ALLART Thibaut
CAIN Florian

INFO0601-INFO0604
2021-2022

Rapport de projet

Table des matières

Structures

Chaque nom de sous-partie correspond au nom du fichier d'en-tête contenant une ou plusieurs structures.

Artefact

Chaque artefact a un nom (de type char*) et possède différentes statistiques (de type int) : points de vie, armure, force, vitesse_attaque et vitesse_deplacement. Chacun peut apporter un ou plusieurs bonus ou malus au joueur qui l'a équipé. Un artefact doit pouvoir être ramassé à condition que le joueur qui veuille l'équiper ait au moins un emplacement libre ; dans le cas contraire, il doit pouvoir poser un artefact au sol avant de pouvoir ramasser celui souhaité.

Il y a 10 artefacts au total (retrouvable dans « artefact.c »), tous cumulables :

```
artefact artefact_vie = {"Artefact de Vie", 15, 0, 0, 0, 0};
artefact artefact_armure = {"Artefact d\'Armure", 0, 3, 0, 0, 0};
artefact artefact_force = {"Artefact de Force", 0, 0, 3, 0, 0};
artefact artefact_vit_att = {"Artefact de Vitesse d\'Attaque", 0, 0, 0, 3, 0};
artefact artefact_vit_dep = {"Artefact de Vitesse de Déplacement", 0, 0, 0, 0, 3};
artefact artefact_roi_rablite = {"Artefact du Roi-Rablite", 10, 10, 5, -3, 0};
artefact artefact_chef_delinquant = {"Artefact du Chef-Delinquant", -5, -2, 15, 3, 3};
artefact artefact_tigre = {"Artefact du Tigre", -5, 0, 3, 0, 3};
artefact artefact_fleur = {"Artefact de la Fleur", 5, -1, 2, 0, 0};
artefact artefact_bougre = {"Artefact du Bougre", -5, -5, -5, -5, -5};
```

Ramasser un artefact qui fait perdre des points de vie maximum ne redonne pas de vie quand on le récupère (le joueur ne peut pas mourir de cet effet mais peut arriver jusqu'à 1 point de vie). Ce choix est dû au fait que si un artefact donnant des points de vie bonus, il est normal de les perdre ; si le joueur perd des points de vie, il n'est pas naturel de les récupérer instantanément. Nous voulions préciser ce point pour que cet effet ne soit pas considéré comme un bug.

Client

La structure « arg_actualiserMap » est réutilisée dans la fonction actualisationMap(), elle permet de garder en mémoire les variables nécessaires à l'actualisation de l'affichage d'un client. Les champs qui la constituent sont les fenêtres d'affichage de la carte et des attributs, les ID et noms des joueurs, un mutex attaché à la map sur laquelle se trouve un joueur et le descripteur de la socket.

Map

Chaque map est constitué d'un tableau de cases. Une case a une position en x et en y (int), un élément (joueur, monstre, trésor, artefact ou obstacle) et un background (sable, eau, montagne ou herbe). Un des principaux points à gérer est qu'un monstre ou un joueur ne doit pas pouvoir se déplacer n'importe où et, dans le cas où un joueur ramasse un trésor ou un artéfact, celui-ci doit pouvoir le récupérer avant d'écraser l'élément de la case. Si un monstre ou un joueur arrive sur une case occupée par un autre monstre ou joueur, un combat doit être déclenché.

Monstre

Un monstre a un nom (de type char*), une position en x et en y (int), des statistiques de combat (tous de type int) : des points de vie, de l'armure, de la force, de la vitesse d'attaque et de la vitesse de déplacement. Chaque monstre possède un montant d'expérience initial qui sera attribué au joueur qui le vaincra en fonction du niveau de ce dernier. Les monstres ne se battent pas entre eux mais peuvent agresser les joueurs inattentifs.

