Ren'Py est un éditeur d'histoires interactives, et plus précisément de visual novel, c'est-à-dire d'histoires interactives illustrées.

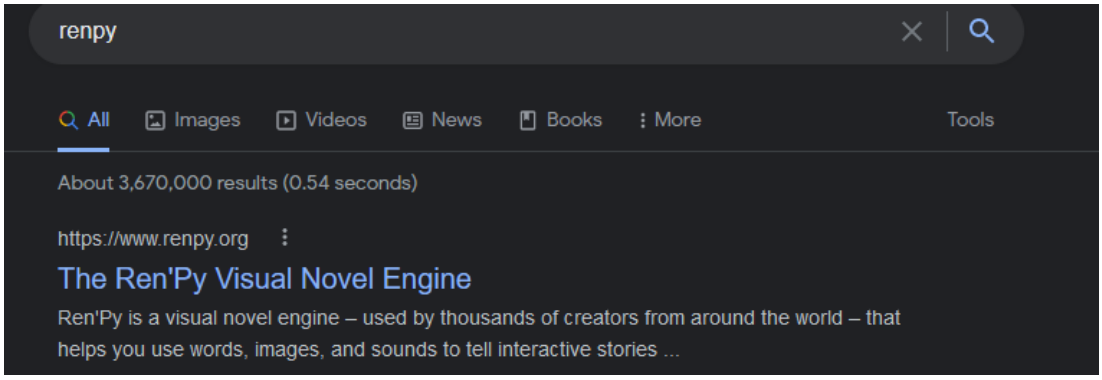
Table des matières

Présentation

Ren'Py est un logiciel libre qui permet de créer des visual novels ou romans vidéoludiques et des sound novels, des types de jeu vidéo populaires au Japon. Le langage utilisé est un dérivé simplifié du python, qui peut d’ailleurs être utilisé pour effectuer des actions plus complexes.

Installation

Se rendre sur renpy.org ou chercher sur le web "Renpy" et cliquer sur le lien suivant :



Cliquer sur « Download Ren’Py » :

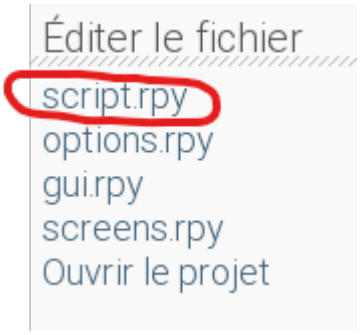
Enregistrer le fichier, le lancer en cliquant sur la flèche bleue en haut à droite de Firefox puis en cliquant sur le nom, et suivre le processus d’installation classique soit cliquez sur Oui puis cochez la case : I accept the agreement puis cliquez sur next — next ---finish

Créer un nouveau projet

Lancer le logiciel, cliquez sur « créer un nouveau projet »

puis choisir un nom pour votre histoire et un dossier de votre ordinateur ou enregistrer les fichiers du jeu (Le dossier « Documents » par exemple).

Pour commencer à éditer le jeu il va falloir ouvrir le script



Et choisir un éditeur de code proposé.

Pour tester le jeu, cliquez sur « Lancer le projet ».

Ecrire des dialogues

Un dialogue se note entre guillemets, précédé du nom du personnage qui parle lui aussi entre guillemets.

Ainsi le code :

```
label start:

    "Sandrine" "Bienvenue dans Ren'Py et félicitation pour ce premier dialogue !"
```

Donnera :

Notons que la commande « `label start:` » démarre le jeu.

Il est possible d'écrire du texte entre deux apostrophes, ce qui donnera un texte affiché sans nom, comme si un narrateur parlait

```
# Le jeu commence ici
label start:
    'Ceci est un tutoriel illustré des bases de Renpy'
    "Sandrine" "Bienvenue dans Ren'Py et félicitation pour ce premier dialogue !"

    return
```

Faire précéder les caractères spéciaux (apostrophe, guillemets) d'un \ pour qu'ils soient interprétés comme du texte.

