Programmation fonctionnelle

Devoir de programmation

2021-2022

1 Introduction

Il s'agit d'implanter un petit jeu de type RPG (*Role Playing Game*) dans lequel le joueur a une classe et le cœur de son activité est de combattre des monstres et de récupérer leurs possessions lorsqu'il les a vaincus.

2 L'aventure de base

1. Un personnage a un nom, une classe (`archer`, `guerrier` ou `magicien`), un genre, un nombre de points de vie, un nombre de points d'expérience, un niveau, et un sac contenant divers objets en diverses quantités. Les objets peuvent être des pièces de monnaie, des cuisses de poulet ou des éponges.
   Définir des types en conséquence.
2. Un monstre a un type (`golem`, `nuée de moustiques` ou `sanglier`) et possède un objet que le personnage pourra ramasser triomphalement. On considérera les nuées de moustiques en nombre variable comme un seul monstre.
   Définir les types en conséquence.
3. Écrire une fonction affichant le contenu du sac. On demande un affichage sous la forme suivante :

   ```
   2 éponges
   1 poulet
   2 pièces
   ```

   Écrire une fonction qui affiche l'état du personnage et qui prend en compte le genre (comme tous les messages du jeu). Par exemple :

   ```
   +----------------------------+
   | Elwina | Guerrière niveau 1 |
   +----------------------------+
   | Points de vie |          18 |
   +----------------------------+
   | Expérience    |          15 |
   +----------------------------+
   | Sac                        |
   |  2 éponges                 |
   |  1 poulet                  |
   |  2 pièces                  |
   +----------------------------+
   ```

4. La fonction `Random.int` prend un argument entier `n`, et renvoie un nombre aléatoire dans l'intervalle $[0..n[$.

   Pour obtenir des tirages pseudo-aléatoires différents entre deux lancements du programme, vous devez utiliser la fonction `Random.self_init()` une fois au début du programme.

   Les monstres ont un nombre de points de vie qui dépend du type de monstre :

   — Golem : 25 + 1D6
   — Moustique : 2 + 1 point par moustique
   — Sanglier : 10 + 1D4

   où xDn veut dire la somme de x nombres tirés aléatoirement entre 1 et n.

   Écrire une fonction qui génère aléatoirement un monstre.

5. Écrire une fonction `frapper`, qui calcule le nombre de points de vie enlevés à l'adversaire lors d'une attaque. Un `barbare` fait 10 points de dégâts, un `archer` 4 et un `magicien` 5. Cependant, avant de faire des dégâts, il doit réussir à toucher sa cible. Un `barbare` a 30% de chances de toucher, un `archer` 70% et un `magicien` 50%. Le personnage gagne 5% de bonus au toucher par niveau.
6. Écrire une fonction `monstre_frapper`, sachant qu'un `golem` fait 4 points de dégâts, chaque `moustique` d'une armée $\frac{1}{2}$ point et un `sanglier` 2 points.
7. Écrire une fonction `manger`, qui prend en argument un personnage, et qui enlève un poulet dans le sac du personnage et augmente ses points de vie de 2 points (sans jamais dépasser 20 points de vie). Cette fonction renvoie un couple formé d'une valeur booléenne, qui indique si le personnage a réussi à manger, et du nouvel état du personnage. Si le sac ne contient pas de poulet, le personnage ne peut pas manger.
8. Écrire une fonction `dormir` qui permet au personnage de récupérer 4 points de vie. Attention il y a 5% de chances qu'un monstre surprenne le personnage dans son sommeil et le tue. La fonction renvoie une valeur qui contient soit le nouvel état du personnage ou le monstre qui l'a tué.
9. Un personnage commence l'aventure avec 20 points de vie. Après chaque combat, si le personnage n'est pas mort, il gagne des points d'expérience selon le type de monstre vaincu, ainsi que le potentiel objet possédé par le monstre (les moustiques ne possèdent rien). L'expérience gagnée dépend du monstre tué :

   — Golem : 8
   — Nuée de moustiques : 2
   — Sanglier : 4

   À la fin d'un combat, on vérifie si le personnage gagne un niveau. Le personnage commence au niveau 1, et il passe au niveau n avec $2^n \times 10$ points d'expérience.

