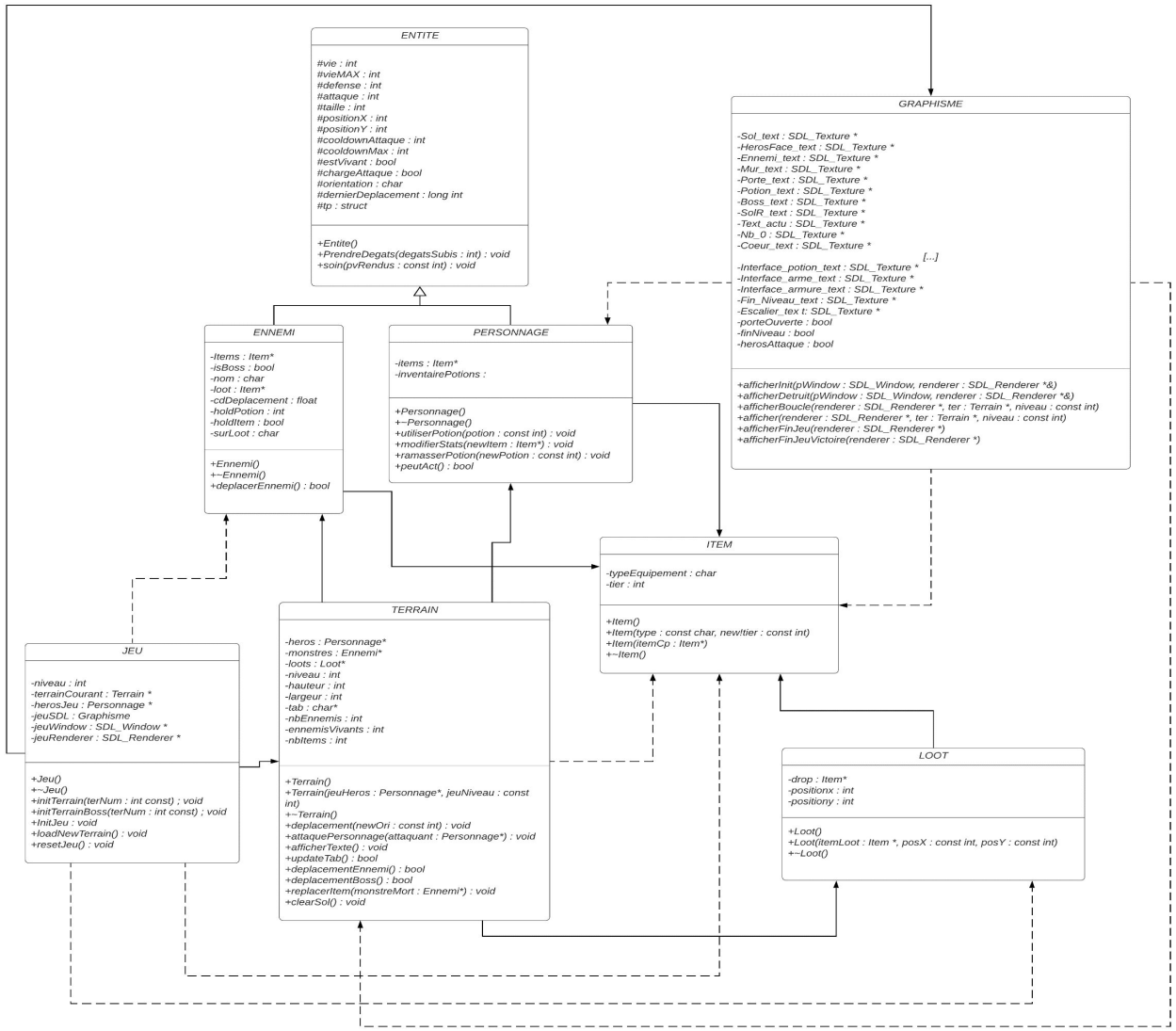
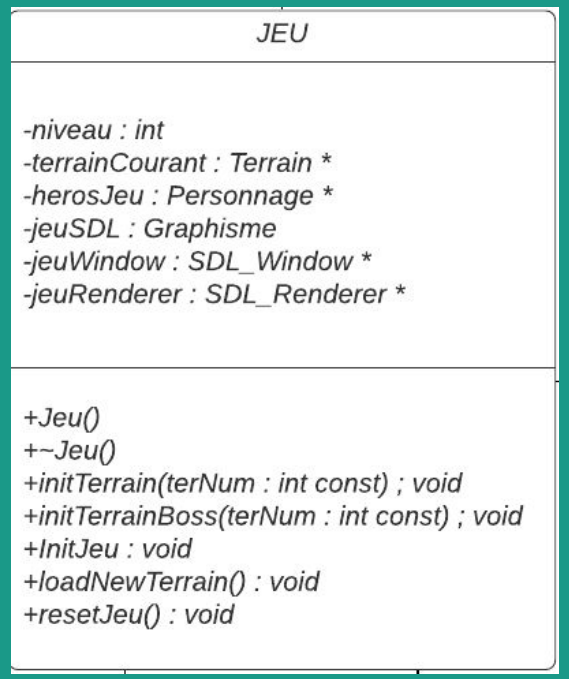
Rogue Dungeon (rogue-like)