Définir des personnages

Seulement écrire à la main le nom de tous les personnage est long et multiplie le risque d'erreur, il est plus simple de définir des personnages en début de script, ce qui permet également de les personnaliser (leur donner une couleur de nom particulière par exemple).

Pour définir les personnages de Sandrine et Error, nous allons écrire :

```
# Déclarez les personnages utilisés dans le jeu.
define san = Character("Sandrine", color="#FF0000")
define err = Character ("Err0r", color="#FF1493")


# Le jeu commence ici
label start:
    'Ceci est un tutoriel illustré des bases de Renpy'
    "Sandrine" "Bienvenue dans Ren'Py et félicitation pour ce premier dialogue !"
    err "Et pour la définition des personnages !"
```

Ce qui donne :

On voit que le raccourcis de nom « err » défini au début du jeu affiche bien le nom « Err0r » dans la couleur souhaitée.

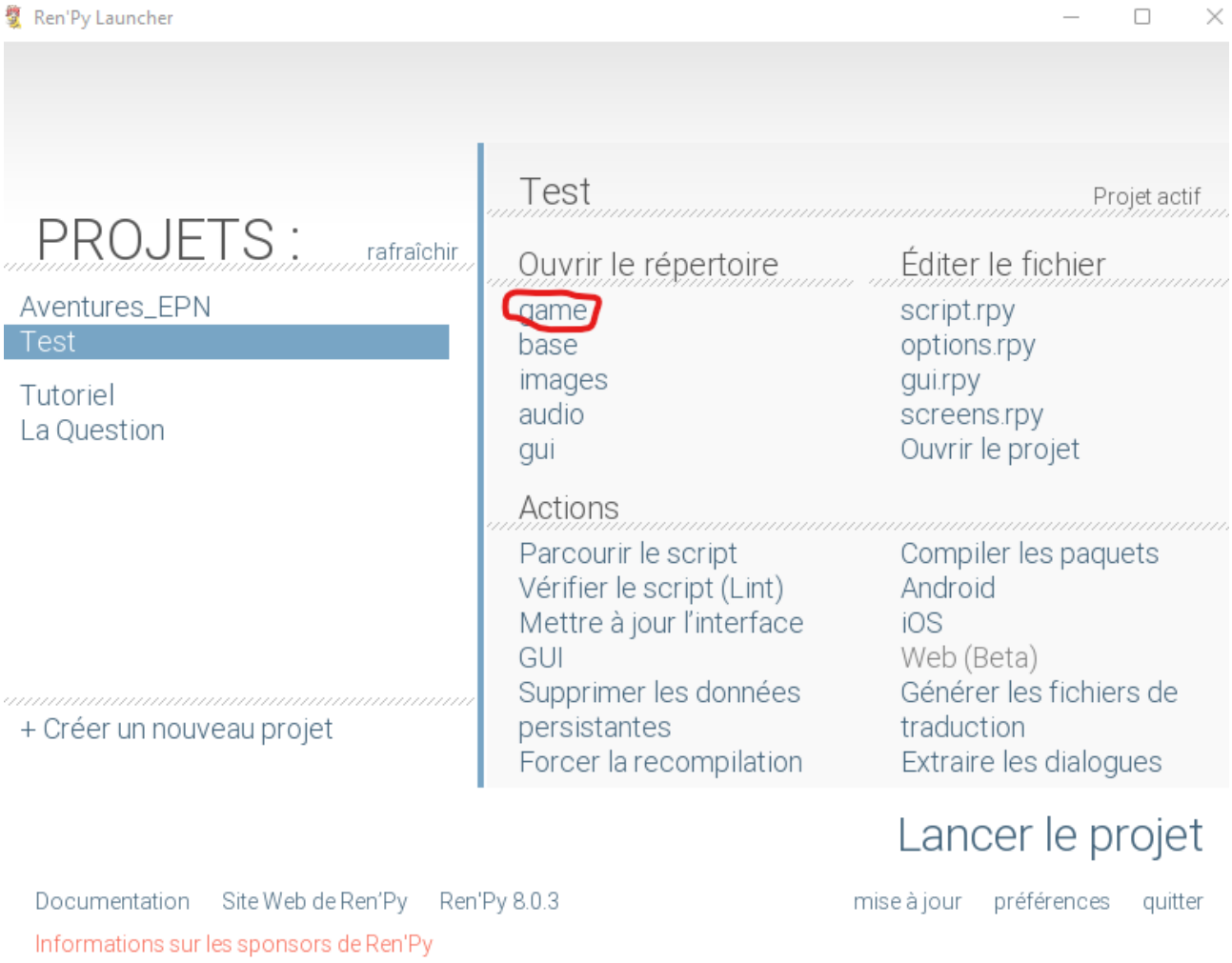
Insérer des images et de la musique

Un visual novel sans images reste un peu triste, nous allons donc voir ici comment ajouter une musique d'ambiance, une image d'arrière-plan et des personnages à notre histoire.

Musique

Pour ajouter une musique d'ambiance qui se jouera tout au long de l'histoire, il faut tout d'abord ouvrir le répertoire du projet (il s'agit de la ligne « game » dans l'accueil de Renpy).

