|  |  |  |  |
| --- | --- | --- | --- |
| Développement d’application AVEC BASE DE DONNÉES  420-4DB-HY  **Travail personnel #1**  **Partie serveur de l’application**  **(15 pts)** | | | |
| Pier Luc Ducahrme | | **Bureau :** B2335 | **MIO** |
|  | | | |
| **Description** | | | |
| **But :** | Implémenter un jeu vidéo de type RPG avec l’Entité Framework. | | |
| **Objectifs** | 1. Utiliser d’un modèle relationnel de base de données dans le but de créer une application. 2. Utiliser l’entité Framework pour gérer les éléments du CRUD d’un jeu. 3. Utiliser des procédures stockées avec l’entité Framework. | | |
| **Durée :** | 15 h | | |
| **Pondération :** | Travail évalué sur 100 points mais ramené sur 15 points. | | |
| **Remise :** | La semaine 5. | | |
| **Note** | * **Ce travail se fait en équipe maximum de trois (3).** * Conserver une copie de sécurité. Il est de **votre responsabilité** de conserver une copie de sécurité dans l’éventualité où la lecture des données serait impossible. Cette copie doit être **disponible sur demande**. * Les règles de la PDEA et de la PIEA s’appliquent**.** * Le plagiat, sous toutes ses formes, est interdit. | | |

# Préparation :

# Une fois que vous aurez décidé des membres de votre équipe, vous devez me transmettre cette information par MIO. Je vous donnerai accès à une base de données(MS SQL) et un serveur pour votre code source(Azure DevOP ou GitHub Classroom).

# Vue d’ensemble :

Le but de ce travail est de créer les éléments de base du jeu **Hugo Land.** Ce travail sera réutilisé tout au long de la session et permettra d’implémenter les éléments vus en classe.



Figure : Saisie d'écran du jeu final de Hugo Land

Vous trouverez avec l’énoncé du travail une version de départ de l’application que nous modifierons pour vous donner un aperçu du genre de jeu que nous réaliserons.

Globalement, voici les étapes de développement :

* Lors du **travail pratique #1**, nous réaliserons les fonctions de bases permettant d’interagir avec la base de données en utilisant l’Entity Framework Core.
* Lors du **travail pratique #2**, nous réaliserons un éditeur qui permettra de faire la conception des mondes. Ce travail réutilisera les fonctions développées dans le travail pratique #1.
* Lors du **travail pratique #3**, nous réaliserons le jeu en mode client-serveur.
* Lors du **travail pratique #4**, nous compléterons le jeu mais avec la gestion de la concurrence pour le rendre multijoueur.



TP3+TP4

INTERNET

Service SOA  
(TP#3)

Serveur

Éditeur de monde

(TP#2)

BASE DE DONNÉES

Libraire pour l’accès aux données   
(TP#1)

Dans un premier temps (TP#1), nous allons implémenter la partie serveur qui permettra une interaction avec la base de données.

Ce jeu possèdera une partie serveur et une partie client. La partie serveur s’occupera de gérer :

* **L’interaction avec la base de données**
* **La persistance des données**
* **La gestion de mises à jour réalisées par les clients**

Il est préférable de créer une classe pour chaque objet.

# Modélisation de la base de données

Pour vous économiser du temps, une base de données a été créée pour votre équipe. Voici le modèle relationnel associé. VOUS POUVEZ LE MODIFIER COMME VOUS VOULEZ.

