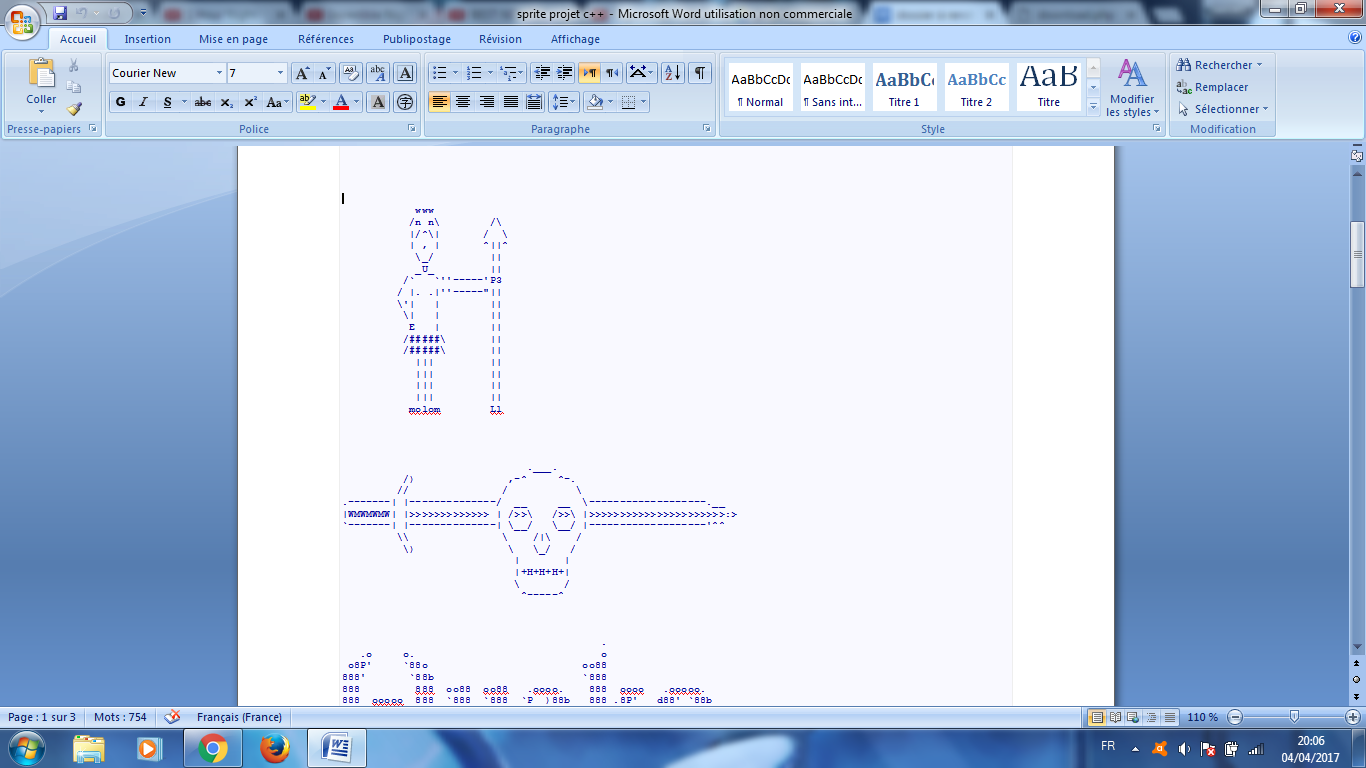
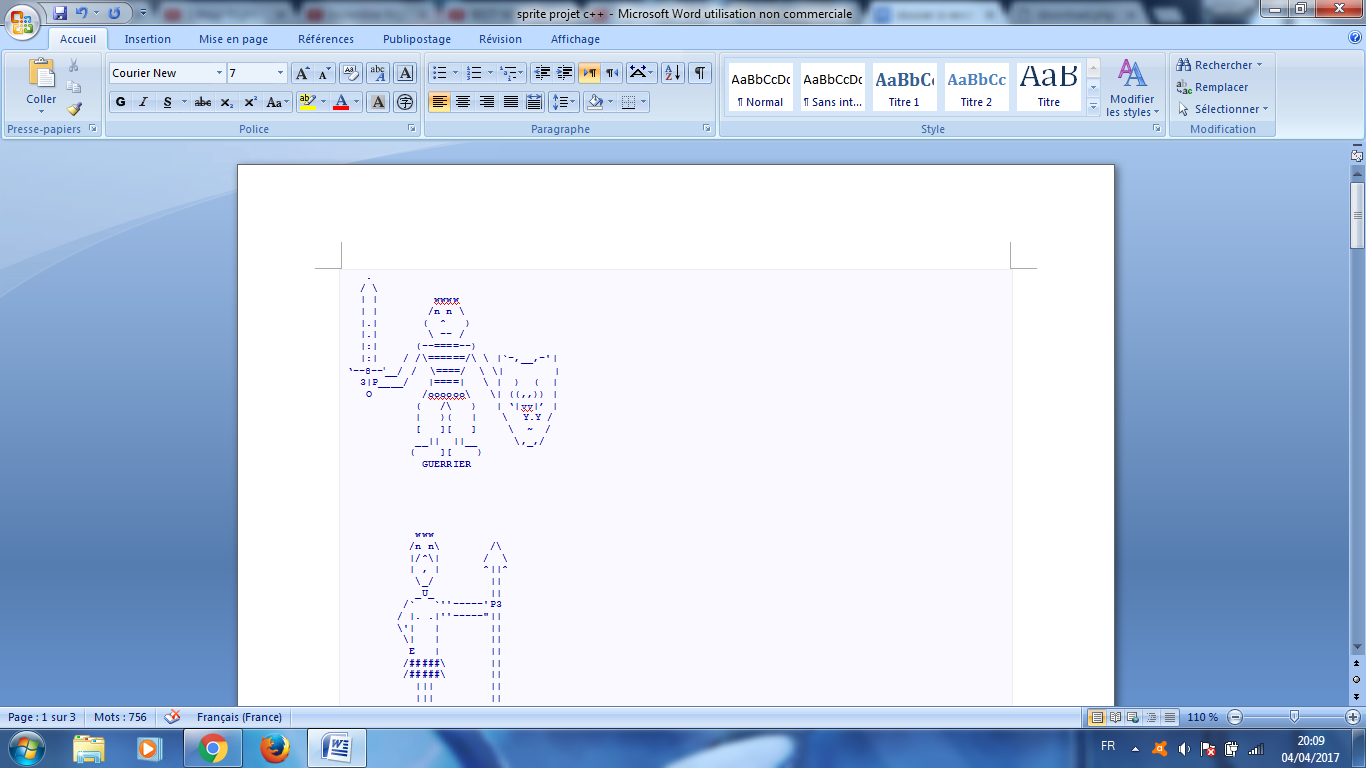
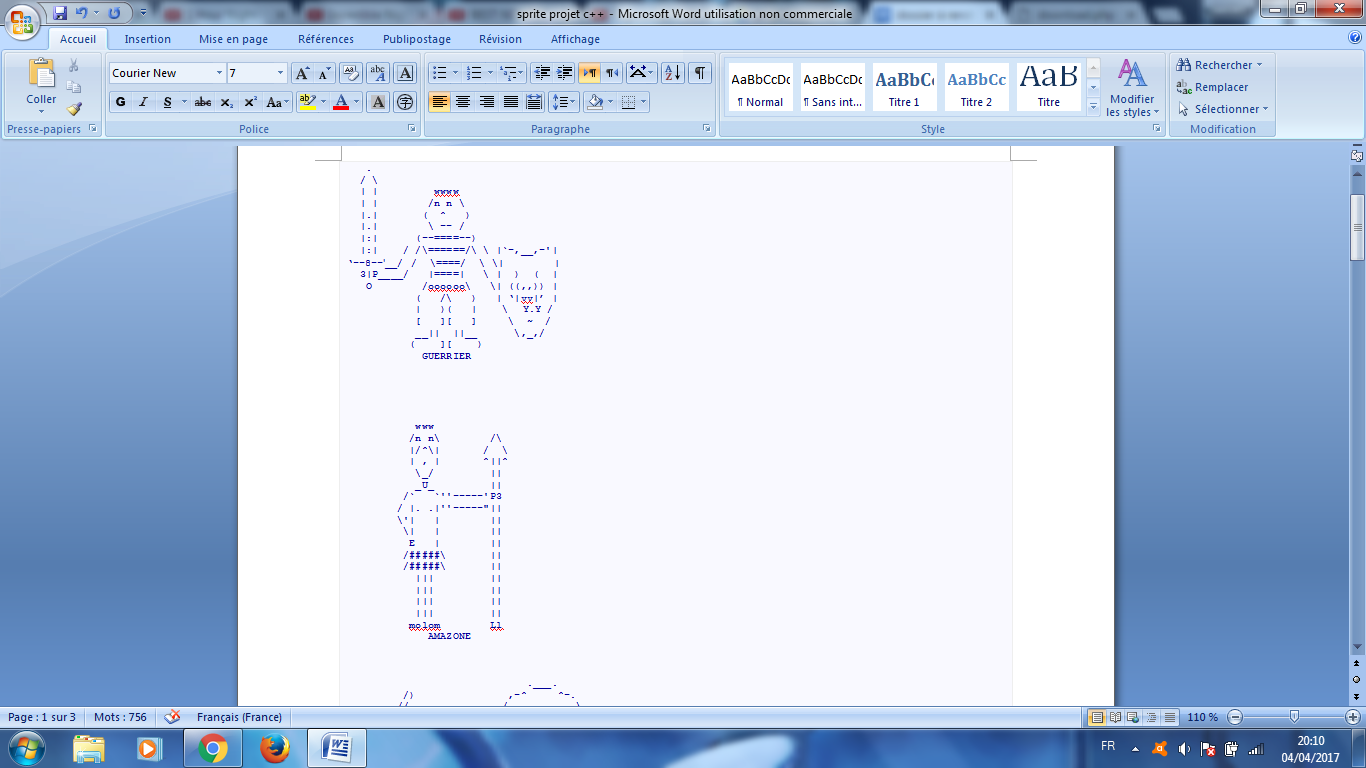
**DONJON RUSH**