Notons qu'il faut au préalable avoir le fichier son désiré. Il est possible de trouver des musiques libres de droit ici : https://www.auboutdufil.com/



Il faut ensuite ajouter les musiques souhaitées à la racine de ce répertoire.

Retournons au code, avant nos dialogues nous allons ajouter une ligne avec pour instruction de jouer la musique précédemment ajoutée :

```
# Le jeu commence ici
label start:

    play music "LaserPointer.mp3"

    'Ceci est un tutoriel illustré des bases de Renpy'
```

Notons qu'il est très important de respecter l'indentation du code dans Renpy, l'indentation se fait avec la touche tabulation du clavier.

Vous pouvez lancer votre jeu, la musique devrait désormais se lancer au démarrage.

Personnages et positionnement

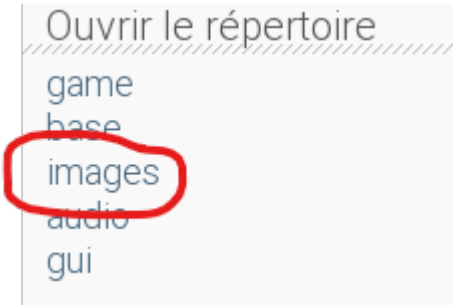
Il est temps de mettre des visages sur nos deux personnages !

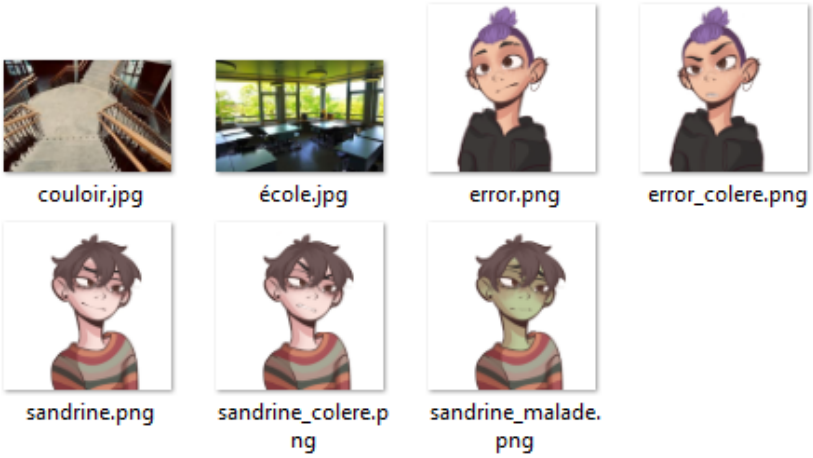
Si dessiner n'est pas votre point fort, pas de panique, il existe des sites de création d'avatar qui font tout à fait l'affaire :

https://avatarmaker.com
https://picrew.me/image_maker/626197

Pensez à les détourer dans un logiciel comme Gimp afin qu'ils n'aient pas d'arrière-plan. Selon l'histoire que vous voulez créer, il est conseillé de créer plusieurs images selon les émotions. Par exemple : error_joie ; error_triste ; error_colere...etc.

