INF1035 – Concepts avancés en objet

Session AUT-15

TRAVAIL PRATIQUE #2

Indications principales	
Date de présentation de l'énoncé	11 novembre 2015
Date de remise du travail	22 décembre 2015
Politique sur remises tardives	-25% par jour de retard
Éléments et format de remise	Rapport et fichiers complet de la solution ou du projet remis sous forme électronique sur le portail de cours dans un seul fichier compressé (.zip)
Pondération	20%
Nb d'étudiants par équipe	1 à 4, sans exception

INTRODUCTION

Nous aurons vu d'ici la fin de le session les notions d'héritage, de polymorphisme, de gestion d'exceptions, d'interfaces, de sérialisation et d'événements.

Dans ce second travail double, vous serez appelés à créer un jeu de rôle japonais classique (JRPG) dans lequel vous devrez créer une aventure pour un groupe de combattants avec des combats tour par tour contre des ennemis et des fichiers de sauvegarde, tout en mettant à profit les connaissances acquises pendant la session.

Ce travail doit absolument être réalisé en C# et doit être exécutable et consultable à partir des versions 2013 ou 2015 de Visual Studio.

DESCRIPTION GÉNÉRALE DU TRAVAIL EXIGÉ

Dans le cadre de ce travail, on vous demande de créer une **application console ou Windows Form/WPF** qui permet à un joueur de jouer à un jeu de rôle japonais.

Le joueur qui lance l'application peut soit débuter une nouvelle aventure où il créera son équipe d'aventuriers, soit charger une aventure déjà commencée.

Le joueur est par défaut dans un village où il peut soit gérer son groupe, se rendre à la forge pour acheter de l'équipement, se rendre à un lieu sacré pour ressusciter un membre décédé du groupe, se rendre à la guilde pour renvoyer ou remplacer un membre décédé de l'équipe, sauvegarder ou charger une partie, ou partir pour une aventure d'un certain niveau de difficulté.

Lorsque parti à l'aventure, le joueur doit faire face à un certain nombre de batailles dont la dernière comporte aussi un boss. Au fil des batailles, le joueur fait des gains en pièces d'or et en points d'expérience. Tous les aventuriers du groupe qui survivent à l'aventure se séparent les points d'expérience à la fin, pour éventuellement gagner des niveaux d'expérience et améliorer leurs statistiques, les rendant plus forts et aptes à tenter des aventures plus difficiles.

On peut ainsi séparer le jeu en plusieurs volets à concevoir et développer, dont les exigences minimales seront décrites par la suite:

- La gestion des aventuriers et du groupe d'aventurier;
- La gestion des ennemis et des groupes d'ennemis;
- La gestion des batailles;
- La gestion des aventures;
- La gestion du village.

Note : le travail est présenté sous le point de vue d'un monde médiéval fantastique, mais vous pouvez faire un monde analogue équivalent sous un thème spatial, cyberpunk, ou moderne, par exemple, selon ce qui vous inspire. Bien évidemment, restez dans le spectre de la décence dans vos choix!

SPÉCIFICATIONS DE LA GESTION DES AVENTURIERS ET DU GROUPE D'AVENTURIER (2%)

Un **aventurier** est défini minimalement comme suit :

- Il possède (sous forme de champs/propriétés selon les cas) :
  - Un nombre de **points d'expérience**;
  - Un **niveau d'expérience**, qui peut être déduit à partir du nombre de points d'expérience (au minimum jusqu'à 10 niveau d'expérience doivent être atteignables);
  - Un **nom**, choisi par l'utilisateur au moment de sa création;
  - Une **classe** parmi au minimum 4 classes différentes (p. ex. Magicien, Rogue, Guerrier et Druide) :
    - Chaque classe doit offrir au minimum 3 pouvoirs/sortilèges/habiletés spéciales qu'elle pourra faire appel pendant les combats;
    - Ces compétences peuvent inclure l'utilisation de mécanismes ou de champs (caractéristiques) spéciaux pour ces classes *(p.ex. une classe Berserker qui cumule des points de rage à chaque tour de combat qui sont nécessaires pour utiliser les habiletés, une classe Mage qui utilise des points de mana comme réservoir à des fins de lancement de*

sortilège, une classe Mage de sang qui utilise plutôt son réservoir de Points de vie pour lancer ses sortilèges, etc.);

- Vous devez utiliser des notions d'héritage pour représenter ces classes.

