Livrable 1 du projet de combats types “RPG classique” en ligne

Sommaire :

Introduction :

Le concept du jeu est pour le moment une suite de combats. Les deux joueurs choisissent leur personnage puis commencent un combat tour par tour. Le choix du joueur qui commence pourrait s'effectuer au shi-fu-mi (pierre papier ciseaux). La partie se terminant lorsque les points de vie (PV) d'un joueur tombent à 0 (le vainqueur étant alors l'autre joueur) ou que l'un des joueur se déconnecte.

A chaque tour, on observera un gain de points d'actions (PA), et selon les actions magiques ou les objets utilisés, on pourra observer un gain ou une perte de PV ansi que de Mana (ressource utilisée pour la magie). Les attaques pourront être esquivées selon certaines probabilités et un système de coups critiques (faible chance de multiplication des dégâts) sera mis en place.

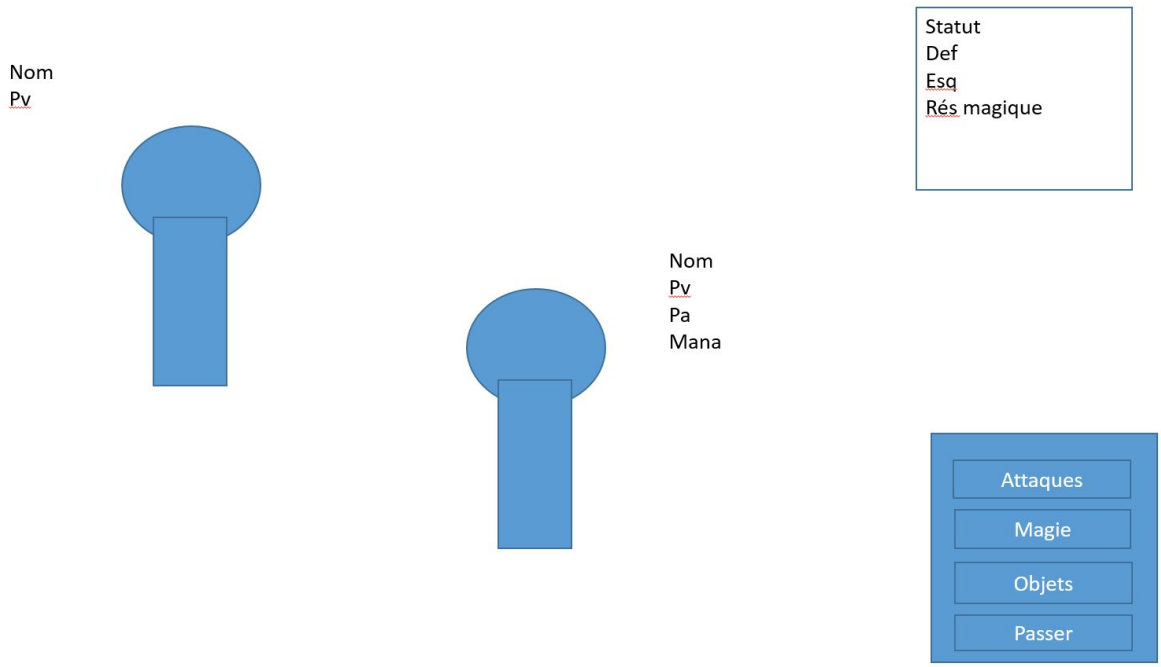
Un système de sort permettra de se régénérer de la vie et de mettre en place des stratégies sur le moyen terme avec un système de buffs/débuffs (effet respectivement bénéfique/néfaste qui modifie les caractéristiques d'un personnage pendant plusieurs tours) qui permettra par exemple d'infliger des dégâts à chaque tour pendant un certain temps ou de régénérer des PV pendant plusieurs tours. Des sorts pourront aussi par exemple consister à retourner une partie des dégâts infligés par l'adversaire. Ces sorts seront limités en utilisation par un coût en points de Mana.

Les objets pourront soit régénérer PV ou Mana, soit infliger des dégâts à l'adversaire, soit booster certaines caractéristiques des personnages. La limitation à l'utilisation des objets sera induite par un coût en PA pour leur utilisation et un nombre limité d'objets.

