Rapport Projet Final

Quelques consignes sur le jeu avant de débuter :

CTRL + E : termine votre tour

Cliquer sur le bouton Play pour créer un compte si vous en avez pas

Comment le jeu atteint chaque exigence :

1. La classe CombatHandler.cs contient un enum avec au moins 3 éléments qui s’appelle Entity\_Types. Celui-ci représente les types d’entity qu’il peut y avoir dans le jeu.
2. Le jeu contient des dizaines et des dizaines d’instructions d’itérations. Suffit d’ouvrir une classe et il risque d’y en avoir pleins. Exemple : CombatHandler.cs
3. Dans la classe CombatHandler.cs, dans la méthode GenerateRandomPosition, on crée un tableau de coordonnées à retourner.
4. Le constructeur de la classe Entity.cs contient plusieurs arguments nommés.
5. Le constructeur de la classe Entity.cs contient plusieurs arguments facultatifs.
6. Toutes les classes de la hiérarchie du UI dans le package Interfaces.Utils contiennent des constructeurs surchargés
7. Héritage
	1. Entity.cs est une classe abstraite car nous ne pouvons pas avoir seulement un entity, il doit être obligatoirement un Player ou un NPC
	2. Sprite dérive de ISprite, Button dérive de Sprite, InputButton dérive de Button, Tile dérive aussi de Button, LabelSprite dérive aussi de Sprite, etc.
	3. Les méthodes Draw et Update des classes dérivées sont tous en override avec celles des classes de base.
	4. Dans CombatHandler.cs, la liste d’entity (Lst\_Entities\_incombat) est une liste d’entity, mais celle-ci contient des Player et des NPC. Lorsqu’on fait des manipulations sur la liste, on utilise le polymorphisme pour avoir accès aux membres supplémentaires de Player et NPC. Exemple : CombatHandler.cs à la ligne 206 dans la méthode UpdateCombat.
8. La classe Button.cs contient un délégué et un événement.
9. La classe Interface.cs contient un dictionnaire de combinaire (string, Button) ainsi qu’un dictionnaire de combinaison (string, InputButton) et (string, ISprite). Dic\_Buttons, Dic\_Inputbuttons et Dic\_Sprites.
10. Les configurations du serveur sont sauvegardées dans config.json, ainsi que les comptes créés dans Data.PlayerFiles.NOM\_DU\_JOUEUR.json
11. Le joueur peut utiliser le clavier pour se créer un compte dans les inputs ainsi que le clavier pour chatter avec les autres joueurs
12. Le joueur doit utiliser sa souris pour se déplacer, attaquer, sélectionner, etc.
13. Le jeu n’est pas terminé, mais même avec seulement un combat comme présenté, il y a facilement une minute de jeu. On doit se créer un compte, se connecter, se déplacer, battre toutes les slimes ainsi que le boss.
14. Le projet contient plus de 50 textures différentes. Allez faire un tour dans le dossier Content.
15. Le joueur ainsi que les monstres ne peuvent pas aller sur une case où il contient déjà une entité
16. Dans CombatHandler.cs, on peut constater qu’il y a un timer pour les tours des joueurs ainsi que pour les tours des monstres.

Hiérarchie avec diagramme :



Design du jeu :

Le jeu a été produit avec exemple Dofus. Cela veut donc dire un MMORPG en tour par tour basé sur la stratégie, le jeu d’équipe, le combat, etc. Évidemment le temps donné pour effectuer le projet était assez restreint alors la partie équipe et la partie MMO n’ont pas été implémentées dans le projet encore. Le but du jeu est de se déplacer en se créant une stratégie pour vaincre les monstres sur le terrain quadriller présenté lorsqu’on se connecte en tour par tour. Vous pouvez cliquer sur une case avec votre souris pour vous déplacer ou cliquer sur un monstre à une distance d’un maximum de 2 pour l’attaquer.

Votre dégât, dégât reçu ainsi que quelques autres messages apparaîtront dans le chat en bas à gauche de votre écran de jeu. Vous pouvez aussi discuter avec vous-même, pour l’instant, dans le chat.



Il va y avoir une liste de vous et tous les monstres à gauche de l’écran de jeu, il s’agit de l’ordre de passage des tours.



Vous allez aussi avoir une étoile qui représente le nombre de points d’action que vous avez. Chaque attaque coûte 2 points d’action.



