1, sticky='W', padx="5")
228 
229         # Bonus aux Dommages - Malus Armure - Seuil de Constitution - Seuil de Sustentation
230         for i in range(3, 7):
231             self.Entry_P.append(IntVar())
232             Label(frame3, text=personnage.point(i, 1) + ':').grid(row=1, column=2*i-6, sticky='E')
233             Entry(frame3, textvariable=self.Entry_P[i], justify='right', width=3, state='disabled')\
234                 .grid(row=1, column=2*i-5, sticky='W', padx="5")
235 
236         # frame 4 : Compétences
237         frame4 = LabelFrame(root, text=" Compétences ", borderwidth=2, relief='ridge', height=200, width=800)
238         frame4.grid(row=3, column=0, columnspan=2, sticky='NW', padx="10", pady="5")
239         frame40 = LabelFrame(frame4, text=' Générales ', borderwidth=2, relief='ridge', height=200, width=300)
240         frame40.grid(row=0, column=0, rowspan=2, sticky='NW', padx="5", pady="5")
241         frame41 = LabelFrame(frame4, text=' Particulières ', borderwidth=2, relief='ridge', height=200, width=300)
242         frame41.grid(row=0, column=1, rowspan=2, sticky='NW', padx="5", pady="5")
243         frame42 = LabelFrame(frame4, text=' Spécialisées ', borderwidth=2, relief='ridge', height=200, width=300)
244         frame42.grid(row=0, column=2, rowspan=2, sticky='NW', padx="5", pady="5")
```

```
245         frame43 = LabelFrame(frame4, text=' Connaissances ', borderwidth=2, relief='ridge', height=200, width=300)
246         frame43.grid(row=0, column=3, sticky='NW', padx="5", pady="5")
247         frame44 = LabelFrame(frame4, text=' Draconic ', borderwidth=2, relief='ridge', height=200, width=300)
248         frame44.grid(row=1, column=3, sticky='SW', padx="5", pady="5")
249         frame45 = LabelFrame(frame4, text=' Combat Mélée ', borderwidth=2, relief='ridge', height=200, width=300)
250         frame45.grid(row=0, column=4, rowspan=2, sticky='NW', padx="5", pady="5")
251         frame46 = LabelFrame(frame4, text=' Combat Tir-Lancer ', borderwidth=2, relief='ridge', height=200, width=300)
252         frame46.grid(row=0, column=5, rowspan=2, sticky='NW', padx="5", pady="5")
253         self.Entry_A = []
254 
255         # Colonne 0 : Générales
256         for i in range(0, 11):
257             self.Entry_A.append(IntVar())
258             Label(frame40, text="           "+personnage.competence(i, 2)+':').grid(row=i, column=0, sticky='E')
259             Entry(frame40, textvariable=self.Entry_A[i], justify='right', width=3)\
260                 .grid(row=i, column=1, sticky='W', padx="5")
261         for i in range(11, 15):
262             Label(frame40, text=' ').grid(row=i, column=0, sticky='E')
263 
264         # Colonne 1 : Particulières
265         for i in range(11, 26):
266             self.Entry_A.append(IntVar())
267             Label(frame41, text="      "+personnage.competence(i, 2)+':').grid(row=i-11, column=0, sticky='E')
268             Entry(frame41, textvariable=self.Entry_A[i], justify='right', width=3)\
269                 .grid(row=i-11, column=1, sticky='W', padx="5")
270 
271         # Colonne 2 : Spécialisées
272         for i in range(26, 36):
273             self.Entry_A.append(IntVar())
274             Label(frame42, text="       "+personnage.competence(i, 2)+':')\
275                 .grid(row=i-25, column=0, sticky='E')
276             Entry(frame42, textvariable=self.Entry_A[i], justify='right', width=3)\
277                 .grid(row=i-25, column=1, sticky='W', padx="5")
278         for i in range(10, 15):
279             Label(frame42, text=' ').grid(row=i+1, column=0, sticky='E')
```