SORCIÈRE et MOINE non fait.

**I) Introduction**

Pour le projet informatique en C++, nous devions réaliser un jeu de type dungeon crawler sur console avec comme demande :

* 4 personnages différents jouables : guerrier, moine, sorcière, amazone.
* Les personnages doivent avoir une barre de santé, avoir une caractéristique rendant aléatoire les combats, un sac de 4 objets et un nom.
* 3 ennemis : les personnages non joués.
* Lorsque 2 personnages sont sur la même case, ils doivent engager le combat.
* Une carte de 10x10.
* Chaque pièce doit montrer les objet présents dans la salle.
* Dans une salle il est possible d’utiliser un objet de son sac, poser dans la pièce des objets de son sac et prendre de la pièce des objets et les mettre dans son sac.
* Les types d’objets disponibles demandés sont : les poisons et les médicaments, les armes et les boucliers, les clés de téléportation.
* Faire un jeu tour par tour après un déplacement ou sur choix du joueur.
* Le jeu doit être fait au moins de façon textuelle.

Le projet s’est effectué après une série de TP variés qui permettaient de réaliser le travail demandé hors création/interaction des classes où seul un cours à était fait mais les exemples furent suffisant pour réaliser le travail demandé.

Il est dommage par contre de ne pas avoir vu la méthode pour pouvoir sauvegarder des données dans un fichier extérieur.

**II) Fonctionnalités**

Dans l’ensemble du programme, 3 parties se distinguent des autres de part leurs importances ou les difficultés rencontrées lors de leurs implémentations.

