Chevasson Raphaël
Cousseau Axel
Barus Eliot

COMPTE-RENDU PROJET D'INFORMATIQUE:
JEU DE RÔLE

I / Reflection sur le sujet

Nous nous sommes d'abord concertés afin de voir ce que nous voulions faire. Nous avons décidé de mettre dans le labyrinthe des ennemis, un marchand, quelques objets (comme de l'équipement) ainsi qu'un boss de fin du jeu. Nous avons également fait une ébauche de scénario: Un mercenaire allant dans un donjon afin de soutirer le butin d'un terrible sorcier. Nous sommes partis avec l'idée de séparer l'écran de jeu en 5 parties, l'une avec une description de la salle et des action possibles, une autre avec le statut (vie, or,...), une autre avec la carte de la salle ainsi qu'un plan du donjon et la dernière avec un visuel. Nous avons ensuite élaboré le plan de travail afin de pouvoir plus facilement nous répartir les taches :

1) affichage des phrases descriptives
2) affichage des actions
3) reconnaissance des actions entrées
4) faire les procédures déclenchées par les actions
5) système de combat
6) faire le donjon, déplacements fonctionnels
7) faire les objets et déclencher les bonus associés
8) faire un écran de statut primitif (santé, or…)
9) faire les créatures
10) faire le boss et la fin du jeu: choix entre garder ou détruire le *sceptre de kartoffel*
11) afficher la carte de la pièce
12) diviser proprement l'écran
13) affichage circonstancié du visuel

II / Structure et organisation du programme

Tout d'abord, afin de pouvoir utiliser les variables dans les fonctions, nous avons décidé de toutes les stocker dans trois dictionnaires: **Joueur**, **Donjon** et **Systeme**. Ces dictionnaires fonctionnent comme des listes mais avec des chaînes de caractères à la place des indices. Ainsi, l'inventaire correspondra à **Joueur["Inventaire"]**. Joueur, Donjon et Systeme sont parfois référencées comme des variables globales, lorsqu'il est trop fastidieux de les passer en paramètre de chaque fonction. La fonction en question traite alors ces variables comme si elles lui étaient propres.

Le programme se déroule dans une boucle **tant que** qui relance une partie après chaque victoire ou game over, tant que l'on a pas explicitement demandé de sortir de la boucle en assignant **False** au booléen Systeme["quitter"].

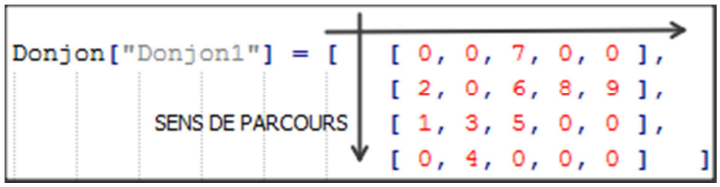
Chaque partie commence par l'initialisation (ou la réinitialisassions) des variables via la procédure **nouvelle_partie(…)**. La structure de celles-ci sera expliquée par la suite. Puis vient l'affichage de l'écran titre, et enfin l'entrée dans le corps du programme : la boucle de jeu.

La procédure **boucle_de_jeu()** représente la succession des tours de jeu : c'est une boucle **tant que** (dont on sort en à la fin de la partie en affectant **True** à **Systeme["fin_partie"]**) qui, à chaque tour de jeu, appelle la procédure d'affichage, puis demande au joueur ce qu'il veut faire, et finalement réagit en conséquence en appelant la procédure **action(entrée du joueur)**.

Représenter informatiquement une chose aussi complexe qu'un jeu de rôle n'a rien d'évident. Voici les choix que nous avons fait:

/ Le joueur est représenté par :

-Sa position dans le donjon Joueur["pos_donjon_X"] et Joueur["pos_donjon_Y"]: Elles correspondent aux coordonnées de la pièce dans le donjon. Nous avons choisi de renverser l'axe Y afin de respecter le sens d'itération de la liste :



-Sa position dans la pièce Joueur["pos_piece_X"] et Joueur["pos_piece_Y"]: Elles permettent principalement d'afficher le joueur dans la carte de la salle. L'axe Y suit la même convention que pour la position dans le donjon.

