Rapport projet C

The Binding Of Briatte

Table des matières

Table des matières
Introduction
Une analyse rapide de l'application
Structure du projet
- I - Menu principale et sous menus
- II - CRUD des pièces du donjon
- III - CRUD des objets du jeu
- IV - CRUD des monstres du jeu
- V - Génération des étages

Dossier d'installation de votre application
Bilan du projet

Introduction

Étudiants: Vincent PHAN, Jonathan MALLIA et Agustin GOMEZ DEL TORO

Le projet **"The Binding of Briatte"** a été mené en utilisant le langage de programmation C. Le but du projet était de créer un jeu dans lequel le joueur doit s'échapper des griffes de ses animaux domestiques satanistes.

Une analyse rapide de l'application

La première étape du projet a été de définir le concept du jeu et d'établir les règles de base. Nous avons ensuite décidé de représenter l'état du jeu en utilisant des **structures de données pour stocker** les informations sur les **pièces, les items et les monstres.**

Pour ce faire, nous avons développé une interface utilisateur simple et intuitive qui permettait aux joueurs de créer les différents **monstres et items,** de sélectionner ceux qu'ils souhaitent utiliser sur les combats. Le déroulement du jeu était développé avec des algorithmes basés sur les caractéristiques des items choisis.

Nous avons commencé par la partie 1 et avons avancé au fur et à mesure.

Nous avons pris la décision de faire le CRUD des pièces de jeu d'abord pour ensuite passer aux Items et monstres.

Structure du projet

Pour la structure de données, nous avons organisé notre projet de la manière suivante:

LE DOSSIER RESSOURCES

Dans ce dossier, nous avons mis les fichiers texte **(.mtbob / .rtbob / .itbob).** Ces fichiers permettent de stocker les données du jeu.

LE DOSSIER SOURCES

Dans ce dossier nous avons créé 2 dossiers, C et H.

Le dossier H regroupe tous les prototypes de nos fonctions et nos structures.

Le dossier C regroupe tous les codes sources de toutes les fonctions de notre programme.

A la racine du projet, nous avons placé le fichier **main** qui permet d'assembler toutes les différentes fonctions pour le lancement du jeu.

I - Menu principale et sous menus

Nous avons décidé de faire un menu principal et des sous-menus différents selon l'option choisie.

Le menu principal se trouve dans le fichier **main** à la racine du projet.

```
while (choice != -1)
    {
        do{
            printf("*********************************************************\n");
            printf("**************** The Binding of Briatte ***************\n");
            printf("*********************************************************\n\n");
            printf("1 - Interface des Pieces\n");
            printf("2 - Interfaces des Items\n");
            printf("3 - Interfaces des monstres\n");
            printf("4 - JOUER\n");
            printf("5 - Fermer le programme\n");
            printf("Faites votre choix et appuyé sur entrer : ");
            scanf("%d",&choice);
        }while(choice<=0 || choice>5);
        switch (choice) {
            case 1:
                pieceMain();
                break;
            case 2:
                itemMain();
                break;
            case 3:
                monsterMain();
                break;
            case 4:
                game();
                break;
            case 5:
                choice = -1;
                system("clear");
                printf("A bientot...\n");
```

```
                break;
            }
        }
```

Les sous-menus se trouve le dossier C.

Menu des étages ——> etageMain.c

Menu des monstres ——> monsterMain.c

Menu des Items ——> itemsMain.c

Menu des Pieces ——> pieceMain.c

II - CRUD des pièces du donjon

Jonathan s'est occupé de cette partie.

On a dans un premier temps structurer notre Piece. Une Piece est constitué d'un matrice à 2 dimension de caractères, d'un Id, d'une largueur et d'une longueur représenté par un entier.

```
struct Piece {
    char** piece;
    int id;
    int width;
    int height;
};
```

Par défaut, la matrice de caractères contient uniquement les murs ainsi que les quatre portes.

On compte pour cette partie du projet 6 grandes fonction:

- createPiece() : Cette fonction permet de créer une pièce en mémoire
- printPiece(Piece* p, int mode) : Affiche la pièce dans la console, le paramètre mode permet de passe en mode édition de la pièce
- modifyPiece(int id) : Permet de modifier une pièce
- freePiece(Piece *p) : Libère l'espace de la pièce prise dans la mémoire
- deletePiece(int id) : Permet de supprimer une pièce
- addPieceToFole(Piece* p, char* file) : Permet d'ajouter une pièce au fichier

Bien sûr, d'autre fonction existent mais elles sont considérées comme secondaire. Elles aident les fonctions énumérer ci-dessus.

