XenoKart Eco+ Ultimate (Not) Definitive Edition
Licence Informatique *2ème année*

Mano Brabant
Benjamin Riviere
Erwan Serelle

19 avril 2019

Table des matières

1 Introduction

Pour ce projet, notre objectif était de faire un jeu fun avec des mécaniques plus originale qu'un dungeon crawler classique. Le but du jeu est d'obtenir le meilleur score possible. On obtient des points en échange d'objet récupérés pendant l'exploration de différentes cartes et pendant des combats.

2 Organisation du travail

Nous avons défini les différentes tâches à réaliser pour le projet et nous les avons réparties de la manière suivante :
- Le menu : Benjamin Riviere
- Le lobby : Erwan Serelle
- La carte :
    - Génération, Pathfinding : Benjamin Riviere
    - Utilisation : Erwan Serelle
- Le système de combat : Mano Brabant

Le projet est stocké sur GitHub.[1]

Le dépôt a été structuré en plusieurs parties :
- SDL2 : contient la bibliothèque Simple Direct Layer
- code : contient les différents .c et .h qui constituent le programme
    - menu
    - lobby
    - map
    - combat
- data : contient les différents textes, images et autres données nécessaires au programme.
- docs : contient les documents annexes.

3 Analyse

Le but du jeu est d'obtenir le meilleur score possible à la fin de la partie. On obtient des points en échange d'objets qu'on récupère en expédition.
On peut aussi choisir de vendre une partie de ses objets contre de l'argent. Cette argent nous permet d'acheter des objets utilitaires à un marchand qui nous aideront au cours de notre partie.

On peux ensuite partir à l'exploration du lieux de notre choix.
On arrive alors sur une carte composée de cases hexagonales de différents type (forêt, désert, eau ...) et on doit rechercher et atteindre la case d'arrivée. Il y aura des ennemis a combattre sur le chemin, et des cases spéciales à explorer pour obtenir le plus d'objets possible.

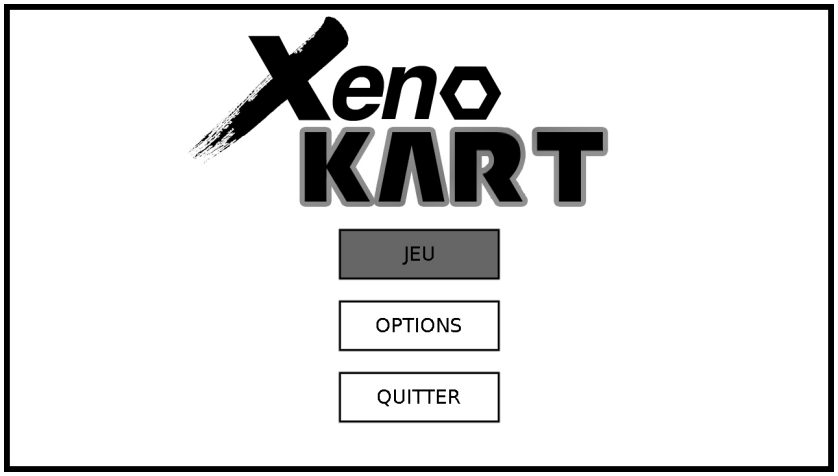
4 Conception

4.1 Menu

Le menu est la première interface sur laquelle l'utilisateur arrive au lancement du jeu. Tout d'abord il y a le logo du jeu et une invitation à appuyer sur n'importe quelle touche afin de lancer le menu.
Ensuite l'utilisateur arrive sur une interface avec des boutons qui peuvent être sélectionnés avec les touches du clavier :

- Z ou Flèche vers le haut : pour aller vers le haut.
- S ou Flèche vers le bas : pour aller vers le bas.
- Entrée ou Espace : pour sélectionner le bouton sur lequel on est positionné.
- Echap ou Retour : pour retourner en arrière.

Le bouton sur lequel on est positionné est symbolisé par le bouton de couleur rouge.
Le menu est composé de sous-menus, tel que le menu des options ou le menu du jeu. Leur fonctionnement est le même que pour le menu principal. Le menu est affiché comme le schéma ci-dessous avec des rectangles pour représenté les boutons, sur lesquelles sont inscrit les textes qui indiquent leurs fonctions.



