

# Bilan projet – Malloc World

## I/ Introduction

Ce projet fait suite à la Piscine de C qui a eu lieu la première semaine d'Octobre 2021. Il consistait à développer en C un RPG (nommé Malloc World) avec une version en CLI donc en C natif et une autre en graphique en utilisant SDL.

Nous avons donc démarré ce projet la seconde semaine d'Octobre pour un rendu le 28 Novembre à 23h59 avec notre groupe composé de **Bill LOUIS-CHARLES**, **Mathieu FEREIRA** et **Jonathan FUENTES**

## II/ Analyse de l'application

Cette partie va être plus logue que les autres, nous allons ici détailler les structures et fonctions principales du projet.

Commençons par la structure la plus importante du projet : la structure Game qui contient la totalité des données utiles pour faire tourner le jeu correctement.

```
typedef struct game{
    int height[3];
    int width[3];
    int *onPortal;
    int **currentPos;
    int ***map;
    int ***npcs;
    int *nbNpcs;
    int *isCraftable;
    int startPos[3][2];
    int *xpWin;
    int *mobCount;
    int *currentMap;
    int *itemCount;
    int *craftCount;
    PNJ *pnj;
    Item *items;
    Craft *crafts;
    Mob *mobs;
    Player *player;
}Game;
```

Cette structure a énormément de valeur et a été créée pour limiter les passages des valeurs dans les fonctions (certaines fonctions avaient plus de 15 paramètres ce qui est aberrant) nous avons corrigé cela grâce à cette structure.

Maintenant qu'on a la base nous allons dérouler les fonctions dans l'ordre de l'énoncé pour s'y retrouver plus aisément.

## 1) La map

La première fonction est donc la fonction de génération de map.

```
int** initMap(Game **game, int stage){
```

Cette fonction permet de générer une map (un tableau a deux dimensions) en fonction de paramètre stage qui est passé, car comme nous le savons ce qui doit apparaître sur chaque map dépend du stage sur lequel on est.

Afin de générer la map de manière procédurale nous avons utilisé un algorithme de génération procédural appelé « Diffused limited aggregation » qui a partir du milieu s'expand petit a petit afin que tous les points de la map soit accessible, cela nous paraissait plus cohérent cela donne une sorte de plaine.

Un extrait du code :

```
while(nbZero < (*game)->width[stage-1]*(*game)->height[stage-1] * 0.7){
    i = rand()%(*game)->height[stage-1];
    j = rand()%(*game)->width[stage-1];
    if (i == 0){
        if(j == 0){
            if( (map[i+1][j] == 0 || map[i][j+1] == 0) && map[i][j] == -1) {
                nbZero += 1;
                map[i][j] = 0;
            }
        }else if(j == (*game)->width[stage-1]-1){
            if( ( map[i+1][j] == 0 || map[i][j-1] == 0 ) && map[i][j] == -1) {
                nbZero += 1;
                map[i][j] = 0;
            }
        }
    }
}
```

Nous générons d'abord une map composé de 0 et de -1 puis nous venons placer aléatoirement les « pnj » ce qui comprend les monstres, les ressources et tout autre chose présent sur la map.

Lors du placement des pnj, un autre tableau est repli en parallèle afin de garder en mémoire l'emplacement des pnj pour gérer le respawn plus tard.

```
for(i=0;i<(*game)->height[stage-1];i++){
    for(j=0;j<(*game)->width[stage-1];j++){
        if(map[i][j] == 0){
            if(rand() % (npcSize-count) == 0){
                map[i][j] = npc[count];
                (*game)->npcs[stage - 1][count][0] = i;
                (*game)->npcs[stage - 1][count][1] = j;
                (*game)->npcs[stage - 1][count][2] = npc[count];
                (*game)->npcs[stage - 1][count][3] = 0;
                (*game)->npcs[stage - 1][count][4] = 0;
            }
        }
    }
}
```