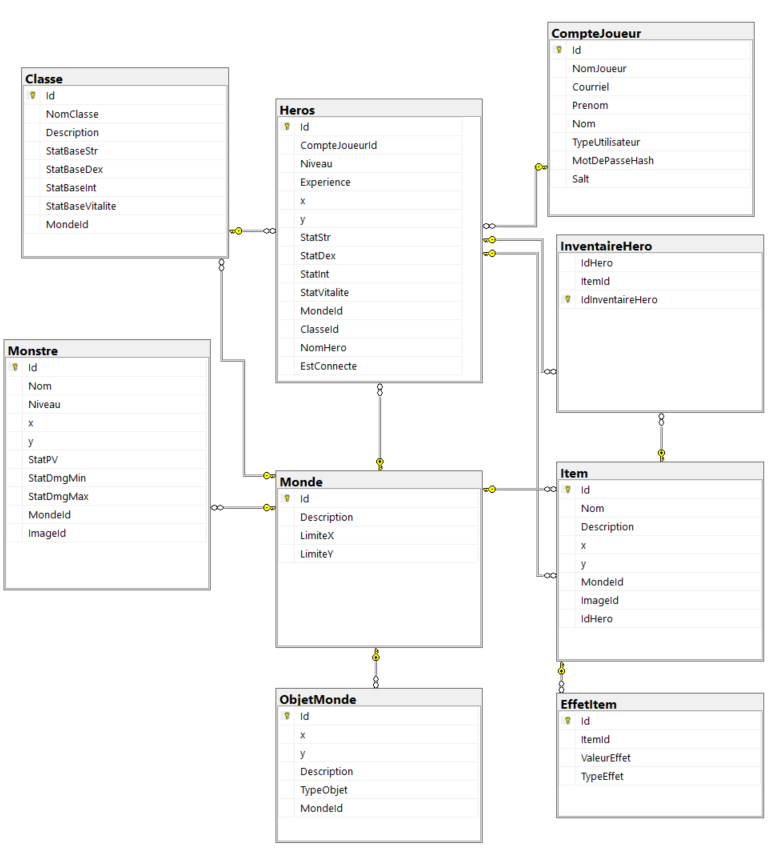


Figure : Modélisation de la base de données fournie

# Explication du travail pratique

Pour chacune des tables dans la base de données, vous devez implémenter les méthodes qui permettent de faire les actions décrites ci-dessous. Attention, vous ne devez pas passer en paramètre la classe modifiée(entitée) et ensuite la lier au contexte. Vous devez passer les bons paramètres et ensuite affecter les entités.

Ensuite, vous devez créer une autre classe qui permet de tester le développement que vous avez fait. Cette classe, par exemple *MondeTestUnitaire*, permettra de tester les cas limites pour les méthodes que vous avez développées. Ensuite, tous les tests unitaires devront être exécutés dans la classe *Program*.

## Monde :

**Objectif** : Cette table permet de stocker les informations relatives à un monde (carte) dans lequel évoluera le jeu. Cette carte spécifie les limites x, y du monde.

***Tâches :*** Vous devez implémenter les méthodes suivantes :

* Fonctions:
  + Créer un monde
  + Supprimer un monde
  + Modifier les limites et la description d’un monde
  + Retourner la liste des mondes
* Tests unitaires :
  + Pour chacune des méthodes ci-dessus, faites une méthode qui appelle vos méthodes que vous venez de développer et affiche les informations dans la console.

## ObjetMonde :

**Objectif** : Cette table permet de définir les éléments du décor dans le monde. Par exemple, cette table définira un rocher, de l’eau, etc. à une position x, y.

***Tâches :*** Vous devez implémenter les méthodes suivantes :

* Fonctions :
  + Créer un objet pour un monde donné.
  + Supprimer un objet pour un monde donné.
  + Modifier la description d’un objet pour un monde donné.
* Tests unitaires :
  + Pour chacune des méthodes ci-dessus, faites une méthode qui appelle vos méthodes que vous venez de développer et affiche les informations dans la console.

## Monstre :

**Objectif** : Cette table permet de définir les monstres dans le monde.

***Tâches :*** Vous devez implémenter les méthodes suivantes :

* Fonctions :
  + Créer un monstre avec des valeurs aléatoires.
  + Supprimer (tuer) un monstre.
  + Modifier les informations d’un monstre
* Tests unitaires :
  + Pour chacune des méthodes ci-dessus, faites une méthode qui appelle vos méthodes que vous venez de développer et affiche les informations dans la console.