**a) La fonction affichage\_donjon**

La fonction pour afficher la carte du jeu (fichier fonction\_globale.cpp ligne 76) a pris un peu de temps à réaliser à cause de plusieurs détails :

* Savoir la bonne taille des boucles et éviter de faire des itérations qui bloqueraient le programme (le donjon étant de 10x10, il a fallu un peu de temps avant de trouver comment faire l’affichage des 5 items dans la pièce) et faire ainsi un switch/case adapté.
* Ensuite, pour valider le prérequis permettant de voir les objets (de façon plus précise qu’avec une lettre) présent dans la salle, rajouter un switch/case a été préféré à mettre un if dans le précédent switch/case (car les 300 itérations étaient déjà assez “longues” (réduite à 30 pour cette fonctionnalité)).
* La condition pour vérifier si le personnage est passé dans la pièce a permis d’éviter de voir la répartition des loot dès le début du jeu et ainsi augmenter le sentiment d’aléatoire.
* Enfin, un gros problème (qui touchait tous les affichages importants) fut gênant : utilisé cout faisait que les affichage n’étaient pas instantanés (très long pour cette fonction : environ 1 seconde avec un mouvement de montée => très peu agréable et on pouvait avoir du mal à se rappeler où l’on était) de ce fait il fallut modifier le format de la fonction : void=>string (avec le return associé), créé une variable dj pour stocker la très longue chaîne de caractères voulue, modifier les << en +, les cout << en dj +=, les endl en “\n” (car sinon ils étaient lus avant l’affichage de dj).

string affichage\_donjon( Piece \*playground[10][10] , Personnages \*ut )

{

string dj = "" ;

for ( int i = 0 ; i <= 30 ; i++ )

{

dj += spam\_space( 10 ) ;

for ( int j = 0 ; j < 10 ; j++ )

{

switch ( i )

{

case 0 :

case 3 :

case 6 :

case 9 :

case 12 :

case 15 :

case 18 :

case 21 :

case 24 :

case 27 :

case 30 :

if ( j == 0 )

{

if ( i == 0 )

{

dj += " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" ;

}

else

{

dj += "|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_" ;

}

}

break ;

case 1 :

case 4 :

case 7 :

case 10 :

case 13 :

case 16 :

case 19 :

case 22 :

case 25 :

case 28 :

if ( playground[i\*10/30][j]->passage\_perso\_utiliser == "oui" )

{

dj += "|" + playground[i\*10/30][j]->icon\_arme\_et\_armure + playground[i\*10/30][j]->icon\_accessoire + playground[i\*10/30][j]->icon\_potion ;

}

else

{

dj += "| " ;

}

break ;

case 2 :

case 5 :

case 8 :

case 11 :

case 14 :

case 17 :

case 20 :

case 23 :

case 26 :

case 29 :

if ( playground[i\*10/30][j]->passage\_perso\_utiliser == "oui" )

{

dj += "|" + playground[i\*10/30][j]->icon\_poison\_et\_tp\_et\_parchemin + playground[i\*10/30][j]->icon\_perso\_et\_piege\_actif + playground[i\*10/30][j]->icon\_piege ;

}

else

{

dj += "| " ;

}

break ;

}

}

if ( i != 0 )

{

dj += "|" + spam\_space( 6 ) ;

}

switch ( i )

{

case 1 :

dj += "objet dans la piece :" ;

break ;

case 5 :

if ( playground[ut->position\_y][ut->position\_x]->icon\_arme\_et\_armure != " " )

{

dj += playground[ut->position\_y][ut->position\_x]->nom\_arme\_et\_armure ;

}

break ;

case 9 :

if ( playground[ut->position\_y][ut->position\_x]->icon\_accessoire != " " )

{

dj += playground[ut->position\_y][ut->position\_x]->nom\_accessoire ;

}

break ;

case 13 :

if ( playground[ut->position\_y][ut->position\_x]->icon\_potion != " " )

{

dj += playground[ut->position\_y][ut->position\_x]->nom\_potion ;

}

break ;

case 17 :

if ( playground[ut->position\_y][ut->position\_x]->icon\_poison\_et\_tp\_et\_parchemin != " " )

{

dj += playground[ut->position\_y][ut->position\_x]->nom\_poison\_et\_tp\_et\_parchemin ;

}

break ;

case 21 :

if ( playground[ut->position\_y][ut->position\_x]->icon\_piege != " " )

{

dj += playground[ut->position\_y][ut->position\_x]->nom\_piege ;

}

break ;

}

dj+= "\n" ;

}

return dj ;

}

**b) Le constructeur Piece**

Le constructeur de chaque pièce (fichier donjon.cpp) prenant 430 lignes nous n’en mettons qu’une petite partie (variable + cas des pièges).

Ce constructeur est le plus important de tout ce qui fut codé car il a permi de comprendre comment faire plusieur objets dans une seule salle (et donc comment afficher la carte), il possède plusieurs variable :

* Un semblant de rng réaliste (malgré que dans les faits, il arrive souvent que plusieurs salles proches possèdent le même objet (même arme/armure/consommable/etc...)) pour chance\_loot\_X\_slot et pour choisir un par un l’objet à créer, un deuxième randint est utilisé avec type\_X\_loot.
* Les variables icon\_X et nom\_X sont utilisés pour l’affichage de la carte et de la partie objet dans la salle de l’inventaire, c’est pour cela qu’ils sont respectivement initialisés “ et “.
* Le fait de rajouter une variable numéro\_objet\_X\_slot a permi de faire un switch/case dans la fonction de ramassage d’objet pour la création des objets (et ainsi éviter une chaîne de if/else if avec comme test une chaine de caractères (traitement long avec beaucoup d’items lorsque l’on tombe sur les derniers de la liste)).
* La variable espace\_X\_slot permet de gérer les cas de figure où l’on veut ramasser un objet qui n’existe pas ou que l’on veut poser un objet dans la pièce (et ainsi éviter de modifier un objet au sol).
* Le booléen pour vérifier si la pièce du coffre est présente dans la salle fait partie des variables mais aurait très bien pu être fait indépendamment de la pièce (car seulement des coordonnées sont testées dans le main), il a était fait juste par envie.

Ensuite, lorsque l’on a la chance qui est dans l’encadrement, on voit des case avec des { }, cela est dû au moment où à la place de modifier des variables, l’on créé l’objet résultant (sans cela on résultait à une erreur lors de la compilation), ce fut remplacé par des variables (malgré la complication des modifications induites par cette méthode) à cause du transfert de l’objet du sol au sac : bug corrigeable mais après réflexion, du fait qu’il aurait fallu créer 500 objets de type Equipement, il fut préférable de minimiser la demande en mémoire en ajoutant des variables (actuellement seulement 13 objets de type Equipement sont créés au maximum (pour l’Amazone)).