## 2) Le joueur

Passons maintenant au joueur en lui-même, il a sa propre structure pour ses données qui est :

```
typedef struct Player{  
    double hp;  
    double hpMax;  
    int exp;  
    int lvl;  
    Item inventory[10];  
    int nbItem;  
} Player;
```

Nous sommes donc capables d'initialiser le joueur avec la fonction :

```
Player *initPlayer(Item *items){  
    Player* player = malloc(sizeof(Player));  
  
    player->hp = 100;  
    player->hpMax = 100;  
    player->exp = 0;  
    player->lvl = 4;  
    player->nbItem = 0;  
    Item item = createItem( id: 0, name: "", type: 0, durability: 0, maxDurability: 0, effect: 0);  
  
    for (int i = 0; i < 10; ++i) {  
        player->inventory[i] = item;  
    }  
  
    addInv( id: 1, player, items);  
    addInv( id: 2, player, items);  
    addInv( id: 3, player, items);  
    addInv( id: 4, player, items);  
  
    return player;  
}
```

Pour tout ce qui est gestion des Items nous avons fait une structure qui est :

```
typedef struct Item{  
    int id;  
    char name[255];  
    int type; //arme, outil, ressource de craft ou soin  
    double durability;  
    double maxDurability;  
    double effect;  
    int amount;  
} Item;
```

Et nous créons au lancement du jeu un tableau d'items que nous récupérons via un fichier item.txt ce qui nous permet de retirer aisément un item du fichier ou d'en rajouter par la suite et évite une fonction avec un switch de 100 lignes.

```
Item* declareItem(int* nbItem){
    FILE* f = fopen( _Filename: "items.txt", _Mode: "r+");

    if (f) {
        int count = 0;
        char buffer[255];
        int id;
        char name[255];
        int type;
        double durability;
        double maxDurability;
        double effect;

        while (fgets(buffer, sizeof(buffer), f)) {
            count += 1;
        }
        Item *items = malloc( _Size: sizeof(Item) * count);
        rewind(f);

        for (int i = 0; i < count; i++) {
            fgets(buffer, sizeof(buffer), f);
            sscanf(buffer, _Format: "id: %d, name: %[^\n], type: %d, durability: %lf, maxDurability: %lf, effect: %lf", &(id),
                name, &(type), &(durability), &(maxDurability), &(effect));
            items[i] = createItem(id, name, type, durability, maxDurability, effect);
        }
        fclose(f);
        *nbItem = count;
        return items;
    } else {
        printf( _Format: "No such file");
        return NULL;
    }
}
```

Exemple de fichier :

```
id: 1, name: Epee en bois, type: 1, durability: 10, maxDurability: 10, effect: 1
id: 2, name: Pioche en bois, type: 2, durability: 10, maxDurability: 10, effect: 0
id: 3, name: Serpe en bois, type: 2, durability: 10, maxDurability: 10, effect: 0
id: 4, name: Hache en bois, type: 2, durability: 10, maxDurability: 10, effect: 0
```

### 3) Le PNJ

```
void addToChest(PNJ *pnj, Player *player, int index){
    if(pnj->chestSize == 0){
        Item* temp = malloc(sizeof(Item));
        temp[0] = player->inventory[index];

        removeItem(player, index, player->inventory[index].amount);

        free(pnj->chest);
        pnj->chest = malloc(sizeof(Item));
        pnj->chest = temp;
        pnj->chestSize = pnj->chestSize + 1;
    }else{

        Item* temp = malloc(_Size: sizeof(Item) * (pnj->chestSize + 1));

        for (int k = 0; k < pnj->chestSize; ++k) {
            temp[k] = pnj->chest[k];
        }

        temp[pnj->chestSize] = player->inventory[index];

        removeItem(player, index, player->inventory[index].amount);

        free(pnj->chest);
        pnj->chestSize +=1 ;
        pnj->chest = malloc(_Size: sizeof(Item) * pnj->chestSize);
        pnj->chest = temp;
    }
}
```

La fonction la plus importante est la fonction d'ajout au chest, celle qui a été la plus compliquée à gérer notamment la gestion du chest infini. Mais globalement cela est fonctionnel et suffit pour l'ampleur de ce projet.

