Sullivan Honnet

Jules Vittone

26/01/2019

**Rapport succinct du projet C++**

Présentation générale de nos choix

Pour ce projet, nous avons fait plusieurs choix dès le début du projet. Notre objectif était qu’il soit très facile de rajouter de nombreux modules car nous voulions rajouter de nombreuses choses. Le premier objectif était que des unités puissent être ajoutées sans problème. Pour ce faire, nous avons créé des super-classes : Unité, Bâtiment et Soldat.

![Une image contenant texte, tableau blanc

Description générée automatiquement]()Ces super-classes sont utilisées par le reste du programme qui n’a pas à savoir quelles sont les différentes unités qui existent. Actuellement, je voudrais rapatrier un certain nombre de fonction dans différents morceaux du programme dans ces super-classes mais par manque de temps ça ne sera pas possible. Par ailleurs, la plupart des fonctions sont redéfinies dans les classes-filles.

Nous avons également choisi de rajouter des âges. Actuellement un seul âge existe mais il ne serait pas très compliqué de rajouter plusieurs autres âges. Il est à noter que chaque changement d’âge entraînera un changement de musique.

Nous n’avons pas prévu que les mécaniques du joueur puissent évoluer grandement, la plupart des mécaniques du joueur sont ainsi fixées et compliquées à changer.

Il nous semblait en revanche important de pouvoir modifier la taille du terrain, c’est pourquoi la taille du terrain est une variable générale qui peut être modifiée à l’envie sans que cela pose de problème. De même, nous avons choisi de faire un terrain composé de case qui sont une classe spécifique car nous voulions nous offrir la possibilité de créer des cases avec des effets spéciaux. Ainsi, il est possible en créant une classe qui hérite de Case de très rapidement créer un type de Case spéciale.

Enfin nous voulions pouvoir rajouter des bâtiments supplémentaires et des unités pour construire et réparer ces bâtiments mais nous avons manqué de temps pour le faire. Mais on retrouve dans le code tous les éléments qui permettrait de le faire, notamment une super-classe Bâtiment qui hérite de la même que Soldat, Unité et la possibilité de stocker des Bâtiments en plus des Soldats sur les Cases. De même, les joueurs possèdent une base, une liste de soldat et une liste de bâtiment. La base sert à savoir quel est la base du joueur et la liste de soldat quels sont les unités de ce joueur. La liste de bâtiment aurait eu le même rôle que la liste de soldat si nous avions eu le temps de les rajouter.

La classe Terrain est un singleton, en effet, le constructeur ne peut pas être appelé en dehors de la classe mais il est possible d’obtenir une instance de Terrain. L’intérêt de ce Terrain est double, premièrement, il centralise toutes les informations qui peuvent être nécessaire n’importe où dans le programme, deuxièmement, il sert à lancer les différentes fonctions de toutes les classes qui existent dans le projet. Par exemple, c’est lui qui lance le tour du joueur 1 puis du joueur 2.

Cahier des charges et solutions

Objectif donné (Objectif bonus) : Solution (Solution bonus)

Avoir un ou deux joueurs humains : Créer une classe Joueur qui offre une interface pour que le joueur puisse entrer ses choix. Et laisser la possibilité au lancement du jeu de choisir son nombre de joueur.

Avoir une IA (La rendre intelligente) : Créer une classe IA qui hérite de joueur et qui se contente de créer des fantassins dès qu’elle a assez d’argent pour en créer un. (Non fait)

Avoir une aire de jeu de douze cases (Rendre la taille du terrain modifiable) : Créer un vector de Case, chaque case pouvant contenir un joueur et un bâtiment, ce vector étant contenu par le Terrain. (Créer une variable globale qui peut être modifiée dans le code et qui change la taille du terrain.)

Avoir quatre types unités différentes + une base (Avoir plus de bâtiments et d’unités) : Créer des classes pour chaque type différents (sauf supersoldat voir ci-dessous) et les faire hériter des mêmes super-classes pour qu’elles aient les mêmes fonctions. (Non fait mais optimisé pour que ce soit facile.)

Avoir des fantassins qui se transforment en supersoldat une fois qu’ils ont tués un ennemi : Utiliser un booléen qui devient vrai si le fantassin a tué un ennemi et qui modifie le comportement du fantassin.

Avoir une portée variable selon l’unité (Permettre à l’utilisateur de modifier toutes les valeurs à volonté) : Créer une fonction que l’on redéfinit selon le comportement de l’unité et qui détermine quels sont les soldats touchés quand l’unité attaque. (Lecture du fichier de l’époque courante contenu dans Donnees/Epoque.)

Faire jouer le tour des unités : Créer une fonction pour chaque étape du tour des unités et les lancer à tour de rôle.

Faire s’affronter des unités : Créer des fonctions d’attaque qui sont appelées par les fonctions de portée.

Finir le jeu et désigner un vainqueur : Vérifier à chaque tour de jeu si un des joueurs est mort et si le dernier tour est atteint. Si l’un des deux est mort, désigné l’autre vainqueur et si le nombre de tours max est atteint annoncer l’égalité.

