

Dossier ISN

Meragon - La forêt interdite



Gaëtan BESSOT TS2 ●

Brice HEILMANN TS2

I – Présentation

Notre projet, Meragon, est un jeu vidéo dit RPG, « role playing game », ou jeu de rôle en français. En effet, le joueur incarnera un personnage qui évoluera à travers les niveaux en suivant une histoire définie par les développeurs du jeu. Les jeux-vidéos RPG sont très populaires comme le montre le succès de certains titres comme Pokémon, Final Fantasy ou Zelda.

Le personnage principal, Arken, est un guerrier parti à la recherche d'une relique ancienne : le Codex Pragmazoïrien. En effet, ce dernier a été dérobé par d'affreuses créatures, les orcs, lors de l'invasion de sa terre natale. Il devra traverser de multiples dangers pour enfin affronter le chef de guerre de cette terrible armée. Dans notre jeu, nous suivons une partie du voyage d'Arken : la traversée de la forêt interdite. Un lieu plein de danger où notre héros ne manquera pas de se battre pour sauver sa vie.

Pour nous, passionnés de fantastique et de jeux-vidéos, le projet demandé en classe d'ISN nous est apparu comme une opportunité de lier nos centres d'intérêts avec nos compétences nouvellement acquises. Nous avons décidé de créer un RPG car c'est ce type de jeu qui répondait le mieux à nos envies.

II – Analyse des besoins

Dès le début de nos recherches, nous avons analysé un jeu populaire pour en comprendre les mécanismes afin de les reproduire dans notre projet. Nous prendrons donc en exemple le jeu-vidéo Pokémon.



Figure 1 : Capture d'écran du jeu Pokémon Rubis.

Comme on peut le voir sur cette image, le personnage évolue dans un monde en deux dimensions. Il peut se déplacer horizontalement et verticalement. Son environnement est composé de cases possédant une texture. Nous avons donc décidé de faire évoluer le personnage dans un tableau.

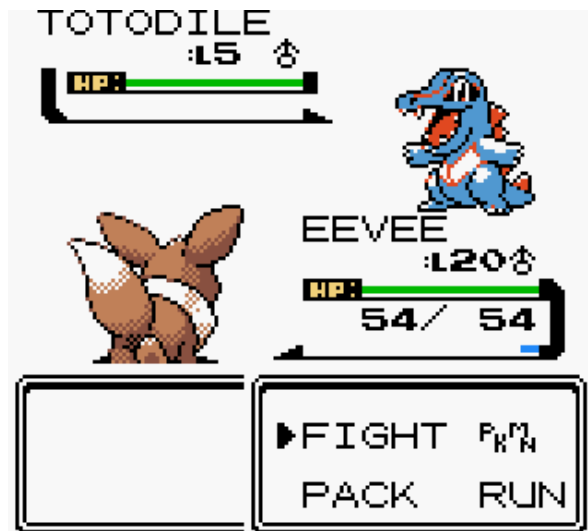


Figure 2 : Un combat dans Pokémon Bleu.

Ici, nous voyons une scène de combat. Nous ne voyons plus à l'image le personnage et son environnement, mais les acteurs du combat. On peut voir une image les représentant ainsi que leurs points de vie. Le joueur doit alors effectuer un choix afin d'orienter sa stratégie de combat. Ce système est appelé « combat tour par tour » et c'est celui que nous avons utilisé dans notre jeu.

Nous allons donc faire progresser notre personnage dans un tableau de texture tout en déclenchant, sous certaines conditions, des scènes de combat.