Une fois les images créées, ajoutez les au dossier image de votre projet :





Notons que c'est également là que nous déposerons nos images d'arrière-plan.

Pour ajouter notre image au jeu nous allons retourner dans le script et lui demander d'afficher nos personnages :

```
    show sandrine
    with dissolve
    "Sandrine" "Bienvenue dans Ren'Py et félicitation pour ce premier dialogue !"
```

Remplacez « sandrine » par le nom de votre image. Ce qui nous donne :

Dans le code, la commande « `with dissolve` » ajouter un fondu lors de l'apparition de l'image, ce qui est simplement plus joli.
Remarquez que par défaut le personnage apparaît centré, ce qui peut être gênant si on veut afficher deux personnages qui dialoguent.
Pour changer cela nous allons définir des positions pour chaque personnage au début du jeu, sous la définition des personnages :

```
transform positionsan:
    xalign 0.0
    yalign 1.0

transform positionerr:
    xalign 1.0
    yalign 1.0
```

Le « `xalign` » possède une valeur entre 0.0 et 1.0 avec le 0 désignant la gauche et le 1 la droite. Si on tape 0,5 nous aurons donc un personnage centré
Le « `yalign` » possède une valeur entre 0.0 et 1.0 avec le 0 désignant le haut et le 1 le bas. Si on tape 0,5 nous aurons donc un personnage centré

Une fois les positions définies elles s'utilisent dans le code comme suit :

```
show error at positionerr
with dissolve
err "Et pour la définition des personnages !"
```

Ce qui nous donne bien deux personnages se faisant face dans le dialogue :
Pour cacher un personnage, utiliser la commande « `hide nomduperso` » de la même manière. Tant que le personnage n'aura pas été caché, il continuera d'apparaître à l'écran.

Arrière plan

Une image d'arrière-plan rendra notre jeu plus vivant.

Vous trouverez des images libres de droit sur : https://unsplash.com/

Les dimensions de la scène sont de 1920x1080 pixels, il est donc conseillé de redimensionner vos arrières plan dans un logiciel comme Gimp afin qu'ils correspondent à ces dimensions. Appliquez également un filtre « aquarelle » pour leur donner un aspect dessiné qui se fondra mieux avec vos avatars.

Une fois votre image aux bonnes dimensions, déposez-le également dans le dossier « images » de votre jeu, au même endroit où vous avez mis les personnages.

Pour l'insérer dans le jeu, retournez au script et notez la commande suivante :

```
# Le jeu commence ici
label start:

    play music "LaserPointer.mp3"
    scene école

    'Ceci est un tutoriel illustré des bases de Renpy'
```

Remplacez évidemment le mot « école » par le nom de votre image.

Et voici le résultat :

Faire des choix et changer de scène

Un des principal intérêt des histoires interactives, c'est la possibilité de laisser le joueur faire des choix et influencer le cours de votre narration. C'est ce que nous allons voir dans cette partie.

La commande « `menu:` » permet d'indiquer un choix à venir, indiquez ensuite les choix entre guillemets, comme pour un dialogue, et terminez la ligne par le symbole « : ».

```
menu:
    san "Bon, et si on passait à la suite ? Que veux tu apprendre maintenant ?"
    "Les choix":
        err "Je voudrait apprendre à faire des choix et changer de scène"
    "Les variables":
        err "J'ai bien tout compris, on passe aux variables ?"
```

Voici ce que le code ci-dessus donne en jeu :

Dans le code, vous pouvez ajoutez un dialogue sous les choix, ce qui donne :

Les label correspondent à des séquences de jeu, ils servent à découper le jeu en passages. En effet, Renpy lit le script dans l'ordre, créer des labels différents permet de sauter certains passages selon les choix des joueurs, d'y revenir plus tard...etc.