```
280
281         # Colonne 3: Connaissances
282         for i in range(36, 43):
283             self.Entry_A.append(IntVar())
284             Label(frame43, text="         "+personnage.competence(i, 2)+':')\
285                 .grid(row=i-35, column=0, sticky='E')
286             Entry(frame43, textvariable=self.Entry_A[i], justify='right', width=3)\
287                 .grid(row=i-35, column=1, sticky='W', padx="5")
288         Label(frame43, text=' ').grid(row=8, column=0, sticky='E')
289
290         # Colonne 3 : Draconic
291         self.Draconic = []
292         for i in range(0, 4):
293             self.Entry_A.append(IntVar())
294             Label(frame44, text="             "+personnage.competence(i+43, 2)+':')\
295                 .grid(row=i, column=0, sticky='E')
296             self.Draconic.append(Entry(frame44, textvariable=self.Entry_A[i+43], justify='right', width=3))
297             self.Draconic[i].grid(row=i, column=1, sticky='W', padx="5")
298         Label(frame44, text=' ').grid(row=4, column=0, sticky='E')
299
300         # Colonne 4 : Combat Mélée
301         for i in range(47, 60):
302             self.Entry_A.append(IntVar())
303             Label(frame45, text=personnage.competence(i, 2) + ':') \
304                 .grid(row=i - 46, column=0, sticky='E')
305             Entry(frame45, textvariable=self.Entry_A[i], justify='right', width=3) \
306                 .grid(row=i - 46, column=1, sticky='W', padx="5")
307         for i in range(13, 15):
308             Label(frame45, text=' ').grid(row=i+1, column=0, sticky='E')
309
310         # Colonne 5 : Combat Tir
311         for i in range(60, 66):
312             self.Entry_A.append(IntVar())
313             Label(frame46, text="         " + personnage.competence(i, 2) + ':') \
314                 .grid(row=i - 59, column=0, sticky='E')
```

```
315             Entry(frame46, textvariable=self.Entry_A[i], justify='right', width=3) \
316                 .grid(row=i - 59, column=1, sticky='W', padx="5")
317         for i in range(6, 15):
318             Label(frame46, text=' ').grid(row=i+1, column=0, sticky='E')
319 
320         # frame5 : table de résolution et lancer de dé
321         frame5 = LabelFrame(root, text=" Résolution et Lancer de Dés ", borderwidth=2, relief='ridge', height=200, width=
    600)
322         frame5.grid(row=0, column=1, rowspan=3, columnspan=2, sticky='NW', padx="10", pady="5")
323 
324         # Listbox caractéristiques
325         Label(frame5, text='         Caractéristique:').grid(row=0, column=0, columnspan=2, padx="10", sticky='NW')
326         self.liste1 = Listbox(frame5, height=13, width=18, relief='sunken')
327         self.liste1.grid(row=1, column=1, sticky='NW', pady="5")
328         for i in range(0, 18):
329             self.liste1.insert(i, personnage.caracteristique(i, 1))
330         self.liste1.bind('<<ListboxSelect>>',self.sel_liste1)
331 
332         # Listbox compétences
333         Label(frame5, text='Compétence:').grid(row=0, column=2, columnspan=2, padx="10", sticky='NW')
334         self.liste2 = Listbox(frame5, height=13, width=18, relief='sunken')
335         self.liste2.grid(row=1, column=3, sticky='NW', pady="5")
336         for i in range(0, 66):
337             self.liste2.insert(i, personnage.competence(i, 2))
338         self.liste2.bind('<<ListboxSelect>>', self.sel_liste2)
339 
340         # Zone de résulats
341         self.Entry_R_C_Val = IntVar()
342         Entry(frame5, textvariable=self.Entry_R_C_Val, justify='right', width=3,) \
343             .grid(row=16, column=0, sticky='E', padx="10")
344         self.Entry_R_C_Name = StringVar()
345         Entry(frame5, textvariable=self.Entry_R_C_Name, justify='left', width=18, state='disabled') \
346             .grid(row=16, column=1, sticky='W')
347         self.Entry_R_A_Val = IntVar()
348         Entry(frame5, textvariable=self.Entry_R_A_Val, justify='right', width=3,) \
```