I) Règles du jeu

I.1) Les actions possibles lors d'un tour

Lorsque le combat débute, le joueur qui commence a un nombre d'actions possibles limité par ses points d'actions (PA) qui se régénèrent à chaque tour. Chaque personnage dispose aussi de points de mana (Mana) qui eux nécessitent l'utilisation de potions pour se régénérer.



Voici ce à quoi ressemblerait l'écran lors de la phase d'action du joueur. Les 4 catégories d'actions possibles sont les suivantes :

1. Attaques
2. Magie
3. Objets
4. Passer (qui sert à terminer son tour)

I.1.1) Attaques

Les attaques sont des actions qui ne nécessitent que des points d'actions pour être exécutées. Parmi elles ont pourra retrouver des attaques rapides (nécessitant peu de PA mais infligeant peu de PV) ou des attaques lourdes (nécessitant beaucoup de PA et infligeant de lourds dommages).

Pour déterminer les dégâts d'une attaque, sont pris en compte :

- Les dégâts de base de l'attaque.
- La défense de l'ennemi.
- Les bonus de dommages ou de défense.

I.1.2) Magies

Les magies sont des actions qui ont un coût en PA et en Mana. Celles-ci pourront être de différents types : feu (offensives) ou eau (défensives). On pourra retrouver parmi les sorts des sorts offensifs infligeant des dégâts sur un ou plusieurs tours, des sorts qui permettent de régénérer de la vie, ou même des sorts qui permettront de blesser l'adversaire lorsqu'il attaquera.

Pour déterminer les dégâts d'un Sort, sont pris en compte :

- Les dégâts de base du sort.
- La résistance magique de l'adversaire.
- Les bonus de dommages ou de résistance magique

I.1.3) Objets

Ici on peut retrouver plusieurs catégories d'objets : des potions de régénération ou qui apportent des bonus de dommage, de défense ou de résistance; mais aussi des objets qui peuvent infliger des dégâts à l'adversaire.

Les objets seront disponibles en nombre limité pendant chaque combat, mais utilisables de manière illimité pendant ce dernier : par exemple, un joueur disposant de 3 potions pourra les utiliser toutes les 3 pendant le même tour, cependant, elle disparaîtront définitivement de ses objets.

I.1.4) Coups critiques et esquives

A chaque action offensive d'un joueur, il y aura une probabilité qu'il fasse coup critique symbolisée par … et qui pourra être modifiée par des sorts ou des objets. Lors d'un coup critique, l'action du joueur infligera beaucoup plus de dégâts que ses dégâts de base.

De même, à chaque action offensive d'un joueur, il y aura une probabilité que l'adversaire esquive son action symbolisée par esq et qui pourra être modifiée par des sorts ou des objets. Lors d'une esquive, l'adversaire ne subit aucun dommage.

I.2) Glossaire et caractéristiques des personnages:

I.2.1) Caractéristiques

Points de vie (PV) : quand un personnage a subi autant de dégâts que ses Points de Vie, il perd le combat.

Points d'action (PA) : les points d'action permettent d'attaquer les ennemis et d'accomplir différentes actions. Ces points seront régénérés à chaque tour.

La Mana : c'est l'énergie magique nécessaire pour utiliser des actions de magie.

Coût de l'action : c'est le nombre de PA qu'il faut dépenser pour exécuter une action (attaques, magie et objets).

Défense : c'est une résistance physique qui permet de subir moins de dégâts lorsque l'on est la cible d'une attaque.

Résistance magique : c'est une résistance qui permet de réduire les dégâts subis par un sort de magie.

Esquive : c'est la probabilité d'esquiver l'attaque de l'adversaire (Échec critique de l'attaque adverse). Chaque personnage possède une valeur de base d'esquive aux attaques qui peut être modifiée lors du combat par des sorts.

Renvoie de dommages : une caractéristique qui permet de renvoyer une partie des dommages infligés par l'adversaire.

I.2.2) Glossaire

Buff/débuff: c'est un "état", un "effet" bénéfique (buff) ou néfaste (débuff) qui modifie les caractéristiques d'un personnage pendant plusieurs tours (par exemple un bonus/malus sur la régénération des PA ou un état de saignement qui fait perdre des PV à chaque tour).