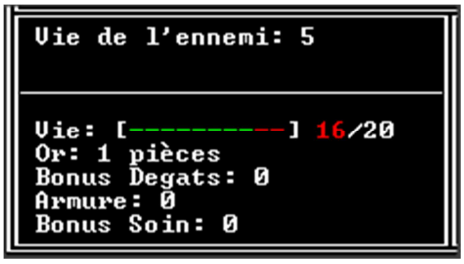
-Son inventaire Joueur[“Inventaire”], une liste de booléens avec un indice par objet que le joueur peut obtenir.

patate	épée	bouclier	anneau	gemme	épée de feu

-Ses caractéristiques Joueur[“Statut”] représentée par une liste d’entiers organisée comme suit :

Vie	Vie maximum	Or	Bonus aux dés d’attaque	Armure (réduit les dés adverse)	Bonus aux dés de soin

-Son statut au tours précédent Joueur[“Ancien_statut”] afin de pouvoir afficher les variations :



-La phrase récapitulative du résultat de l’action du joueur Joueur[“derniere_action”]. Celle-ci est une liste de trois éléments : la phrase en elle-même, le temps pour que l’action se déroule (que nous n’avons finalement pas (encore ?) implémenté) et un booléen qui spécifie si l’action doit être écrite avant la description de la pièce (comme pour “Vous entrez dans une nouvelle pièce”) (True) ou à la suite de la description sans saut de ligne (action classique) (False). Nous avons finalement préféré afficher toutes les actions avant la description.

-Le nom de l’ennemi Joueur[“creature_combatue”] permet de connaitre le nom de l’ennemi ou de vérifier si on est en phase de combat (si oui : “chaine”, si non : False)

-Joueur[“actions_supplementaires”]: Contient les actions possibles qui se réinitialisent à chaque affichage. Une action supplémentaire est une liste structurée de la façon suivante :

phrase à proposer au joueur	procédure à déclancher	raccourcis que le joueur peut entrer à la place de la phrase

-Joueur[“actions_persistantes”]: Fonctionne de la même façon, mais ne se réinitialise pas à chaque affichage. Il faut donc penser à les retirer manuellement. De plus la structure n’est pas exactement la même :

phrase à proposer au joueur	procédure à déclancher	l’action est-elle disponible hors combat ?	l’action est-elle disponible en combat ?

-La phrase Joueur["question"] affichée avant la zone de frappe de l’utilisateur.

/ Le Donjon est représenté par :

-Une liste de liste Donjon["Donjon1"] se présentant comme un tableau en deux dimensions contenant l'indice de la salle pour chaque coordonnée X et Y dans le donjon. Par exemple,

```
Donjon["Donjon1"] = [   [ 0, 0, 7, 0, 0 ],
                        [ 2, 0, 6, 8, 9 ],
SENS DE PARCOURS        [ 1, 3, 5, 0, 0 ],
                        [ 0, 4, 0, 0, 0 ]   ]
```

donne

-La liste Donjon["piece"] dont l'élément d'indice i est une liste décrivant la salle i de la façon suivante :

nom de la salle (nous avons retiré son affichage pour conserver tout le suspens de l'exploration)	description de la salle	liste des actions ajoutées par la salle :

phrase proposée	procédure à déclencher

-La liste Donjon["Pieces"] dont l'élément d'indice i est une liste décrivant le contenu de la salle i (contrairement à Donjon["piece"] qui décrit la pièce en elle-même) de la de la façon suivante :

Liste des objets dans la salle :

Objet	position X dans la salle	position Y dans la salle

↪ un élément

Liste des créatures dans la salle :

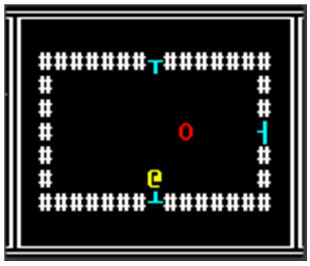
Créature	position X dans la salle	position Y dans la salle

↪ un élément

Par exemple,

```
[ [[Porte_droite, 7, 0], [Porte_haut,
```

donne



On peut de cette façon réutiliser plusieurs fois le même objet ou la même créature à des coordonnées différents. Les nôtres étant initialisés en amont, on peut même les utiliser dans plusieurs salles différents.