Player

Un joueur a un nom (de type char*) que l'utilisateur choisit au lancement du client, celui-ci est envoyé au serveur pour qu'il soit affiché dans la fenêtre des attributs. Il a aussi une position en x et en y (int). Il possède une liste d'artefacts pouvant en contenir 5 au maximum. Comme les monstres, il possède des statistiques de combat (de type int) : points de vie, points de vie maximum, armure, force, vitesse d'attaque, vitesse de déplacement. Certaines statistiques alternatives (type int), soient le nombre de pièces du Grand-Tout et l'expérience actuellement possédée font aussi partie de la liste des informations du joueur. Toutes ces données sont représentées dans la fenêtre des attributs et sont mises à jour suivant la progression du joueur.

Lorsqu'un joueur possède plus de 100 points d'expérience, il améliore aléatoirement une statistique et perd 100 points jusqu'à en avoir moins de 100. Les statistiques améliorables sont l'armure, la force, les points de vie maximum, la vitesse d'attaque et la vitesse de déplacement. Si les points de vie maximum sont augmentés, l'équivalent en points de vie (le montant que possède actuellement le joueur) est attribué à celui-ci qui améliore cette statistique.

Requête

Une requête permet au serveur de recevoir différentes informations de la part d'un client. La structure nommée « reponse_map_et_player » permet de communiquer dans le sens inverse, soit du serveur vers le client.

Chaque requête possède deux coordonnées (de type int) indiquant la position d'une map (cf. la sous-partie WorldMap), un ID correspondant au client qui envoie la requête, une commande (de type int) qui sera soit une demande de nouvelle carte, un déplacement, une déconnexion ou une actualisation. L'action sert uniquement à indiquer la touche pressée par le joueur et permettra de d'afficher le bon résultat à l'écran : soit un déplacement, soit une activation des pièces du Grand-Tout.

Lors d'une réponse, le serveur envoie la map (map) actualisée au joueur (player) qui a envoyé la première requête.

WorldMap

5 structures composent le fichier d'en-tête de WorldMap.

Les 3 premières sont des listes d'artefacts, monstres et joueurs. Ces listes ont tous pour but de répertorier les différentes entités présents sur une map.

Une completeMap est une map (cf. Map) à laquelle on ajoute les 3 listes vues précédemment, permettant d'obtenir toutes les informations d'une zone de la carte du monde. Les monstres y circulent dès leur création, les joueurs peuvent aussi se déplacer librement, ramasser et déposer des artefacts.

Algorithmes principaux

Les noms des sous-parties se rapportent au nom des fichiers contenant le code C des fonctions exposées.

Editeur

L'éditeur permet de créer des cartes en choisissant l'élément et le background de chaque case, et de modifier des cartes existantes enregistrer en fichier binaire.

Map

Il était prévu de faire une fonction de compression de l'affichage d'une map afin de pouvoir en mettre 20 dans une file de messages, permettant de communiquer entre le serveur principal et ses fils qui gèrent un client chacun. La fonction de compression devait retenir le background et l'élément d'une case et ajouter un caractère entre A et X qui correspondait aux attributs de la case. La fonction de décompression devait opérer le procédé inverse. 2 fonctions permettaient de gérer ce code (détection des attributs vers code et inversement).