   Écrire une fonction `combattre`, qui effectue un combat entre un personnage `p` et un monstre `m`. Cette fonction renvoie la mise à jour du personnage, s'il ne meurt pas. La mort d'un personnage peut se gérer avec la levée d'une exception, ou par l'utilisation d'un type spécifique (à vous de choisir).
10. Écrire une fonction `malheureuse_rencontre`, qui génère aléatoirement un monstre et le fait s'affronter avec le personnage.
11. Le jeu commencera par la création du personnage. Ensuite il comprendra au minimum des rencontres aléatoires qui pourront mener à des combats. Le jeu s'arrête si le joueur décide de s'arrêter, si son personnage meurt, ou s'il atteint le niveau 10.
    Écrire la fonction principale du jeu.

Le déroulement d'une partie après la création du personnage pourra ressembler à ceci :

```
Vous entendez du bruit à l'orée de la forêt et vous apercevez
un sanglier.

Que faites-vous ?
(A) Attaquer
(F) Fuir
(V) Visualiser l'état de votre personnage
<?> F

Dans votre précipitation à fuir vous perdez 1 poulet.

Soudain, vous tombez nez à nez avec une nuée de 7 moustiques.
Ils vous attaquent !
La nuée de moustiques fait mouche, vous perdez 3 points de vie.
Vous ripostez, mais vous manquez la cible.
L'ennemi attaque, mais vous manque.
Vous frappez et infligez 10 points de dégât.
La nuée de moustiques est terrassée.
Vous gagnez 2 points d'expérience.
Vous progressez au niveau 2 !

Que voulez-vous faire ?
(C) Continuer votre chemin
(D) Dormir
(M) Manger
(V) Visualiser l'état de votre personnage
(Q) Quitter l'aventure
<?> M

Vous mangez 1 poulet et gagnez 2 points de vie.

Que faîtes-vous ?
(C) Continuer votre chemin
(D) Dormir
(M) Manger
(V) Visualiser l'état de votre personnage
(Q) Quitter l'aventure
<?> D

Vous installez votre campement et tombez rapidement endormie.
Pendant votre sommeil, un golem surgit et vous fracasse le crâne.
Vous êtes morte !
```

3 L'aventure étendue

Étendez le jeu comme vous le voulez. Cela peut être par exemple :

1. une grande variété et qualité des messages affichés,
2. de nouveaux objets avec de nouveaux effets dans le jeu (potions, autres aliments, ...),
3. de nouvelles classes de personnages avec leurs avantages et désavantages propres,

4. de nouveaux monstres,
5. des monstres avec des capacités spéciales,
6. des rencontres et des combats avec des équipes de monstres plutôt qu'un seul monstre,
7. la gestion d'une équipe de personnages plutôt qu'un seul personnage et des combats à plusieurs,
8. la gestion de personnages non-joueurs qui peuvent aussi combattre le personnage joueur,
9. la gestion de personnages non-joueurs avec lesquels on peut discuter et qui peuvent aider ou échanger/acheter/vendre des objets au personnage,
10. un système plus flexible de combat, avec la prise en compte d'armes,
11. un système de magie et de lancer de sorts pour les mages,
12. la possibilité d'acheter des objets dans des villages,
13. d'autres types d'évènements aléatoires, heureux ou dangereux, au cours de l'aventure,
14. un mode scripté d'aventure (comme dans un livre dont vous être le héros)
15. la gestion du déplacement sur une carte définie ; vous pouvez représenter les déplacements par un graphe, où les arêtes sont étiquetées par le choix que le personnage peut faire ;
16. la gestion du déplacement avec des choix éventuellement conditionnés ; par exemple, un choix pourrait être `(O) Ouvrir la porte massive qui vous fait face au nord` si le personnage possède dans son sac la clé de cette porte, ou `Une porte massive vous fait face au nord, mais elle est fermée` si le personnage n'a pas la bonne clef,
17. la gestion du mode scripté ou carte en lisant le scénario ou la carte depuis le disque et donc la possibilité de proposer plusieurs aventures différentes (qu'il faut néamoins concevoir),
18. un mode donjon avec déplacements, mais généré aléatoirement,
19. la sauvegarde de la partie en cours d'aventure, et la reprise depuis une sauvegarde,
20. la gestion virtuelle du passage du temps et la gestion plus réaliste de la santé du personnage (il faut manger, boire et dormir un minimum),
21. gérer une mise en forme des messages plus agréable visuellement et avec des couleurs