```
349             .grid(row=16, column=2, sticky='E', padx="10")
350         self.Entry_R_A_Name = StringVar()
351         Entry(frame5, textvariable=self.Entry_R_A_Name, justify='left', width=18, state='disabled') \
352             .grid(row=16, column=3, sticky='W')
353         Label(frame5, text='   Seuil de Réussite:').grid(row=17, column=0, sticky='NE')
354         self.Entry_R_Seuil = IntVar()
355         Entry(frame5, textvariable=self.Entry_R_Seuil, justify='right', width=3, state='disabled')\
356             .grid(row=17, column=1, sticky='W', padx="10")
357         Label(frame5, text='Tirage:').grid(row=17, column=2, sticky='NE')
358         self.Entry_R_Tirage = IntVar()
359         Entry(frame5, textvariable=self.Entry_R_Tirage, justify='right', width=3, state='disabled') \
360             .grid(row=17, column=3, sticky='W', padx="10")
361         Label(frame5, text='Résultat Spécial:').grid(row=18, column=0, sticky='NE')
362         self.Entry_R_Special = StringVar()
363         Entry(frame5, textvariable=self.Entry_R_Special, justify='left', width=30, state='disabled') \
364             .grid(row=18, column=1, columnspan=2, sticky='W', padx="10")
365         Label(frame5, text=' ').grid(row=19, column=4, sticky='NE')
366 
367         # Bouton pour le lancer de Dés
368         Button(frame5, text="Lancer les Dés", command=self.lancer) \
369             .grid(row=18, column=3, columnspan=3, sticky='W', padx="10")
370 
371         # La mascote
372         # On la fait déborder sur le frame4 pour gagner en largeur totale
373         self.dragon = PhotoImage(file='./dragon3.gif')
374         logo = Canvas(root, width=200, height=181, bd=1, relief='ridge')
375         logo.grid(row=3, column=1, columnspan=2, sticky='SE', padx="10", pady="3")
376         logo.create_image(0, 0, image=self.dragon, anchor='nw')
377 
378         # L'ecran étant initialisé, on peut créér un premier personnage par défaut
379         self.creer()
380         return
381 
382     # Fonction de recopie de la sélection depuis la Listbox des caractéristiques
383     # Met à jour les 2 champs points et nom de caractéristique pour le calcul de résolution
```

```
384     def sel_liste1(self, event):
385
386         if self.liste1.curselection() != ():
387             index = self.liste1.curselection()[0]
388             self.Entry_R_C_Name.set(self.liste1.get(index))
389             self.Entry_R_C_Val.set(self.Entry_C[index].get())
390         return
391
392     # Fonction de recopie de la sélection depuis la Listbox des compétences
393     # Met à jour les 2 champs points et nom de compétence pour le calcul de résolution
394     def sel_liste2(self, event):
395
396         if self.liste2.curselection() != ():
397             index = self.liste2.curselection()[0]
398             self.Entry_R_A_Name.set(self.liste2.get(index))
399             self.Entry_R_A_Val.set(self.Entry_A[index].get())
400         return
401
402     # Fonction de changement d'etat haut-rêvant
403     def sel_revant(self):
404
405         if self.Entry_HRevant.get() != 1:
406             for i in range(0, 4):
407                 self.Entry_A[i+42].set(-11)
408                 self.Draconic[i].configure(state='disabled')
409         else:
410             for i in range(0, 4):
411                 self.Draconic[i].configure(state='normal')
412         return
413
414     # Nouveau jeu
415     # Il faut préalablement fermer le jeu en cours
416     def nouveau(self):
417
418         if self.fermer():
```

```
419             self.jeu.nouveau()
420         return
421 
422     # Ouvrir jeu
423     # Il faut préalablement fermer le jeu en cours
424     # On reçoit le nom du jeu suivi d'une liste de personnages ou None si rien d'ouvert par le jeu
425     def ouvrir(self):
426 
427         if self.fermer():
428             names = self.jeu.ouvrir()
429             if names != None:
430                 numero = -1
431                 for person in names:
432 
433                     # index 0 : nom du fichier jeu
434                     if numero < 0:
435                         self.root.title('Rêve de Dragon - ' + person)
436 
437                     # autres index : personnages
438                     # index vaudra le nombre de personnages reçus
439                     else:
440                         self.viewmenu.add_command(label=person, command=lambda index=numero: self.selectionner(index))
441                     numero += 1
442 
443                 # On affiche le premier personnage
444                 if numero > 0:
445                     self.selectionner(0)
446 
447         return
448 
449     # Fermer le jeu en cours
450     # On efface tous les personnages
451     # on cree un nouveau personnage vide pour obtenir un affichage vierge
452     def fermer(self):
453 
```