III – Répartition des tâches

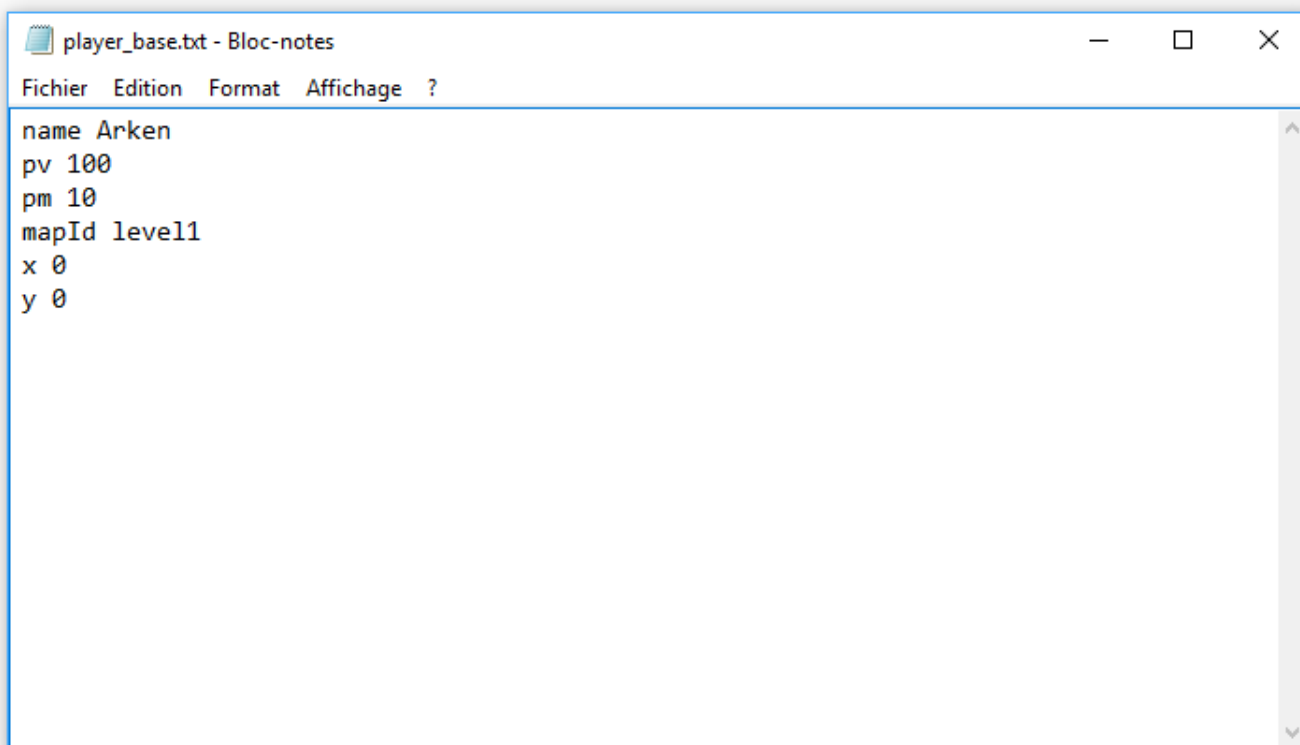
Une fois le cahier des charges terminé, nous nous sommes répartis les tâches. Brice était responsable de la création des niveaux de jeu, de l'environnement mais aussi des messages s'affichant à l'écran et de la musique de fond. Quant à moi, j'étais en charge du système de sauvegarde, de la gestion des caractéristiques du joueur et du système de combat. Nous nous réservions deux semaines pour la création de chaque fonctionnalité et nous avons pu tenir ce rythme tout au long du projet.

Travaillant chacun sur un fichier différent, nous nous sommes fréquemment échangés nos programmes afin de compléter nos versions respectives. De plus, certaines fonctions que je devais réaliser avait besoin de celle de mon partenaire pour fonctionner. Par exemple, le système de combat devait afficher des messages à l'écran. Nous nous retrouvions donc parfois pour faire le point sur l'avancement du jeu.

IV – Travail personnel

Au commencement de notre travail, je me suis directement penché sur la création du système de sauvegarde. Ce dernier est assez simple. Les informations du joueur sont contenues dans un tableau (voir figure 4). Lorsque le jeu se ferme, le tableau est copié dans un fichier texte, « player.txt » grâce à la fonction `player_setInfo()`. Cette dernière récupère pour colonne du tableau l'identifiant de la variable (première ligne) et sa valeur (deuxième ligne). Elle les écrit ensuite dans le fichier, séparées d'un espace, avant de revenir à la ligne.

Lorsqu'on lance le jeu pour la première fois, le programme fait appel à la fonction `player_getInfo()`. La fonction teste premièrement l'existence d'un fichier de sauvegarde « player.txt ». Si il n'existe pas, c'est que le joueur lance le jeu pour la première fois. Elle copie alors le fichier « player_base.txt » qui contient les informations initiales du personnage. Le programme ouvre ensuite le fichier « player.txt ». Pour chaque ligne, il récupère les deux informations séparées d'un espace (à l'aide de la fonction `split()`). Il place le nom de la variable dans la première ligne du tableau et sa valeur dans la seconde.



```
name Arken
pv 100
pm 10
mapId level1
x 0
y 0
```

Figure 3 : Le fichier `player_base.txt`.

