Compte rendu Bataille en forêt

Le Roux Amélie, Gonichon Lucas, Jacquet Julien

Janvier 2019

1 Comment jouer?

1.1 Compiler et exécuter le code

- Utilistation de CMake pour compiler le code:
'sudo apt install cmake cmake-gui'
- Aller dans le dossier /BEF:
'cmake .'
'make'
L'exécutable est placé dans le dossier bin sous le nom 'bef'.

- La librairie graphique choisie est la SFML, il est nécessaire de l'installer pour compiler le code.
'sudo apt update'
'sudo apt install libsfml-dev'

Dans la version rendue, le programme est déjà compilé, il suffit de rentrer bin/bef dans le terminal. Dans l'arboresence du projet vous devez vous trouver dans le repertoire BEF. A tout moment la touche "Echap" ou la croix permettent d'arrêter le programme.

Une fois lancé on a accès à un menu, sélectionner "Play" pour jouer une partie, Create pour générer de nouveaux niveaux et Tests pour lancer une batterie de tests. Le résultat des test apparaît dans le terminal. Pour se déplacer dans le menu on peut utiliser "Z S". Appuyer sur "Entrée" pour sélectionner.

1.2 Le mode éditeur/create -fonctionnel-

Dans l'éditeur vous pouvez charger un ancien terrain ou en créer un nouveau. Il est possible de sauvegarder le terrain que vous avez créé afin de jouer dessus ou de le rééditer plus tard. Le mode éditeur est implémenté avec une interface graphique.
En mode création appuyer sur "A" permet d'ajouter une entité depuis le terminal (grosarbre,petitarbre,grosrocher,petitrocher,bazooka,pistolet), "E" supprime l'entité dans le carré de sélection rouge, "L" charge le nom du fichier entré dans le terminal, et enfin "W" enregistre le niveau créé avec un nom

spécifié dans le terminal.
Le catalogue est composé des entités plaçables suivantes:

1. petitrocher
2. grosrocher
3. petitarbre
4. grosarbre
5. pistolet
6. bazooka

Les personnages ne sont pas plaçables car toujours placés aléatoirement en début de partie.

1.3 Le mode jeu -partiellement fonctionnel-

Si play est sélectionné un sous menu s'ouvre, appuyer sur "L" et tapper le nom d'un niveau dans le terminal (ex: default) appuyer sur "Entrée", une fois le niveau chargé, il faut choisir un mode de jeu, appuyer sur "I" pour le mode Joueur contre IA, et "P" pour le mode Joueur contre Joueur. Si aucun niveau n'est choisi le terrain sera vide.

Utiliser "Z Q S D" pour faire tourner un personnage, "Entrée" pour avancer dans la direction actuelle, et "Espace" pour tirer tout droit. Ce mode est partiellement fonctionnel car le déplacement du joueur ne fonctionne pas à l'affichage, ce défaut est expliqué dans la partie 3: Difficultés rencontrées. Le déplacement est cependant fonctionnel hors affichage et testé dans la batterie de tests. Sélectionner "Tests" dans le menu pour les lancer.

2 Détails sur le code

2.1 La classe Terrain

Pour représenter le monde dans lequel nos personnages évoluent, nous avons décidé de créer une matrice de pointeurs sur des objets qui composeraient ensuite le niveau. Les cases vides sont alors du sol. Pour cela nous avons utilisé les packages ¡memory¿ et ¡vector¿. Memory nous permet d'utiliser les smart pointers , qui ont l'avantage par rapport aux pointeurs classiques de gérer l'allocation et la désallocation mémoire de façon automatique, et apportent certaines fonctions utiles pour ce que nous voulons faire. Vector permet de créer des vecteurs d'objets dynamiques, avec encore une fois une gestion de la mémoire automatique. Ces deux structures utilisant des templates, cela nous permet d'avoir une matrice dynamique au niveau de son contenu, et la gestion de la mémoire au sein de ces structures permet de stocker des objets de tailles différentes dans la

matrice (les pointeurs stockés dans la matrice pointent vers l'exterieur de cette dernière). Ainsi, l'intérêt de pouvoir stocker des objets de tailles différentes est intéressant dans notre cas, nous souhaitions pouvoir stocker personnages, obstacles et armes qui ont tous des attributs différents même s'ils héritent tous de la même classe : entité.