```
454         if not self.saved:
455             self.saved = askyesno('Fermer', 'Voulez-vous vraiment fermer ce Jeu ?\nLes données non enregistrées seront
    perdues')
456         if self.saved:
457             last = self.viewmenu.index("end")
458             if last is not None:
459                 for i in range(last + 1):
460                     self.viewmenu.delete(0)
461             self.root.title('Rêve de Dragon')
462             self.jeu.fermer()
463             self.creer()
464         return self.saved
465 
466     # Quitter le programme
467     # onh détruit la fenêtre et on quitte
468     def quitter(self):
469 
470         if askyesno('Quitter', 'Voulez-vous vraiment quitter le programme ?'):
471             self.root.destroy()
472             self.root.quit()
473         return
474 
475     # Fonction interne d'affichage des données d'un personnage
476     # Copie toutes les données du dictionnaire local dans les variables associées aux champs de saisie
477     def affiche(self):
478 
479         self.Entry_Nom.set(self.pod["Fiche"]["Nom"])
480         self.Entry_Age.set(self.pod["Fiche"]["Age"])
481         self.Entry_Heure.set(self.pod["Fiche"]["Heure_Naissance"])
482         self.Entry_Taille.set(self.pod["Fiche"]["Taille"])
483         self.Entry_Poids.set(self.pod["Fiche"]["Poids"])
484         self.Entry_Sexe.set(self.pod["Fiche"]["Sexe"])
485         self.Entry_Cheveux.set(self.pod["Fiche"]["Cheveux"])
486         self.Entry_Yeux.set(self.pod["Fiche"]["Yeux"])
487         self.Entry_Beaute.set(self.pod["Fiche"]["Beaute"])
```

```
488         self.Entry_Ambidextre.set(self.pod["Fiche"]["Ambidextre"])
489         self.Entry_HRevant.set(self.pod["Fiche"]["Haut_Revant"])
490         self.Entry_SignesP.set(self.pod["Fiche"]["Signes_Particulier"])
491         for i in range(0, 18):
492             self.Entry_C[i].set(self.pod["Caracteristique"][personnage.caracteristique(i, 0)])
493         for i in range(0, 7):
494             self.Entry_P[i].set(self.pod["Point"][personnage.point(i, 0)])
495         for i in range(0, 66):
496             self.Entry_A[i].set(self.pod["Competence"][personnage.competence(i, 0)][personnage.competence(i, 1)])
497         if self.Entry_HRevant.get() != 1:
498             for i in range(0, 4):
499                 self.Draconic[i].configure(state='disabled')
500         return
501 
502     # Création d'un nouveau personnage
503     # On demande au jeu de créer un nouveau personnage dans la liste
504     # On initialise toutes les variables de saisie aux valeur reçues
505     def creer(self):
506 
507         self.pod = self.jeu.creer()
508         self.affiche()
509         return
510 
511     # Validation des données du personnage
512     # On reconstitue le dictionnaire qui est envoyé au jeu pour vérification
513     # Le jeu répond avec un dictionnaire contenant
514     # - l'index du personnage
515     # - Le nom de personnage
516     # - Un message d'erreur ou d'acceptation
517     def valider(self):
518 
519         if len(self.Entry_Nom.get()) < 1:
520             return
521 
522         self.pod["Fiche"]["Nom"] = self.Entry_Nom.get()
```