Un objet se présente comme suit :

Description de l'objet	Caractère représentatif	liste des actions ajoutées par l'objet :

phrase proposée	procédure à déclencher

Une créature se représente de la même façon, mais avec deux propriétés (représentées par deux éléments) en plus :

Description	Caractère repré-sentatif	liste des actions ajoutées par l'objet :	Nom de la créature	Celle-ci engage-elle le combat si vous essayez une autre action ?

phrase proposée	procédure à déclencher

Le nom de la créature est important : il doit correspondre au nom du fichier du visuel à afficher, et se retrouvera dans la variable creature_combatue qui est utilisée et dans certaines phrases de combat et pour identifier le monstre (lors de sa suppression de la pièce par exemple).

III / Réalisation des étapes

Nous avons commencé par faire un affichage des phrases décrivant la situation du joueur (1). Pour commencer, la fonction **description()** renvoie les phrases décrivant son environnement. Elle concatène simplement la description de chaque objet trouvé dans la salle du joueur (obtenue par la fonction **salle_courante()**), ainsi que celle de la pièce, avec un début et une fin de phrase (“Vous voyez ” et “.”)

La fonction **texte_derniere_action()** renvoie les phrases décrivant le résultat de la dernière action du joueur (2). Elle renvoie directement la phrase de la variable **Joueur["derniere_action"]**, à laquelle on rajoute des phrases (en les concaténant avec les précédentes) via la procédure **ajouter_phrase(“chaine”)**.

La procédure **action(“entree”)** parcourt tous les objets susceptibles de proposer des actions au joueur (la pièce courante ainsi que les objets et créatures qu’elle contient) et vérifie si l’entrée clavier du joueur correspond à la phrase de l’une d’entre elles. Si oui, elle exécute l’action correspondante (3). Pour ce faire, elle passe par une variable de transition **a** (représentant une procédure) afin de ne garder que la dernière occurrence. La procédure contenue dans **a** est ensuite exécutée, sauf si celle-ci est inexistante, auquel cas il faut relancer le joueur et ne pas actualiser l’état du donjon. Plus tard dans le projet, nous l’avons modifié pour qu’elle vérifie qu’aucune créature agressive ne déclenche un combat si le joueur tentait de faire autre chose que de la combattre. Ce test est réalisé en comparant **a** avec la procédure de la première action proposée par cette créature (par convention, le combat si elle est agressive). Le cas échéant, elle remplace **a** et ajoute une phrase explicative.

Chaque action est une procédure avec un impact différent sur le programme (4). A la fin de la procédure, la boucle de jeu reprend la main et relance la procédure d’affichage. Il faut donc déclencher tout ce qui doit être effectué avant celui-ci, et stocker tout les résultats dont on aura besoin ultérieurement ; c’est ce qui nous a imposé l’utilisation de variables globales. Nous avons commencé par réaliser les procédures de déplacement (notamment **deplacer(x, y)**) - nous avions alors trois pièces dans le donjon - puis le catalogue de procédures s’est étoffé au fur et à mesure.

En parallèle, une autre partie du groupe a réalisé le système de combat (5). Celui-ci fonctionne grâce à des lancer de dés. Initialement, le joueur a le choix entre attaquer et se soigner. Si le joueur se soigne, le montant de vie récupérés dépendra d’un lancé de dés et de son équipement (fonction **lance_des_soin()**) ; l’ennemi attaquera ensuite et fera perdre au joueur un montant de vie égal à son lancé de dés. Si le joueur décide d’attaquer, il fera un lancé de dés dépendant de son équipement (**lance_des_joueur()**) en même temps que son ennemi (**lance_des_creature()** affectée par l’armure du joueur). Si les résultats des lancés sont égaux, aucun dégât ne sera infligé. Si le score du joueur est supérieur à celui du monstre, le monstre perdra un montant de vie égal au score du joueur. Il se produira le contraire si le score du joueur est inférieur à celui du monstre.

