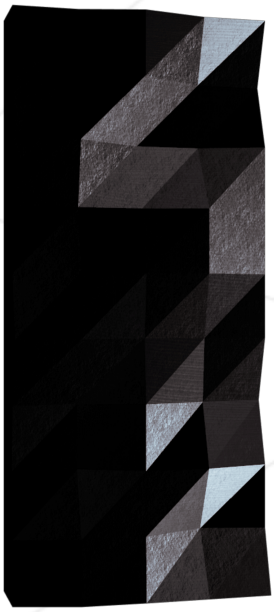

Initiation à la programmation - 42

Ruby - Création d'un RPG texte

Staff 42 pedago@42.fr

Résumé: Ce document est le sujet du mini-projet RPG texte du programme d'initiation à Ruby.

Table des matières

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices.
- Vous serez corrigés par les autres participants du programme d'initiation. C'est le "peer-correcting" de la pédagogie 42 !
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement specifiés par les énoncés des exercices.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Votre manuel de référence s'appelle `Google / man / Internet / ...`.
- Lisez attentivement les exemples. Les exercices pourraient bien requérir des choses qui y sont précisées, et non dans le sujet...

Chapitre II

Procédure de rendu : Git

- Votre rendu est collecté via un serveur distant. Cela signifie que vous devrez envoyer sur ce serveur votre repertoire de rendu avec vos exercices suivant une arborescence précise.
- Le logiciel utilisé pour ce faire s'appelle `git`.
- Git permet de faire beaucoup de choses, vous trouverez ci-dessous les principales commandes :
- `Git clone` vous permet de `cloner` un répertoire présent sur un serveur directement dans votre répertoire courant. Cela signifie que tous les fichiers présents que vous y aviez mis depuis un autre poste se retrouveront sur votre ordinateur et pourront être modifiés et réenvoyés avec les commandes git suivantes. Cette commande est aussi importante quand vous souhaitez vérifier ce que vous avez "pushé" sur le serveur, n'hésitez pas à en abuser.

```
?> git clone vogsphere@vogsphere.42.fr:piscine/truc/machin/creme
Cloning into 'piscine'[...]
?>
```

- `Git add` vous permet d'ajouter à une liste de fichiers surveillés un ou plusieurs fichiers/repertoires. Ils seront ajoutés en l'état ce qui veut dire que si vous les modifiez par la suite, il faudra les ajouter à nouveau pour mettre à jour la liste.

```
?> git add file directory
?>
```

- `Git commit -m` vous permet de "fixer" votre liste de fichiers surveillés afin de préparer leur envoi sur le serveur. Le message est obligatoire et permet d'avoir un historique des créations et modifications que vous avez effectué au fur et à mesure de votre travail.

```
?> git commit -m "Ajout des exercices X Y Z"
[master (root-commit) 4e8b2aa] Ajout des exercices X Y Z
3 file changed, 500 insertions(+)
create mode 100755 X Y Z
?>
```

- `Git push origin master`, la commande ultime, celle qui vous permet d'envoyer sur le serveur vos exercices qui y seront collectés par la Moulinette. Vérifiez bien son retour, le moindre message d'erreur signifie que vous n'avez rien pu envoyer et que par conséquent votre répertoire sur le serveur est vide (ce qui serait dommage pour votre note). Si vous avez le moindre doute, contactez un membre du staff 42.

```
?> git push origin master
[...]
X files written.
?>
```

L'adresse git de votre dépôt se trouve sur l'intranet et dans le fichier config du répertoire .git de votre dossier de rendu, c'est avec elle que vous pourrez "git clone".

Chapitre III

Préambule

Voici un extrait de Bill & John

```
Up to mighty London
Came an Irishman one day.
As the streets are paved with gold
Sure, everyone was gay,
Singing songs of Piccadilly,
Strand and Leicester Square,
Till Paddy got excited,
Then he shouted to them there:
It's a long way to Tipperary,
It's a long way to go.
It's a long way to Tipperary
To the sweetest girl I know!
Goodbye, Piccadilly,
Farewell, Leicester Square!
It's a long long way to Tipperary,
But my heart's right there.
(repeat)
Paddy wrote a letter
To his Irish Molly-O,
Saying, "Should you not receive it,
Write and let me know!"
"If I make mistakes in spelling,
Molly, dear," said he,
"Remember, it's the pen that's bad,
Don't lay the blame on me!
It's a long way to Tipperary,
It's a long way to go.
It's a long way to Tipperary
To the sweetest girl I know!
Goodbye, Piccadilly,
Farewell, Leicester Square!
It's a long long way to Tipperary,
But my heart's right there.
```

```
Molly wrote a neat reply
To Irish Paddy-O,
Saying "Mike Maloney
Wants to marry me, and so
Leave the Strand and Piccadilly
Or you'll be to blame,
For love has fairly drove me silly:
Hoping you're the same!"
It's a long way to Tipperary,
It's a long way to go.
It's a long way to Tipperary
To the sweetest girl I know!
Goodbye, Piccadilly,
Farewell, Leicester Square!
It's a long long way to Tipperary,
But my heart's right there.
```

Ce projet est beaucoup plus facile si vous le réalisez en regardant `Bill & John`.