```
523         self.pod["Fiche"]["Age"] = self.Entry_Age.get()
524         self.pod["Fiche"]["Heure_Naissance"] = self.Entry_Heure.get()
525         self.pod["Fiche"]["Taille"] = self.Entry_Taille.get()
526         self.pod["Fiche"]["Poids"] = self.Entry_Poids.get()
527         self.pod["Fiche"]["Sexe"] = self.Entry_Sexe.get()
528         self.pod["Fiche"]["Cheveux"] = self.Entry_Cheveux.get()
529         self.pod["Fiche"]["Yeux"] = self.Entry_Yeux.get()
530         self.pod["Fiche"]["Beaute"] = self.Entry_Beaute.get()
531         self.pod["Fiche"]["Ambidextre"] = self.Entry_Ambidextre.get()
532         self.pod["Fiche"]["Haut_Revant"] = self.Entry_HRevant.get()
533         self.pod["Fiche"]["Signes_Particulier"] = self.Entry_SignesP.get()
534         for i in range(0, 18):
535             self.pod["Caracteristique"][personnage.caracteristique(i, 0)] = self.Entry_C[i].get()
536         for i in range(0, 7):
537             self.pod["Point"][personnage.point(i, 0)] = self.Entry_P[i].get()
538         for i in range(0, 65):
539             self.pod["Competence"][personnage.competence(i, 0)][personnage.competence(i, 1)] = self.Entry_A[i].get()
540         retour = self.jeu.valider(self.pod)
541         index = retour["index"]
542 
543         # On a bien un index valide : alors on met à jour le menu et on va chercher les données du personnage
544         if index is not None:
545             if self.viewmenu.entrycget(index, 'label') != self.Old_Nom:
546                 self.viewmenu.entryconfigure(index, label=retour["nom"])
547             elif self.viewmenu.entrycget(index,'label') != retour["nom"]:
548                 self.viewmenu.add_command(label=retour["nom"], command=lambda index=index: self.selectionner(index))
549             self.pod = self.jeu.selectionner(index)
550             self.affiche()
551 
552         # En cas d'erreur ou retour ok, il y a un message du jeu
553         if len(retour["message"]):
554             showerror("Validation", retour["message"])
555 
556         self.saved = False
557         return
```

```
558
559     # Sélection d'un personnage depuis le menu
560     # On envoie au jeu l'index du menu qui correspond à l'index de la liste du jeu
561     # Les données du personnage reçu sont ensuite affichées
562     def selectionner(self, code):
563
564         self.pod = self.jeu.selectionner(code)
565         self.affiche()
566         return
567
568     # Nouvelle partie
569     # On demande au Jeu de changer de partie
570     # Le jeu renvoie le contenu du personnage courant
571     def partie(self):
572
573         if askyesno('Nouvelle Partie', 'Voulez-vous vraiment terminer cette partie ?\nLes données non enregistrées seront
    perdues'):
574             self.pod = self.jeu.partie()
575             self.affiche()
576         return
577
578     # Lancer de dé
579     # Calcul de résolution puis lancer des dés
580     # On passe au jeu les valeurs de caractéristiques et compétences sélectionnées
581     # Le résultat sera récupéré en retour et affiché
582     def lancer(self):
583
584         resultat = self.jeu.lancer(self.Entry_R_C_Val.get(),self.Entry_R_A_Val.get())
585         self.Entry_R_Seuil.set(resultat["seuil"])
586         self.Entry_R_Tirage.set(resultat["tirage"])
587         self.Entry_R_Special.set(resultat["special"])
588         self.saved = False
589         return
590
591     # Affichage de la boite de dialogue A propos
```

```
592     # Le texte est dans le fichier A_PROPOS.TXT
593     def a_propos(self):
594
595         fp = open("A_PROPOS.TXT", "r")
596         texte_a_propos = fp.read()
597         fp.close()
598         showinfo("A Propos de...", texte_a_propos)
599         return
600
601     # Dialogue d'aide pour connaitre les règles du jeu
602     # Le texte est dans le fichier REGLES.TXT
603     def regles(self):
604
605         file = "REGLE.TXT"
606         titre = "Règles du Jeu Rêve de Dragon."
607         self.aide(file,titre)
608         return
609
610     # Le texte est dans le fichier AIDE.TXT
611     def utilise(self):
612
613         file = "AIDE.TXT"
614         titre = "Utilisation de Rêve de Dragon."
615         self.aide(file,titre)
616         return
617
618     # Aide du jeu
619     # Affiche une boite de dialogue avec un widget texte contenant l'aide
620     def aide(self, file, titre):
621
622         # On ouvre une fenêtre fille de celle du jeu
623         self.wdw = Toplevel()
624         self.wdw.geometry('+400+100')
625         self.wdw.title(titre)
626
```