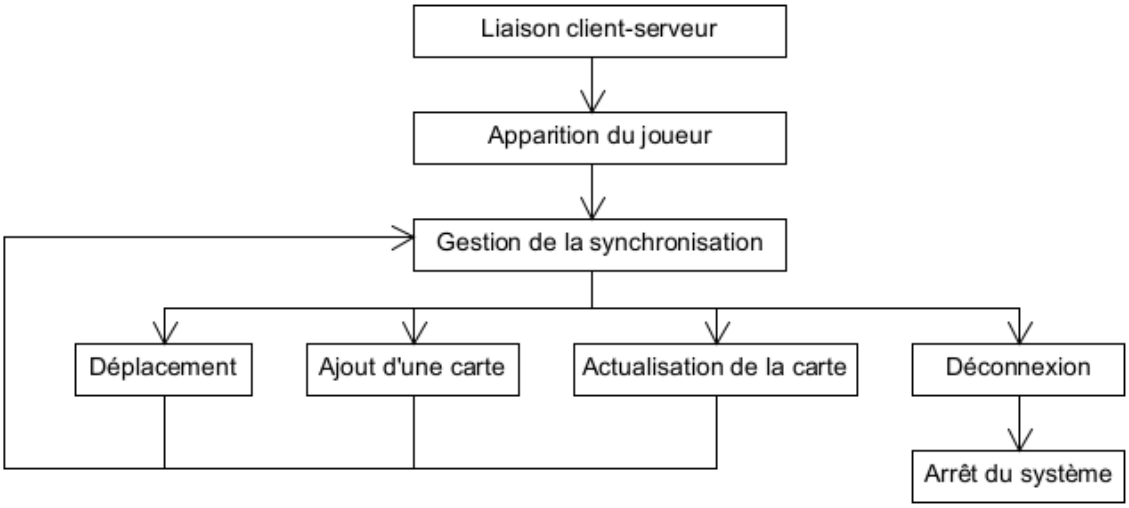
Nous avons décidé de laisser ces fonctions en commentaire même si elles ne serviront pas car elles permettaient d'utiliser des éléments vu en INFO0603. Toutes les données d'une map pouvaient être compressées jusqu'à 801 octets à la place de plus de 4600 octets, ce qui était plutôt intéressant au niveau de la gestion de la mémoire.

Elles ne sont pas fonctionnelles car nous voulions finir ce dont nous avions besoin en premier lieu et n'avons pas eu le temps de les compléter.

Serveur

Player_thread()

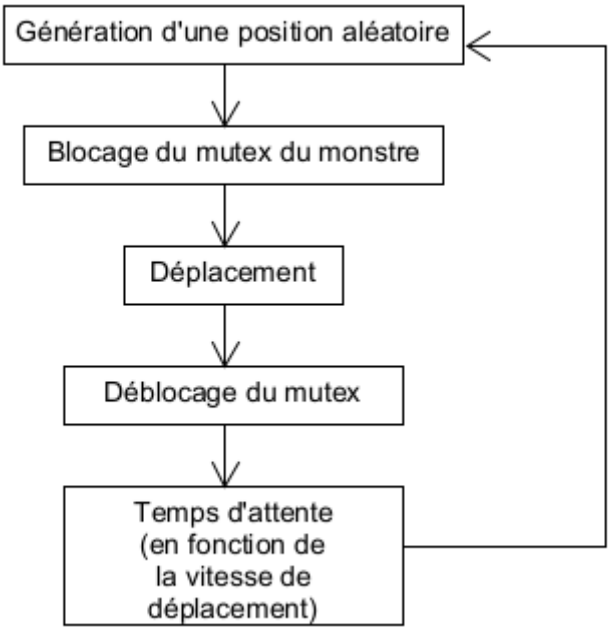
Le thread associé à chaque joueur permet de s'occuper d'un client. Il récupère les données envoyées par le client nécessaire au bon fonctionnement de l'application côté serveur à la création du personnage. Son rôle est d'assurer l'apparition du joueur sur une case valide puis de faire en sorte que le jeu se déroule correctement pour lui.



WorldMap

monsterMove()