II) Structure préliminaire des programmes clients et serveurs

II.1) Types de messages échangés entre le client et le serveur

Une architecture présentant un logiciel client et un logiciel serveur, ce dernier étant lancé par un des clients (voire par un 3ème ordinateur) a été retenue ici. Pour simplifier le tri des messages échangés à travers le protocole TCp suivant leur utilisation, nous les avons séparés en “types” distincts, représentés par un nombre entre **0** et **6** apposé en début de chaque message échangé. Chaque type englobe une sorte de message et des sous-types sont éventuellement définis pour certains types.

0 => **c**onnexion/**d**éconnexion TCP
Entre superviseur et client
1 => gestion de la partie : **d**ébut/**f**in/**o**k/**r**efus
Entre superviseur et client
2 => choix possibles d'**a**ction/**p**ersonnage
De maitre_jeu vers un seul client
(remarque : ce type n'est jamais envoyé par un client, toujours par maitre_jeu)
3 => choix retenu par le client d'**a**ction/**p**ersonnage
d'un client vers maitre_jeu puis de maitre_jeu vers les deux clients
(remarque : une fois reçu, ce type est toujours retransmis par le maitre_jeu à tous les clients à la fois en tant qu’acquittement pour l'un et comme information pour l'autre afin qu'il sache ce qu'a choisi son opposant et puisse afficher son choix)
4 => « état » des personnages : vie, mana (servant à effectuer des actions « magiques »), points d'action (servant à toutes les actions), défense, chances de coups critiques, chances d'esquiver...
(de maitre_jeu vers les deux clients
(remarque : ce type n'est jamais envoyé par un client, toujours par maitre_jeu)
5 => gestion du « pierre-papier-ciseaux » du début de partie permettant de savoir qui commence : **d**ébut/**f**in/**c**hoix
entre superviseur et client
6 => Mise à jour de la bibliothèque (l'initialisation est une « longue » mise à jour)
de maitre_jeu vers les deux clients (les deux devant avoir les descriptions des actions possibles par chacun des joueurs

II.2) Structure de ces messages

La structure est toujours du type : (int) *type* (entre **0** et **6**) + reste du message

Légende :

- (type de la variable) *nom de la variable*
- « + » fait référence à la concaténation
- *****string***** veut dire une liste sous forme de string avec un séparateur à définir ultérieurement

0 : **0 +** (bool) *type0* + (int) *id_joueur*

avec : -*type0* = 1 si **0c** (connexion) : **0 +** 1 + (int) *id_joueur*

-*type0* = 0 si **0d** (déconnexion) : **0 +** 0 + (int) *id_joueur*

-*id_joueur* = 0 si id inconnu (lorsque le client contacte le superviseur pour la première fois), puis l'objet aiguillage du serveur remplacera ce 0 par l'id qu'il choisi lorsqu'il transmet à superviseur

1: **1 +** (bool) *typeP1* + (bool) *typeP2* + (bool) *information/demande* + (int) *id_joueur*

avec :

-*id_joueur* = 0 si envoyé par le serveur suite à une erreur(exemple : fin de partie parcequ'un joueur a été déconnecté), sinon *id_joueur* du joueur qui demande le début, la fin de partie , qui acquiese à la demande (de début ou de fin) ou qui la refuse, même si c'est le superviseur qui retransmet cette information au second client .

-(comme 2 booléens permettent de faire 4 options :)

typeP1 = 0 et *typeP2* = 0 si **1d** (début) : **1** + 0+0+ (int) *id_joueur*

typeP1 = 0 et *typeP2* = 1 si **1f** (fin) : **1** + 0+1+ (int) *id_joueur*

typeP1 = 1 et *typeP2* = 0 si **1o** (OK) : **1** + 1+0+ (int) *id_joueur*

typeP1 = 1 et *typeP2* = 1 si **1n** (NOK) : **1** + 1+1+ (int) *id_joueur*

- *information/demande* valant 1 si c'est une information (« la partie débute ou termine » maintenant) ou 0 si c'est une demande (« commencer la partie maintenant ? ») pour les types OK ou NOK, ce bit est toujours à 1. Si quelqu'un demande a terminer la partie (sans gagner) ce bit est à 0 , si la partie est terminée parcequ'il y a un gagnant , le superviseur ajoute l'id du gagnant et met ce bit à 1.