Par Work_in_progress
- BERNOT Camille : p1908800
- GABRIEL Justin : p1920022
- NOUVEL Alexandre : p1710464

Diagramme des classes



La classe Jeu



`initTerrain`

- La méthode `void initTerrain(const int terNum)` :
  - Est appelée à chaque étage du donjon.
  - Ouvre un fichier terrain désigné par l'indice terNum.
  - Attribue au terrain le Personnage de la classe Jeu.
  - Initialise le tableau de caractères représentant le terrain ainsi que ses dimensions.
  - Instancie les ennemis avec des statistiques dépendant du niveau.
  - Génère de façon aléatoire le loot qui sera présent sur les ennemis.
  - A une version alternative `initTerrainBoss` adaptée aux niveaux de Boss.

initTerrain

- Un fichier map tel qu'on les trouve dans l'archive.

```
21
15
5
ooooooooooooooovvvvvvvvvvvvvvoovvvvvvvEvvvvvvvoovvvvvvvvvvvvvvoovvvvoovoovvvvoovvvvvvvvvvvvvoovvvvEvv
vEvvvvoovvvvvvvvvvvvvvoovvvovvvvvvovvvoovEvovvvvvvovEvoovvvovvvvvvovvvoovvvovvvvvovvvoovvvovvvvvovvvoo
vvvovvvvvovvvoovvvvvvvvvvvvvvoovvvvvvvvvvvvvvoovvvvvvPvvvvvvo
ovvvvvvvvvvvvvoovvvvvvvvvvvvvoovvvvvvvvvvvvvvoooooooooooooooo
7
16
7
2
5
6
9
6
2
9
12
9
```

initTerrain

- Une version plus lisible d'un fichier map

```
hauteur:21
largeur:15
nbEnnemis:3
oooooooooooooo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvEvvvvEvvo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvvEvvvvvvo
ovvvvvvvvvvvvo
ovvovvvvvovvvo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvvvvvvvvvo
ovvvvvvPvvvvvo
oooooooooooooo
Xpersonnage:7
Ypersonnage:19
XEnnemi1:5
YEnnemi1:3
XEnnemi2:10
YEnnemi2:3
XEnnemi3:6
YEnnemi3:6
```

Autres méthodes

- La méthode `void initJeu`
  - Est appelée au lancement du jeu.
  - Appelle la méthode `afficherInit` de l'attribut jeuSDL
  - Appelle la méthode `afficherBoucle` de l'attribut jeuSDL et gère sa valeur de retour.
- La méthode void `loadNewTerrain`
  - Appelle la méthode `afficherBoucle` de l'attribut jeuSDL et gère sa valeur de retour.
  - S'appelle elle-même si le héros a passé l'étage
  - Appelle `initTerrain` avec une valeur aléatoire ou `initTerrainBoss` tous les 5 niveaux.

Autres méthodes

- La méthode `void resetTerrain()`
  - Est appelée à la fin d'une partie
  - Réinitialise le niveau et le personnage.
  - Appelle `initTerrain` avec une valeur aléatoire.
  - Appelle la méthode `afficherBoucle` de l'attribut jeuSDL et gère sa valeur de retour.
  - Appelle `loadNewTerrain` si le héros a passé l'étage.