Il a ensuite fallu découper la grosse procédure de combat en plus petites afin de coller à la structure du programme (sortie de la procédure à chaque tour afin de déclencher l'affichage) et de palier à tout les scénarios (différents adversaires, affichage des actions de combat seules au premier tour…).

Toujours en suivant notre plan de travail, nous réalisé le donjon (6) d'après un plan:

		7 : salle piège		
2 : salle du vendeur		6	8	9 : BOSS
1:VIDE (Entrée)	3	5		
	4			

Les procédures d'affichage et d'actions ayant été écrites pour parcourir les liste de créature, d'objet et de pièces, il nous a suffi de les initialiser (Donjon["Donjon1", "piece", "Pieces"] et les objets portes) pour modifier le donjon.

Après avoir initialisé les pièces vides, nous les avons rempli d'objets (7). D'abord, une *épée* augmentant l'attaque du joueur, ensuite un *bouclier*, diminuant les dégâts reçut par celui-ci. Nous avons également mis un *anneau*, augmentant les soin et la vie, ainsi qu'une *gemme*, enflammant l'épée. En dernier; nous avons ajouté un objet, utilisable une seule fois, infligeant d'énormes dégâts: une *patate*. Les objets sont représentés dans la liste de booléens Inventaire:

patate	épée	bouclier	anneau	gemme	épée de feu

L'effet des équipements (principalement des bonus) sont déclenchés grâce à **obtenir_objet("nom de l'objet")**.

Nous avons réalisé la liste représentant le statut du joueur (vie restante et maximale, or, bonus d'attaque d'armure et de soin) ainsi que sa procédure d'affichage (8). L'or est ramassée après certains combats et pourra être dépensée chez le vendeur.

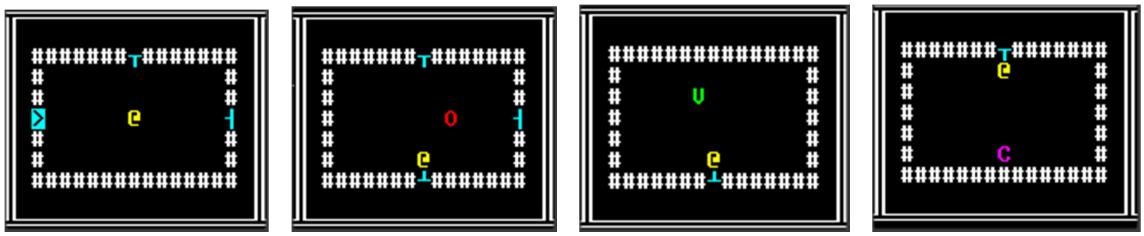
Puis nous avons ajouté des monstres agressifs (9) : un *lapin*, un *gobelin*, un *orc*, une *licorne* et un *squelette*. Nous avons également réalisé le vendeur, une créature amicale permettant au joueur d'acheter l'*épée* et le *bouclier* (**achat_epee()** et **achat_bouclier()**) via **commercer()**. Chaque monstre possède un nombre de points de vie, un intervalle de score de lancer de dés, et un butin prédéfini. Nous avons fait le choix de ne pas rendre ces propriétés aléatoires afin de pouvoir mieux contrôler l'avancée du joueur dans le donjon. Cela aboutit au plan suivant :

		7 :Licorne Gemme		
2 : vendeur (épée/bouclier)		6:Orc	8:Squelette	BOSS
1:VIDE (Entrée)	3:lapin	5:gobelin; Anneau		
	4 : objet(Coffre) (patate)			

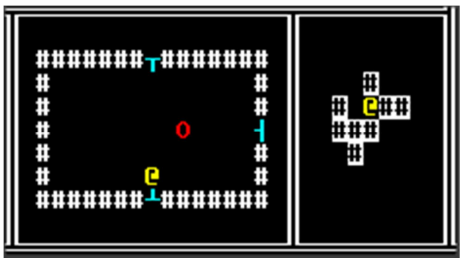
(oui, le piège s'est transformé en licorne, et alors ?)