Le menu se compose ainsi :

- Jeu
    - Charger (charge une partie)
    - Nouveau (commence une nouvelle partie)
- Options
    - Coop (permet de jouer en multijoueurs)
        - Serveur (indique que nous serons un serveur)
        - Client (indique que nous serons un client)
    - Solo (permet de jouer tout seul)
- Quitter (quitte le programme)

4.2 Lobby

Le lobby est une suite d'interface qui permet au joueur de se préparer pour l'expédition. Elle est composée de 3 modules : le premier sert à trier son inventaire, le second lui permet de choisir le lieu qu'il va explorer, et il pourra acheter des objet à un marchand dans le dernier. L'inventaire du joueur, un tableau de structure objet, y est initialisé. La structure objet est composée du nom de l'objet, du nombre présent dans l'inventaire, de sa valeur en argent et de sa valeur en score.

4.2.1 Trie

Le premier module du lobby permet de trier son inventaire. Les objets de l'inventaire du joueur apparaisse un par un sur l'écran. On attend que le joueur choisisse quoi faire de son objet avant de passer au suivant. Il a le choix entre garder l'objet, l'échanger contre des points ou contre de l'argent.Une fois que tous les objets sont triés, on passe au choix de la carte.

4.2.2 Choix de carte

L'interface est constituée d'une carte du monde avec des drapeaux symbolisant les différents lieux explorables. On peux naviguer entre les différents drapeaux pour sélectionner la zone de notre choix. la valeur de la zone sélectionnée sera utilisée plus tard pour générer la carte. Une fois ce choix effectué, on passe au commerce.

4.2.3 Commerce

Dans cet dernière étape du lobby, le joueur pourra acheter des objets à un marchand. L'inventaire du joueur et celui du marchand sont affichés sur l'écran. Le joueur peut déplacer un curseur sur l'inventaire du marchand. Il peut acheter les objets du marchand tant qu'il a assez d'argent. Les objets achetés sont alors transférés dans son inventaire.

4.3 Carte

La carte est ce sur quoi l'utilisateur va se déplacer en jeu afin de récolter des objets ou combattre en expédition, elle est composé de case en hexagone ayant chacun un type (eau, plaine, forêt, etc). Il y a une case ou on apparaît et une case de fin pour finir l'expédition.

4.3.1 Génération

La génération de la carte se fait case par case avec un nombre de cases par carte défini ainsi qu'un type de carte. Pour chaque case généré on lui assigne un type de case aléatoirement (influencer par le type de la carte et par ses voisins) et elle a une chance défini de contenir un ennemi ou un objet. A la fin on place aléatoirement une case de début et de fin.

4.3.2 Pathfinding

Dans ce jeu on utilise le pathfinding pour la génération de la carte afin de vérifier que la carte soit jouable (pas de carte impossible à finir a cause des cases d'eau) ou soit trop facile (nombre de cases entre le début et la fin trop court). On s'en sert aussi pour la gestion des déplacement pour savoir quelle cases sont accessible a l'utilisateur en un déplacement.

4.3.3 Utilisation

On gère dans cette partie le déplacement du personnage sur la carte ainsi que leur affichage sur l'écran en fonction de la position de la caméra. Le joueur doit simplement cliquer sur une case de la carte pour y déplacer le personnage et recentrer la caméra sur celui-ci.

4.4 Combat

Le système de combat est une reprise du système de Xenoblade Chronicles un jeu sorti le 10 juin 2010 sur la Wii.

On contrôle, dans un environnement en 2D vu de dessus, une équipe de trois personnages qui vont se battre contre un ou plusieurs ennemis. Pour gagner il faut réduire les PV des ennemis à zéro. Si les PV des trois personnages de notre équipe sont réduits à zéro on perd.

4.4.1 Statistiques

Les personnages et les ennemis possèdent plusieurs statistiques à prendre en compte durant le combat :
- Point de Vie.
- Force : Détermine la puissance des auto-attaques et des compétences physiques.
- Magie : Détermine la puissance des compétences magiques.
- Défense physique : Détermine la résistance aux dégâts physiques.
- Défense magique : Détermine la résistance aux dégâts magiques.
- Vitesse d'attaque : Détermine la fréquence des auto-attaques.
- Critique : Détermine le pourcentage de chance qu'une auto-attaque ou qu'une compétence provoque un coup critique.
- Agilité : Détermine le pourcentage de chance d'esquiver une attaque.
- Dextérité : Détermine le pourcentage de chance de toucher sa cible.
- Portée : Détermine la portée des auto-attaques.
- etc.

4.4.2 Auto-attaques/Arts

Pour réduire les PV des ennemis on utilise deux moyens
- Les auto-attaques
- Les arts

Les auto-attaques sont des attaques que les personnages réalisent automatiquement quand ils sont à portée de l'ennemi qu'ils ciblent. Les dégâts que les auto-attaques infligent sont déterminés par la force de l'attaquant ainsi que la défense physique de la victime. Les auto-attaques sont réalisées de façon régulière à une fréquence déterminée par la vitesse d'attaque.