III - CRUD des objets du jeu

Vincent s’occupe de cette partie.

Tous les prototypes du CRUD se trouvent dans le fichier item.h.

Pour le CRUD des items du jeu, nous avons d’abord créé une structure Item. Cette structure est composée d’un nom qui est une chaîne de caractères, de hpMax, shield et dmg qui sont des floats et de ps, ss et flight qui sont des entiers.

```
typedef struct Item {
    char nameItem[20];
    float hpMax;
    float shield;
    float dmg;
    int ps; //bool pour les tirs perçants
    int ss; //bool pour les tirs spectrals
    int flight; //bool pour le vol
} Item;
```

“**nameItem**” correspond au nom de l’objet.

“**hpMax**” correspond au bonus de points de vie donné par l’objet.

“**shield**” correspond au bonus d’armure donné par l’objet.

“**ps**” correspond au bonus de tirs perçants.

“**ss**” correspond au bonus de tirs spectrals.

“**flight**” correspond au bonus de vol.

```
Item* createItem();

void printItem(Item* obj);

void modifyItem();

void freeItem(Item* obj);

void addItemToFile(Item* obj, char* file);

void deleteItem();

void printAllItem();

int foundItem(char nameItem[20]);

float checkFloat();

int checkInt();

Item* randomItem();
```

Voici les déclarations de toutes les fonctions du CRUD. On y compte 11 fonctions :

- createItem() qui permet de créer un objet puis de le retourner.

- printItem(Item* obj) qui prend en paramètre un objet puis l'affiche dans la console.
- modifyItem() qui permet de modifier un Item dans le fichier item.itbob en entrant son nom dans la console.
- freeItem(Item* obj) qui permet de libérer la mémoire allouée à un objet.
- addItemToFile(Item* obj, char* file) qui prend en paramètre un objet et le nom d'un fichier puis va ajouter l'objet dans le fichier en vérifiant qu'il n'y est pas déjà.
- deleteItem() qui permet de supprimer un objet du fichier item.itbob.
- printAllItem() qui permet d'afficher tous les objets contenus dans item.itbob dans la console.
- foundItem() qui prend en paramètre un nom d'objet et nous permet de savoir si l'objet existe ou non dans le fichier.
- checkFloat() qui permet de gérer les cas d'erreurs sur un float lorsqu'il est entré par l'utilisateur, par exemple si une lettre est entrée.
- checkInt() gère les cas d'erreurs sur un int lorsqu'il est entré par l'utilisateur.
- randomItem() qui permet de retourner un objet du fichier aléatoirement.

IV - CRUD des monstres du jeu

Agustin s'occupé de cette partie.

Une structure "Monster" qui declare les différentes caractéristiques dans le fichier "monster.mtbob".

```
struct Monster {
    char nameMonster[20];
    float hpMax;
    int shoot; //bool pour les tirs
    int ss; //bool pour les tirs spectrals
    int flight; //bool pour le vol
};
```

"**nameMonster**" correspond au nom de l'objet.

"**hpMax**" correspond au bonus de points de vie donné par l'objet.

"**shoot**" correspond au bonus de tirs normal.

"**ss**" correspond au bonus de tirs spectrals.

"**flight**" correspond au bonus de vol.

Pour faire ce crud nous avons commencé par la création des monstres avec la fonction **createMonster()** et les afficher en console avec la fonction **printAllMonster()**, une fois cette partie était fini, nous avons développé la fonction **addMonsterToFile()** pour pouvoir afficher les monstres et ces caractéristiques uniquement en TRUE sur notre fichier **monster.mtbob.**

Ensuite nous avons continué notre CRUD avec le reste des fonctions.

Toutes les fonctions dans ce dossier sont essentielles:

```
typedef struct Monster Monster;

Monster* createMonster(); //permet de créer un monster

void printMonster(Monster* obj); //Permet d'afficher

void modifyMonster(); //permet de modifier un monster

void freeMonster(Monster* obj);  //liberer la memoire allué

void addMonsterToFile(Monster* obj, char* file); //ajoute un monster to .mtbob

void deleteMonster(); // permet d’effacer un monster

void printAllMonster(); // Permet afficher tous les monsters sur la console

int foundMonster(char nameItem[20]); // permet de vérifier le nom des monstres

Monster* randomMonster();/permet de choisir 2 monster aléatoirement sur .mtbob
```

V - Génération des étages

Jonathan s’est occupé de cette partie.

