malloc world

Rapport de projet

Elsa FIRMIN, Yann BEMBA, Owen ANCELOT

école supérieure de
génie informatique

TABLE DES MATIÈRES :

Introduction :

Suite à la semaine de piscine début octobre un projet nous avait été donné. Il nous a donc fallu construire entièrement un jeu “Malloc World” qui contient un joueur, 3 zones générées procéduralement, des ressources, des monstres et des PNJ.

Le personnage doit tuer des monstres pour monter en niveaux, miner des ressources pour crafter de l’équipement, parler aux différents PNJ pour utiliser son coffre, crafter et réparer son équipement. Le but étant d’anéantir le boss final de “Malloc World”.

Nous devions donc créer le jeu en plusieurs parties avec :

- La création de la Map, du joueur et des déplacements
- La collecte de ressource
- Le craft & les réparations avec le PNJ
- Le levelling et combat
- La sauvegarde et restauration du jeu
- L’interface graphique

Les étudiants de notre groupe sont Owen ANCELOT, Yann BEMBA et Elsa FIRMIN.

Analyse de l'application :

Liste des structure de données :

- ```
  typedef struct Game{
     World* world;
     Player* player;
     Position* position;
     Respawn* respawn;
     Storage* storage;
  } Game;
  ```
  - ➢ La structure du jeu permet de créer un monde, un joueur, la position du joueur, un respawn et un stockage
- ```
  typedef struct World{
     Zone** world;
  } World;
  ```
  - ➢ La structure du monde permet de créer une zone
- ```
  typedef struct Zone{
     int row;
     int column;
     int** map;
     TypeZone type;
  } Zone;
  ```
  - ➢ La structure de la zone contient un nombre de colonne et de ligne, une tableau à 2 dimensions map et le type de zone (1,2,3)

- ```
  typedef struct Player{
     int currentHp;
     int maxHp;
     Stack** inventory;
     int sizeInventory;
     int level;
     int xp;
     int xpNext;
  } Player;
  ```
  - ➢ La structure du joueur permet de créer un joueur. Elle contient la vie actuelle du joueur et le max qu'elle peut atteindre, un inventaire, la taille de celui ci, un level, un xp et le prochain xp
- ```
  typedef struct Inventory{
     int id;
  ```
  ```

```
    char* name;
    int value;
    float durability;
    float durabilityMax;
    int maxStack;
    InventoryType type;
} Inventory;
```

➢ La structure de l'inventaire contient un id, un nom, une valeur, la durabilité actuelle et celle max ainsi que le nombre max de pile

❖
```
typedef struct Stack{
    int id;
    int maximum;
    int length;
    Inventory** inventory;
} Stack;
```

➢ La structure d'une pile contient un id, un maximum et une taille

❖
```
typedef struct Monster{
    int id;
    char* name;
    MonsterBreed breed;
    int xp;
    int damage;
    int idInventory;
    int hp;
} Monster;
```

➢ La structure d'un monstre permet de créer un monstre. Elle contient un id, un nom, une race, un xp, un dommage d'attaque, un id Inventaire et un hp

❖
```
typedef struct Position{
    int x;
    int y;
    int zone;
} Position;
```

➢ La structure d'une position contient un x et un y (coordonnées verticale-horizontale) et une zone

❖
```
typedef struct Respawn{
    int id;
    int roundLeft;
    struct Position* position;
    struct Respawn* child;
} Respawn;
```

➢ La structure Respawn est une liste chaînée qui permet de repop les ressources et les monstres sur la map ainsi que les portails pour le changement de zone. Elle contient l'id de la ressource, le nombre de tour restant avant de repop et sa position sur la map.

❖
```
typedef struct Storage{
    int id;
    int quantity;
```