Après notre choix nous allons donc créer deux labels, un correspondant au premier choix, et l'autre au second :

```
label choix:
    san "Pour changer de scène il faut créer un nouveau label et utiliser la commande jump"

label variables:
```

Et pour passer automatiquement à une scène différente du jeu, il faut utiliser la commande « `jump nomdulabel` » comme suit :

```
menu:
    san "Bon, et si on passait à la suite ? Que veux tu apprendres maintenant ?"
    "Les choix":
        err "Je voudrait apprendre à faire des choix et changer de scène !"
        jump choix
    "Les variables":
        err "J'ai bien tout compris, on passe aux variables ?"
        jump variables
```

Ainsi, lorsqu'on fera un choix, on passera ensuite automatiquement au passage de notre histoire qui lui correspond sans forcément suivre l'ordre dans lequel on a écrit nos scènes.

A noter que on peut également donner un nom aux menu de choix (`menu nom:`), et utiliser la commande jump avec les noms des menus pour accéder directement à un choix précédent.

Terminer le jeu

La commande `return` met fin au jeu et le joueur retourne à l'écran titre.

```
return
```

Les commentaires

Pour insérer des commentaires dans le code (c'est-à-dire des lignes qui ne seront pas prises en compte dans le jeu) placer un # en début de ligne comme dans l'exemple ci-dessous

```
# Le jeu commence ici
label start:
```

Les variables

Les variables sont un moyen de mettre en mémoire des informations pour s'en resservir plus tard. Elles peuvent être booléennes, textuelles ou comporter des nombres entiers.

Variables textuelles

Les variables textuelles sont utilisées pour stocker du texte, par exemple pour laisser le joueur choisir son nom !

Voici le code à recopier :

```
# Laisser le joueur écrire son nom :
    $ nomduhero = renpy.input("Entrez votre nom :")
# ôter les potentiels espaces avant et après le texte :
    $ nomduhero = nomduhero.strip()
# Donner un nom par défaut si le champ est laissé vide par le joueur :
    if not nomduhero:
        $ nomduhero = "Michel"
```

Pour utiliser ensuite la variable, il faut la noter entre crochets comme suit :

```
show hero at positionerr
with dissolve
hero "Je m'appelle [nomduhero], enchanté !"
```

Ce qui nous donne :

Note : Pensez bien à définir ce nouveau personnage en haut de votre script !

```
define hero = Character("[nomduhero]", color="#09cf2d")
```

Variable booléenne

Une variable booléenne est une information pouvant avoir deux états : vrai (« True ») et faux (« False »). Elle se note : `$ mavariable = True/False`. Et son état peut être changé selon les choix des joueurs avec la même commande.

```
label booleen:
    hide error_joie
    with dissolve

    $ compris = False

    san "Une variable booléenne est une information pouvant a

    menu varbool:
        san "As-tu bien compris les variables booléennes  ?"
        "Oui tout à fait!":
            $ compris = True
        "Euh, je ne suis pas tout à fait sûr·e..."
            $ compris = False
```

Dans notre exemple, on peut se remémorer si le joueur pense avoir compris l'explication donnée par Sandrine ou pas.

Pour vérifier une variable booléenne on utilise une autre commande python, la commande `if`. Elle se note

```
if mavariable == True :
     dialogue
else :
     dialogue
```

Comme dans l'exemple qui suit :

```
label exercice:
    san "Pour en être sûr, voici un petit exercice : Combien d'état peut avoir une variable booléenne ?"
    menu exo:
        "Deux !":
            if compris == True:
                san "Tu as en effet compris, on peut passer à la suite !"
                jump var
            else:
                san "Tu as compris finalement, bravo !"
                jump var
        "Trois !":
            if compris == True:
                san "Je croyais que tu avais compris ? On recommence !"
                jump booleen
            else:
                san "En effet, tu as peut-être besoin de revoir les explications encore une fois."
                jump booleen
```

Ainsi, selon la réponse du joueur nous avons 4 options de dialogue possible, selon si le joueur pense avoir compris les variable et si il donne la bonne réponse à l'exercice.

