
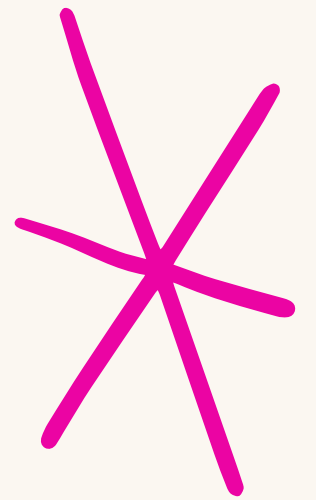
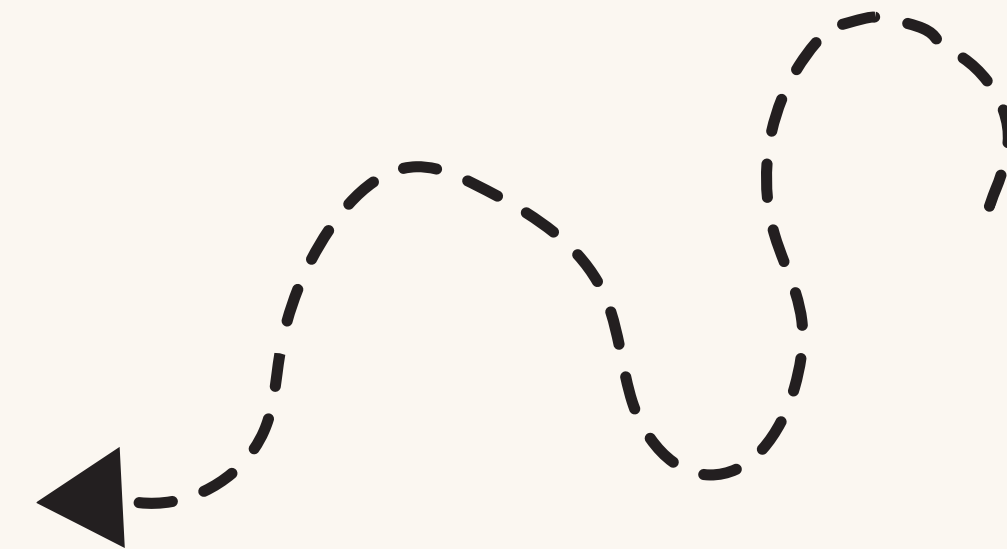


ADAM RODRIGUEZ
LE KIM YEN



Projet



CDKA



Règle du jeu

Règle du jeu

Toute les actions utilisent le clavier. En dialogue les réponses possible sont affichés en dessous du texte. En combat, les nom des sorts ne sont pas disponible, il faut les avoir retenu en avance (ou noté). Une tentative de sort invalide utilise quand même un tour

Avec agidyne on met une valeur au début collé au mot et ça tape l'ennemi concerné, kougaon ne fait que soigner le joueur, et makajama tape aléatoirement