```
627         # Le texte de l'aide est stocké dans un fichier Atexte
628         fp = open(file, "r")
629         texte_aide = fp.read()
630         fp.close()
631 
632         # On l'affiche dans un widget Text avec une barre de défilement
633         # Ne fonctionne que si on utilise grid pour placer les Widgets
634         # Il faut mettre le widget en état disabled pour éviter que l'on y entre du texte
635         self.S = Scrollbar(self.wdw, orient='vertical')
636         self.T = Text(self.wdw, height=50, width=100, font=('TkDefaultFont',10))
637         self.T.grid(row=0, column=0, sticky='NW')
638         self.S.configure(command=self.T.yview)
639         self.T.configure(yscrollcommand=self.S.set)
640         self.S.grid(row=0, column=1, sticky='SN')
641         self.T.configure(state='normal')
642         self.T.insert('end', texte_aide)
643         self.T.configure(state='disabled')
644 
645         # La nouvelle fenêtre est ouverte en modal
646         # Il faudra la fermer pour reprendre le contrôle de la fenêtre principale
647         self.wdw.transient(self.root)
648         self.wdw.grab_set()
649         self.root.wait_window(self.wdw)
650         return
651 
652     # fonction qui ne fait rien (pour les tests)
653     def void(self):
654         return
```

```
# Rêve de Dragon
# ISN Terminale S Lycée Murat Issoire 2016-2017
# Par Jikael Larriven, Marguerite Sobkowicz, Cédric Mongiat
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
#
# jeu.py
# Classe du jeu
#
# Cette classe contient les données du jeu
# Elle inclus les objets de type personnage
# On peut:
# - L'initialiser avec un nouveau jeu vide (création par défaut)
# - Lui ajouter des personnages
# - Lui retirer des personnages
# - Sélectionner un personnage
# - Mettre à jour les données du personnage
# - La charger depuis un fichier texte
# - La sauvegarder dans un fichier texte
#
# On stocke les données dans des fichiers de type texte au format JSON (Javascript Object Notation)
# Les fichiers sont sous le même répertoire que l'application
# Afin de discriminer les fichier invalides on ajoute un entête unique à chaque fichier
#
```

```
36 # La gestion des fichiers est volontairement simplifiée et pourra être améliorée
37 # - On ne demande pas de confirmation si on ecrase un fichier existant
38 # - On considère les données du fichier valide à partir du moment ou l'entête correct est présent
39 # - On ne contrôle pas que les données ont été correctement sauvées avant de fermer ou ouvrir un autre fichier
40 #
41 # La création depuis le corps de programme main est:
42 #  from jeu import Jeu
43 #  job = Jeu()
44 # Les appels depuis la classe fenêtre sont:
45 #  job.nouveau()
46 #  liste = job.ouvrir()
47 #  job.enregistrer()
48 #  job.enregistrer_sous()
49 #  data = job.fermer()
50 #  personnage = job.creer()
51 #  personnage = job.selectionner(index)
52 #  string = job.valider(personnage)
53 #  dictionnaire = job.lancer(integer, integer)
54 #
55
56 import os
57 import json
58 import random
59 from tkinter import filedialog, messagebox
60 import personnage
61
62 class Jeu:
63     def __init__(self):
64
65         # Entête magique des fichiers jeux
66         self.magic = "MAGIC-ISN-V3.0"
67
68         # par défaut les fichiers sont stockés dans le répertoire du jeu
69         self.rep = os.getcwd()
70
```