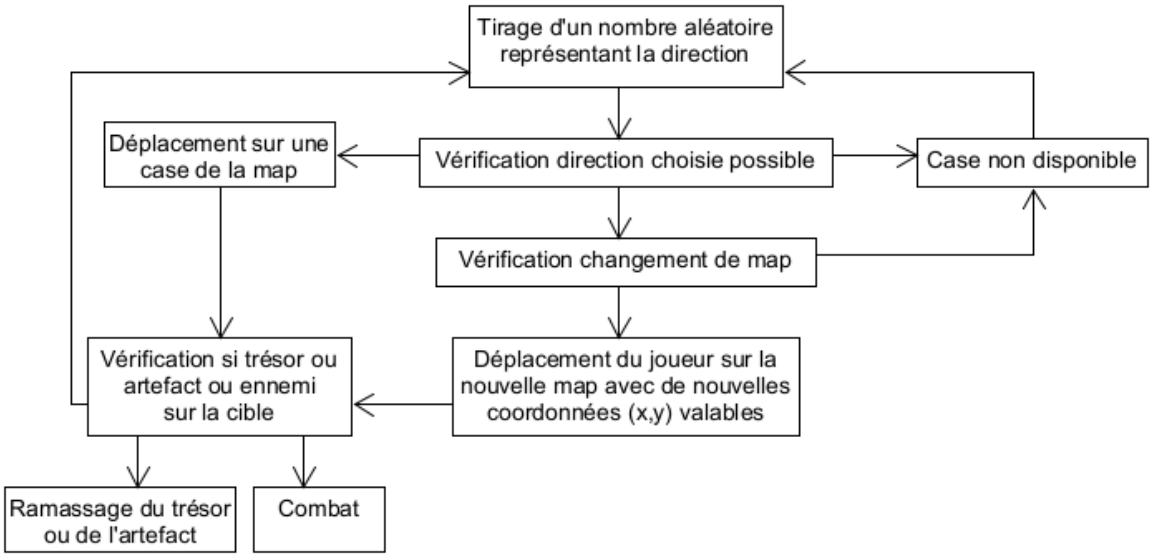
Le monstre se déplace aléatoirement d'une case en fonction de sa vitesse de déplacement (si la case sélectionnée le permet).



Dans le cas où le monstre meurt, le thread gérant le monstre arrête proprement la fonction de déplacement.

playerMove()

Le joueur peut se déplacer sur une case vide, si un trésor ou artefact se trouve sur la case choisie, ou si un ennemi est positionné sur la case cible. Dans le cas où il y a un trésor ou artefact, le joueur le ramasse si son inventaire le permet. Dans le cas où un ennemi (monstre ou joueur) est présent, l'attaquant et le défenseur frappent l'adversaire une fois chacun.



pve()/pvp()

Sur une carte, des combats peuvent avoir lieu entre des monstres et des joueurs ou des joueurs entre eux. Pour cela, il faut qu'un joueur se dirige vers une case occupée par un monstre ou un joueur, un combat est alors déclenché et les dégâts infligés sont équivalent à la formule suivante (A = entité qui attaque ; B = entité qui subit les dégâts) : $dmgSubisParB = (forceA * vitesse\ d'attaqueA * vitesse\ de\ deplacementA)/(defB * vitesse\ de\ deplacementB)$. Pour tuer une entité sur la map, il faut attaquer plusieurs fois celle-ci si elle ne meurt pas dès le premier coup (se déplacer plusieurs fois dans sa direction). L'expérience gagnée par un joueur à la fin du combat dépend de l'ennemi vaincu : +100 XP pour un joueur battu et +N XP pour un monstre vaincu où N est la valeur de la variable de l'XP du monstre.

Dans un combat, il n'y a pas besoin de bloquer un joueur ou un monstre avec un mutex car la map est déjà bloquée et tant que celle-ci est bloquée, il ne peut rien se passer d'autre. Si la vie d'un monstre ou d'un joueur n'est pas strictement positive, sa routine le détecte et il mourra instantanément.

Bonus :

- Quand un monstre meurt, il laisse derrière lui un trésor ;
- Un monstre peut engager un combat contre un joueur.

Améliorations possibles

3 éléments des consignes n'ont pas été respectés :

- Création d'un fichier .sav : trop long et complexe à mettre en place pour le temps qu'il restait sans avoir de problèmes importants à gérer ;
- Effets des pièces du Grand-Tout : nous avons préféré éviter de les ajouter pour éviter l'apparition de nouveaux bugs (où il est possible que nous n'aurions pas eu le temps de les corriger) ;
- Détails de certaines actions dans la fenêtre d'informations : nous avons préféré les derniers bugs qu'il restait et cela nous aurait demander de remodifier plusieurs fonctions.

Nous aurions pu ajouter des éléments tels que l'affichage des caractéristiques d'un monstre ou d'un joueur dans le fenêtre d'informations, empêcher les monstres d'une même famille de se battre mais qu'ils se battent contre d'autres types.