. . .ou toute autre extension que vous jugerez pertinente pour rendre le jeu plus intéressant !

Soyez créatifs ! Les extensions proposées par votre groupe doivent être uniques.

4 Travail demandé

4.1 Groupe

Le devoir est à programmer par groupe de *trois* étudiants, appartenant au même groupe de TP. Si le nombre d'étudiants dans le groupe de TP n'est pas divisible par trois, un ou deux binômes seront autorisés. *Les groupes de plus de trois étudiants sont strictement interdits.* La composition de votre groupe est à communiquer au plus tard le vendredi 25 mars à votre chargé de TP.

4.2 Rendu

Le devoir sera déposé sur Celene sous la forme d'un dossier nommé d'après le groupe défini sur Celene (`groupeN`, où `N` est le numéro de groupe, sans espace), et archivé au format **zip** (et uniquement ce format). Attention l'archive zip doit contenir à sa racine uniquement le dossier `groupeN`.

Le dossier contiendra :

— le ou les fichiers sources, éventuellement organisés en sous-répertoires, et avec si besoin le nécessaire pour générer un fichier exécutable (par exemple avec l'outil `dune`) ; si votre contribution contient plusieurs fichiers, le fichier principal doit se nommer `main.ml` ;

— un document au format *markdown*, intitulé `devoir.md`, qui contiendra deux parties :

  — la répartition précise du travail dans le groupe ;

  — un bref mode d'emploi du jeu développé en insistant sur l'extension.

Si les fichiers fournis contiennent des erreurs de syntaxe OCaml ou de typage, la note sera automatiquement 0. Si l'archive n'est pas au format **zip**, elle ne sera pas ouverte et la note sera 0.

Si les autres parties de la spécification ne sont pas respectées, un malus de 5 points sera appliqué (par exemple non respect de l'organisation des répertoires, format de document non respecté, *etc.*).

4.3 Date limite

Le devoir doit être déposé sur Celene au plus tard le 23 avril 2022.

4.4 Barème et critères de notation

La notation est **individuelle** : la participation de chaque membre du groupe sera appréciée individuellement notamment en se basant sur la répartition du travail indiquée par vos soins dans le document et surtout lors de la présentation et des réponses aux questions posées.

— Soutenance : 8 points

  Critères : clarté de la présentation, correction et concision des réponses aux questions.

— L'aventure de base : 8 points

  Critères : correction du code (le programme doit fonctionner sans lever d'exceptions, et les fonctions doivent renvoyer des résultats conformément à leurs spécifications) ; qualité du code (en particulier clarté et pertinence de l'organisation du code, qualité de l'indentation, qualité des noms de fonctions et de variables) ; qualité de la documentation de chacune des fonctions (qui doit être fournie dans tout le code en format `ocamldoc`) ; utilisation de fonctions de la bibliothèque OCaml.

— L'aventure étendue : 4 points

  Critères : les mêmes que pour l'aventure de base.

4.5 Intégrité académique

Pour ce devoir, comme pour toutes vos activités académiques, vous devez respecter les règles de l'intégrité académique. Il est en particulier interdit de copier du code d'autres groupes ou sur Internet.

Les correcteurs utiliseront un logiciel pour calculer automatiquement la similarité de code entre groupes. Une similarité supérieure à 20% entre deux ou plusieurs groupes entraînera automatique la saisie du conseil de discipline.