## compte joueur :

**Objectif** : Cette table permet de créer les paramètres de connexion du joueur. Une fois connecté, le joueur se connectera et sélectionnera le héros avec lequel il désire jouer.

***Tâches :*** Vous devez implémenter les méthodes suivantes :

* Fonctions :
  + Créer un nouveau joueur (doit utiliser la procédure stockée)
  + Supprimer un joueur
  + Modifier les paramètres d’un joueur
  + Valider la connexion d’un joueur (doit utiliser la procédure stockée)
* Tests unitaires :
  + Pour chacune des méthodes ci-dessus, faites une méthode qui appelle vos méthodes que vous venez de développer et affiche les informations dans la console.

## Pour créer un joueur, vous devez utiliser la procédure stockée que j’ai créée. Cette procédure stockée utilise un champ binaire, un hash et un salt pour encoder le mot de passe. Pour mieux comprendre, voir le site suivant(<https://www.mssqltips.com/sqlservertip/4037/storing-passwords-in-a-secure-way-in-a-sql-server-database/>). Si la création fonctionne, le mot clé SUCCES sera retourné. Inversement, pour vous connecter, vous devez passer en paramètre à la procédure stocké le nom du joueur et e mot de passe en claire. Si c’est le bon mot de passe, le mot clé SUCCES sera retourné.

Voici le code des procédures stockées déjà créées :

Créer un joueur :

CREATE PROCEDURE [dbo].[CreerCompteJoueur]

@pNomUtilisateur NVARCHAR(50),

@pCourriel NVARCHAR(255),

@pPrenom NVARCHAR(50) = NULL,

@pNom NVARCHAR(50) = NULL,

@pTypeUtilisateur int,

@pMotDePasse NVARCHAR(50),

@Message NVARCHAR(250) OUTPUT

AS

BEGIN

SET NOCOUNT ON

DECLARE @salt UNIQUEIDENTIFIER=NEWID()

BEGIN TRY

INSERT INTO dbo.[CompteJoueur] (NomJoueur, Courriel, MotDePasseHash, Salt, Prenom, Nom, TypeUtilisateur)

VALUES(@pNomUtilisateur, @pCourriel, HASHBYTES('SHA2\_512', @pMotDePasse+CAST(@salt AS NVARCHAR(36))), @salt, @pPrenom, @pNom, @pTypeUtilisateur)

SET @Message='SUCCES'

END TRY

BEGIN CATCH

SET @Message=ERROR\_MESSAGE()

END CATCH

END

**Connexion :**

CREATE PROCEDURE [dbo].[Connexion]

@pNomJoueur NVARCHAR(50),

@pMotDePasse NVARCHAR(50),

@Message NVARCHAR(250)='' OUTPUT

AS

BEGIN

SET NOCOUNT ON

DECLARE @userID INT

IF EXISTS (SELECT TOP 1 Id FROM [dbo].[CompteJoueur] WHERE NomJoueur=@pNomJoueur)

BEGIN

SET @userID=(SELECT Id FROM [dbo].[CompteJoueur] WHERE NomJoueur=@pNomJoueur AND MotDePasseHash=HASHBYTES('SHA2\_512', @pMotDePasse+CAST(Salt AS NVARCHAR(36))))

IF(@userID IS NULL)

SET @Message='INCORRECT'

ELSE

SET @Message='SUCCESS'

END

ELSE

SET @Message='INVALIDE'

END

## item :

**Objectif** : Cette table représente les items que peuvent utiliser les héros. Lorsqu’un joueur met un item dans son inventaire, la position x, y de l’item est mise à zéro et l’id du héros est différent de null.

***Tâches :*** Vous devez implémenter les méthodes suivantes :

* Fonctions :
  + Créer un item
  + Supprimer un item.
  + Modifier les informations d’un item.
* Tests unitaires :
  + Pour chacune des méthodes ci-dessus, faites une méthode qui appelle vos méthodes que vous venez de développer et affiche les informations dans la console.