Comme vous pouvez le voir, ce fichier contient bien les informations de notre personnage. Dans le programme, elles sont représentées dans le tableau `playerData`:

name	pV	pM	mapId	x	y
Arken	100	10	level1	0	0

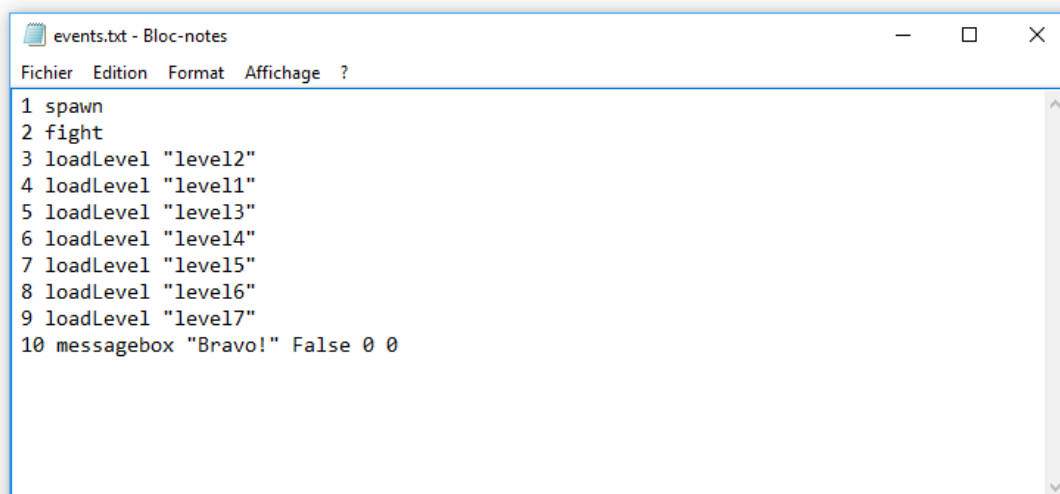
Figure 4 : Le tableau `playerData`.

Dès qu'une de ses caractéristiques doit changer dans le jeu (quand le personnage se blesse, quand il change de position, etc), on fait appel aux fonctions dites « setters » (de l'anglais « set », définir) propres à chaque caractéristique : `player_setName()`, `player_setPv()`, `player_setPm()`, `player_setMap()`, `player_setX()`, `player_setY()` qui prennent en entrée la nouvelle valeur de la caractéristique en question.

Le fonctionnement d'une setter est assez simple. Elle cherche dans la première ligne du tableau le nom de la variable à modifier, par exemple, « pv ». Une fois l'indice de cette dernière trouvée, elle se rend à la ligne suivante, à la même colonne, et remplace la valeur actuelle par l'argument donné. On appelle ensuite la fonction `player_setInfo()` qui sauvegarde le tableau dans le fichier de sauvegarde.

Mais lors d'un combat ou lors de l'affichage de la texture du personnage, le programme a besoin d'informations concernant le personnage : ses points de vie et d'attaque dans le cadre d'un combat, ou sa position dans le cadre de l'affichage. On utilise alors les fonctions dites « getters » (de l'anglais « get », obtenir). Encore une fois, les getters sont propres à chaque caractéristique : `player_getName()`, `player_getPv()`, `player_getPm()`, `player_getMap()`, `player_getX()`, `player_getY()`. Leur fonctionnement est assez proche de celui des setters. En effet, elles cherchent dans la première ligne du tableau `playerData` le nom de leur variable respective. Une fois son indice trouvé, elles se rendent à la seconde ligne, à la même colonne pour retourner la valeur désirée. Si c'est une valeur numérique, on procède à une conversion en entier avant de la retourner.

J'ai ensuite dû programmer un système d'événement. En effet, Brice avait terminé la programmation du système de niveau. Nous avons donc un tableau dont chaque cellule contenait les informations de la case (texture, accessibilité et événement). Lorsque le joueur avance sur la case, si un événement existe, il doit se lancer. Pour cela, j'ai programmé la fonction « `runEvent()` » qui prend en argument la troisième information de la case, correspondant à l'identifiant de l'événement. On ouvre alors le fichier « `events.txt` » qui contient pour chaque identifiant possible le nom de la fonction associée et les arguments potentiels. On lit le fichier. Pour chaque ligne, on sépare par l'espace l'identifiant, la fonction et les arguments. Si l'identifiant est celui passé en argument, on lance la fonction.



```
1 spawn
2 fight
3 loadLevel "level2"
4 loadLevel "level1"
5 loadLevel "level3"
6 loadLevel "level4"
7 loadLevel "level5"
8 loadLevel "level6"
9 loadLevel "level7"
10 messagebox "Bravo!" False 0 0
```

Figure 5 : Le fichier events.txt.