> Lorsque vous travaillez avec l'héritage, il est attendu de vous que vous créez des classes ou membres abstraits lorsqu'il convient de le faire, que vous redéfinissiez les méthodes dans les sous-classes lorsqu'aussi nécessaires.

- Un **état** général parmi les suivants au minimum :
  - Normal;
  - Étourdi : un personnage étourdi n'est pas en mesure de faire quelconque action à son tour;
  - Mort

- Un nombre de **Points de vie actuel** (PV) : si à 0, alors le personnage devient dans l'état « Mort »;

- Un ***certain nombre de caractéristiques minimales*** :
  - **Points de vie de base** (maximum) :
  - Points d'**initiative de base**: habileté d'un personnage à attaquer rapidement (détermine l'ordre d'attaque);
  - Points de **précision de base**: habileté d'un personnage à atteindre sa cible;
  - Points d'**esquive de base**: habileté d'un personnage à esquiver une attaque qui aurait autrement touchée le personnage;
  - Point de **force de base** : plus un personnage est fort, plus lorsqu'il touche un ennemi il lui infligera des dégâts à ses points de vie;
  - Point de **défense de base** : plus un personnage a une haute défense, plus il réduit les dégâts qui lui sont infligés s'il ne réussit pas à esquiver l'attaque ennemie;
  - Des versions totales de ces caractéristiques (**PV totaux**, **initiative totale**, **précision totale**, **force totale**, **esquive totale**, **défense totale**) qui prennent en considération la valeur de base pour la caractéristique et tous les bonus également octroyés par une ou plusieurs des pièces d'équipements équipées par le personnage (voir « Équipement »)

- Un **équipement**, c'est-à-dire une liste d'items « équipable » dont au minimum :
  - Une **arme**; augmente la force et possiblement la précision, au minimum;
  - Un **bouclier** (si l'arme est à une seule main); augmente au moins l'esquive;

- Une **armure**; augmente au moins la défense;
- Et optionnellement d'autres (casque, bottes, accessoire, …).

- Il offre minimalement les méthodes suivantes ou l'équivalent (à vous de voir pour les paramètres):
  - **Attaquer()** :
    - Attaque un ennemi et retourne un résultat de l'attaque :
      - Soit l'attaque est esquivée par l'ennemi (on doit pouvoir indiquer cela);
      - Soit l'attaque a fonctionnée et on retourne le nombre de dégâts et potentiellement un état à infliger (comme « étourdis »);
    - L'attaque est calculée de la sorte :
      - À la base, le personnage a 50% de probabilités de toucher un ennemi;
      - L'ajustement par la suite se fait en ajoutant le résultat de Personnage.PrecisionTotale - Ennemi.Esquive, et est toujours situé entre 10% et 90% peu importe le cas;
      - Ensuite, on génère une valeur au hasard entre 0.0 et 1.0 et si la valeur est inférieure à la probabilité de toucher, le personnage inflige des dégâts;
      - Le nombre de dégâts est calculé en faisant Personnage.ForceTotale – Ennemi.Defense, et est au minimum de 1;
      - Si l'arme inflige également un statut particulier, faites votre propre calcul : soit l'arme est accompagné d'un % de conférer un statut, soit vous donnez toujours le même % (p. ex. 25 ou 50%).