```
    struct Storage* next;
} Storage;
```

➢ La structure Storage est une liste chaînée qui permet de gérer le stockage. Elle contient un id et une quantité.

❖
```
typedef struct Craft{
    Inventory* inventoryCraft;
    int inventory1;
    int quantityInv1;
    int inventory2;
    int quantityInv2;
    int zoneMin;
} Craft;
```

➢ La structure Craft permet de crafter, c'est-à-dire de créer une arme avec les ressources que l'on a récoltés. Elle contient un inventaire, l'id du premier inventaire ainsi que sa quantité et idem pour l'inventaire 2, et la zone minimum pour pouvoir crafter l'arme

❖
```
typedef struct ListCraft{
    Craft** list;
    int length;
} ListCraft;
```

➢ La structure ListCraft contient la liste des crafts ainsi que sa taille.

Choix d'implémentation

Nous avons eu différents choix d'implémentation :

- Pour la liste de l'inventaire, des monstres et du crafting nous avons utilisé des CSV. Cela nous a permis d'économiser de la mémoire, nous n'avons pas besoin de recompiler le projet pour modifier/ajouter un monstre, un inventaire ou un craft. D'autre part, cela est plus simple à maintenir et à récupérer.
- Vous pourrez observer que nous avons beaucoup de fichiers. Nous avons fait ce choix pour bien découper chaque partie, pour que le code soit plus simple à relire et pour mieux s'y retrouver.
- Elsa a créé sa propre branche sur le projet pour que l'on puisse relire et vérifier son code si besoin. Cela nous a aussi permis d'éviter des conflits de code.
- Tous les include de nos fichier header se trouvent dans "global.h". Cela nous permet d'utiliser ce fichier pour tous les .c, et de réutiliser n'importe quel header. En outre, cela nous a permis de régler le problème de la double inclusion entre 2 fichier **.h**

Liste des fonctions principales :

- ★ ***main(int argc, char *argv[])*** dans main.c
  - C'est la fonction principale du jeu, elle permet d'afficher le nom du jeu et d'appeler la fonction pour lancer le jeu
- ★ ***void runGame()*** dans game.c
  - Cette fonction lance le jeu, elle nous permet de créer une nouvelle partie, de charger une partie ou de la quitter
- ★ ***void play(Game *game)*** dans game.c
  - Cette fonction permet l'interaction avec le joueur, pour qu'il puisse jouer son personnage
- ★ ***int executeActionPlayer(char action, Game* game)*** dans game.c
  - Cette fonction exécute l'action du joueur et permet donc aussi de savoir si le jeu continue ou non (sauvegarde, affiche la map ou l'inventaire, quitte, fait bouger le personnage)
- ★ ***Game *createVoidGame()*** dans game.c
  - C'est la fonction qui permet de créer le jeu à partir de zéro avec un monde, un joueur et une position
- ★ ***void executeAction(Position newPosition, Game *game)*** dans game.c
  - Cette fonction permet d'exécuter l'action en fonction de la case où veut aller le joueur (miner, faire un combat, changer de zone, interagir avec le PNJ ou avancer)
- ★ ***void freeGame(Game *game)*** dans game.c
  - Cette fonction libère le jeu de la mémoire (le monde, la position, le joueur et le respawn)
- ★ ***void mining(Game *game, int id, Position position)*** dans mining.c
  - Cette fonction permet de miner une ressource (cherche si le joueur a un outil adéquat, gère la réapparition de la ressource et l'ajout dans l'inventaire)
- ★ ***void startFight(int id, Game* game, Position position)*** dans fight.c
  - C'est la fonction qui commence le combat contre un monstre et qui ajoute l'inventaire du monstre dans l'inventaire du joueur s'il a gagner le combat
- ★ ***void changeZone(Game* game, Element portal)*** dans moving.c
  - Cette fonction fait changer de zone le joueur quand il passe un portail en fonction du niveau du personnage
- ★ ***void actionMove(Position position, Game* game)*** dans moving.c
  - Cette fonction permet de changer la position du joueur suite à son déplacement
- ★ ***void saveGame(Game game)*** dans save.c
  - Cette fonction sauvegarde le jeu (le monde, le joueur, le storage)
- ★ ***void updateAllRespawn(Game *game)*** dans respawn.c
  - Cette fonction modifie tous les respawn dans la liste chainée en diminuant le "Tour restant" (roundleft) de 1. De plus, si "Tour restant" est à 0, on remet l'élément sur la map et on le supprime de la liste chaînée
- ★ ***void interactionWithPnj(Game* game)*** dans pnj.c
  - Cette fonction gère les intéractions du joueur avec le PNJ (craft, réparation, stockage, affichage, récupération)
- ★ ***void repairPlayerInventory(Player* player)*** dans pnj.c

- C’est la fonction qui répare tout l’inventaire du joueur (remet la durabilité des armes au max)

★ ***void stockStorage(Game* game)*** dans pnj.c
- Cette fonction stocke l’inventaire dans le storage avec son id et la quantité voulue

★ ***void retrieveStorage(Game* game)*** dans pnj.c
- C’est la fonction qui récupère un inventaire du storage (id et quantité)

Dossier d’installation précis:

Pour installer le jeu “**MallocWorld**”, veuillez suivre les étapes dans l’ordre ci-dessous :

1. Téléchargez le fichier **.zip** “***MallocWorld.zip***”