Parmi les quatre fonctions visibles ci-dessus, j'en ai programmé trois. Je vais détailler leur rôle :

- Spawn() symbolise le point d'apparition du personnage sur un niveau.
- Fight() lance un combat contre un monstre aléatoire.
- LoadLevel() téléporte le personnage sur un autre niveau.

LoadLevel() prend en argument le niveau à charger. Elle efface tout les éléments du canvas, détruit tout les éléments pouvant être à l'écran, puis recharge un niveau à partir du fichier texte en argument (par exemple, « level2.txt »). On affiche ensuite le niveau à l'écran et on place le personnage.

Passons maintenant à la plus grosse partie de mon travail, le système de combat. Lorsque le joueur marche dans les buissons, on choisit un chiffre entre 1 et 5. Si le résultat est inférieur ou égal à 2, le combat se lance. On met la variable global « pInFight » à True pour empêcher le joueur de se déplacer dans le niveau en plein combat. On efface ensuite tout les éléments à l'écran et on dessine les éléments nécessaires au combat en commençant par un fond vert foncé, pour symboliser la forêt, qui ne prend pas toute la hauteur de l'écran. En effet, on laisse un espace noir en bas pour placer les messages destinés au joueur et les choix qu'il peut prendre.

On récupère ensuite les informations sur le monstre. A l'image de playerData, on récupère d'un fichier texte les informations sur un monstre aléatoire. On ouvre le fichier « monsters.txt », on choisit une ligne au hasard et on récupère les points de vie, les points d'attaque, le nom et l'image du monstre. On trace ensuite une ellipse vert clair sur laquelle on place le monstre, puis un cadre gris. Dans ce cadre, nous plaçons le nom du monstre, puis une barre symbolisant ses points de vie.

On réitère le même processus pour placer le joueur sur la scène de combat.

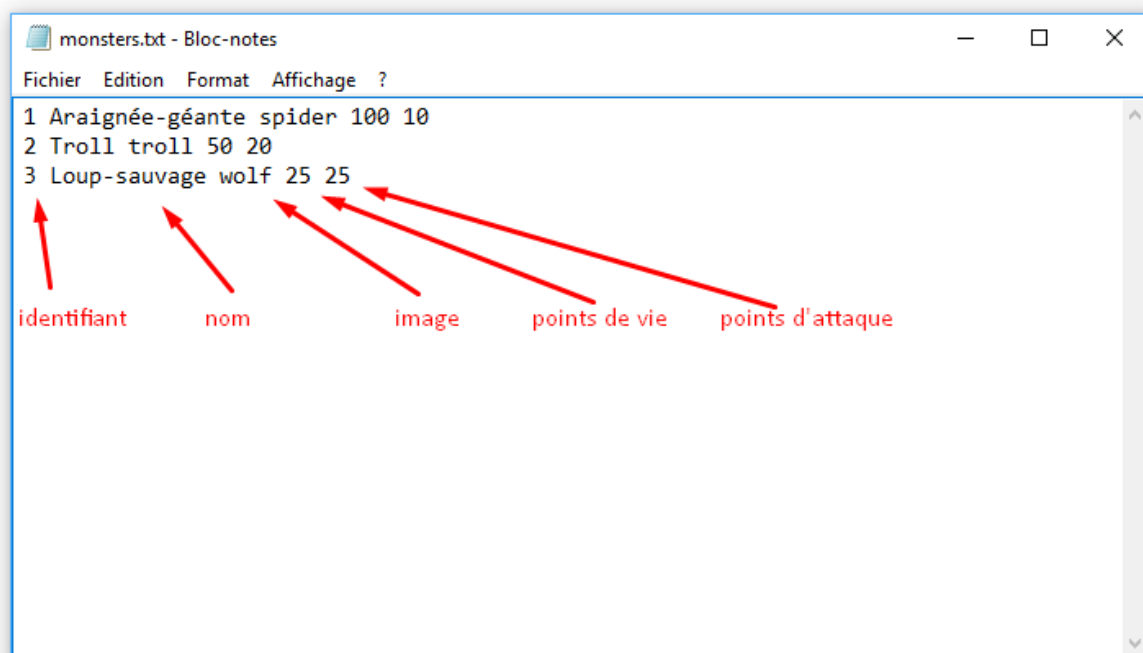


Figure 6 : Le fichier monsters.txt.