  - **UtiliserCompetence()** :
    - Vous devez décider de quelle façon vous procédez pour appeler chacune des compétences;
    - Vous pouvez avoir 3 méthodes (UtiliserCompetenceA(), UtiliserComptenceB(), UtiliserCompetenceC()) qui sont définies pour chaque classe, ou encore utiliser une seule méthode mais un paramètre pour indiquer laquelle des 3 compétences on tente d'utiliser;
    - À vous de voir si les pré-requis nécessaires pour utiliser la compétence sont remplis et quoi faire s'ils ne le sont pas;
    - Si vous le souhaitez, le niveau d'expérience et/ou certaines caractéristiques peuvent avoir une influence sur les effets de l'utilisation de la compétence.

  - **MonterNiveauExperience()** :

        - Lorsque l'ajout de points d'expérience fait en sorte de dépasser certains seuils, le personnage est réputé monter de niveau d'expérience;
        - Lorsque c'est le cas, cette méthode est appelée et les caractéristiques de base du personnage sont augmentées selon la classe du personnage;
            - Vous pouvez soit faire des augmentations fixes par niveau, ou un peu au hasard (p.ex. entre 3 et 5 forces supplémentaires, …).
    - Un **constructeur** permettant de générer les caractéristiques de base pour la classe en question :
        - Les valeurs peuvent être prédéterminées selon la classe ou laissées partiellement au soin du hasard.

Un **groupe d'aventurier** est défini minimalement comme suit :

- Possède :
    - **Une liste ou un tableau d'aventuriers** (pas plus de 3) et les **méthodes de délégations nécessaires** pour accéder aux aventuriers et aux items de l'inventaire (voir plus bas);
    - Un nombre de **pièces d'or** ou une monnaie quelconque;
    - **Un inventaire** :
        - L'inventaire contient une **collection d'items** qui peuvent appartenir à différentes catégories (voir définition plus bas);
        - Soit les items dans cette collection ont un statut « équipé » ou « non équipé » (utilisé ou non par un des aventuriers du groupe), soit seuls apparaissent les items non équipés (équiper un item enlève donc l'item de l'inventaire et le déséquiper le remet) : à vous de voir!
- Offrent les fonctionnalités suivantes :
    - **AjouterAventurier()** et **RetirerAventurier()** :
        - Ajoute ou retire un aventurier;
        - Vous devez vérifier à ce qu'on ne puisse jamais retirer un aventurier s'il est le seul du groupe, ou qu'on ne puisse pas ajouter un 4^{e} aventurier.
    - **UtiliserItem()** : utilise un consommable comme une potion ou de la nourriture, par exemple, sur le groupe en entier ou un aventurier particulier.

Un **item** est défini minimalement comme suit :

- Possède :
    - Un **nom**;
    - Une **catégorie** (arme, armure, accessoire, consommable, … et possiblement une classification plus poussée (arme à une main et arme à deux mains) selon vos besoins; il est très possible qu'une partie de cette catégorisation/classification soit résolue par des relations d'héritage; par exemple, les armes n'ont pas les mêmes caractéristiques nécessairement que les armures;
    - Des **caractéristiques en lien avec les catégories d'items** en question :

- Les pièces équipables peuvent donner des caractéristiques bonus;
- Les items consommables peuvent simplement guérir un état ou restaurer des points de vie.

- Une **valeur d'achat** (en pièces d'or, p. ex.).

SPÉCIFICATIONS DE LA GESTION DES ENNEMIS ET DES GROUPES D'ENNEMIS (2%)

Les ennemis partagent beaucoup de similitudes avec les aventuriers.