## 4) Les déplacements

Gestion des déplacements grâce à la fonction move

```
void move(Game *game, char dir) {
    int horizontal = 0;
    int vertical = 0;
    switch (dir) {
        case 'l':
            horizontal = -1;
            break;
        case 'r':
            horizontal = 1;
            break;
        case 'u':
            vertical = -1;
            break;
        case 'd':
            vertical = 1;
            break;
        default:
            printf( _Format: "Direction non autorisee\n");
            return;
    }
}
```

En fonction de la direction entrée on incrémente/décrémente horizontal ou vertical

Puis grâce à nextmap on définit vers où va le joueur et en fonction de la case je lance différentes fonctions, c'est notamment ici que l'on peut lancer les combats / la récolte etc etc

```
int nextMap = game->map[(game->currentMap)][game->currentPos[(game->currentMap)][0] + vertical][(game->currentPos[(game->currentMap)][1] + horizontal)];
```

## 5) Le respawn

Pour le respawn, un tableau à trois dimensions a été défini, dans ce dernier du format npc[3][x][5] où x est le nombre de npc (ressources etc) générer procéduralement la première dimension [3] est pour chacune des maps et le [5] sert à stocker les données sur les npc (position, numéro de ressource sur la map, le nombre de tours de respawn et un booléen récolté ou non)

```
void respawn(Game *game){
    int i;
    for (i = 0; i < game->nbNpcs[(game->currentMap)][0]; i++) {
        if (game->npcs[(game->currentMap)][i][4] == 1) {
            game->npcs[(game->currentMap)][i][3] -= 1;
            if (game->npcs[(game->currentMap)][i][3] <= 0 &&
                game->map[(game->currentMap)][game->npcs[(game->currentMap)][i][0]][game->npcs[(game->currentMap)][i][1]] == 0) {
                game->map[(game->currentMap)][game->npcs[(game->currentMap)][i][0]][game->npcs[(game->currentMap)][i][1]] = game->npcs[(game->currentMap)][i][2];
                game->npcs[(game->currentMap)][i][4] = 0;
            }
        }
    }
}
```

## 6) Le leveling

Pour ce qui est du leveling une simple fonction a été faite pour vérifier l'xp du joueur

```
void levelUp(Player *p){
    int check = 0;
    int i;
    if(p->exp >=100 && p->lvl == 1 ){
        p->lvl = 2;
        p->exp -= 100;
        p->expMax = 200;
        p->hpMax += 10;
        check = 1;
    }
}
```

Dans laquelle on enchaîne les if et on manipule les données du joueur.

## 7) Combats

Les combats se gèrent facilement grâce à la structure game qui englobe une grande partie des données nécessaires. Tout commence dans la fonction de move puis évolue dans plusieurs fonctions de fight comme :

```
initStuff(nbWeapons,nbChest,nbPotions,&weapons,&potions,&chest,p,count);
firstStuff(nbWeapons,nbChest,weapons,chest,chosenWeapon,chosenChest,p);
startFight(mob,nbMobs,mobId,currentMob,mobHp,nbWeapons,p,chosenWeapon,nbChest,armor,chosenChest);
while(1){
    printf( _Format: "Que voulez vous faire ?\n  -Attaquer (tapez 1)\n  -Boire une potion (tapez 2)\n  -Fuir (tapez 3)\n");
    scanf( _Format: "%d",&choice);
    switch (choice) {
        case 1:
            attack(p,chosenWeapon,mobHp,nbWeapons,count,weapons,&newWeapons,check);
            break;
        case 2:
            takePotion(nbPotions,p,potions,check);
            break;
        case 3:
            if(runAway(check)){
                return 0;
            }else {
                break;
            }
    }
}
```