La boucle du combat commence alors. Un message s'affiche à l'écran : « Un monstre attaque ! Que faites-vous ? ». Le joueur est alors face à deux choix possibles : attaquer ou parer. On choisit ensuite un nombre entre 1 et 3 qui symbolise le choix stratégique du monstre.

Si le résultat est 1, le monstre attaque. Si le joueur n'a pas choisi de parer l'attaque, il subit l'attaque du monstre et perd autant de point de vie que le monstre n'a de point d'attaque. Le monstre, lui, subit l'attaque du joueur. Si le joueur pare, il ne se passe rien.

Si le résultat est 2, le monstre ne bouge pas. Si le joueur attaque, le monstre subit les dégâts, sinon, il ne se passe rien.

Si le résultat est 3, le monstre pare. Ainsi, que le joueur attaque ou non, il ne se passe rien.

Dans les deux cas où il peut se passer quelque chose, si l'un des deux combattants a perdu un tiers de sa vie, sa barre devient orange, si il en perd deux tiers, elle devient rouge. La barre change aussi de taille en fonction du nombre de point de vie restant.

A chaque tour, nous vérifions l'état du monstre et du joueur. Si les points de vie du joueur sont inférieurs ou égal à 0, il décède. On remet alors la variable « pInFight » à False, on détruit tout les éléments affichés à l'écran et on redémarre le jeu. La fonction « restartGame() » détruit la sauvegarde du joueur et le replace au début du jeu. Si les points de vie du monstre sont inférieurs ou égal à 0, le joueur gagne son combat. A nouveau, « pInFight » est remis à False, l'écran est effacé et l'on recharge le niveau actuel du joueur en le replaçant à ses coordonnées précédentes.

V – Projet final

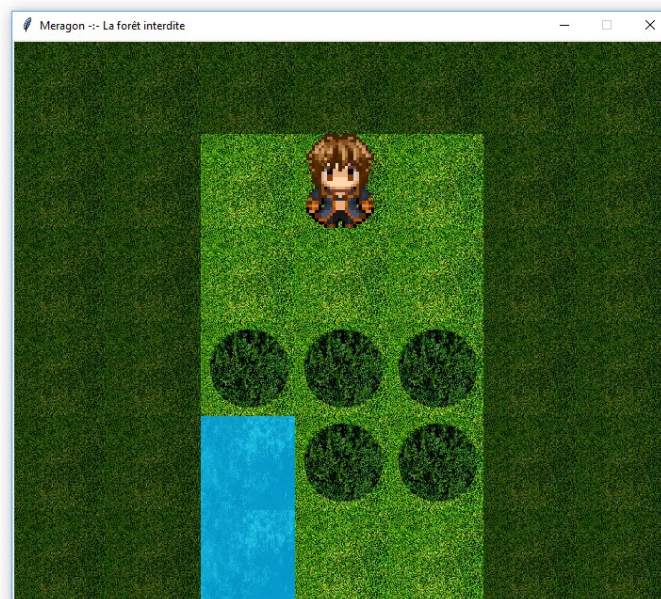


Figure 7 : Le personnage dans la forêt interdite.

Le jeu est lancé, un fichier de sauvegarde est bien créé et le personnage est placé sur la case contenant l'événement « spawn() ». On remarque que grâce au travail de mon partenaire, le personnage peut se déplacer dans le niveau à travers les cases que l'on peut voir.