Vous allez avoir une chaussure qui représente le nombre de points de mouvement que vous avez. Une case équivaut à un point de mouvement.

Finalement, vous allez avoir un cœur qui représente le nombre de vie que vous avez.

Difficultés rencontrées lors du projet et mes solutions :

1. L’algorithme qui a été très périlleux pour moi durant ce projet est l’algorithme pour les tours de jeu des monstres. Un tour de monstre s’agit de : bouger tant qu’il lui reste des points de mouvements, ainsi que d’attaquer le joueur si celui-ci est dans le range d’attaque du monstre jusqu’à temps que le monstre n’ait plus de points d’action. Les tours doivent se faire automatiquement avec une intelligence puisque ceux-ci sont contrôler par l’ordinateur. Je n’avais jamais fait ça auparavant, alors beaucoup de problèmes ont été rencontrés. Premièrement, le premier problème que j’ai rencontré est que les collisions ne fonctionnaient pas puisque l’ordinateur change directement la valeur des X et des Y sans regarder s’il y a déjà une entité présente sur la case. Ceci était un gros problème puisque les monstres s’empilaient entre eux. J’ai dû ajouter une association d’une entité à une case. En d’autres mots, j’ai dû créer une relation bidirectionnelle entre les cases (Tile.cs) et les entités (Entity.cs). Donc maintenant, l’ordinateur prévisualise le mouvement et regarde si la case contient déjà une entité, si oui, il n’effectue pas ce mouvement. Premier problème résolu. Le deuxième problème que j’ai rencontré est que l’update du des entités pour leurs sprites se faisait seulement lorsque le joueur cliquait sur une case, alors lorsque les monstres jouaient, leur tour de jeu n’affichait que lorsque c’était le tour du joueur, ce qui n’est vraiment pas bon et optimale. Alors j’ai dû créer une méthode qui update toutes les sprites des entités que je peux appeler lorsque l’ordinateur effectue un tour de jeu ( LoadCombatEntities(); ) dans CombatInterface.cs
2. Un deuxième algorithme que j’ai dû programmer est OptimizeText. OptimizeText est une fonction static dans la classe Misc.cs qui met en majuscule la première lettre d’un mot. Cette fonction m’a été très utile pour le nom du joueur. Il est très important de mettre une majuscule en début de phrase ou de mettre le nom d’un joueur avec sa première lettre en majuscule car cela fait très professionnel. Quand les utilisateurs se crée un compte, il ne mette pas nécessairement leur première lettre en majuscule, alors nous devons le faire pour eux. J’ai aussi eu un problème avec cet algorithme puisque les lettres sont dans une table qui s’appelle la table ASCII. Il est plus difficile qu’on le pense de mettre une lettre en majuscule puisque par exemple a = 65 mais A = 85. On aurait tendance à faire + 20, mais cela ne fonctionne pas pour toutes les lettres. Alors après un certain temps de tests et de debug, j’ai trouvé une solution logique pour cet algorithme. Il suffit que d’enlever 0x20 (en hexadécimal) transformé en caractère à notre lettre minuscule pour l’obtenir en majuscule. Comme ceci : valeur\_lettre -= (char) 0x20;
3. Une des difficultés que j’ai rencontrées lié à la langue de programmation est les délégués et les évènements. Je n’avais jamais vraiment fait d’évènements et de délégués dans ce langage et ils sont très différents dans les autres langages. Au début, je n’avais aucune idée qu’un délégué n’était en fait qu’une variable mais qui store la référence vers une fonction ou méthode. Évidemment, puisque je n’avais aucune idée à quoi il servait, je n’arrivais pas à faire fonctionner mes évènements de boutons (Button.cs). J’ai donc posé des questions en classe afin d’avoir une clarification sur les délégués et les évènements. Après une deuxième explication qui était très différente de la première, j’ai compris rapidement à quoi ils servaient et comment les utiliser. J’ai réussi très rapidement après à effectuer mon évènement de bouton.

Sources utilisées pour le projet :

<https://stackoverflow.com/>

<https://docs.microsoft.com/en-us/dotnet/standard>

<https://www.youtube.com/>

<https://msdn.microsoft.com/en-us/library/>

Ma soeur (pour tout ce qui est design graphique)