Les arts sont des compétences spéciales utilisables au cours du combat. Les arts peuvent réaliser différentes choses :
- Infliger des dégâts.

- Soigner les membres de l'équipe.
- Améliorer les capacités des membres de l'équipe.
- Réduire les capacités des ennemis.

Les arts ne peuvent pas être utilisés plusieurs fois à la suite, après leur utilisation ils ont un temps de recharge durant lequel ils ne peuvent pas être utilisés. Les arts ont des effets bonus, en fonction de l'orientation du personnage par rapport à l'ennemi au moment où ils sont utilisés, par exemple certains arts font plus de dégâts quand ils sont utilisés dans le dos de l'adversaire.

Quand des dégâts sont infligés ou quand on soigne des personnages, etc, le nombre de dégâts ou de soins apparaît au dessus du personnage ou de l'ennemi concerné.

4.4.3 Hostilité

Les ennemis disposent uniquement des auto-attaques pour infliger des dégâts à l'équipe.
Les ennemis vont cibler un personnage en particulier grâce au système d'hostilité.
Quand on inflige des blessures à un ennemi, quand on soigne notre équipe, etc, on augmente la valeur de notre hostilité. Chaque ennemi va donc cibler le personnage qui lui semble le plus menaçant. Il est impératif de gérer cette valeur afin que les ennemis attaquent les personnages les plus résistants.

5 Réalisation

5.1 Menu

Le menu fonctionne exclusivement via la SDL. On crée des `SDL_Rect` pour chaque bouton et leurs coordonnées sont calculés afin de les centrer. On utilise une variable `nb_choix` qui contient le numéro du choix actuel de l'utilisateur et on utilise un switch quand on valide.

`nb_choix` varie en fonction de l'appuie des touches grâce a la variable `state` qui contient le statut du clavier. En fonction de `nb_choix` et de la validation on appelle les fonctions adéquates tel que fonctionJeu() ou fonctionOption().
Dans le menu les rectangles et les textes sont contenus au même endroit dans un tableau de structure nommé `affichage` qui contient un rectangle et un texte. Pour afficher le menu on parcours ce tableau pour les afficher un par un et les coordonnées du rectangle de choix (le rectangle rouge) sont calculés en fonction de `nb_choix`.

5.2 Lobby

Les différentes interfaces du lobby sont gérées avec SDL et les actions du joueurs se font au clavier grâce à `SDL_Event`. Chaque étape possède sa fonction d'affichage.

5.2.1 Trie

Le tableau inventaire fournis en paramètre de la fonction du trie de l'inventaire contient tous les objet existant dans le jeu. Chaque objet possède un nom, une valeur en argent et en score ainsi que son effectif dans l'inventaire. La fonction va afficher les objet présents dans l'inventaire du joueur un par un, en ignorant ceux dont l'effectif est 0. Le joueur peux décider d'échanger l'objet contre du score en appuyant sur la touche 1 ou de le vendre en appuyant sur la touche 3. L'effectif de l'objet diminue donc et on incrémente la valeur d'argent ou de score du joueur. Si il décide de garder l'objet en appuyant sur la touche 2, l'effectif afficher dans la fonction diminue mais celui enregistrer dans le tableau ne change pas (pour qu'il reste présent dans l'inventaire). La fonction boucle tant que l'utilisateur n'a pas effectuer d'action pour chaque objet. On test la valeur de state à chaque itération de la boucle en attendant un évènement au clavier.

5.2.2 Choix de carte

Les deux images nécessaires à l'affichage (le drapeau et la carte) sont chargées dans des `SDL_Surface`. La carte est afficher en boucle avec les drapeaux dispersés dessus. A chaque itération, on teste la valeur de state pour vérifier les actions de l'utilisateur. Une variable correspond au drapeau sélectionner par le joueur. Elle est s'incrémente ou se décrémente quand l'utilisateur appuie sur les flèches directionnelles du clavier. On affiche le type de carte à coté du drapeau sélectionné. L'utilisateur peut valider son choix en appuyant sur entrée. La fonction retourne alors un entier correspondant au type de carte sélectionné.