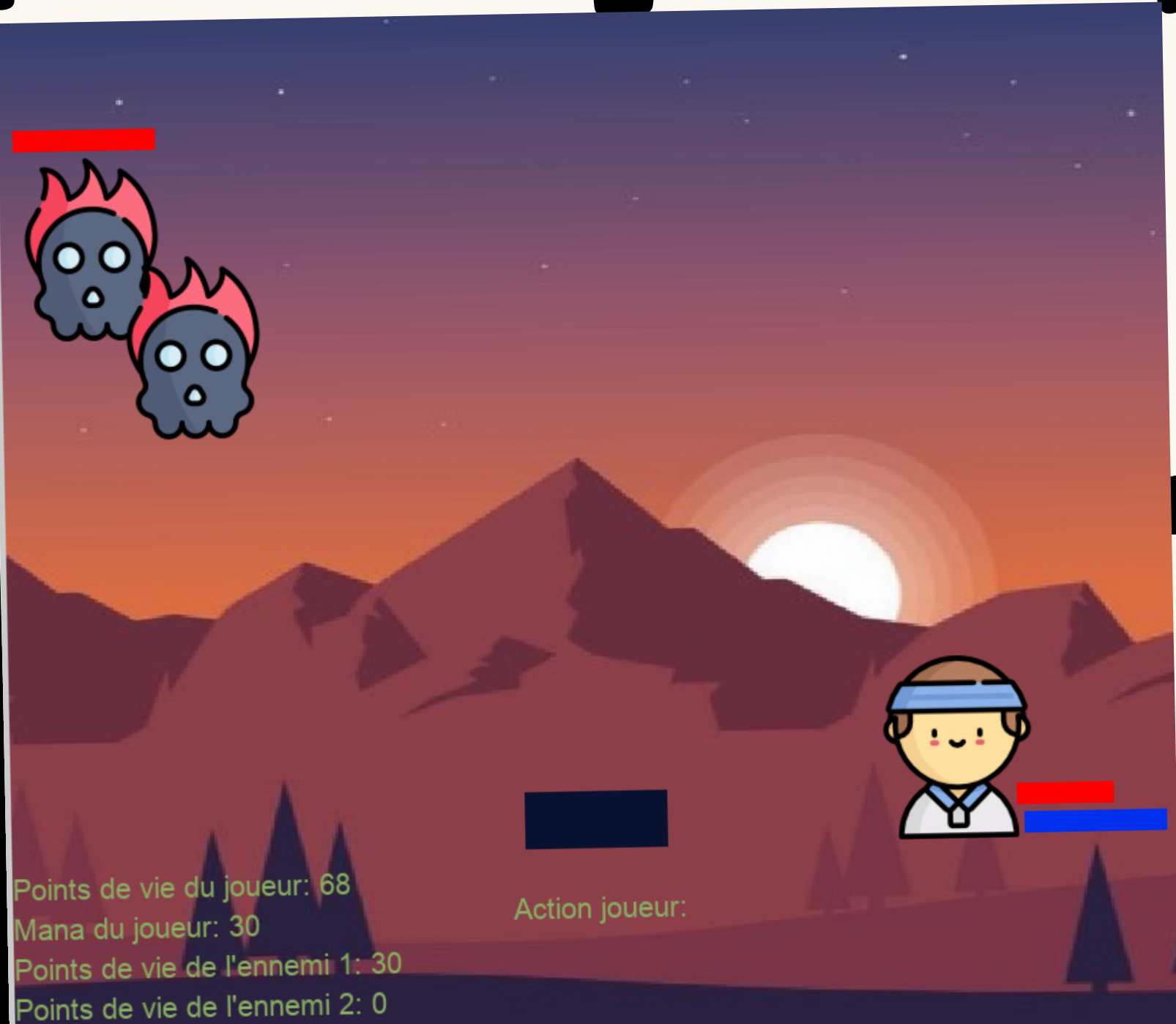
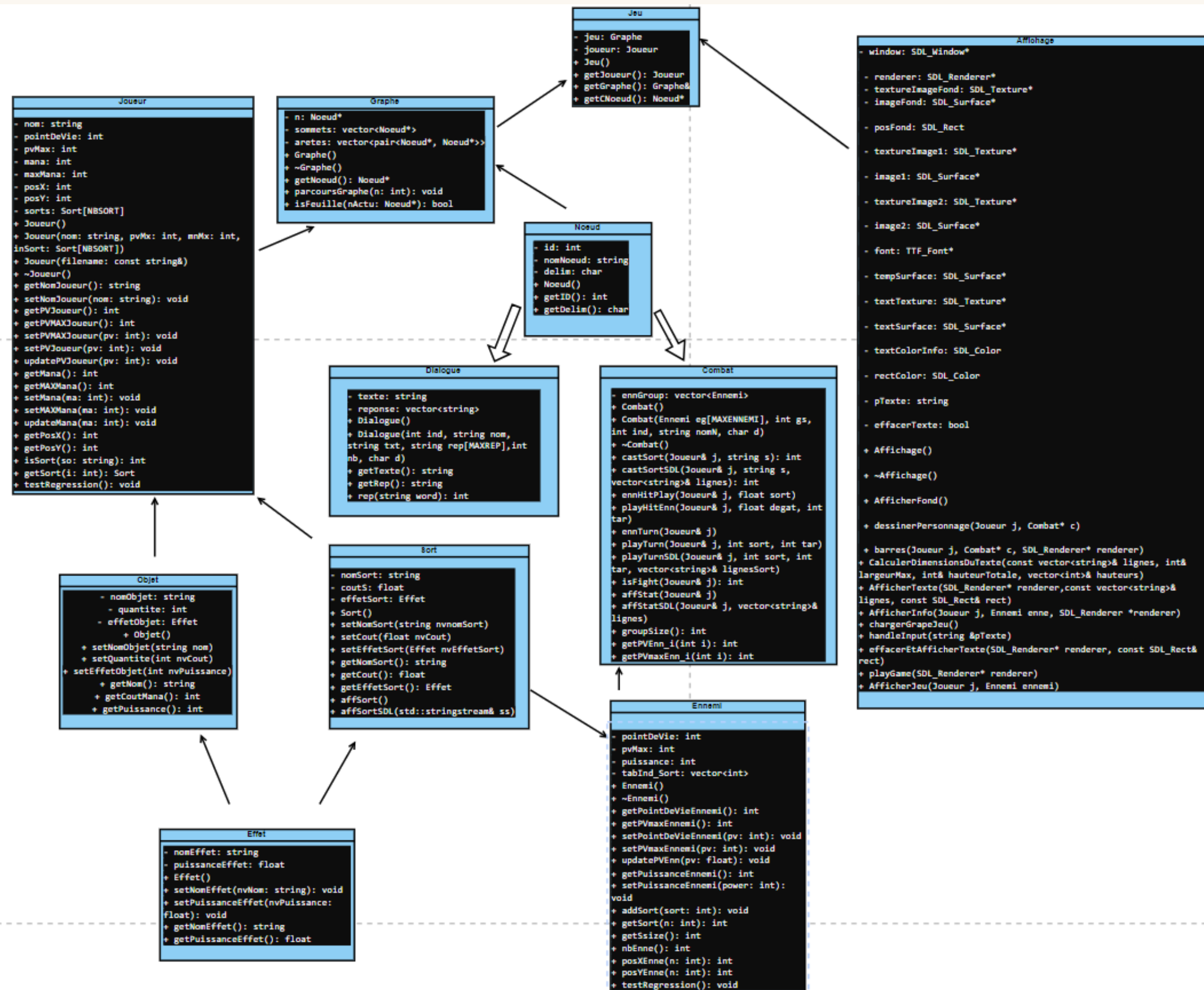