Piece::Piece()

{

chance\_loot\_premier\_slot = 1 + rand()%100 ;

chance\_loot\_second\_slot = 1 + rand()%100 ;

chance\_loot\_troisieme\_slot = 1 + rand()%100 ;

chance\_loot\_quatrieme\_slot = 1 + rand()%100 ;

chance\_loot\_sixieme\_slot = 1 + rand()%100 ;

type\_arme\_loot = rand()%16 ;

type\_armure\_loot = rand()%15 ;

type\_accessoire\_loot = rand()%5 ;

type\_piege\_loot = rand()%3 ;

type\_poison\_et\_tp\_loot = rand()%5 ;

type\_parchemin\_loot = rand()%4 ;

icon\_arme\_et\_armure = " " ;

icon\_accessoire = " " ;

icon\_potion = " " ;

icon\_piege = " " ;

icon\_poison\_et\_tp\_et\_parchemin = " " ;

icon\_perso\_et\_piege\_actif = " " ;

nom\_arme\_et\_armure = "" ;

nom\_accessoire = "" ;

nom\_potion = "" ;

nom\_poison\_et\_tp\_et\_parchemin = "" ;

nom\_piege = "" ;

nom\_piege\_actif = "" ;

passage\_perso\_utiliser = "non" ;

espace\_premier\_slot = "vide" ;

espace\_second\_slot = "vide" ;

espace\_troisieme\_slot = "vide" ;

espace\_quatrieme\_slot = "vide" ;

espace\_sixieme\_slot = "vide" ;

piece\_du\_coffre = false ;

numero\_objet\_premier\_slot = 0 ;

numero\_objet\_second\_slot = 0 ;

numero\_objet\_troisieme\_slot = 0 ;

numero\_objet\_quatrieme\_slot = 0 ;

numero\_objet\_sixieme\_slot = 0 ;

if ( chance\_loot\_sixieme\_slot >= 23 && chance\_loot\_sixieme\_slot <= 47 )

{

espace\_sixieme\_slot = "plein" ;

switch ( type\_piege\_loot )

{

case 0 :

{

numero\_objet\_sixieme\_slot = 1 ;

nom\_piege = "piege runique" ;

icon\_piege = "T" ;

break ;

}

case 1 :

{

numero\_objet\_sixieme\_slot = 2 ;

nom\_piege = "arbalette piegee" ;

icon\_piege = "T" ;

break ;

}

case 2 :

{

numero\_objet\_sixieme\_slot = 3 ;

nom\_piege = "piege incapacitant" ;

icon\_piege = "T" ;

break ;

}

}

}

} ;

**c) La fonction inventaire**

L’inventaire et son affichage (fichier Personnages.cpp ligne 1004) sont la pierre angulaire de la gestion des objets dans le programme car ils servent de hub pour les fonctions ramasser, poser, équiper, utiliser, voir ses caractéristiques et les caractéristiques des objets dans son inventaire.

Cette fonction ne fut pas tellement compliqué à développer car ce n’est qu’une succession de cout avec l’utilisation de 2 fonctions personnelles : spam\_endl( int n ) qui consistent à faire n fois l’action cout << endl et spam\_space( int m ) qui consiste à renvoyer une chaine de caractères comportant m espace.

\_getch() est utilisé dans tout le programme pour 2 principaux avantages : éviter les bugs dus à un input de mauvais type, mais aussi avoir un panel d’input beaucoup plus grand.

Puis le while permettant d’éviter que le menu se rafraîchisse inutilement quand le chat marche sur le clavier.

Et finalement, le switch/case qui permet d'accéder au sous menu correspondant avec comme condition que l’entrée dans le sous menu soit justifiée (avoir un objet dans son sac pour pouvoir équiper, avoir un espace libre dans son sac pour ramasser, etc...).

void Personnages::inventaire( Piece \*salle , Personnages \*perso\_fictif )

{

int action\_inventaire ;

do

{

spam\_endl( 10 ) ;

cout << spam\_space( 10 ) << "objet equiper :" ;

spam\_endl( 2 ) ;

cout << spam\_space( 10 ) << "armes :" << spam\_space( 46 ) << "utiliser : u" << endl ;

for ( int i = 0 ; i < this->slot\_armes ; i++ )

{

cout << spam\_space( 10 ) << this->stuff\_arme[i]->nom\_equipement << endl ;

}

cout << endl << spam\_space( 10 ) << "accessoires :" << spam\_space( 40 ) << "poser par terre : t" << endl ;

for ( int j = 0 ; j < this->slot\_accessoires ; j++ )

{

cout << spam\_space( 10 ) << this->stuff\_accessoire[j]->nom\_equipement << endl ;

}

cout << endl << spam\_space( 10 ) << "casque :" << spam\_space( 45 ) << "equiper : e" << endl ;

for ( int k = 0 ; k < this->slot\_casques ; k++ )

{

cout << spam\_space( 10 ) << this->stuff\_casque[k]->nom\_equipement << endl ;

}

cout << endl << spam\_space( 10 ) << "plastron :" << spam\_space( 43 ) << "ramasser : r" << endl ;

for ( int l = 0 ; l < this->slot\_plastron ; l++ )

{

cout << spam\_space( 10 ) << this->stuff\_plastron[l]->nom\_equipement << endl ;

}

cout << endl << spam\_space( 10 ) << "bottes :" << spam\_space( 45 ) << "statistiques du personnage : c" << endl ;

for ( int m = 0 ; m < this->slot\_bottes ; m++ )

{

cout << spam\_space( 10 ) << this->stuff\_botte[m]->nom\_equipement << endl ;

}

cout << endl << spam\_space( 10 ) << "jambiere :" << spam\_space( 43 ) << "statistiques objet : s" << endl ;

for ( int n = 0 ; n < this->slot\_jambieres ; n++ )

{

cout << spam\_space( 10 ) << this->stuff\_jambiere[n]->nom\_equipement << endl ;

}

cout << endl << spam\_space( 10 ) << "gantelets :" << spam\_space( 42 ) << "fermer inventaire : espace" << endl ;

for ( int o = 0 ; o < this->slot\_gantelets ; o++ )

{

cout << spam\_space( 10 ) << this->stuff\_gantelets[o]->nom\_equipement << endl ;

}

spam\_endl( 5 ) ;

cout << endl << spam\_space( 10 ) << "inventaire :" ;

spam\_endl( 2 ) ;

for ( int p = 0 ; p < this->slot\_objets ; p++ )

{

cout << spam\_space( 10 ) << this->sac[p]->nom\_equipement << endl ;