Un **ennemi** est défini minimalement comme suit :

- Il possède (sous forme de champs/propriétés selon les cas) :
  - Un **niveau d'expérience**, qui détermine la difficulté de l'instance du monstre et sert à déterminer les valeurs de ses caractéristiques;

  - Un **nom**, qui représente le nom de l'ennemi;

  - Un **état** général parmi les suivants au minimum :
    - Normal;
    - Étourdi : un ennemi étourdi n'est pas en mesure de faire quelconque action à son tour;
    - Mort

  - Un nombre de **Points de vie actuel** (PV) : si à 0, alors l'ennemi devient dans l'état « Mort »;

  - Un ***certain nombre de caractéristiques minimales*** :
    - **Points de vie** (maximum) :
    - Points d'**initiative**;
    - Points de **précision**;
    - Points d'**esquive ;**
    - Point de **force** ;
    - Point de **défense** ;

  - Une **compétence spéciale optionnelle** (comme guérir, lancer un sortilège, etc.; peut être null);

  - Une **stratégie d'action** (Guérir d'abord, Attaquer le plus faible, Attaquer le plus fort, Attaquer le guérisseur, etc.);

    - Au plus simple, vous pouvez toujours renforcer la même stratégie et si elle ne s'applique pas, appliquer une stratégie par défaut;

        - Pour quelque chose de plus complexe, vous pourriez avoir un tableau de paires stratégie-valeur dont la valeur représente un poids; parmi toutes les stratégies applicables à la situation lorsque vient le temps pour l'ennemi d'agir, la probabilité d'utiliser chacune des stratégies possibles est normalisée pour totaliser 100% en fonction des poids et une stratégie est sélectionnée au hasard.

    - **Gain en expérience** lorsque défait; cela peut être un domaine de valeurs possibles à déterminer au hasard après la bataille;

    - **Gain en pièces d'or** lorsque défait : idem;

    - **Items potentiellement récupérables** par les aventuriers lorsque défait :
        - Il s'agit d'une liste de paires items-probabilité.

    - ***Suggestion optionnelle*** : faites-en sorte d'avoir également différentes versions plus ou moins dangereuses de l'ennemi, du genre « Normal, Leader, Mini-Boss, Boss » qui modifie la puissance de l'instance de l'ennemi et modifie les gains si battus.

    - ***Créez obligatoirement au minimum 5 ennemis différents, c'est-à-dire 5 classes dérivées de la classe Ennemi qui utilisent des magnitudes différentes pour leurs caractéristiques, qui donnent plus ou moins de gains en points d'expérience, en pièces d'or et en items.***

- Il offre minimalement les méthodes suivantes ou l'équivalent (à vous de voir pour les paramètres):
    - **Agir() :**
        - On doit déterminer quelle sera l'action posée par l'ennemi en fonction de sa stratégie d'action à privilégier ou au hasard par rapport aux probabilités des stratégies applicables (voir plus haut);

    - **Attaquer()** :
        - Attaque un personnage et retourne un résultat de l'attaque :
            - Soit l'attaque est esquivée par le personnage ciblé (on doit pouvoir indiquer cela);
            - Soit l'attaque a fonctionnée et on retourne le nombre de dégâts et potentiellement un état à infliger (comme « étourdis »);
        - L'attaque est calculée de la sorte :
            - À la base, l'ennemi a 50% de probabilités de toucher un personange;

- L'ajustement par la suite se fait en ajoutant le résultat de Ennemi.Precision - Personnage.EsquiveTotale, et est toujours situé entre 10% et 90% peu importe le cas;
- Ensuite, on génère une valeur au hasard entre 0.0 et 1.0 et si la valeur est inférieure à la probabilité de toucher, l'ennemi inflige des dégâts;
- Le nombre de dégâts est calculé en faisant Ennemi.Force – Personnage.DefenseTotale, et est au minimum de 1;
- Si l'ennemi inflige un statut particulier, on doit le calculer et l'affecter au personnage concerné.

- **UtiliserCompetence()** : l'ennemi applique sa compétence.

- Un **constructeur** permettant de générer les caractéristiques de base pour l'ennemi en question :
  - Les valeurs peuvent être prédéterminées selon la classe ou laissées partiellement au soin du hasard.