5.2.3 Commerce

Dans cette fonction, on utilise un second inventaire pour le marchand en plus de celui du joueur. La fonction boucle tant que l'utilisateur ne quitte pas en appuyant sur echap. A chaque itération, on appelle la fonction d'affichage de l'inventaire pour le marchand et pour le joueur. Elle prend en paramètre un tableau d'objets, la fenêtre SDL, ainsi que les coordonnée et la taille du `SDL_Rect` dans lequel on veut afficher l'inventaire. On affiche des cases blanches dans cette emplacement. On parcourt ensuite l inventaire passé en paramètre et on affiche les objet présent dedans dans les cases blanches en ignorant les objets dont l'effectif est 0. On affiche ainsi leur nom, leur prix et leur effectif. De retour dans la fonction commerce, une image de curseur a été chargé dans un `SDL_Rect`. Celui-ci s'affiche dans l'inventaire du marchand. Une variable correspond à sa position dans l'inventaire et l'utilisateur la fait varier en utilisant les flèches du clavier (on vérifie les évènements du clavier a chaque itération de la boucle). Quand elle varie, on l'incrémente ou la décrémente tant qu'elle correspond à un objet dont l'effectif est 0, pour ne pas que l'indice corresponde à un objet non présent dans l'inventaire. Si il a assez d'argent,

l'utilisateur peut appuyer sur entrée pour acheter un objet. L'effectifs dans l'inventaire du marchand diminue donc, l'objet apparaît dans l'inventaire du joueur si il n'était pas présent (sinon son effectif augmente), et l'argent du joueur diminue du prix de l'objet. Si c'était la dernière itération de l'objet dans l'inventaire du marchand, la valeur de l'indice est incrémentée pour réafficher l'inventaire du marchand sans case vide.

5.3 Carte

La carte est un pointeur sur `mapt_t` une structure dynamiquement alloué qui contient un type de carte et une matrice de case `case_t` qui elle même contient un type, des coordonnées ainsi que des valeurs utiles.

5.3.1 Génération

Pour générer la carte on appelle la fonction de création de carte avec le type de carte en paramètre, cette fonction elle même appelle d'autres fonctions pour générer complètement la carte.
On commence par générer case par case la carte en ajoutant les valeurs utiles et en les stockant dans la matrice de la carte.
Dans les valeurs utiles il y a le type de case qui lui est générer aléatoirement en regardant le type des voisins de la carte ainsi que le type de la carte en établissant des probabilités pour chaque type.
À la fin on génère une case de début (généralement la première case de la matrice) ainsi que la case de fin aléatoirement (chaque case a la même chance d'être la case de fin que les autres).
Pour afficher la carte on parcours case par case la matrice de la carte et on affiche via la SDL une image en fonction du type de la case dont les coordonnées sont calculés en fonction de leurs coordonnées dans la matrice.

5.3.2 Pathfinding

Le pathfinding fonctionne grâce à la valeur utile nommé `path` dans les cases ainsi que grâce à une file. On utilise des fonctions qui nous renvoie les cases de début et de fin et on cherche à aller du début à la fin. En partant du début, on regarde la case sur laquelle on est, et on regarde les voisins de cette case, si un voisin est valide (que c'est une case dans la matrice et que ce n'est pas une case d'eau) et que sa valeur de `path` est égale à zéro, on l'ajoute à la file et sa valeur de `path` devient égale à la valeur de la case dont on regarde les voisins plus un. On fait ça pour chaque voisin puis on enlève une case de la file et on recommence avec celle-ci, jusqu'à la fin (ou jusqu'à ce que la file soit vide). Au final la valeur de la case de fin est égale à soit la distance début-fin ou alors à zéro (indiquant alors qu'il y a n'y a pas de chemin possible).

5.3.3 Utilisation

La fonction d'affichage du personnage prend en paramètres le sprite du personnage chargé dans un `SDL_Surface`, ses coordonnées en x et en y, la caméra (un `SDL_Rect`), et la fenêtre SDL. Elle affiche le personnage aux coordonnées voulues en fonction de la position de la caméra. Le déplacement du personnage va s'effectuer à l'aide du clique de la souris. Quand on clique sur la carte, la caméra va se déplacer sur la position de notre souris. La fonction centrage va alors comparer les distances entre chaque case de la carte et la caméra grâce à leurs coordonnées et retourner la case la plus proche de là où l'utilisateur à cliqué. Le personnage et la caméra sont alors positionnés sur cette case.

5.4 Combat

Cette partie du programme est divisée en plusieurs fichiers .c et .h. Dans les .h on définit une structure et les primitives qui lui

correspondent. Dans les .c on écrit le corps des primitives définies dans le .h.