Pour générer un étage, on a dû encore une fois se demander comment pouvons-nous structurer cette donnée.

Une structure “Etage” est constituée d’un tableau d’entier contenant les id des salles prise aléatoirement dans le fichier “piece.rtbob”, d’une largeur et d’une longueur représentée par un entier, une matrice à 2 dimensions d’entiers et d’un tableau de “Piece”.

La matrice sera notre “mini-map”.

```
struct Etage {
    int* Idsalles;
    int** etage;
    int width;
    int height;
    Piece** piece;
};
```

Par défaut, la matrice d’entiers contient uniquement des 0.

Les 0 sont remplacés par des espaces lors de l’affichage.

Toutes les fonctions, ici, sont essentielles:

```
Etage* createMap();

int randomId(Etage* e,int maxSalle); // Renvoie un nombre

int testId(Etage* e,int id); //Renvoie si la salle existe dans l'étage

int randomDirections();

void placerSalles(Etage* e); // Permet de placer les salle dans l'étages: le plus gros du travail

void printEtage(Etage* e);

void freeEtage(Etage* e);

int randomNbMonster();

int* randomCoordonnee();
```

Algorithme de générations d’un étage :

- Création d’un étage vide
- On pioche 10 salle dans le fichier et on stocke les Id dans le tableau “IdSalles” de l’étage
- On ajoute les salle spéciale a la fin du tableau représenté par des id négatif
- On place les salle dans la matrice de l’étage
- Pour chaque Pièce aléatoire on prend un nombre aléatoire entre 0 et 6 et on ajoute autant de monstre que ce nombre aléatoire.

Dossier d'installation de votre application

Pour exécuter un fichier exécutable en C depuis le terminal, vous pouvez suivre ces étapes :

1. Ouvrez une fenêtre de terminal et naviguez jusqu'au dossier où se trouve votre fichier exécutable en utilisant les commandes "**cd**" et "**ls**".
2. Une fois que vous êtes dans le dossier (The Binding of Briatte) où se trouve l'ensemble de fichiers, tapez **" gcc main.c sources/C/*.c -o app && ./app "** pour lancer la compilation du fichier et l'exécutable avec la commande.
3. Si vous rencontrez des erreurs lors de l'exécution du fichier, vous pouvez lui donner les droits d'exécution en utilisant la commande "**chmod +x app**".

Il est important de noter que l'exécution d'un fichier exécutable en C depuis le terminal peut varier en fonction de la configuration de votre ordinateur et du système d'exploitation utilisé.

Conclusion de compilation:

Fichier: /The Bidding of Briatte

Compilation: gcc main.c sources/C/*.c -o app

Exécution: ./app

Choisir un menu pour créer ces différents éléments :

1 - Interface des Pièces

2 - Interfaces des Items

3 - Interfaces des monstres

4 - Fermer le programme

Ensuite, vous pouvez commencez à jouer une fois tous ces éléments créés.

Bilan du projet

Ce projet en groupe a permis de partager les idées et les connaissances de chacun, ce qui a enrichi notre réflexion et a abouti à des solutions plus complètes et adaptées au sujet.

Points techniques non résolus ——> Partie 6 et 7

Comme difficulté technique nous avons trouvé compliqué de générer des tailles de salles aléatoirement, donc nous avons seulement une taille déterminée.

On est également tombé sur des problèmes lors de l'éditions des fichiers ressources (item et monster), surtout pour parvenir à respecter la synthaxe du fichier.
Le respect des tirets " - - -" nous a posé problèmes lorsque le nombre d'éléments passait de 9 à 10, un caractère était écrasé lors du changement. Il n'y avait plus de place dans le fichier.

Finalement, nous avons réussi à corriger le problème en décalant d'un caractère vers la droite le contenu du fichier.

Pour la partie 5, on doit encore trouvé un moyen de se déplacé de salle en salle, nous somme tous de même sur la bonne voie.

En quelques mots, "The Bidding of Briatte" est un projet qui nous a permis de continuer à développer nos compétences de développement en C mais aussi sur l'algorithmie. On a pu également développer nos méthodes de gestions de projets et surtout d'utilisation d'outils de versionning comme GIT.

Le git du projet : https://github.com/Jonathan13127/TBOB.git

CORDIALEMENT.

Vincent PHAN, Jonathan MALLIA et Agustin GOMEZ DEL TORO