Un **groupe d'ennemi** est défini minimalement comme suit :

- Possède :
  - **Une liste ou un tableau d'ennemi** (pas de maximum imposé) et les **méthodes de délégations nécessaires** pour accéder aux ennemis;
  - Un **gain total de pièces d'or** qui récupère tous les gains des ennemis du groupe;
  - Un **gain total de points d'expérience** qui récupère tous les gains des ennemis du groupe;
  - Un **gain d'items qui retourne tous les items à trouver** sur les ennemis;

SPÉCIFICATIONS DE LA GESTION DES BATAILLES (2%)

- Une bataille implique et possède un **groupe d'aventuriers** et un **groupe d'ennemis**;
- Ils **s'affrontent tour par tour** et **s'attaquent mutuellement à chacun d'entre eux** des scores d'initiatives les plus hauts au plus bas :
  - Lorsque c'est le tour d'un ennemi, il fait une action en fonction de sa stratégie privilégiée;
    - Si suite à l'action un personnage est tué, on vérifie à ce qu'il reste au moins un aventurier toujours vivant;
    - Sinon, la bataille prend fin et la partie prend fin également (on doit alors retourner au menu principal du jeu).
  - Lorsque c'est le tour d'un personnage, il doit :
    - Faire le choix de faire différentes actions; au minimum :
      - Attaquer;

        - Utiliser une compétence, où il choisit alors laquelle des 3 compétences il utilise ;
        - Utiliser un item, où il choisit parmi la liste des items consommables à utiliser;
    - Une fois l'action et possiblement la compétence ou l'item sélectionné, le joueur doit indiquer la cible du personnage (un des personnages ou un des ennemis, ou tous les personnages, ou tous les ennemis selon ce qui s'applique);
    - Si suite à l'action un ennemi meurt, on vérifie s'il reste d'autres ennemis dans le groupe d'ennemis;
    - Si c'était le dernier, alors la bataille se termine et les gains sont attribués au groupe d'aventuriers.
- Évidemment, tout cela doit être supporté dans le jeu par un affichage des données pertinentes et des résultats des actions des ennemis et des personnages.

SPÉCIFICATIONS DE LA GESTION DES AVENTURES (2%)

Une aventure :

- Possède une **liste de batailles ordonnées** pour lesquelles les groupes d'ennemis sont prédéterminés (au moins partiellement, avec possiblement des composants au hasard)

- Portent un **nom** , une **description**, et elles ont un **niveau de difficulté** associé (ou un niveau d'expérience suggéré;

- Sont **contenues dans une liste d'aventures** qui décrit toutes les aventures qui font parties du jeu;

- Possède une méthode ou une façon de **démarrer une aventure** qui enchaîne une à une les batailles prévues;

- On doit pouvoir **ajouter des aventures** et **des batailles aux aventures**.

SPÉCIFICATIONS DE LA GESTION DU VILLAGE (2%)

Le village est le point d'accueil après chargement d'une partie ou le commencement d'une partie.

Vous pouvez :

- **Partir à l'aventure :** le joueur sélectionne une des aventures de la liste et débute une aventure;
- **Aller à la maison** : le joueur sauvegarde l'état de son groupe d'aventurier ou charge une nouvelle partie;

- **Visiter un lieu de recueillement** : le joueur peut payer un certain nombre de pièces d'or pour une résurrection d'un des personnages du groupe (qui p. ex. dépend du niveau d'expérience du personnage à ressusciter);
- **Visiter la boutique/l'échoppe** : le joueur peut vendre ou acheter des items;
- **Visiter la guilde/l'auberge** : le joueur peut gérer son équipe (libérer, recruter (possiblement dans le sens de créer un nouveau personnage), équiper des items, etc.

Pour tout le reste, vous êtes libres de laisser aller votre imagination!!!! Amusez-vous!