DIAGRAMME DES CLASSES (UML)



Cette classe gère les combats entre le joueur et les ennemis.

- Gestion flexible de la logique de combat : Les méthodes telles que ennTurn et playTurn permettent à la classe Combat de gérer les actions à chaque tour pour les ennemis et les joueurs de manière flexible. Cela rend la classe adaptable à différentes situations de jeu.
- Les méthodes castSort, ennHitPlay, playHitEnn, ennTurn, playTurn et isFight sont responsables des différentes actions liées au combat, telles que le lancement de sorts, les attaques des ennemis et du joueur, ainsi que la vérification de l'état du combat.
- Les méthodes castSort et castSortSDL permettent d'interagir avec l'utilisateur en lui permettant de choisir et de lancer des sorts pendant le combat. Cela rend le jeu plus dynamique et engageant pour le joueur
- La méthode isFight détermine si le combat se poursuit ou non, en fonction de l'état du joueur et des ennemis. Cela permet à la classe de contrôler le déroulement du jeu et de passer à d'autres phases une fois que le combat est terminé.



CLASSE 'COMBAT'

```
Combat
- ennGroup: vector<Ennemi>
+ Combat()
+ Combat(Ennemi eg[MAXENNEMI], int gs,
int ind, string nomN, char d)
+ ~Combat()
+ castSort(Joueur& j, string s): int
+ castSortSDL(Joueur& j, string s,
vector<string>& lignes): int
+ ennHitPlay(Joueur& j, float sort)
+ playHitEnn(Joueur& j, float degat, int
tar)
+ ennTurn(Joueur& j)
+ playTurn(Joueur& j, int sort, int tar)
+ playTurnSDL(Joueur& j, int sort, int
tar, vector<string>& lignesSort)
+ isFight(Joueur& j): int
+ affStat(Joueur& j)
+ affStatSDL(Joueur& j, vector<string>&
lignes)
+ groupSize(): int
+ getPVEnn_i(int i): int
+ getPVmaxEnn_i(int i): int
```


La classe Graphe, qui représente un graphe de jeu et pour gérer la progression du joueur à travers différents nœuds du graphe, représentant des dialogues et des combats

- Le constructeur Graphe ouvre un fichier texte contenant les informations nécessaires à la création du graphe du jeu. Il lit les données du fichier, crée les nœuds correspondants (soit des dialogues, soit des combats), et établit les arêtes entre ces nœuds pour représenter la structure du jeu, Ex: pour chaque sommet dans le fichier :
 - + Si le sommet est un nœud de dialogue, extrait son texte et ses réponses.
 - + Si le sommet est un nœud de combat, extrait les informations sur les ennemis et leurs compétences.
 - + Ajoute le sommet au vecteur sommets.
 - + Lit et stocke les liens entre les nœuds dans le vecteur aretes.
- La méthode getNoeud permet d'accéder au nœud actuel du graphe
- La méthode parcoursGraphe permet de naviguer dans le graphe en se déplaçant vers un nœud spécifique en fonction de son identifiant. Cela permet de faire avancer le jeu en fonction des actions du joueur
- La méthode Graphe::isFeuille vérifie si un nœud donné est une feuille du graphe