}

spam\_endl( 5 ) ;

cout << spam\_space( 10 ) + "objet dans la piece :" << endl << endl << spam\_space( 10 ) + salle->nom\_arme\_et\_armure ;

cout << endl << spam\_space( 10 ) + salle->nom\_accessoire << endl << spam\_space( 10 ) + salle->nom\_potion << endl ;

cout << spam\_space( 10 ) + salle->nom\_poison\_et\_tp\_et\_parchemin << endl << spam\_space( 10 ) + salle->nom\_piege ;

spam\_endl( 5 ) ;

action\_inventaire = \_getch() ;

while ( action\_inventaire != 32 && action\_inventaire != 99 && action\_inventaire != 101 && action\_inventaire != 114 && action\_inventaire != 115 && action\_inventaire != 116 && action\_inventaire != 117 )

{

action\_inventaire = \_getch() ;

}

if ( action\_inventaire == 99 )

{

this->caracteristiques\_perso() ;

}

else if ( action\_inventaire == 101 )

{

for ( int a = 0 ; a < this->slot\_objets ; a ++ )

{

if ( this->sac[a]->nom\_equipement != "vide" )

{

this->equiper( salle ) ;

break ;

}

}

}

else if ( action\_inventaire == 114 )

{

for ( int b = 0 ; b < this->slot\_objets ; b ++ )

{

if ( this->sac[b]->nom\_equipement == "vide" )

{

this->ramasser\_objet( salle ) ;

break ;

}

}

}

else if (action\_inventaire == 115 )

{

for ( int c = 0 ; c < this->slot\_objets ; c ++ )

{

if ( this->sac[c]->nom\_equipement != "vide" )

{

this->caracteristiques\_objet() ;

break ;

}

}

}

else if ( action\_inventaire == 116 )

{

for ( int d = 0 ; d < this->slot\_objets ; d ++ )

{

if ( this->sac[d]->nom\_equipement != "vide" )

{

this->poser\_objet( salle ) ;

break ;

}

}

}

else if ( action\_inventaire == 117 )

{

for ( int e = 0 ; e < this->slot\_objets ; e ++ )

{

if ( this->sac[e]->type\_equipement\_objet == "objet" )

{

this->utiliser\_objet( salle , perso\_fictif ) ;

break ;

}

}

}

}while ( action\_inventaire != 32 ) ;

}

On peut rajouter une mention spéciale pour la condition tampon des 4 premiers sous menus de la fonction inventaire qui permettent d’avoir un blocage parfait : si on utilise pas le bon input ou que l’input réalisé mène sur une action irréalisable, alors on doit refaire sa commande (chose très appréciable pour un jeu sur la console car cela minimise les rafraîchissements de la console et donc les effets dérangeants lors d’un gameplay).

Ce type de while aurait pu être étendu à tous les while le nécessitant (actuellement présent uniquement sur les sous menu de l'inventaire) mais sa longueur impose de créer des sous variables avec un nom très court (action\_inventaire => a et (varie selon la fonction) espace\_X\_slot\_X => e\_X) pour que le tout garde un minimum de lisibilité.

a = \_getch() ;

while ( ( ( a != 32 && a != 49 && a != 50 && a != 51 && a != 52 && a != 53 && a != 54 ) || ( a == 49 && e\_1 == "vide" ) || ( a == 50 && e\_2 == "vide" ) || ( a == 51 && e\_3 == "vide" ) || ( a == 52 && e\_4 == "vide" ) || ( a == 53 && e\_5 == "vide" ) || ( a == 54 && e\_6 == "vide" ) ) )

{

a = \_getch() ;

}

**III) Déroulement du jeu :**

Lorsque l’on lance le jeu on a le choix entre commencer une partie, charger une partie sauvegardée (non implémenté) ou lire le tutoriel en utilisant les nombres de 1 à 3.

Si l’on choisi de lire le tutoriel alors cela permet de comprendre quelques fonctionnalités du jeu pour un nouvel arrivant en utilisant les nombres de 4 ou 6.

Si l’on choisi de commencer une nouvelle partie, alors on a le choix entre les 4 personnages jouables en utilisant les nombres de 1 à 4.

Lorsque l’on a choisi son personnage alors on doit choisir le mode de difficulté souhaité entre (trop) facile (vagabond), moyen (aventurier) et difficile (chercheur de trésor confirmer) en utilisant les nombres de 1 à 3.

Après avoir choisi sa difficulté de jeu alors on se retrouve sur la carte du jeu dans la salle en haut à gauche du donjon.

On a le choix entre se déplacer en utilisant les nombres 2/4/6/8, ouvrir son inventaire avec la touche i ou de passer son tour avec la touche p.

Si l’on choisit d’ouvrir son inventaire, alors le menu de l’inventaire remplace la carte du donjon et l’on a alors le choix entre ramasser un objet avec la touche r, poser un objet avec la touche t, équiper un objet avec la touche e, utiliser un objet de son sac avec la touche u, voir les caractéristiques des objets dans son l’inventaire avec la touche s, voir les caractéristiques actuelles de son personnages avec la touche c et quitter l’inventaire avec la touche espace.

Pour les 5 premiers sous menu on utilise les nombres de 1 à 6 pour faire l’action du sous menu choisi.Le 6ème sous menu est fixe.

Si lors d’un déplacement on tombe sur la case du coffre alors on gagne le jeu et l’on bascule sur l’écran de fin de jeu, ensuite le programme se ferme au prochain input.

Si lors d’un déplacement on tombe sur la case d’un ennemi alors le combat s’engage directement et l’on bascule sur l’écran de combat nous offrant 4 possibilités : faire une attaque normale, utiliser un sort, utiliser un objet ou fuir en utilisant les nombres de 1 à 4.

Faire une attaque normale ou un sort fait des dommages à l’ennemi et passe le tour pour que l’ennemi frappe.

Utiliser un objet ouvre l’inventaire de combat donnant la possibilité d’utiliser un objet ou non en utilisant les nombres de 1 à 6, ou avec la touche espace, quelque soit l’action choisie le tour est ensuite fini et l’ennemi attaque.

Fuir fait revenir sur la case où votre personnage se trouvait au tour précédent.

Si l’ennemi n’as plus de point de vie, alors vous retournez sur la carte du donjon.

Si votre personnage n’a plus de point de vie alors vous basculer sur l’écran de mort, ensuite le programme se ferme au prochain input.