Variable entières

Les variables entières sont des variables pouvant stocker des nombres entiers. On peut ensuite en ajouter, retrancher et les comparer pour créer des systèmes de points, points de vie, points de fatigue...etc.

Une variable entière se définit comme suit :

`$ mavariable = nombre entier`

```
san "Nous allons maintenant créer un mini jeu avec un système de point de vie"

$ pv = 20
```

Pour ajouter ou retrancher des nombres, voici la commande :

```
label game:

    'Que veux-tu manger ?'

    menu miam:
        "la pomme":
            show hero at positionerr
            with dissolve
            hero "Cette pomme est délicieuse !"
            $ pv += 10
            hide hero
            with dissolve

        "le cocktail":
            show hero_triste at positionerr
            with dissolve
            hero "Je ne sais pas si c'était une bonne idée..."
            $ pv -= 10
            hide hero_triste
            with dissolve
```

Dans notre exemple, le joueur a le choix entre deux aliments, l'un va ajouter des points de vie avec la commande `$ mavariable += nombre` et l'autre va en retirer avec la commande `$ mavariable -= nombre`.

Pour arriver à un résultat différents selon les choix du joueur, il faut maintenant comparer le nombre stocké dans la variable.

```
if pv > 29:
    show hero at positionerr
    with dissolve
    hero "Je suis en pleine forme !"
    hide hero
    with dissolve
    jump var2

else:
    show hero_triste at positionerr
    with dissolve
    hero "Je ne me sens pas très bien..."
    hide hero_triste
    with dissolve
    jump var2
```

Notons l'utilisation du comparateur « plus grand que », mais nous pouvons également utiliser le « plus petit que » (<) ou encore « plus grand ou égal à » (>=) ou « plus petit ou égal à » (>=).
Ainsi, si on choisi de boire le cocktail :

Mais si on prends la pomme :

Ressources

Tutoriel vidéo

Pour écrire ce tutoriel j’ai notamment suivi les vidéos de Game Dev alliance sur Renpy :

https://www.youtube.com/watch?v=XprVZAtPqDI

https://www.youtube.com/watch?v=nqWEN4Z2420

Ressources média

Musique libre de droit :

https://www.auboutdufil.com/

Arrières-plan libres de droit :

https://unsplash.com/

Générateurs d’avatars :