CLASSE 'GRAPHE'

Graphe

```
- n: Noeud*
- sommets: vector<Noeud*>
- aretes: vector<pair<Noeud*, Noeud*>>
+ Graphe()
+ ~Graphe()
+ getNoeud(): Noeud*
+ parcoursGraphe(n: int): void
+ isFeuille(nActu: Noeud*): bool
```

La création d'un jeu en C++ utilisant SDL pour la partie graphique, avec des fonctionnalités telles que l'affichage d'images, de texte, la gestion des interactions utilisateur et le déroulement du jeu



- Le programme définit plusieurs méthodes pour gérer l'affichage des éléments du jeu, y compris le fond, les personnages, les barres de santé et de mana, et les textes. Il y a également des méthodes pour gérer la saisie utilisateur et le déroulement du jeu
- L'aspect principal de la gestion du texte dans la classe Affichage réside dans sa capacité à afficher efficacement les textes à l'écran et à gérer les entrées utilisateur à partir du clavier. Elle utilise une méthode 'AfficherTexte' qui prend en paramètre le renderer SDL et un vecteur de lignes de texte à afficher. Cette méthode calcule les dimensions du texte à afficher, crée une surface de texte à partir des lignes de texte
- La méthode 'handleInput' de la classe gère la saisie utilisateur depuis le clavier, récupérant la chaîne de texte entrée et la stockant pour une utilisation ultérieure. Cette fonction peut être invoquée à chaque trame du jeu pour surveiller les entrées utilisateur
- La méthode 'playGame' gère le déroulement du jeu en affichant les éléments du jeu, en permettant à l'utilisateur d'interagir et en mettant à jour le jeu en conséquence
- La méthode AfficherJeu est utilisée pour démarrer le jeu en affichant les éléments du jeu et en gérant le déroulement du jeu jusqu'à ce que l'utilisateur décide de quitter

CLASSE 'AFFICHAGE'

```
Window: SDL_Window*
- renderer: SDL_Renderer*
- textureImageFond: SDL_Texture*
- imageFond: SDL_Surface*
- posFond: SDL_Rect
- textureImage1: SDL_Texture*
- image1: SDL_Surface*
- textureImage2: SDL_Texture*
- image2: SDL_Surface*
- font: TTF_Font*
- tempSurface: SDL_Surface*
- textTexture: SDL_Texture*
- textSurface: SDL_Surface*
- textColorInfo: SDL_Color
- rectColor: SDL_Color
- pTexte: string
- effacerTexte: bool
+ Afficher()
+ ~Affichage()
+ AfficherFond()
+ dessinerPersonnage(Joueur j, Combat* c)
+ barres(Joueur j, Combat* c, SDL_Renderer* renderer)
+ CalculerDimensionsDuTexte(const vector<string>& lignes, int& largeurMax, int& hauteurTotale, vector<int>& hauteurs)
+ AfficherTexte(SDL_Renderer* renderer, const vector<string>& lignes, const SDL_Rect& rect)
+ AfficherInfo(Joueur j, Ennemi enne, SDL_Renderer* renderer)
+ chargerGrappeJeu()
+ handleInput(string &pTexte)
+ effacerEtAfficherTexte(SDL_Renderer* renderer, const SDL_Rect& rect)
+ playGame(SDL_Renderer* renderer)
+ AfficherJeu(Joueur j, Ennemi ennemi)
```

CONCLUSION

- Bien que le code fonctionne assez bien et que le jeu puisse être joué à la fois en mode texte et en fenêtre SDL comme prévu, nous avons encore besoin de plus de temps pour compléter les classes telles que objets, sorts et graphe afin que le jeu puisse offrir plus d'options et être plus attrayant pour les joueurs. Nous devons nous concentrer sur le développement des fonctionnalités et du contenu du jeu pour créer une expérience de jeu intéressante et variée pour les joueurs
- La gestion de la mémoire dans une application SDL peut être complexe. Même après avoir fait de mon mieux pour résoudre les problèmes de mémoire, il peut encore rester des problèmes à résoudre. Je dois prendre le temps de revoir attentivement le code pour déboguer toutes les erreurs

**MERCI DE VOTRE
ATTENTION**