2. Dézippez le fichier là où vous le souhaitez (dans le “Program Files” par exemple).
3. Un dossier “***MallocWorld***” à dû se créer. Allez dans le dossier “***MallocWorld***”
4. Allez dans le dossier “***bin***”.
5. Vous pouvez faire clique droit l'exécutable “***MallocWorld.exe***” pour créer un raccourci sur le bureau.
6. Vous n’avez plus qu'à double-cliquer sur l’exécutable ou bien sur le raccourci pour que le jeu se lance sur votre invité de commande.

Liste des dossiers :

Dans le dossier d’installation “***MallocWorld’***, vous trouverez 3 dossiers différents :

- ***/bin*** : Contient l’exécutable du jeu
- ***/resources*** : Contient la liste des monstres, des équipements et les créations possibles avec les ressources récoltés
- ***/saves*** : Contient la liste des sauvegardes du jeu que vous aurez effectuées. Ce dossier n’est pas créé par défaut. Il sera créé lors de la première sauvegarde du jeu.

Dossier d'utilisation :

Lancer le jeu :

Pour le jeu, vous avez juste à double-cliquez sur l'exécutable (il se trouve dans le dossier "***bin***" dans le dossier d'installation "***MallocWorld***") ou bien double-cliquez sur le raccourci si vous en avez créé un.

Liste des commandes :

Lors du lancement du jeu, vous arrivez au menu principal et vous avez le choix de créer une partie ce qui créera une partie, ou de quitter, ce qui vous fera quitter le jeu. Voici les commandes pour le menu principale :

- **`create`** : créé la partie
- **`quit`** : quitter la partie

Une fois la partie créée, la partie se lance et vous aurez la possibilité de vous déplacer, de voir les statistiques de votre personnage, de sauvegarder… La liste des commandes sera affichée lors de chaque action effectuée. Vous trouverez ci-dessous la liste des commande :

- **`z`** : se déplacer vers le haut
- **`q`** : se déplacer vers la gauche
- **`d`** : se déplacer vers la droite
- **`s`** : se déplacer vers le bas
- **`i`** : regarder son inventaire
- **`m`** : afficher la carte
- **`j`** : afficher les informations de son personnage
- **`l`** : sauvegarder la partie (les parties seront sauvegarder dans le dossier "***saves***" du dossier d'installation)
- **`p`** : sauvegarder la partie et quitter le jeu
- **`c`** : quitter le jeu sans sauvegarder

Lors de la rencontre avec un monstre, un combat se lancera automatiquement en vous demandant de choisir une arme (si vous en avez une) et une armure (si vous en avez une). La liste des commandes sera affichée à chaque tour du combat. Vous trouverez ci-dessous la liste des commande lors d'un combat :

- **`a`** : attaquer le monstre
- **`h`** : utiliser une potion pour se soigner (il vous sera demandé de choisir la potion)
- **`p`** : changer d'armure (il vous sera demandé de choisir l'armure si vous en avez une)
- **`e`** : essayer de fuir (33% de chance de pouvoir fuir le combat)
- **`w`** : changer d'arme (il vous sera demandé de choisir une arme si vous en avez une)

Lorsque vous rencontrez un PNJ (représenté par un “**2**” sur la carte), vous aurez la possibilité de réparer entièrement votre équipement, de stocker de l’équipement dans votre coffre (accessible uniquement depuis un PNJ) ainsi que de le récupérer ou de créer une arme, armure, outil et portion. Vous trouverez ci-dessous la liste des commande lors d’une rencontre avec un PNJ :

- `r` : réparer l’équipement
- `a` : stocker de l’équipement dans le coffre
- `s` : voir ce que contient le coffre
- `c` : créer de l’équipement (une liste des objets possible à créer sera lister et vous devrez saisir l’ID).
- `t` : récupérer de l’équipement depuis le coffre
- `q` : quitter le PNJ

Liste des cases sur la carte :

Vous trouverez ci-dessous la liste des cases sur la carte du jeu :

- `1` : Joueur (vous)
- `0` : Zone libre
- `-1` : Mur / zone infranchissable
- `3` / `6` / `9` : Plante
- `4` / `7` / `10` : Rocher
- `5` / `8` / `11` : Bois
- `12` à `50` : Les monstres
- `-2` : Portail entre les zones 1 e 2
- `-3` : Portail entre les zones 2 et 3
- `99` : Le Boss de fin

Bilan :

Nous avons rencontré différents problèmes avec la SDL, la restauration du jeu mais aussi des difficultés humaines avec l'absence de Yann.

- Pour la SDL, nous avons eu beaucoup de difficultés à la mettre en place avec CLion et donc cela nous a pris beaucoup de temps. Une fois que nous avions mis en place l'inclusion de la librairie dans le projet, une erreur a été levée lors de l'utilisation d'une fonction SDL. Nous avons donc pris la décision de mettre de côté cette partie et de revenir dessus à la fin du projet si nous avions le temps, pour pouvoir avoir le plus de fonctionnalités dans le projet.
- Pour la restauration du jeu, nous avons eu les mêmes problèmes. Nous avons essayé de la mettre en place à de multiples reprises mais cela nous provoquait toujours une erreur dans le code, nous avons donc préféré mettre cette partie de côté pour avancer sur les autres points.
- L'absence totale de Yann (qui a quitté l'école il y a peu) nous a donné une charge de travail supplémentaire mais nous avons essayé de nous répartir et de nous aider au mieux pour les différentes étapes.