Permettre de sauvegarder (Permettre de modifier facilement la sauvegarde) : Créer un fichier .txt ou supprimer celui qui porte déjà ce nom et écrire dedans ce qui est nécessaire à son fonctionnement. (Afficher en claire les informations pour que l’utilisateur puisse les modifier)

Objectif personnel : Solution

Rajouter de la musique : Fait avec un changement de musique automatique à chaque changement d’époque. Utilisation de la librairie SFML pour le faire.

Rajouter des bruitages : Non fait car impossible de trouver une bibliothèque sonore satisfaisante mais assez simple grâce à SFML.

Rajouter une interface graphique : Non fait car manque de temps mais également disponible dans SFML.

Rajouter des époques : Fait avec un fichier .txt qui contient les paramètres des unités de l’époque et un répertoire de musique pour l’époque.

Cahier des charges final

Un ou deux joueurs, IA, aire de jeu de taille variable, quatre unités, une base, transformation du fantassin en supersoldat, possibilité pour l’utilisateur de modifier toutes les variables à volonté, capacité des unités à jouer chaque étape de leur tour à tour de rôle, capacité des unités à attaquer, définir un point d’arrêt au jeu et un vainqueur, permettre de sauvegarder et charger ses sauvegardes, possibilité de bidouiller ses sauvegardes et ajout de musique et d’époque.

Le Makefile

Le problème dans le makefile est le fait que l'on utilise une librairie extérieure, l'installation de cette librairie n’a pas posé de problème sur notre IDE : Code::Blocks, or le sujet nous impose le fait d'utiliser un makefile pour compiler le projet.

Il a fallu donc refaire tous les liens du projet et refaire toutes les dépendances entre les différents fichiers .o, étape qui a mis du temps, et l'installation de la librairie SFML. Cette dernière est la raison pour laquelle le makefile ne fonctionne pas. En effet, les multiples dépendances de la librairie entre elle et aussi avec mingw32. Pour résoudre ce problème on a appliqué la même méthode que pour son installation sur Code::Blocks en mettant les flags de Code::Blocks dans le make, tout en essayant de faire attention à la localisation des différents fichiers de SFML. Les erreurs que l'on obtient sont de la forme "skipping incompatible SFML/lib/libsfml-audio.a when searching for -lsfml-audio" ce qui entraine d'autre erreurs (car il ne les lies pas) " undefined reference to `sf::seconds(float)'"

Liste des classes et leur rôle

|  |  |
| --- | --- |
| Terrain | Centralise toutes les informations du jeu et les transmet aux différents objets |
| Case | Permet de stocker la position absolue des différentes unités et bâtiments |
| Joueur | Permet de demander au joueur ses actions et s’occupe de gérer les différentes unités du joueur |
| IA | S’occupe de gérer les différentes unités de l’IA et choisit les unités à invoquer |
| Unité | Centralise les fonctions et les variables nécessaires aux bâtiments et soldats |
| Bâtiment | Centralise les fonctions et les variables nécessaires aux bâtiments |
| Base | Sert à définir la base des deux joueurs |
| Soldat | Centralise les fonctions et les variables nécessaires aux soldats |
| CorpsACorps | Centralise les fonctions et les variables nécessaires aux soldats qui se battent au corps à corps |
| Distance | Centralise les fonctions et les variables nécessaires aux soldats qui se battent à distance |
| Fantassin/Catapulte /Archer | Différents types de soldats disponibles |
| Epoque | Centralise les fonctions et les variables nécessaires aux époques |
| Préhistoire | Seule époque disponible pour le moment |

Problème rencontré

Nous avons rencontré quatre problèmes au cours de notre travail :

* Le premier fut les cases qui devaient à l’origine juste stocker des soldats mais qui durent finalement aussi accueillir des unités et qui ont posés un certain nombre de problème au niveau de la surcharge de certaines fonctions. Ainsi que dans leur stockage dans un vector qui se trouvait dans Terrain qui était construit avant le début du main avant de créer la fonction init.
* Le second fut pour le chargement des sauvegardes qui fut particulièrement ardu car toutes les valeurs pouvaient être modifiée par l’utilisateur sans que cela doive nous poser de problème.
* Le troisième fut de réussir à jouer plusieurs musiques différentes. En effet, SFML offre seulement la possibilité de jouer une musique en boucle, il nous a fallu utiliser cette capacité sans pour autant garder toutes les musiques prêtes à être jouées car c’est extrêmement gourmand en mémoire et en calcul. Finalement, la solution fut de garder un pointeur vers le fichier et de charger uniquement la musique en cours puis de refermer le fichier et d’ouvrir le suivant. Le changement entre deux musiques est un peu plus long mais on économise de la puissance. En outre le changement de musique entre deux époques est fluide car on ouvre le nouveau répertoire qu’après la fin de la musique courante.
* Le dernier et que nous n’avons pas pu résoudre est celui du makefile d’une part car nous n’avons essayé sérieusement de le faire que la dernière semaine, pensant cela relativement facile, d’autre part car nous n’avons absolument aucune connaissance des makefile ni aucune expérience.