Le gameplay choisi dépendant du choix de la classe choisie :

- Moine :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PV | Esquive | Chance | Force | Magie | Défense physique | Défense magique |
| 20 | 25 | 25 | 2 | 3 | 1 | 2 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Slot d’armes | Slot d’accessoires | Slot d’armures | | | | | Slot d’objets |
| Casques | Plastrons | Bottes | Jambières | Gantelets |
| 0 | 4 | 0 | 0 | 0 | 1 | 1 | 4 |

- Guerrier :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PV | Esquive | Chance | Force | Magie | Défense physique | Défense magique |
| 15 | 30 | 10 | 5 | 1 | 3 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Slot d’armes | Slot d’accessoires | Slot d’armures | | | | | Slot d’objets |
| Casques | Plastrons | Bottes | Jambières | Gantelets |
| 2 | 0 | 1 | 1 | 1 | 1 | 1 | 3 |

- Sorcière :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PV | Esquive | Chance | Force | Magie | Défense physique | Défense magique |
| 10 | 15 | 10 | 1 | 5 | 0 | 3 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Slot d’armes | Slot d’accessoires | Slot d’armures | | | | | Slot d’objets |
| Casques | Plastrons | Bottes | Jambières | Gantelets |
| 2 | 1 | 1 | 1 | 1 | 0 | 0 | 4 |

- Amazone :

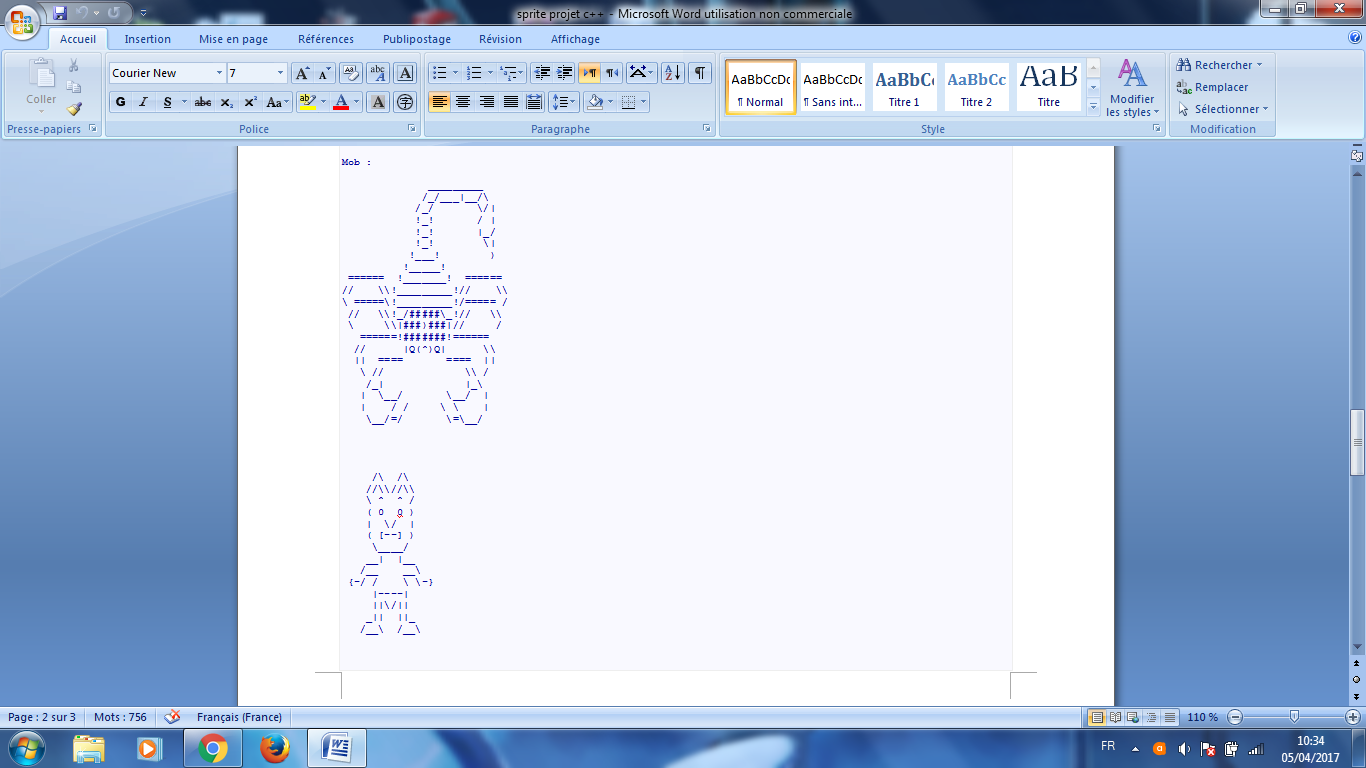
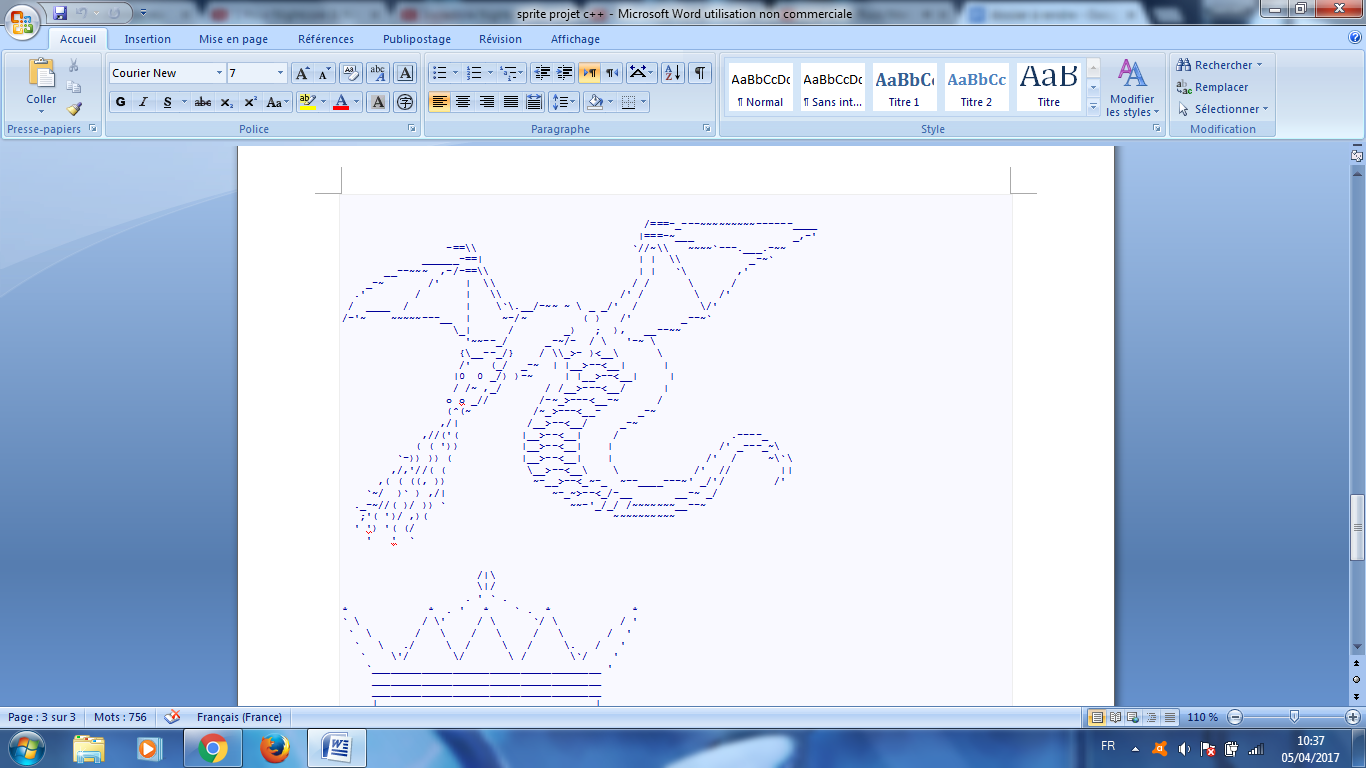
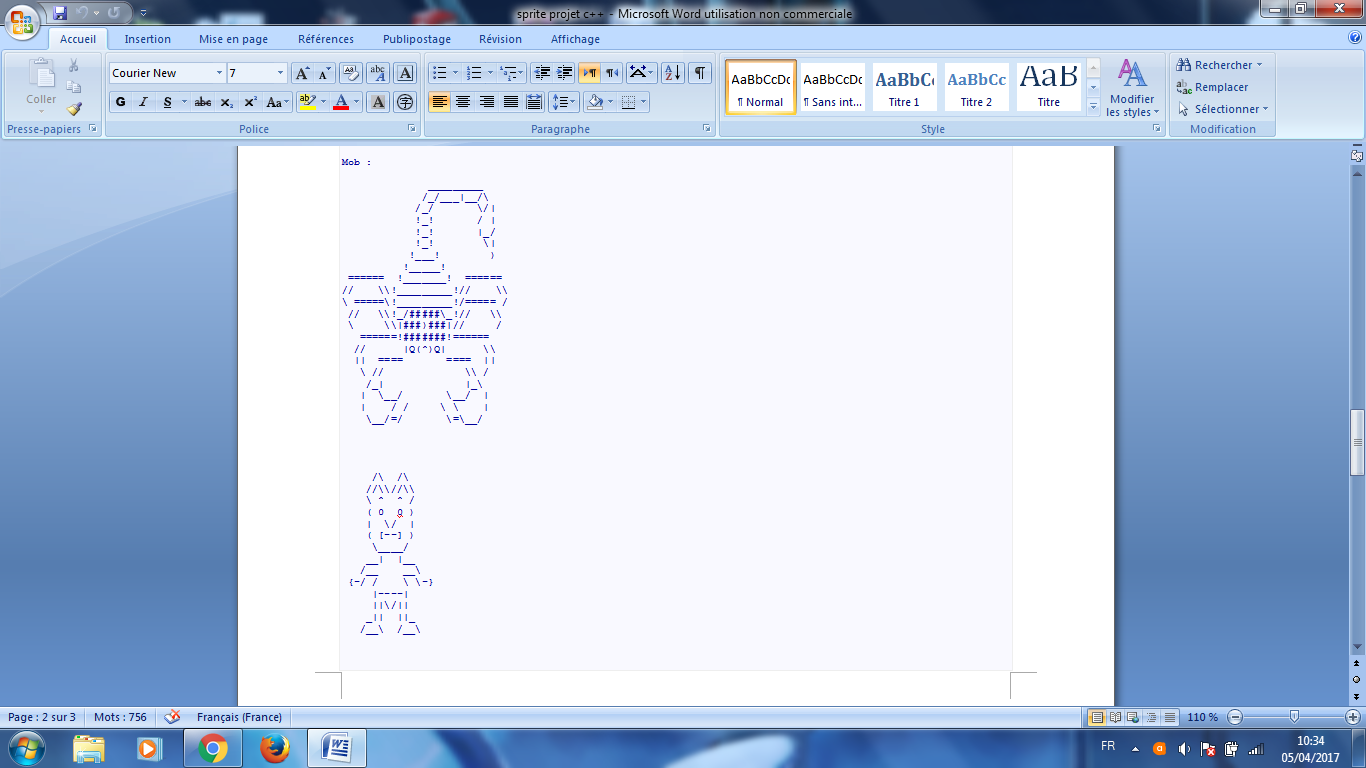
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PV | Esquive | Chance | Force | Magie | Défense physique | Défense magique |
| 12 | 35 | 15 | 4 | 2 | 2 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Slot d’armes | Slot d’accessoires | Slot d’armures | | | | | Slot d’objets |
| Casques | Plastrons | Bottes | Jambières | Gantelets |
| 1 | 3 | 0 | 0 | 1 | 1 | 1 | 6 |