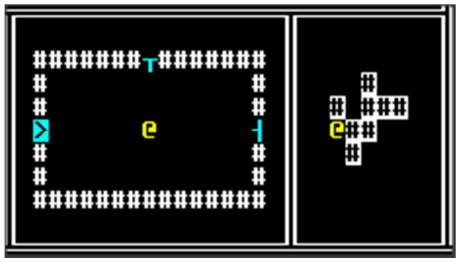
Pour le combat de boss (10), nous affrontons le puissant sorcier *Shlepper* via une procédure dédiée, **shlepper()**. Il possède une santé très élevée (150, contre un maximum de 30 pour le joueur et les autres monstres) et a un meilleur lancé de dés que les autres créatures. Le combat se déroule en 3 phases. La première phase se déroule tant que *Shlepper* a plus de 50 de vie. Lors de cette phase, le Boss lance les dés entre 8 et 12. La seconde phase a lieu quand *Shlepper* à moins de 51 de vie, jusqu'à ce qu'il ne lui en reste plus. Ses dés sont augmentés de deux. Enfin, lorsque sa santé tombe à 0, le nécromancien invoque un guerrier squelette qu'il faut terrasser. Le combat est ensuite gagné. Après avoir triomphé, on a le choix entre ridiculiser le sorcier avant de le tuer (ce qui vous causera des dommages)(**choix2()**) ou de le tuer avec honneur(**choix1()**). On pourra ensuite choisir entre les deux fins: garder le sceptre(**fin1()**) ou le détruire (**fin2()**).

Nous avons ensuite rajouté deux cartes constamment présentes sur l'écran: la carte de la salle et la carte du donjon. La carte de la salle (plan_salle()) affiche le joueur, représenté comme un @, à ses coordonnées dans la salle. Elle affiche également les diverses créatures, les murs, les objets, ainsi que les portes.



La carte du donjon (**plan_donjon()**) affiche, quand à elle, l'agencement global des salles, représentées par des #, dans le donjon ; ainsi que la position du joueur, toujours représentée par un @. Elle est complémentaire à la carte de la salle :





Le plus dur a été de scinder l'écran de jeu principal en 5 parties. La première affiche la description de la salle, des ennemis, et liste les action possibles. La seconde affiche l'écran de statut: la vie, l'or possédé, l'armure, le bonus de dégât et de soin et, si un ennemi est présent, sa vie. Ces informations sont affichées grâce à **statut(…)**. Ensuite viens les deux parties pour les cartes, qui sont affichées grâce aux fonctions vues plus haut. Enfin, la partie visuelle, destinée à l'affichage des images des salles, des ennemis et des objets, que nous avons étendue (en cours de route) à la moitié droite de l'écran, le minimum pour afficher les visuels qui nous plaisaient le plus. Ceux-ci sont réalisé en ASCII, et ont été trouvés sur divers sites, générés grâce à un convertisseur (d'image ou de texte), ou fait main, comme l'écran titre (sauf l'épée) ou de fin. La plupart des visuels a été trouvé sur les sites suivants puis modifiés:

- *http://www.chris.com/ascii/*
- *http://www.retrojunkie.com/asciiart/myth/myth.htm*

Les convertisseurs utilisés sont les suivants:

- *http://patorjk.com/software/taag/#p=display&f=Doh&t=Type%20Something%20*
- *http://www.text-image.com/convert/*

Ces images sont des listes de chaînes de caractères, obtenue à partir de fichier textes, grâce à la fonction **visuel()**. L'écran est séparée en ces 5 parties grâce à **affichage_scinde()**, écrite de façon peu astucieuse (on aurait pu générer un cadre vide puis appliquer les différentes liste de chaînes avec une procédure) mais relativement performante : on concatène les bordures et les chaînes renvoyées par toutes les fonctions d'affichage (après les avoir mis aux bonnes dimensions) suivant la zone de l'écran.