Chapitre IV

Sujet - Partie obligatoire

IV.1 Introduction

Vous avez appris pendant plusieurs jours les bases de Ruby en faisant des petits exercices sans queue ni tête, il est maintenant temps de faire aboutir vos connaissances en réalisant un vrai programme !

Aujourd'hui, vous allez coder un petit RPG... textuel. Adieu les graphismes, vous allez mettre les mains dans le cambouis et développer uniquement les mécaniques de jeu. Pour ceux qui ne savent pas ce qu'est un RPG textuel, représentez-le vous comme les fameux "Livres dont vous êtes le héros" que vous avez pu rencontrer plus jeunes. L'idée est de décrire une scène et ce qui s'y passe, et de proposer des choix d'action au joueur. L'histoire évolue ainsi selon le choix des joueurs... jusqu'à une issue favorable - ou la mort !

Le but de ce mini-projet est de vous faire intégrer toutes vos connaissances au sein d'un vrai programme. Les variables, le traitement des entrées utilisateur, les tableaux, les méthodes... tous ces outils permettent de multiples applications concrètes ! Coder un programme complet vous permettra d'appréhender les usages usuels de ces outils.

IV.2 Comment ça marche ?

- A chaque "scène" existante de votre RPG correspondra une méthode. Ainsi, passer d'une scène à l'autre se traduira par le passage d'une méthode à une autre.
- Dans chaque méthode, vous devez décrire où le joueur vient d'arriver, s'il y a des objets à sa disposition, des ennemis qui l'attaquent, etc... et lui proposer des choix d'action (c'est-à-dire, d'autres méthodes qui peuvent être appelées à la suite.)
- Attention dans le traitement des entrées utilisateur : si vous proposez à votre joueur de choisir entre trois mots-action, et qu'il ne saisit pas un mot valide, vous devez lui indiquer son erreur et lui re-proposer le choix.

Etant donné que ce check va être présent dans toutes les méthodes, vous pouvez songer à factoriser votre code en créant, par exemple, une fonction qui prend en paramètre des inputs valides et ne renvoit un résultat que lorsque l'utilisateur a entré une valeur parmi ces inputs valides...

- Votre jeu doit pouvoir se terminer. Ainsi, il faudra implémenter trois types de fin de jeu :
  - Votre joueur va commencer avec un nombre fixe de points de vie. Au grès des rencontres dans les différentes scènes, il pourra en perdre en rencontrant des ennemis, ou en re-gagner (mais jamais plus que le niveau initial) s'il trouve des médicaments. S'il tombe à 0 points de vie, il meurt. (Attention, donc, à quand faire le check des points de vie !)
  - La partie a une durée limitée, dont le nombre de secondes sera passé en paramètre au programme. (Comment implémenter cela ? Songez que, lorsque vous lancez le programme, vous pouvez récupérer l'heure exacte et la stocker dans une variable... Bien sur, il faudra checker règulièrement si la partie n'est pas terminée !)
  - Et bien sur, un joueur doit être capable de gagner s'il arrive au bout de sa quête !

Chapitre V

Sujet - Partie bonus

Les bonus ne seront évalués que si votre partie obligatoire tient la route. C'est la règle : on n'ajoute des features que lorsque la base est solide ! Tuner une voiture sans permis, ça ne sert à rien. ;)

Voici quelques idées de bonus intéressants à réaliser, voire même utiles. Vous pouvez évidemment ajouter des bonus de votre invention, qui seront évalués à la discrétion de vos correcteurs.

- Rajouter de la couleur à vos outputs, voire représenter des éléments de votre jeu en ASCII art pour rendre cela plus intéractif !
- Avoir un système de points, et une victoire possible lorsque le joueur a marqué suffisamment de points.
- Faire un jeu multijoueur, où deux joueurs qui jouent en même temps peuvent se croiser et interragir dans les différentes scènes.

Chapitre VI

Consignes

- Votre programme doit être lancé avec deux arguments :
  - une difficulté, entre easy / medium / hard, qui définit le nombre de points de vie avec lequel le joueur part.
  - la durée maximum de la partie en secondes.
- Si les arguments passés au programme sont invalides, un message d'erreur doit être affiché et le programme quitter proprement.
- Le script doit s'appeler `rpg.rb`.
- Personnalisez le joueur : demander lui d'entrer un nom en début de partie, et utilisez ce nom pour le désigner pendant la partie.
- Les mécaniques du jeu sont à votre discrétion, soyez créatifs ! Mais il faut au minimum qu'il y ait des ressorts du jeu qui permettent au joueur de perdre des points de vie ou d'en gagner, de posséder des objets, d'explorer des scènes...
- N'hésitez pas à communiquer entre vous, pour savoir comment implémenter les différents aspects du jeu !