La classe Jeu_SDL

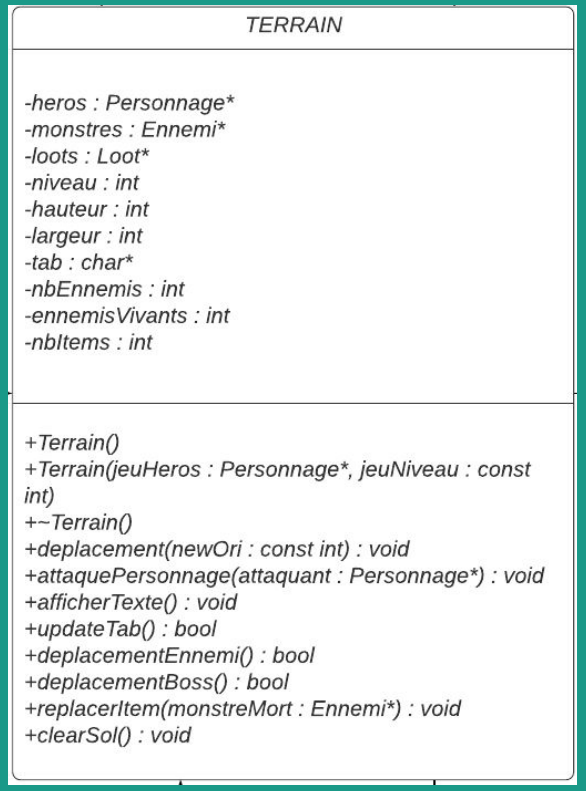
Diverses méthodes

- Une méthode de création de fenêtre et plusieurs d'affichage.
  - `afficherInit` : pour créer une fenêtre et créer les textures
  - `afficher` : pour l'affichage du jeu
  - `afficherFinJeu` : pour l'affichage de la mort du personnage
  - `afficherFinJeuVictoire` : pour l'affichage de la victoire du personnage

Autres méthodes

- Une méthode principale: `int afficherBoucle(SDL_Renderer * renderer, Terrain * ter, const int niveau)`:
  - Gère les inputs de l'utilisateur
  - Appelle les méthodes du personnage d'après ces inputs si son cooldown d'action est écoulé.
  - Appelles `deplacementEnnemi` à chaque tour de boucle.
  - Renvoie un code sous la forme d'un entier décrivant la fin du niveau:
    - 0 : L'utilisateur a appuyé sur échap, fin du programme
    - 1 : Le héros est sorti du niveau, chargement du niveau suivant
    - 2 : Le héros est mort, fin de la partie, affichage de l'écran de mort.
    - 4 :Le héros a passé le 50ème niveau, affichage de l'écran de victoire.
    - 100 : Code de triche : fait passer automatiquement le héros au prochain étage.

La classe Terrain



`deplacement`

- La méthode `void deplacement(const int newOri)` :
  - Est appelée dans la boucle d'affichage.
  - Autorise le déplacement si la case visée n'est pas occupée par un ennemi ou un obstacle.
  - Gère le déplacement du personnage et ses interactions avec l'environnement (looting, "sol rouge" en combat de boss...).
  - Met à jour le tableau de caractères, les coordonnées du personnage et son orientation.

`attaquePersonnage`

- La méthode `void attaquePersonnage(Personnage * attaquant)` :
  - Est appelée dans la boucle d'affichage.
  - Gère la position et l'orientation du personnage pour définir la case cible de l'attaque.
  - Inflige des dégats à l'ennemi s'il était présent sur la case visée.
  - Gère la mort des ennemis :
    - Replacement de l'ennemi via la méthode `replacerItem` si celui-ci se trouvait sur un loot ou une potion.
    - Affichage de la porte de sortie si tous les ennemis sont morts et appel à `clearSol` pour la suppression du "sol rouge" après un combat de boss.
    - Mise à jour du terrain (tableau de caractères et tableau d'ennemis).

`deplacementEnnemi`

- La méthode bool `deplacementEnnemi` :
  - Est appelée à chaque itération de la boucle d'affichage.
  - Parcourt le tableau d'ennemis, ignore l'ennemi s'il est mort ou si son cooldown ne s'est pas écoulé.
  - A une version alternative `deplacementBoss` adaptée aux niveaux de Boss.
  - Gère le déplacement et l'attaque de chaque ennemi :
    - Autorise le déplacement si la case est libre.
    - Déplace l'ennemi en priorité sur l'axe sur lequel il est plus éloigné du héros.
    - Stocke temporairement le caractère représentant la case destination et replace le caractère stocké à la case de départ.
    - Fait subir des dégâts au héros le héros s'il est à une case de l'ennemi. Renvoie vrai si le héros est mort, faux sinon.

Conclusion

Difficultés

- Difficultés matérielles pour configurer un environnement Linux (limitation matérielle pour certains membres du groupe), impossibilité de se réunir physiquement pour travailler.
    - Implementation précoce de certaines classes qui a amené à des choix non pertinents sur lesquels il était difficile de revenir à un stade avancé du projet.
    - Certaines fonctionnalités se sont révélées difficiles à mettre en place dans le temps imparti (arc et projectile).

Bilan

- Nous avons réussi à faire une application fonctionnelle sans bug connu.
- Le jeu est complet avec des fonctionnalités simples mais efficaces comme
  - la difficulté progressive avec une accélération du jeu (cooldown d'action des ennemis et du boss se réduisant, augmentation de leurs points de vie et de leur attaque)
  - ajout de boss avec une mécanique intéressante et adapté au gameplay
  - Rendu agréable, impression de progression dans le jeu (looting d'un meilleur équipement)
  - Difficulté progressive, nécessité de s'adapter au jeu (mécanique de "kiting" : attirer les ennemis pour leur infliger des dégâts puis reculer pour ne pas se faire toucher).
- Apprentissage du travail de groupe et developpement de competences.
  - Travail en distanciel et organisation en parallèle.
  - Compétences en programmation, écriture d'algorithmes adaptés pour résoudre un problème.