Pour ce qui est de la fuite nous avons ajouté un coté réaliste type rpg papier avec un lancé de dé pour réussir a fuir ou non

```
int runAway(int *check){
    int leave;
    int dice;
    printf(_Format: "Vous voulez fuir ? En etes vous sur ?\n    1- Oui\n    2- Non\n");
    scanf(_Format: "%d", &leave);
    switch (leave) {
        case 1:
            dice = 1+(rand()%10);
            printf(_Format: "Vous lancez un d10 pour essayer de fuir, froussard\n");
            sleep(2);
            if(dice > 3){
                printf(_Format: "%d, c'est un echec cuisant\n",dice);
            }else{
                printf(_Format: "%d, Bravo !\n",dice);
                sleep(2);
                return 1;
            }
            break;
        case 2:
            *check = 1;
            break;
        default:
            printf(_Format: "Ce n'est pas une reponse valide\n");
    }
}
```

## 8) La sauvegarde

Un fichier est créé sous le format "pseudo save.txt". Si le fichier n'existe pas, il sera créé dans un dossier "save".

La sauvegarde est composée de 2 fonctions principales:



```

void save(Game *game){
    FILE *fp = NULL;

    char path[255] = "../save/";
    strcat(path, game->playerName);
    strcat(path, "_save.txt");
    fopen_s(&fp, path, "w");

    saveMap(fp, game);
    savePlayer(fp, game);
    saveNPCS(fp, game);

    fclose(fp);
}

```

"save()" : La sauvegarde de la partie qui se fait ligne par ligne et qui est décomposée en plusieurs sous-fonctions qui vont chacun sauvegarder une catégorie (Map, player, autres)

Afin de pouvoir écrire dans le fichier, nous utilisons fprintf qui permet de facilement définir des formats et fputs pour une simple chaîne de caractère.

```

void load(Game **game){
    FILE *fp = NULL;
    char buffer[255];
    int count = 0;
    char path[255] = "../save/";
    strcat(path, (*game)->playerName);
    strcat(path, "_save.txt");
    fopen_s(&fp, path, "r");
    int ***tempMap = malloc(sizeof(int**)*3);

    int *countZoneH = malloc(sizeof(int) * 3);
    int *countZoneW = malloc(sizeof(int) * 3);

    for(int i = 0; i < 3; i++){
        countZoneW[i] = 0;
        countZoneH[i] = 0;
    }
    loadTempZone1(fp, countZoneH, countZoneW);
    tempMap[0] = malloc(sizeof(int *) * countZoneH[0]);
    for (int k = 0; k < countZoneH[0]; ++k) {
        tempMap[0][k] = malloc(sizeof(int) * countZoneW[0]);
    }
    loadTempZone2(fp, countZoneH, countZoneW);
    tempMap[1] = malloc(sizeof(int *) * countZoneH[1]);
    for (int k = 0; k < countZoneH[1]; ++k) {
        tempMap[1][k] = malloc(sizeof(int) * countZoneW[1]);
    }
    loadTempZone3(fp, countZoneH, countZoneW);
    tempMap[2] = malloc(sizeof(int *) * countZoneH[2]);
    for (int k = 0; k < countZoneH[2]; ++k) {
        tempMap[2][k] = malloc(sizeof(int) * countZoneW[2]);
    }
    rewind(fp);
    loadZone1(fp, countZoneH, countZoneW, tempMap);
    loadZone2(fp, countZoneH, countZoneW, tempMap);
    loadZone3(fp, countZoneH, countZoneW, tempMap);
}