https://avatarmaker.com
https://picrew.me/image_maker/626197

Script de l’exemple

A copier/coller dans votre script et à modifier si vous le souhaitez. Attention cependant à l’indentation et aux noms des images.

```
# Vous pouvez placer le script de votre jeu dans ce fichier.
# Déclarez les personnages utilisés dans le jeu.
define san = Character("Sandrine", color="#FF0000")
define err = Character("ErrOr", color="#FF1493")
define hero = Character("[nomduhero]", color="#09cf2d")

# Déclarez ici les position des images de vos différents personnages
transform positionsan:
    xalign 0.0
    yalign 1.0

transform positionerr:
    xalign 1.0
    yalign 1.0



# Le jeu commence ici
label start:

    play music "LaserPointer.mp3"
    scene école

    'Ceci est un tutoriel illustré des bases de Renpy'

    show sandrine at positionsan
    with dissolve
    "Sandrine" "Bienvenue dans Ren'Py et félicitation pour ce premier dialogue !"
```

```
    show error at positionerr
    with dissolve
    err "Et pour la définition des personnages !"


    menu debut:
        san "Bon, et si on passait à la suite ? Que veux tu
apprendres maintenant ?"
        "Les choix":
            err "Je voudrait apprendre à faire des choix et
changer de scène !"
            jump choix
        "Les variables":
            err "J'ai bien tout compris, on passe aux variables ?"
            jump variables


label choix:
    hide error
    with dissolve
    san "Pour changer de scène il faut créer un nouveau label et
utiliser la commande jump"
    jump debut


label variables:
    hide error
    with dissolve
    san "Les variables sont un moyen de mettre en mémoire des
informations pour s'en resservir plus tard. Elles peuvent être
booléennes, textuelles ou comporter des nombres entiers."
    menu var:
        san "De quel type de variable veux-tu apprendres les
subtilités ?"
        "Les variables textuelles":
            show error_joie at positionerr
            with dissolve
            err "Je veux pouvoir laisser le joueur choisir son nom
!"
```

```
            jump texte


label var2:
    menu:
        san "De quel type de variable veux-tu apprendres les
subtilités ?"
        "Les variables textuelles":
            show error_joie at positionerr
            with dissolve
            err "Je veux pouvoir laisser le joueur choisir son nom
!"
            jump texte
        "Les variables booléennes":
            show error_joie at positionerr
            with dissolve
            err "Comment se souvenir d'un choix précédent ?"
            jump booleen
        "les variables entières":
            show error_joie at positionerr
            with dissolve
            err "Comment faire un système de point de vies ?"
            jump entier
        "Je veux sortir du tutoriel et commencer un jeu !":
            jump fin


label texte:
    hide error_joie
    with dissolve

    san "Alors, quel est ton nom ?"

# Laisser le joueur écrire son nom :
    $ nomduhero = renpy.input("Entrez votre nom :")
# ôter les potentiels espaces avant et après le texte :
```

```
    $ nomduhero = nomduhero.strip()
# Donner un nom par défaut si le champ est laissé vide par le
joueur :
    if not nomduhero:
        $ nomduhero = "Michel"


    show hero at positionerr
    with dissolve
    hero "Je m'appelle [nomduhero], enchanté !"
    hide hero
    with dissolve


    jump var2


label booleen:
    hide error_joie
    with dissolve


    $ compris = False


    san "Une variable booléenne est une information pouvant avoir
deux états : vrai (« True ») et faux (« False »). Elle se note : $
mavariable = True/False. Et son état peut être changé selon les
choix des joueurs avec la même commande."


    menu varbool:
        san "As-tu bien compris les variables booléennes  ?"
        "Oui tout à fait!":
            $ compris = True
        "Euh, je ne suis pas tout à fait sûr·e...":
            $ compris = False


label exercice:
    san "Pour en être sûr, voici un petit exercice : Combien
d'état peut avoir une variable booléenne ?"
```

```
    menu exo:
        "Deux !":
            if compris == True:
                san "Tu as en effet compris, on peut passer à la
suite !"
                jump var2
            else:
                san "Tu as compris finalement, bravo !"
                jump var2
        "Trois !":
            if compris == True:
                san "Je croyais que tu avais compris ? On
recommence !"
                jump booleen
            else:
                san "En effet, tu as peut-être besoin de revoir
les explications encore une fois."
                jump booleen


label entier:
    hide error_joie
    with dissolve


    san "Nous allons maintenant créer un mini jeu avec un système
de point de vie"


    $ pv = 20
    hide sandrine
    with dissolve
    jump game


label game:


    'Que veux-tu manger ?'
```

```
menu miam:
    "la pomme":
        show hero at positionerr
        with dissolve
        hero "Cette pomme est délicieuse !"
        $ pv += 10
        hide hero
        with dissolve

    "le cocktail":
        show hero_triste at positionerr
        with dissolve
        hero "Je ne sais pas si c'était une bonne idée..."
        $ pv -= 10
        hide hero_triste
        with dissolve

if pv > 29:
    show hero at positionerr
    with dissolve
    hero "Je suis en pleine forme !"
    hide hero
    with dissolve
    jump var2

else:
    show hero_triste at positionerr
    with dissolve
    hero "Je ne me sens pas très bien..."
    hide hero_triste
    with dissolve
    jump var2
```

```
label fin:
    san "Merci d'avoir étudié avec nous les bases de Ren'Py !"
    show error_joie
    with dissolve
    err "Je vais enfin pouvoir créer mon propre jeu !"

    scene black
    with dissolve
    return
```