2: **2**+(int) *numero_coup_attendu* + (*string*) *liste_possibilites*

2p : 2+(*string*) *liste_personnages_possibles*

2a : **2**+(int) *numero_coup_attendu* + (*string*) *liste_actions_possibles*

3a: **3**+(bool) *id_joueur* + (int) *numero_coup_joue* +(string)*action*

3p: **3**+(bool) *id_joueur* +(string)*personnage choisi*

4: **4**+(int) *numero_coup_joue* + (*string*) *etat_des_persos*

5: **5**+(bool) *type_1* +(bool) *type_2* + options

- **5d** (début): 5+0+0
- **5f** (fin) : **5** +0+1+ (int) *id_joueur*
- **5c** (coup) : **5** +0+1+(string) *coup_joue*

6: **6**+(*string*) *liste*

II.3) Fonction de chaque objet du programme client

Légende :

- *variable*
- objet
- **type_de_message** (voir II.1) à ce sujet)

Interface :
Gère la librairie graphique, fonctionnement à définir. Nous avons choisi d'utiliser la bibliothèque graphique SDl pour l'interface, en effet elle semble convenir à nos besoin en matière de rendu 2D, est utilisable avec le C++ et relativement facile d'accès.

Bibliotheque :
Retiens les descriptions de toutes les actions possibles des deux personnages, ainsi que l'ID de l'action correspondante.

Est initialisée au début de la partie (peut être mise à jour dans ce cas une fonction de remplacement de l'ID est utilisée si il y a conflit).
Lorsqu'il faut choisir quelle action effectuer ou afficher ce que l'adversaire a fait, seule l'ID de l'action est transmise par le serveur, et c'est la bibliothèque
qui permet de revenir au nom et à la description du sort.
Remarque : les buffs/debuffs sont aussi des "actions" et sont donc elles aussi dans la bibliothèque

Client :
Coeur du programme client, il gère l'utilisation et l'affichage des messages reçus depuis le serveur.

Il demande dabord a l'utilisateur d'être Client ou Hote (l'hote étant celui qui lance à la fois le serveur et le client, alors que le "client" ne lance justement que ce dernier processus.
Il lance ensuite le serveur si le joueur a choisi d'être Hote, sinon il demande l'IP du serveur auquel se connecter.
Il contacte le serveur à l'aide d'un message **0**, et l'acquittement du serveur ajoute l' *ID_Joueur du client* à la fin, information que le client conserve par la suite dans *ID_joueur1* ,et utilisera a chacun de ses messages. Cette ID permet de l'identifier et de savoir quel joueur est dans quel "état", effectue quelles actions...
Ensuite pour chaque type de message reçu par le serveur, un traitement spécifique est appliqué :
→ type **2p** (avec un string à la place d'un nombre en deuxième argument)=> affichage de la liste des personnages, ajout d'un champ dans l'interface où le joueur spécifie le nom de son personnage
→ type **3** avec la variable *partie_debutee* de client à faux =>

si l'*ID_jouer* du message != *ID_joueur1* (l'ID du joueur qui fait tourner le processus client en question) on sauvegarde *ID_joueur* dans *ID_joueur2* ainsi que *nom_joueur* dans *nom_2* , sinon on sauvegarde juste *nom_joueur* dans nom_1 . On affiche ensuite chaque choix avec le bon nom.
→ type **1d** avec le (bool) *information/demande* à 0=> demande à l'utilisateur s'il veut commencer la partie l'autre joueur ayant exprimé ce souhait (si oui renvoie d'un **1o** , sinon **In**)
→ type **5d** => demande au joueur de choisir pierre-papier-ciseaux, puis renvoie au serveur avec un **5c**
→ type **5c** => sauvegarde jusqu'à recevoir les 2, puis affiche les 2 coups
→ type **5f** => affiche le gagnant du pierre-papier-ciseaux
→ type **6** => mise a jour de la bibliothèque avec la liste envoyée dans le message
→ type **4** => affichage de l'état des deux personnages (vie, mana...)
→ type **2a** avec la variable *partie_debutee* de client à faux **=>** affichage des actions possibles dans l'interface mais sans demander au joueur d'en choisir une
→ type **1d** avec le (bool) *information/demande* à 1 => mise à 1 de la variable *partie_debutee* interne à client (l'interpretation des messages de type 2 et 3 notamment sera donc différente à partir de maintenant)
→ type **2a** avec la variable *partie_debutee* de client à vrai **=>** affichage des actions possibles dans l'interface ,demande au joueur d'en choisir une et renvoie le tout avec un type **3**
→ type **3** avec la variable *partie_debutee* de client à vrai => affichage du nom du joueur ainsi que de l'action, avec lancement de l'animation adéquate
Lorsque la partie est finie, affichage du nom du gagnant ou d'un message de déconnexion d'un des joueurs.