Nous avons également fait une fonction **cxy(…)** permettant de centrer les visuels. Elle ajoute dans la liste des chaînes vides afin de centrer verticalement, puis dans chaque chaîne des espaces afin de les décaler horizontalement pour que l'image soit au centre de l'écran. La différence est calculée à partir de la taille de l'écran et de la taille de l'image, déterminée dans **dimensions_fichier(fichier)**.

Pour avoir un écran titre dynamique, nous avons fait s'afficher successivement plusieurs images, afin de créer une petite animation.

Pour ajouter de la couleur, nous avons tenté d'écrire un procédé qui fonctionnerait à la fois dans un shell classique (linux, apple, androïd) et une console ms-dos (à la fois car c'est ce que nous avions à disposition et pour pouvoir distribuer notre jeu sans soucis). Nous avons décidé d'un caractère spécial, **&**, qui servirait de balise d'ouverture et de fermeture de commandes maisons. Celles-ci sont pour l'instant au nombre de trois : &couleur [couleur]&, &fond [fond]&, et &couleur [couleur] fond [fond]& (l'idée initiale était de les utiliser également pour simuler une pause dans l'affichage). Nous avions de toute façon un problème de comptage (dans les fonctions pour scinder le texte et centrer les chaînes) avec les caractères

d’échappement AINSI : **\033** représente un caractère, **[** un autre, XX un ou deux autres et **m** un dernier ; la commande **“\033]XXm”** (par exemple) comptait donc comme 5 caractères, contre 0 visibles à l’écran dans un terminal compatible, ce qui avait des conséquences dramatiques. Nous utilisons donc les fonctions et procédures **cprint**, **cinput** et **clen** (pour colored print, input et len) qui parcourent la chaine à la recherche d’une commande tout en l’écrivant pour cprint et cinput, et ne tient pas compte des commandes dans le comptage pour clen. La procédure **commande(“commande”)** se charge d’exécuter la commande adéquate en fonction du système d’exploitation de l’utilisateur.

Nous avons aussi eu le temps d’ébaucher une fonction cscinder qui, en plus de scinder une longue phrase en liste de chaînes de la bonne dimension (tout en tenant compte des sauts de lignes compris dans la chaîne) comme le faisait son homologue scinder, renvoie à la ligne suivante les mots qui devraient être coupés. Elle tient aussi compte des commandes maisons dans son découpage, mais cette partie-là n’est pas encore complète (actuellement cela provoque un saut de ligne ; de plus, si la commande est une couleur, il faudrait la reporter sur les lignes suivantes, tout en repassant en couleur par défaut en fin de ligne pour le visuel).

Résultat :


Vous entrez dans une nouvelle pièce.
Vous poussez avec difficulté la lourde porte de pierre... Vous voyez ce qui s'apparente au butin du maître des lieux, une porte menant vers l'ouest, le terriible Shlepper.
Vie: [----------] 30/30
Or: pas assez
Bonus Degats: ?
Armure: 2
Bonus Soin: 5

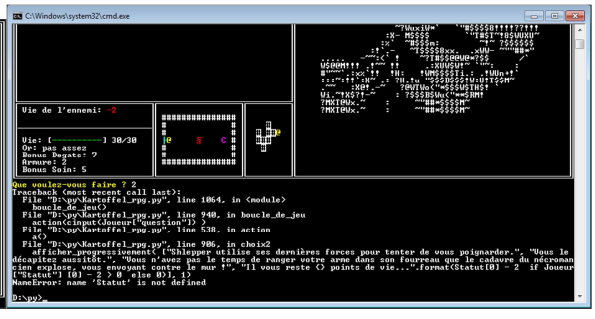

Quoi ? Une majestueuse licorne se dresse devant vous ! Pas de quartier, on est dans un donjon, il faut la tuer !
La licorne a 25 points de vie
Il vous reste 30 points de vie
Vous pouvez faire les actions suivantes :
1: attaquer, 2: lancer sort de soin, manger la patate.
Vie de l'ennemi: 25
Vie: [----------] 30/30
Or: pas assez
Bonus Degats: 5
Armure: 2
Bonus Soin: 5
Que voulez-vous faire ? _