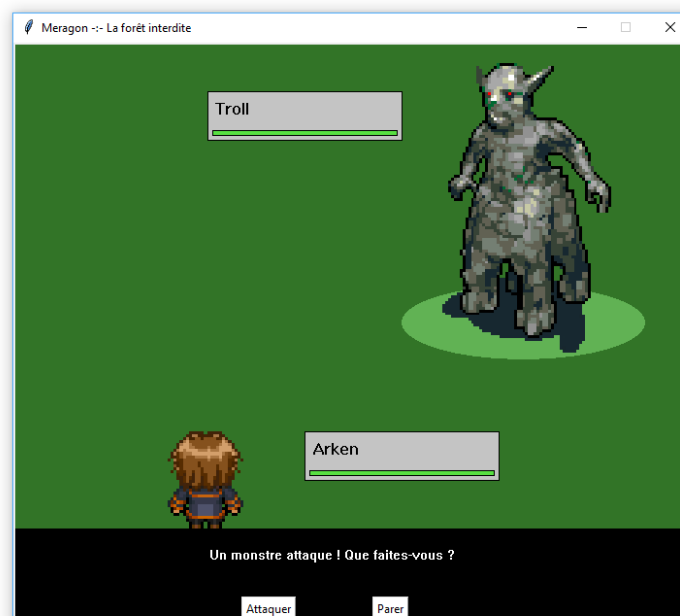


Figure 8 : Début de combat entre le héros et un monstre.

Lorsque l'on marche dans les buissons, parfois, un combat se lance. Nous nous retrouvons ici face à un Troll. Les éléments sont bien affichés à l'écran, le personnage ne peut plus bouger. Un choix est possible en bas de la fenêtre, comme prévu.

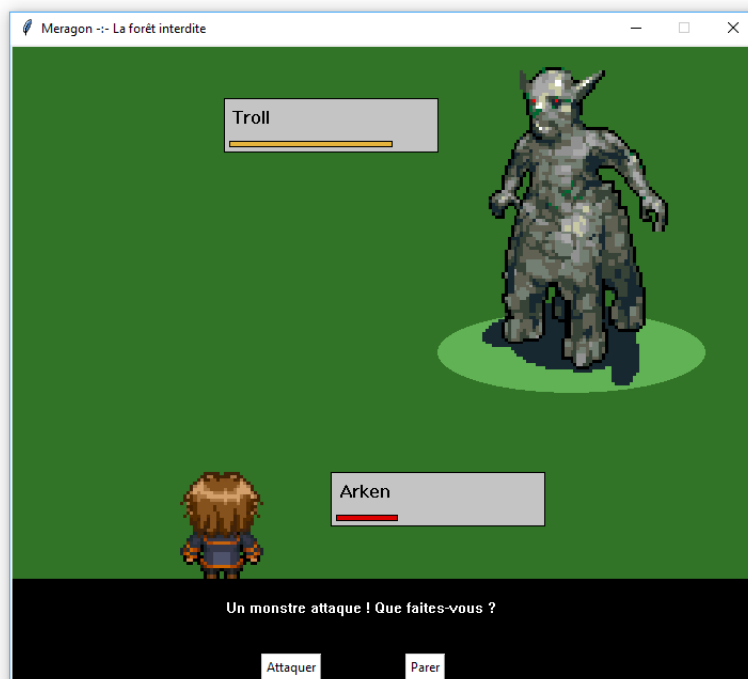


Figure 9 : Combat avancé entre le héros et un monstre.

Au fil du combat, le joueur et le monstre perdent des points de vie. On voit les barres associées à ses valeurs diminuer et passer du vert au orange puis au rouge. Lorsque le monstre meurt, le héros continue son aventure et si le héros meurt, il revient au point de départ.

Notre projet répond donc entièrement au cahier des charges.

VI – Bilan

Nous savions, dès le début de notre travail, que nous ne pourrions pas implémenter dans le jeu la totalité des fonctionnalités dont nous rêvions. En effet, un jeu vidéo de ce type est habituellement créé par une dizaine de développeur au moins et ce, pendant plusieurs années. Hélas, nous devons tenir compte des contraintes de l'épreuve d'ISN et terminer notre jeu dans le temps qui nous était donné. Voici une liste de quelques éléments que nous aurions aimé ajouter :

- Des objets utilisables par le joueur.
- Des personnages avec qui le joueur peut interagir.
- Des missions qu'il aurait pu réussir en échange de récompenses.
- Un système d'expérience qui rend le personnage de plus en plus fort.
- Un choix entre plusieurs types d'attaque pour approfondir l'aspect stratégique.

- Des graphismes adaptés au jeu et de meilleur qualité.
- Une plus grande diversité de niveau pour suivre toute l'aventure du personnage selon l'histoire de notre jeu.

Malgré cela, je suis satisfait de notre travail. Nous avons réussi à nous organiser et à travailler ensemble sur un même projet tout en respectant plusieurs dates d'échéances. Les grandes lignes du jeu vidéo RPG se retrouvent dans notre projet et cela nous rend très fier de nous. Aujourd'hui plus que jamais, je suis motivé pour continuer à programmer divers projets.