Traduction :

Enlève la partie *type* (**0** → **6**) des messages reçus de transfert , retransforme le texte reçu en types de variables adéquats attendus par les fonctions de client et envoie à la bonne fonctions de client.
Il accède à la variable *partie_debutee* de client (qui est donc publique ou possède un getter) et tous les traitements pour différencier les sous-types de messages (ex **2a** et **2p** ou **5d, 5f, 5c ...)** et/ou ceux qui impliquent de séparer suivant la valeur de la variable *partie_debutee* de client sont en réalité réalisé par traduction au préalable et client possède une fonction différente pour chacun de ces cas.
Lorsqu'un message lui est transmis depuis client vers transfert il effectue le procédé inverse en transformant tous les messages en texte.

Transfert :

Gère la communication TCP entre client et serveur, effectue une connexion vers l'ip qui lui est donnée, puis fais passer tous les messages quels qu'ils soient sans discrimination aucune dans un sens et dans l'autre. Relais et envoie un message **0** à traduction s'il y a un problème de connexion avec le serveur. Lorsque traduction lui demande à travers un message **0** , transfert se déconnecte du serveur.

II.4) Fonction de chaque objet du programme serveur

Légende :

- *variable*
- objet
- **type_de_message** (voir II.1) à ce sujet)

II.4.1) Objets principaux

Transfert :
Gère la communication TCP entre client et serveur, ouvre ses connexions, puis en accepte deux .
Lorsque chaque premier message **0c** d'acquittement de connexion veut être renvoyé par l' aiguillage vers un des client il garde en mémoire l'*id_joueur* qui a été attribué à ce client et fera par la suite tout seul la conversion *id_joueur* <=> *IP_joueur*.
Fais ensuite passer tous les messages quels qu'ils soient sans discrimination aucune dans les deux sens.
Relais et envoie un message **0** à aiguillage s'il y a un problème de connexion avec un client.

Aiguillage :
Enlève la partie *type* (**0** → **6**) des messages reçus de transfert , retransforme le texte reçu en types de variables adéquats attendus par les fonctions de superviseur et maitre_jeu et envoie à la bonne fonctions des deux objets. Les messages sont grossièrement répartis de manière préliminaire
comme suit :
0,1,5 => superviseur ; **3** => maitre_jeu
Lors de la première communication **0c** avec chaque client, lorsqu'il transfère leur message de connexion vers le superviseur, il remplace la variable *id_joueur* de leur message par une id que chaque objet récupérera au cours du trajet du message et réutilisera.
Les traitements de tri sont ici très simples car il y a peu de types de messages envoyés par le client, et il n'y a donc pas besoin d'utiliser de variable *partie_debutee* pour le tri.
Lorsqu'un message lui est transmis depuis *superviseur* ou *maitre_jeu* vers transfert il effectue le procédé inverse en transformant tous les messages en texte.

Superviseur :
Gère tout ce qui ne concerne pas le combat à proprement parler : la connexion des clients, la réception de leurs requêtes de début et de fin de partie, le jeu de pierre-papier-ciseaux pour décider qui commence à jouer (il fait passer cette information à maitre_jeu à travers

une fonction publique de ce dernier)… C'est lui qui demande a transfert d'ouvrir ses connexions, et c'est lui qui demande à maitre_jeu de proposer les personnages disponibles, puis lui indique le début effectif du combat.
A noter qu'il possède une fonction publique que maitre_jeu utilise pour lui signifier la fin du combat, dans ce cas superviseur envoie un message de type **1f** avec le bit *information/demande* à 1 et ajoute l'*id_joueur* du gagnant ce qui confirme alors au client que c'est le gagnant.