```
71          # A l'initialisation aucun fichier jeu n'est chargé et les données sont vierges
72          # Il n'y a pas de personnages
73          self.filename = ''
74          self.data = []
75          self.current = None
76
77      # Nouveau jeu de données
78      # On efface la liste des personnages
79      def nouveau(self):
80
81          self.data = []
82          self.current = None
83          return
84
85      # Commande d'ouverture d'un fichier jeu
86      # On va utiliser une boite de dialogue système pour choisir un fichier
87      def ouvrir(self):
88
89          # Choix d'un fichier dans le répertoire courant
90          # Les fichiers sont de type texte (.txt) ou tous types (.*)
91          filename = filedialog.askopenfilename(
92              title="Ouvrir",
93              initialdir=self.rep,
94              filetypes=(("Text File", "*.txt"), ("All Files", "*.*")))
95
96          # Le dialogue renvoie une chaine vide si abandon de l'utilisateur
97          if len(filename) > 0:
98
99              # On conserve le nom du fichier utilisé
100             self.filename = filename
101
102             # Lecture du fichier
103             try:
104                 fp = open(self.filename, "r")
105             except:
```

```
106                 messagebox.showwarning(
107                     "Ouvrir",
108                     "Impossible d'ouvrir le fichier\n(%s)" % self.filename
109                 )
110                 return None
111 
112             # Vérification de l'entête magique (avec version du fichier)
113             magic_string = fp.read(len(self.magic))
114             if magic_string != self.magic:
115                 fp.close()
116                 messagebox.showwarning(
117                     "Ouvrir",
118                     "Le fichier n'est pas au bon format"
119                 )
120                 return None
121 
122             # Lecture du Dictionnaire au format JSON
123             json_string = fp.read()
124             fp.close()
125 
126             # Conversion du format JSON en liste
127             self.data = json.loads(json_string)
128 
129             # Extrait la liste des noms de personnages
130             # La liste names sera envoyée à Fenêtre pour affichage
131             # En premier on met le nom du fichier
132             # Ensuite les noms de personnages
133             names = []
134             names.append(filename)
135             for person in self.data:
136                 names.append(person["Fiche"]["Nom"])
137             return names
138 
139         # Pas de fichier ouvert on renvoie None
140         # Qui sera traité comme un abandon de la commande par Fenêtre
```

```
141         else:
142             return None
143 
144     # commande enregistrer sous le nom par défaut
145     def enregistrer(self):
146 
147         # L'enregistrement n'est possible que si un fichier existe
148         if len(self.filename) > 0:
149 
150             # Conversion du Dictionnaire en JSON
151             json_string = json.dumps(self.data)
152 
153             # Ecriture dans le fichier
154             # On ecrase tout fichier existant
155             fp = open(self.filename, "w")
156 
157             # Ecriture de l'entête magique
158             fp.write(self.magic)
159 
160             # Ecriture du Dictionnaire
161             fp.write(json_string)
162             fp.close()
163 
164         else:
165             messagebox.showwarning(
166                 "Enregistrer",
167                 "Aucun fichier sélectionné pour enregistrer"
168             )
169         return
170 
171     # commande enregistrer sous un nouveau nom
172     def enregistrer_sous(self):
173 
174         # Choix d'un fichier dans le répertoire courant
175         filename = filedialog.asksaveasfilename(
```

```
176             title="Enregistrer sous",
177             initialdir=self.rep,
178             initialfile=self.filename,
179             filetypes=(("Text File", "*.txt"), ("All Files", "*.*"))
180         )
181         if len(filename) > 0:
182             # On conserve le nom du fichier utilisé
183             name, extension = os.path.splitext(filename)
184             if extension == '':
185                 extension = 'txt'
186             self.filename = name + '.' + extension
187             self.enregistrer()
188         return
189 
190     # commande fermer le fichier jeu
191     def fermer(self):
192 
193         self.filename = None
194         self.data = []
195         return None
196 
197     # commande de création d'un personnage
198     # il faut un jeu actif
199     def creer(self):
200 
201         # On ajoute un personnage à la liste de données du jeu (data)
202         # la position courante est mémorisée (fin de la liste)
203         # on renvoie le personnage à la fenêtre
204         pod = personnage.creer()
205         self.data.append(pod)
206         self.current = len(self.data)-1
207         return pod
208 
209     # selection d'un personnage dans la liste
210     def selectionner(self, index):
```