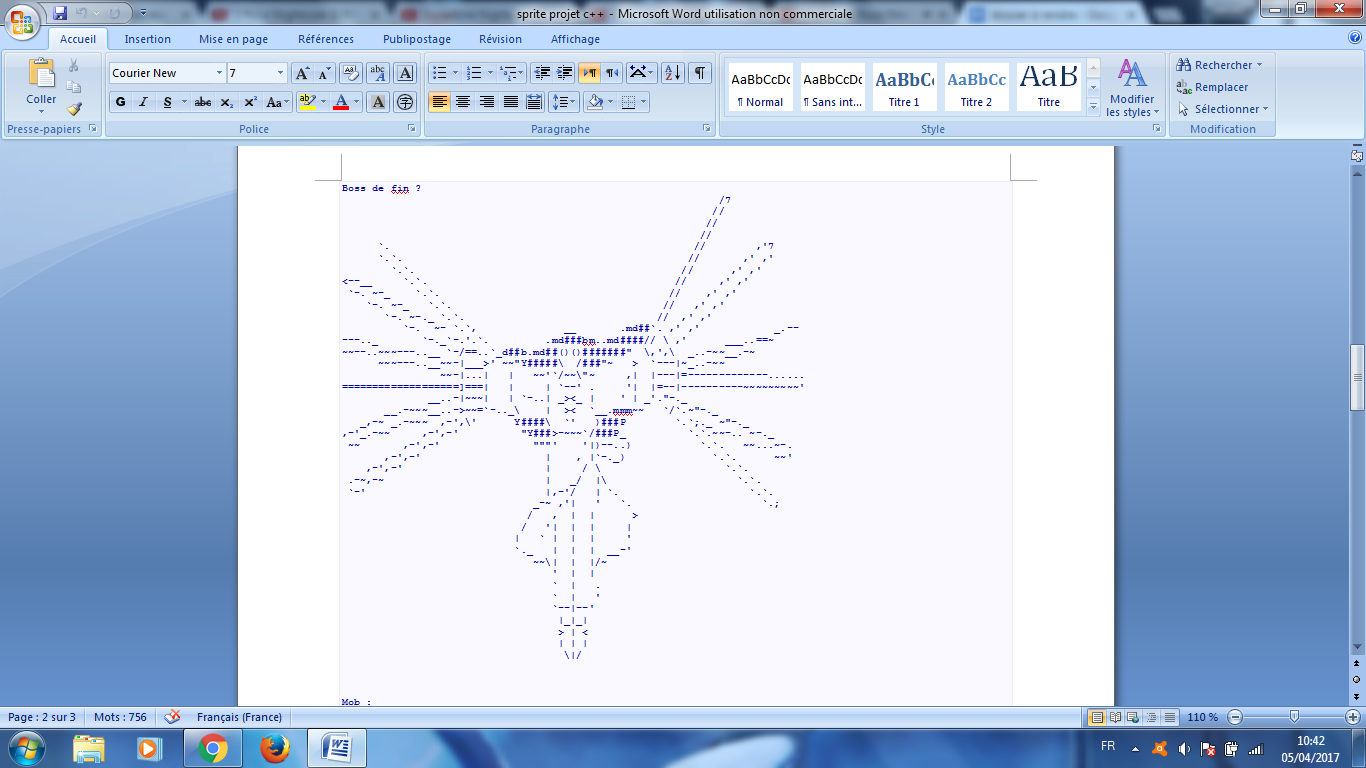
**IV) Améliorations possibles**

Le programme propose actuellement suffisamment de fonctionnalités pour être jouable mais il manque encore quelques bouts de programme comme les sprites de la sorcière et du moine, les altérations d’états avec le piège incapacitant (initialement prévu sous 3 types : paralysie, mutisme et saignement (dommage uniquement sur la carte) ), les divers médicaments pour enlever les altérations d’états, faire que l’enduit applique un buff de poison sur l’arme équipée et non avoir la même utilisation que la fiole, l’utilisation de la clé de téléportation, il manque aussi l’histoire lors du début et de la fin du jeu pour rendre le tout cohérent.

De plus pour rendre le jeu plus intéressant, nous pensions rajouter des monstres en partant sur une base de 3 : un faible, un qui empoisonne et un puissant.

Dans la même idée, pour éviter les victoires “accidentelle”, un gardien du coffre (boss de fin) fut prévu.



L'intérêt de mettre des monstres aurait permi de développer un système d’expérience, un système de sort différent (pas simplement utilisé la caractéristique de magie) en ajoutant des capacités propres à la classe utilisée comme un coup de bouclier pour assommé avec le guerrier, divers buffs pour le moine, un bouclier bloquant les dommages pour la sorcière et une détection de piège pour l’amazone (ceux ci n’auraient été que les sorts lv 1 des personnages, d’autres auraient été développé), avec bien sur une amélioration des statistiques par niveau, cela aurait aussi permi de mettre une chance pour fuir (car actuellement, en difficile, si l’on tombe sur un ennemi dès le 1° déplacement, on peut presque mourir en 1 seul coup suivant les conditions).

Ces deux systèmes posant d’autres questions : laisser les personnages non joueur en ennemi et ainsi leurs défaites entraînant un bonus sur le personnage joué, ou en faire des alliés lorsque l’on les trouvent (et ainsi faire un donjon avec plusieurs étages de taille différente (5x5 puis 7x7 puis 10x10 et enfin le couloir de fin en 5x1)) et ainsi créer d’autres objets pour éviter que l’aléatoire rende le jeu trop simple très tôt.

L’avantage d’avoir des personnages en plus dans son équipe aurait permi de rendre le jeu beaucoup plus punitif, faire un meilleur tour par tour, d'équilibrer plus facilement le jeu pour éviter les combats à sens unique en travaillant sur les synergie entre les personnages.

Ensuite à la place d’être contraint des désavantages de l’utilisation de la console pour réaliser une interface graphique, faire une fenêtre extérieure ne montrant que les sprites désirés (long mais réduisant au final le temps nécessaire pour coder certaines parties dans le cadre d’un jeu très complexe (à partir de 10+ arrière plan différent), gérer l’affichage de chacun d’eux peut prendre du temps quand on doit gérer des cas de nettoyage d’écran (endl massif) et des cas d’affichage de texte visible).

Une interface graphique aurait permis d’avoir une variété de sprite pour les personnages et ainsi d’avoir des combat plus animés (sans le saccadement obtenu avec la console) ainsi que d’avoir des sprites pour les armes dans le menu des caractéristiques des objets.

Avec toutes ces fonctionnalités, il aurait aussi fallu une fonction de sauvegarde car le jeu aurait pu durer entre 30 minutes et 1 heures .. voir plus.

**V) Conclusion :**

Le jeu respecte actuellement environ 85% des demandes initiales (dont beaucoup de contenus non demandés) et pourtant il est encore loin d’être légitime du nom de jeu de part ses défauts pour le genre choisit (peu de choix stratégique en combat, une variété faible d’équipement ainsi que peu d’ennemi n’est que peu adapté à un rogue like de type dungeon crawler), il aurait fallu pour cela s’investir beaucoup plus tôt dans le projet (fait 3 ou 4 semaines avant la fin du temps prévu : 50+ heures hors cours n’ont pas suffit (mauvaise estimation de notre part car nous nous attendions à ce que le jeu soit fini entre 2000 et 3000 lignes (en comparaison du projet python ayant environ le même temps pour être fait mais se terminant en 500- lignes))).