Maitre_jeu :
Gère tout ce qui concerne le cœur du jeu : le combat. C'est lui qui possède les « adresses » des 2 personnages , il leur demande la liste des choix possibles pour chaque action du tour et leur envoie le choix du joueur jusqu'à ce que le joueur n'ai plus suffisamment de points d'actions pour effectuer une action durant ce tour, ou jusqu'à ce qu'il choisisse de passer son tour. Il renvoie chaque message **3a** vers les deux joueurs à chaque fois afin que l'un ait la confirmation, et l'autre sache ce que son adversaire a joué.

Personnages_possibles :

Un objet remplissant exactement la même fonction que traducteur_action, mais pour les personnages : un tableau contenant dans une colonne un string *id_personnage* et dans l'autre une manière d'accéder à l'objet personnage dont l'*id* correspond à *id_personnage* . (Il y a à priori plusieurs manières d'implémenter cela.)

Personnage :
Classe possédant les différents conteneurs des caractéristiques et actions de chaque personnage jouable, ainsi que des méthodes publiques permettant de modifier les valeurs des variables de ces conteneurs de manière contrôlée. Par exemple la méthode *recevoir_degats* permet de prendre en compte notamment la défense du personnage dans le calcul des dommages reçus... Par la suie ce sont ces méthodes qui sont utilisées par les objets de classe action (qui est une classe abstraite pour les attaque,sort,objet) pour interagir avec les caractéristiques des personnages de manière contrôlée.
Elle possède donc des conteneurs de type traducteur_attaques, traducteur_magie, traducteur_objets et traducteur_buffs qui sont 4 tableaux de type traducteur_action qui référencent les actions possibles à un personnages et permettent d'accéder aux objets action à travers la seule variable *id_action*.
Un conteneur_caracteristiques, qui contient toutes les caractéristiques essentielles d'un personnage (vie, mana...)
Un tableau buffs_effectifs qui liste tous les buffs(et débuffs) actifs lors du tour courant sur le personnage, et pour combien de tours ils le seront. Une fois par tour , une méthode spécifique de personnage utilise ce tableau pour effectuer l'action de tous les buffs qu'il contient (et uniquement d'eux)et décrémente le compteur *nombre_tours_restants* puis supprime les entrées lorsque ce compteur tombe à 0.

A noter que pour des questions d'optimisation de la mémoire, les 4 tableaux traducteurs (traducteur_attaques, traducteur_magie, traducteur_objets et traducteur_buffs) sont créés

par le constructeur de la classe personnage mais ne sont PAS remplis à ce moment là, une méthode spécifique est appelée pour cela plus tard lorsque le combat débute vraiment. (Sinon lors du choix du personnage, on aura une surcharge complètement inutile de la mémoire.

II.4.2) Objets nécessaires à la classe personnage

action :
Une classe abstraite pour tous les types d'actions possibles, même un buff (qui est simplement une action qui se répète durant un certain nombre de tours, et est stockée dans un tableau spécifique).

conteneur_caracteristiques:
Un simple conteneur d'int contenant toutes les valeurs des caractéristiques du personnage.

traducteur_actions :
Une classe abstraite pour traducteur_magie, traducteur_objets, traducteur_attaques et traducteur_buff qui est un tableau contenant dans une colonne un string *id_action* et dans l'autre une manière d'accéder à l'objet abstrait action dont l'*id* correspond à *id_action* . (Il y a à priori plusieurs manières d'implémenter cela.)

tableau_buff effectif :
Un tableau contenant dans une colonne un string *id_buff* et dans l'autre un int *nombre_tours_restants ,* les deux colonnes étant liées bien entendu. Il sert à savoir quels buffs sont actifs à un tour donné et pour encore combien de tours (si c'est définitif , alors 99999 tours devrait être suffisant).
Ce tableau doit pouvoir être utilisé afin de pouvoir accéder à un objet action complet (nom,description, effet...) simplement à partir de son *id_action*. Il ne doit aussi contenir aucun doublons au niveau des *id_action* .