Note : *la sauvegarde n'est possible que dans le village.*

AUTRES SPÉCIFICATIONS (8%)

GESTION DES TYPES ET DES MEMBRES (2%)

Vous devez faire en sorte de :

- Utiliser les modificateurs (d'accès, de paramètres, etc.) qui font du sens au contexte;
- Utiliser les bons principes d'encapsulation;
- Faire usage des propriétés autant que possible, et non de méthodes GetX() et SetX(), en lecture seule ou sous forme de propriété automatique lorsque cela fait du sens;
- Faire bon usage de l'héritage et du polymorphisme, mais aussi de la délégation;
- Utiliser les types appropriés pour les membres, les paramètres et les valeurs de retour.

GESTION DES PERSISTENCES (2%)

Vous devez faire en sorte que :

- La « partie » (le groupe d'aventuriers) soit sauvegardée et chargeable sous format binaire;
- Tous les items du jeu soient décrits en format XML;
- Les aventures soient également décrites en format XML (par exemple elles contiennent les détails sur les différents paramètres d'instanciation des groupes d'ennemis et des récompenses);
- Idem pour ce qui est vendu dans les boutiques du village (en XML).

GESTION DES EXCEPTIONS (2%)

Vous devez faire en sorte de gérer les exceptions (ou utiliser des instructions using) lorsque nécessaires, minimalement :

- Pour tout ce qui concerne les inputs/outputs utilisateur;
- Pour tout ce qui concerne les écritures/lectures vers des streams ou des fichiers;
- Créez au moins une exception personnalisée qui doit être lancée et capturée.

GESTION DES ÉVÉNEMENTS (1%)

Vous devez intégrer au moins un événement personnalisé à votre projet.

GESTION DES INTERFACES (1%)

Vous devez trouver une façon d'intégrer au minimum une interface que vous créez à votre projet.

RAPPORT ET COMMENTAIRES DANS LE CODE (2%)

Vous devez également remettre un **rapport Word ou PDF** qui doit contenir les éléments suivant (1%) :

- Page de présentation;
- Problèmes, difficultés rencontrés ou justification supplémentaires (s'il y a lieu);
- Instructions spéciales d'exécution du programme (s'il y a lieu).
- Guide d'utilisation COMPLET;

Les justifications peuvent être en lien avec des choix de structures de données particuliers sur lesquels vous voudriez élaborer davantage que dans les commentaires. Elles sont optionnelles.

Vous devez aussi supplémenter votre code de commentaires pertinents (1%).

INSTRUCTIONS SUPPLÉMENTAIRES

Éléments à remettre

Vous devrez remettre l'ensemble des extrants exigés qui seront énumérés ci-dessous dans un seul fichier compressé (en format .zip) dans la section de dépôt des travaux sur le portail du cours **avant le début du cours de la remise.**

Dans le cas où vous remettiez le travail en retard, vous ne pourrez alors le déposer sur le portail et vous devrez me le retourner via courriel. Une pénalité de 25% par jour de retard, à compter de l'heure de remise, serait alors appliquée.

Les éléments à remettre sont les suivants :

- Tous les fichiers relatifs au projet de l'application : vous devez remettre le répertoire contenant la solution, le ou les projets associés, les fichiers contenant le code source, les exécutables, etc.

    - **Assurez-vous que le projet soit prêt à être exécuté directement à l'intérieur de la plateforme de développement, c'est-à-dire sans erreur de compilation et sans configuration spéciale non explicitement donnée**. Aucun débogage ne sera fait lors de la correction. C'est TRÈS important. Si je dois reconstruire un

ouveau projet et importer les classes, les fichiers de 12121212iguration et OSĂgurer des paramètres pour vous, il y aura d'importantes pénalités appliquées.

- Le rapport PDF.

N.B. Des points peuvent être retranchés si le travail est remis en retard ou s'il y a des problèmes d'exécution (selon l'ampleur des problèmes rencontrés).