5.4.1 Structures

On dispose de plusieurs structures pour organiser le combat :
- Personnage
- Ennemi
- Art

Ainsi que de plusieurs tableaux :
- ennPool contient tout les ennemis que l'on peut rencontrer
- ennemis contient tout les ennemis contre lesquels on est en train de se battre
- equipe contient les trois personnages de l'équipe
- artJeu contient les huit arts qu'utilisent les personnages - dgtsTxt contient tous les textes informatifs (dégâts, soin, esquive, coup critique, etc) durant le combat

5.4.2 Organisation

Le programme est organisé comme ci-dessous :
- Détection des évènements clavier à l'aide de SDL.
- Utilisations de différentes fonctions qui gèrent le combat (auto-attaque, utilisation des arts, ordre que l'on donne à ses alliés, etc).
- Déplacement des personnages.
- Affichage de tous les éléments.
- Destruction des éléments temporaires.

5.4.3 Réseau

Le jeu peut se jouer en solo ou à trois joueurs. Quand on joue en coop on définit un serveur et deux clients. Quand le personnage que le serveur contrôle se déplace, lance une auto-attaque, utilise un art, etc, on envoie aux deux clients le résultat de ces actions (la position du personnage quand celui-ci s'est déplacé, les dégâts

infligés par une auto-attaque, si elle a été esquivée, bloquée, si c'était un coup critique, etc), et les clients mettent à jour leur programme. Si c'est un client qui réalise une action alors aucun calcul n'est fait, on envoie au serveur une requête pour qu'il fasse le calcul, puis il renvoie le résultat aux deux clients.

6 Résultats

Les structures Ennemi et Personnage ainsi que les fonctions qu'utilisent ses structures se ressemblent beaucoup (ex : typeCoupPerso et typeCoupEnnemi (qui détermine si une auto-attaque est un coup critique, si elle est esquivée, bloquée, etc)), on pourrait fusionner toutes les fonctions qui se ressemblent.

Les améliorations possibles pour le système de combat : - Différents effets en fonction de l'environnement :

- case d'eau (peu profonde, profonde), sable, pierre, terre, forêt, feu, boue, buisson, neige, glace, trou, obstacle, etc...

- météo (pluie, neige, tempête, foudre, éruption, tremblement de terre)

- Tension des personnages : Attitude d'un personnage en fonction de comment se passe le combat (ex : découragé, terrifié,
- Niveau de relation
- Trait de caractère
- Lien spontané (QTE pendant le combat)

7 Conclusion

Nous avons pu implémenter les principaux modules du programme pendant la conduite de projet. Cependant, nous aurions aimé avoir plus de temps pour finir le jeu.
En ce qui concerne ce projet, le planning prévisionnel n'a pas été respecté, le jeu fonctionne mais n'est pas réellement jouable. Nous pensons que des améliorations sont possibles, comme l'histoire du

jeu, les personnages non-joueurs et les détails tel que le déplacement qui est pour le moment infini. Ce projet nous as permis de travailler en groupe sur un sujet qui nous plaisait tous.
C'était très enrichissant au niveau du code grâce à la découverte et l'utilisation de la SDL et du réseau en C. Nous sommes très heureux d'avoir pu travailler dessus.

8 Annexe

```
valgrind --leak-check=full ./incroyable.c
==20181== 35,392 (1,152 direct, 34,240 indirect) bytes in 12 blocks are definitely lost in
loss record 169 of 184
==20181==    at 0x4C31B25: calloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20181==    by 0x4EBF110: SDL_calloc_REAL (SDL_malloc.c:5344)
==20181==    by 0x4F0F40E: SDL_CreateRGBSurfaceWithFormat_REAL (SDL_surface.c:75)
==20181==    by 0x4F0F40E: SDL_CreateRGBSurface_REAL (SDL_surface.c:166)
==20181==    by 0x53EE671: TTF_RenderUTF8_Solid (SDL_ttf.c:1309)
==20181==    by 0x53EE9C9: TTF_RenderText_Solid (SDL_ttf.c:1275)
==20181==    by 0x111C5C: hudEnnemi (ennemi.c:376)
==20181==    by 0x10C592: main (incroyable.c:1342)
```

`Une SDL_Surface n'était pas libérée après son utilisation.`

```
valgrind --leak-check=full ./izi.c
==22832== 8 bytes in 1 blocks are definitely lost in loss record 5 of 1,299
==22832==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22832==    by 0x109F98: main (izi.c:80)
```

`Un pointeur n'était pas libéré après son utilisation.`

Références

[1] Lien du GitHub du Projet