## effet item :

**Objectif** : Cette table permet de définir les bonus/malus d’un item.

***Tâches :*** Vous devez implémenter les méthodes suivantes :

* Fonctions :
  + Ajouter un effet à un item.
  + Supprimer un effet à un item.
  + Modifier un effet à un item.
* Tests unitaires :
  + Pour chacune des méthodes ci-dessus, faites une méthode qui appelle vos méthodes que vous venez de développer et affiche les informations dans la console.

## hero :

Cette table permet de définir le héros d’un joueur.

***Tâches :*** Vous devez implémenter les méthodes suivantes :

* Fonctions :
  + Créer un héros.
  + Supprimer un héros.
  + Modifier les valeurs d’un héros
  + Lister tous les héros pour un joueur.
* Tests unitaires :
  + Pour chacune des méthodes ci-dessus, faites une méthode qui appelle vos méthodes que vous venez de développer et affiche les informations dans la console.

**Bonus :**

* + Pour un héros, retourner tous les éléments (ObjetMonde, Monstre, Item, Héro) qui se trouvent **dans les quatre tableaux autour du tableau(8 x 8) où se trouve le héro**.
  + Déplacer un héros de manière optimale.

## classe :

**Objectif** : Cette table permet de définir les classes associées aux héros du jeu. Chaque classe attribue des bonus/malus à un héros.

Vous devez implémenter les méthodes suivantes :

* Fonctions :
  + Créer une classe.
  + Supprimer une classe.
  + Modifier une classe.
  + Recevoir toutes les classes pour un monde donné (clé).
  + Trouver la classe d’un héros.
* Tests unitaires :
  + Pour chacune des méthodes ci-dessus, faites une méthode qui appelle votre fonction que vous venez de développer et affiche les informations dans la console.

## Évaluation du travail de vos collègues :

Dans le but d’être juste avec tous vos collègues, vous devrez évaluer la performance de vos collègues dans ce travail. Vous devrez décider, par consensus, la distribution du résultat du travail sachant que vous avez 300 % à distribuer entre trois étudiants.

1. Vous allez donc assigner un pourcentage de participation au travail à chaque membre de l’équipe. Par exemple, si les trois membres de l’équipe (Alice, Bob, Claude) ont travaillé équitablement, alors Alice aura 100 %, Bob 100 % et Claude 100 %. Par contre, si les trois membres de l’équipe (Alice, Bob, Claude) n’ont pas travaillé équitablement, alors on pourrait avoir Alice 100 %, Bob 130 % et Claude 70 %. Ce pourcentage servira à calculer la note associée à chaque étudiant de telle sorte que la note sera multipliée par ce facteur (Note du travail \* facteur). Ce pourcentage devra être décidé par consensus en équipe.
2. Si vous n’arrivez pas à obtenir un consensus, j’utiliserai un ratio entre le nombre de lignes de code utile développé (pas de commentaire, de vide ou de changement de ligne) par une personne divisée par le nombre total de lignes de code utile.

Dans tous les cas, vous devez me remettre un document (avec une page de présentation) qui précise la distribution et explique votre choix.

## Devis à remettre :

Ce travail devra être remis électroniquement au plus tard lors du cours de la semaine #5 à **23h59**.

Ce travail se fait en équipe maximum de trois(3) personnes. Dans le cadre du travail pratique #1, il vous est demandé de :

1. Le code source de votre application sur le serveur GIT d’Azure DevOp ou GitHub Classroom.
2. La pondération associée à chacun des étudiants (Document Word avec une page de présentation et une très brève explication de votre choix).

# Voici la pondération :

1. *Création correcte du modèle (1* ***point****)* ;
2. *Application correcte des méthodes CRUD (12* ***points****)* ;
3. *Utilisation et création des procédures stockées et d’une requête SQL (2* ***points****);*
4. *Bonus (2* ***points****)*

Les critères d’évaluation sont :

- la complétude;

- la qualité fonctionnelle (les fonctions livrées donnent les bonnes réponses et sont correctes);