```

"load()" : Le chargement d'une partie sauvegardé qui se fait ligne par ligne et qui est décomposé en plusieurs sous-fonctions qui vont chacun charger une catégorie (Map, player, autres) .

```

void savePlayer(FILE *fp, Game *game){
    fprintf(fp, "=== PLAYER ===\n");
    fprintf(fp, "%d\n", game->player->lvl);
    fprintf(fp, "%d/{%d}\n", game->player->exp, (game->player->lvl * 100));
    fprintf(fp, "%f/{%f}\n", game->player->hp, game->player->hpMax);
    fputs("-- INVENTORY --\n", fp);

    for (int k = 0; k < 10 ; ++k) {
        Item actualSlot = game->player->inventory[k];
        if(actualSlot.id == 0){
            fprintf(fp, "{0}@{0}@{0}\n");
        }else {
            fprintf(fp, "%d@%d@%f\n", actualSlot.amount, actualSlot.id, actualSlot.durability);
        }
    }
    fputs("-- STORAGE --\n", fp);
    int chestSize = game->pnj->chestSize;
    for (int k = 0; k < chestSize ; ++k) {
        Item actualSlot = game->pnj->chest[k];
        fprintf(fp, "%d@%d@%f\n", actualSlot.amount, actualSlot.id, actualSlot.durability);
    }
}

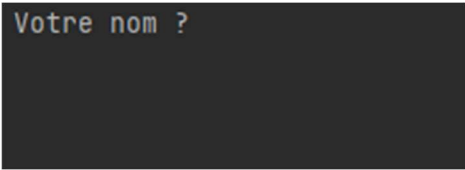
```

### III/ Dossier d'installation

Pour notre application il n'y a rien à installer il suffit de lancer l'exécutable pour faire fonctionner le tout.

### IV/ Dossier d'utilisation

Afin d'utiliser notre application il faut tout d'abord renseigner son nom (ou pseudo) qui sera utilisé pour générer un fichier de sauvegarde personnalisé

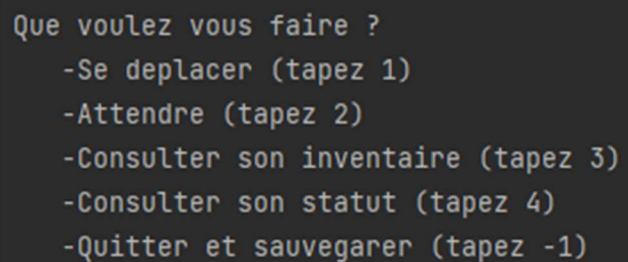


```
Votre nom ?
```

Par la suite une map sera générée procéduralement et vous pourrez commencer à jouer.

Les déplacements et toutes les interactions sont précisés dans le chat, il suffit de rentrer les numéros correspondants à ce que l'on veut faire.

Les menus ressemblent globalement à cela :



```
Que voulez vous faire ?  
-Se déplacer (tapez 1)  
-Attendre (tapez 2)  
-Consulter son inventaire (tapez 3)  
-Consulter son statut (tapez 4)  
-Quitter et sauvegarder (tapez -1)
```

Voici les différentes cases disponibles :

0 case vide ou vous pouvez aller librement

-1 case inaccessible

2 c'est le pnj vous aurez différentes interactions avec lui

-2 et -3 il s'agit des portails de zone accessible uniquement au niveaux 3 et 7

3 à 11 il s'agit de ressources récoltables si vous avez les outils nécessaires

12+ il s'agit des monstres faites attentions !

## V/ Bilan

Mathieu :

Pas de problème majeur au niveau du projet. Des difficultés personnelles au niveau du langage C, pallié par l'expérience générale du groupe. Une belle expérience qui m'a apporté beaucoup tant au niveau du langage C, du relationnel et de la gestion de versioning d'un projet.

Bill :

Le projet c'est très bien passer dans l'ensemble, aucun problème majeur à signaler. Tous les obstacles rencontrés durant la programmation ont pu être adressés durant les réunions régulières. Sauf à la fin ou un bug qui m'empêche d'exécuter le programme est apparu. L'absence de la SDL est dû à une mauvaise gestion de temps et la difficulté d'installation. En conclusion ce fut une belle expérience

Jonathan :

Un projet très agréable, je me suis beaucoup amusé à le réaliser. Pas de problèmes de groupe mais forcément un petit écart de niveau en C (notamment du côté de Mathieu) qui a peut-être un peu ralenti le projet et m'a forcé à prendre un peu plus mon temps. Cependant une bonne gestion de notre temps et un groupe très investi. Quelques problèmes techniques (aucun de nous n'a réussi à installer SDL) donc malheureusement pas de partie graphique ce qui me rend vachement triste mais dans l'ensemble un super projet pour moi, j'ai vraiment senti une énorme montée en compétence en C.