```
211
212         # ne peut être que de 0 à nombre-1 de la liste
213         if index < len(self.data):
214             self.current = index
215             return self.data[index]
216         else:
217             return None
218
219     # Changement de partie
220     # On doit modifier les compétences de tous les personnages sans toucher aux caractéristiques
221     # Renvoie le personnage courant
222     def partie(self):
223         # Nécessite de créeer la feuille d'archétype.
224         # Fait appel à la feuille d'archétype pour archiver les compétences du personnage
225         return self.data[self.current]
226
227     # demande de validation et enregistrement des données du personnage
228     # On vérifie la validité des données
229     # Si les données sont correctes:
230     # - le personnage est stocké dans la liste
231     # - l'index, le nom et un message de succès sont renvoyés à l'interface
232     # Sinon le message d'erreur fourni par la vérification du personnage est renvoyé
233     def valider(self, pod):
234
235         message = personnage.verifier(pod)
236         if message is not None:
237             return {"index": None, "nom": None, "message": message}
238
239         # On recalcule tous les champs
240         pod = personnage.recalculer(pod)
241
242         self.data[self.current] = pod
243         Nom = pod["Fiche"]["Nom"]
244         return {"index":self.current, "nom":Nom, "message":''}
245
```

```
246     # Lancer des dés
247     # On commence par calculer le seuil de réussite à partir des points en caractéristiques et compétences
248     # On détermine ensuite les seuils de jugement de réussite ou échec (significatif, particulier, total)
249     # On lance les dés et on compare les résultats aux seuils
250     # On renvoie à l'interface les résultats
251     def lancer(self, caracteristique, competence):
252
253         # Seuil obtenu à partir du tableau de résolution
254         seuil = self.calcul(caracteristique, competence)
255
256         # Calcul des seuils de jugement
257         # - c'est une "réussite particulière" si le tirage est inférieur à 20% du seuil
258         # - c'est une "réussite significative" si le tirage est inférieur à 50% du seuil
259         # - c'est une "réussite" si le tirage est inférieur ou égal au seuil
260         # - on ne peut pas échouer si le seuil est supérieur à 100
261         # - c'est un "échec total" si le tirage est supérieur à 90% de 100-seuil
262         # - c'est un "échec particulier" si le tirage est supérieur à 80% de 100-seuil
263         # - c'est un "échec" si le tirage est supérieur au seuil
264         reussi_particulier = 0.2 * seuil
265         reussi_significatif = 0.5 * seuil
266         if seuil > 100:
267             echec_total = 180
268             echec_particulier = 180
269         else:
270             echec_total = seuil + 0.9 * (100 - seuil)
271             echec_particulier = seuil + 0.8 * (100 - seuil)
272
273         # lancer des dés
274         des = self.des()
275
276         # jugement
277         if des <= reussi_particulier:
278             special = "réussite particulière"
279         elif des <= reussi_significatif:
280             special = "réussite significative"
```

```
281         elif des <= seuil:
282             special = "réussite"
283         elif seuil < 100:
284             if des > echec_total:
285                 special = "échec total"
286             elif des > echec_particulier:
287                 special = "échec particulier"
288             else:
289                 special = "échec"
290         else:
291             special = "réussite"
292 
293         # renvoi du résultat du tirage à l'interface graphique pour affichage
294         # on retourne un dictionnaire : seuil, tirage et special
295         tirage = {"seuil": seuil, "tirage": des, "special": special}
296         return tirage
297 
298     # Fonction de calcul du Seuil de points de 0 à 170
299     # Ce seuil sera utilisé pour déterminer le succès ou non de l'action en cours pour le personnage
300     def calcul(self,caracteristique,competence):
301 
302         # On utilise la formule plutôt que le tableau du livre
303         return int(caracteristique * (1 + .5 * (competence + 8)))
304 
305     # Lancer de dés
306     # On lance 2 dés à 10 faces donnant un résultat entre 1 et 100
307     def des(self):
308 
309         # On lance bien 2 dés
310         unit = random.randrange(0, 9)
311         dix = random.randrange(0, 9)
312         score = 10 * dix + unit
313 
314         # Le résultat 0-0 est interprèté comme 100
315         if score == 0:
```

```
316             score = 100
317         return score
```