2.2 Un FileManager pour la sauvegarde

Le FileManager est une classe permettant de gérer les fichiers d'entrée et de sortie du jeu. Il permet de charger un terrain ainsi que de charger le catalogue d'obstacles et arme plaçables sur la matrice lors de l'édition. Le format utilisé pour la sauvegarde d'un terrain est une le suivant:

TailleX TailleY
x1 y1 typedentite1 nomentite1
x2 y2 typedentite2 nomentite2
...

Le format de sauvegarde type pour une entité est le suivant:
Pour un obstacle,
nomobstacle force.
Pour une arme,
nomarme force portee.

2.3 Utilisation de la librairie graphique

La bibliothèque que nous avons choisis d'utiliser est la SFML (https://www.sfml-dev.org/index-fr.php), elle nous a paru plus simple à utiliser que Qt, SFML est écrite en C++. SFML gère aussi les entrées clavier pour le jeu.
Fonctionnement:
On crée une fenêtre graphique, puis on fait une grande boucle qui sera notre boucle de jeu, cette boucle vérifie que la fenêtre est ouverte. Dans notre boucle de jeu il y a une plus petite boucle qui gère les évènement, notamment les entrées clavier. Pour la partie affichage à chaque tour de boucle de jeu on efface la fenêtre graphique et on dessine les différents éléments activés. Enfin on affiche ce qui à été dessiné. Pour dessiner le terrain on dessine des cases que l'on va texturer avec des sprites libres de droits (License CC0) on les positionne ensuite. Le sol est un mélange de deux textures qui sont aléatoirement choisies au moment de leur placement ce qui donne un effet continu. Par dessus le sol on va venir dessiner nos entités. Le reste de l'affichage est principalement du texte on va donc définir une police, un texte, et une position. La partie graphique est également la partie qui va venir rassembler toutes les autres, c'est pourquoi au niveau de la gestion des événements la majeure partie de nos classes sont utilisées.

3 Difficultés rencontrées

Actuellement, lorsqu'on lance une partie (Joueur vs Joueur ou Joueur vs IA), si on essaye de déplacer un personnage, ce dernier disparaît de l'écran à partir du moment où il se déplace. Ce problème n'a été détecté que très tard (une fois toutes les fonctions d'affichages terminées), car en effet si on observe l'état de la matrice une fois le personnage déplacé, il se trouve bien à la position voulue. De même, lorsqu'on inspecte les coordonnées contenues dans le personnage, ces dernières sont bien mises à jour. Du point de vue jeu, sans affichage, nous pensions que tout marchait pour le mieux. En réalité ce n'est pas tout à fait vrai : en effet les coordonnées du personnage sont bien mises à jour et le personnage à la case A passe à la case B ... sauf que ce n'est plus le même personnage. Cela est dû à une fonction particulière : terrain::ajoutEntite.
Cette fonction prend en argument une entité (ou toute autre objet d'une classe héritée), et en place une copie dans la matrice du jeu. Le personnage passe donc bien de la case A à la case B, mais ce dernier n'est alors qu'une copie de sa version en case A. Lorsqu'on affiche les personnages à l'écran, la fonction d'affichage prend en argument un terrain (= la matrice de jeu avec tout ce qu'elle contient) ainsi que 2 personnages (J1 et J2 ou J1 et IA). Tant que les personnages ne se déplacent pas, ils restent le même objet , donc la fonction affiche correctement les personnages sur la grille. Mais une fois qu'ils sont mis à jour, des copies les remplacent et eux deviennent alors obsolètes. La fonction n'affiche plus les personnages, ne les trouvant simplement plus.

Pour régler ce problème, nous avons envisagé trois solutions :

1. La première étant de réécrire la classe personnage pour qu'elle soit indépendante de la grille, et ainsi garder le même objet tout au long de l'exécution du programme. Il survolerait alors la grille au lieu d'en être une partie intégrante.

2. La deuxième était de modifier la fonction terrain::ajoutEntite afin qu'elle ne place plus une copie de l'objet mais l'objet lui-même dans la grille, en modifiant quel pointeur pointe sur cette entité.

3. La dernière solution consistait à modifier la fonction d'affichage pour qu'elle ne prenne plus de personnages en arguments mais uniquement la grille. Cette solution pose un autre problème : avec deux personnages dans la grille, comment les différencier lors de l'action du joueur ? Cela impose de revoir toute la gestion des inputs clavier et des types de chaque entité.

Ces trois solutions sont envisageables mais sont toutes longues à mettre en place, et nous manquons de temps avant le rendu au moment auquel nous écrivons ces lignes, c'est pourquoi cette bogue ne sera pas résolue au moment du rendu, malgré le fait que nous en connaissions sa source.