L'orc vous agresse avant que vous n'ayez pu 'ouvrir porte nord' !
Une ombre vous fait face. À l'odeur nauséabonde, vous reconnaissez un orc. Celui ci dispose d'une armure en acier. À voir l'hostilité énanant de son regard, il n'a pas l'air d'apprecier de vous trouver ici.
L'orc a 20 points de vie
Il vous reste 30 points de vie
Vous pouvez faire les actions suivantes :
1: attaquer, 2: lancer sort de soin.
Vie de l'ennemi: 20
Vie: [----------] 30/30
Or: 6 pièce
Bonus Degats: 0
Armure: 0
Bonus Soin: 5
Que voulez-vous faire ? _


Vous obtenez un score de 3
Votre adversaire obtient un score de 2
Vous atteignez votre cible !*
Vous obtenez 5 pièces d'or !
Vous obtenez un anneau !
Une créature, au fond de la salle, vous regarde fixement. Vous voyez une porte menant vers l'ouest, une porte menant vers le nord.
Vous pouvez faire les actions suivantes :
ouvrir porte ouest, ouvrir porte nord.
Vie: [----------] 30/30
Or: 6 pièce
Bonus Degats: 0
Armure: 0
Bonus Soin: 5
Que voulez-vous faire ? _


Le voilà. Devant vous se dresse le Tyran, Shlepper. Vous apercevez son trésor derière lui. Votre regard se pose sur son sceptre. Il en émane une aura troublante.
«Alors ç'est toi, l'impétueux mercenaire tentant de me détrôner. de t'attendais» Vous lance-t-il. Vous vous lancez alors dans le combat, attiré par tant de pouvoir.
Vie de l'ennemi: 150
Vie: [----------] 30/30
Or: pas assez
Bonus Degats: ?
Armure: 2
Bonus Soin: 5
... _


Shlepper tombe à genoux. Il a perdu ce combat, et il le sait. Ses forces l'ont abandonné.
Vous vous approchez.
Vous pouvez faire les actions suivantes :
1 : Asséner le coup de grâce au nécromancien, en saluant sa combativité. 2 : Narguer votre ennemi avant de le tuer comme un péon.
Vie de l'ennemi: -2
Vie: [----------] 30/30
Or: pas assez
Bonus Degats: ?
Armure: 2
Bonus Soin: 5
Que voulez-vous faire ? _


C:\Windows\system32\cmd.exe
Vie de l'ennemi: -2
Vie: [----------] 30/30
Or: pas assez
Bonus Degats: ?
Armure: 2
Bonus Soin: 5
Que voulez-vous faire ? 2
Traceback (most recent call last):
File "D:\py\Kartoffel_rpg.py", line 1064, in <module>
boucle_de_jeu()
File "D:\py\Kartoffel_rpg.py", line 940, in boucle_de_jeu
actionCinputCJoueur!("question")) ?
File "D:\py\Kartoffel_rpg.py", line 538, in action
i()
File "D:\py\Kartoffel_rpg.py", line 906, in choix2
afficher_progressivement(("Shlepper utilise ses dernières forces pour tenter de vous poignarder.", "Vous le décapitez aussitôt.", "Vous n'avez pas le temps de ranger votre arme dans son fourreau que le cadavre du nécromancien explose, vous envoyant contre le mur !", "Il vous reste {} points de vie...".format(Statut[0] - 2 if Joueur["Statut"] [0] - 2 > 0 else 0), 1)
NameError: name 'Statut' is not defined
D:\py>

oups !

IV / Anexe