Rapport de projet

Vie dans un labyrinthe

BODINEAU Bastien, DURAN Alizée et NGATCHOU Junior

Novembre 2015 - Janvier 2016

Table des matières

Chapitre 1

Présentation du sujet

Dans cette première partie, nous allons expliquer le sujet ainsi que les modifications effectuées.

1.1 Sujet du projet

Pour l'UE Conduite de projet, il nous faut réaliser un jeu en C sur terminal avec possibilité de le faire avec une interface graphique (SDL). Parmi plusieurs sujets, nous avons choisi "Vie dans un labyrinthe".

Le but de ce jeu est de faire évoluer des insectes à l'intérieur d'un labyrinthe, en s'inspirant du jeu de la vie, un jeu que l'on a déjà réalisé au cours de l'UE Algorithme et Programmation. Nous devons également informer l'utilisateur sur les statistiques de la population présente dans le labyrinthe, générer des déplacements semi-aléatoires et nourrir ces insectes.

1.2 Modifications apportées au sujet

Dans ce sujet, nous avons apporté des modifications afin de mieux étudier le sujet du jeu à réaliser. En premier lieu, nous avons préféré mettre des entités comme un joueur et un/plusieurs monstre(s). Les insectes peuvent tout à fait être des monstres que le joueur doit combattre.

Étant un point clé du jeu, une part du rapport est attribuée à la manière dont la génération du labyrinthe a été construite.

Chapitre 2

Organisation du travail

Ici, nous allons détailler la répartition du travail entre les étudiants en affectant chacun un module différent.

2.1 Répartition des tâches

Afin de concevoir correctement le programme du jeu, nous avons réparti les tâches en fonction de modules. En effet, trois modules principaux sont importants : le module du labyrinthe, le module du joueur et le module du monstre. Voici la répartition des tâches en fonction des étudiants :

Table 2.1 – Répartition des tâches.

	Alizée	Junior	Bastien
Module Joueur	×		
Module Monstre		×	
Module Labyrinthe			×
Module Menus	×		×
Makefile			×
Documentation	×		

Chapitre 3

Analyse du problème

Dans cette partie, nous allons indiquer les structures de données que nous avons établies ainsi que la génération du labyrinthe, qui est un élément important du projet.

3.1 Structures de données

Dans un premier temps, nous donnons les différentes structures de données que chaque module possède :

```
typedef struct inventaire{int etat; int contenu[10];}
    t_inventaire;
typedef struct {int id; int hp; int action; t_inventaire
    inventaire; int orientation;}entity;
typedef struct {int etat; int isivisit; int haut; int bas;
    int gauche; int droite; t_inventaire objets; entity
    entite;}t_salle;
t_salle labyrinthe[N][N];
entity joueur;
entity monstre;
```

Explications :

— t_inventaire est la structure correspondant à l'inventaire de chaque module. Elle contient une variable *etat* pour déterminer si l'objet est accessible ou non et une variable *contenu* pour sa nature.

— t_entity est également une structure possédant quatre variables : *id* pour l'identifiant d'une entité, *hp* pour ses points de vie associés, *action* pour ses points d'actions et *orientation* pour définir son sens de direction. t_inventaire représente l'inventaire.

— t_salle représente la salle comportant le labyrinthe. Cette structure reprend les deux structures de type t_inventaire et *entity* ci-dessus qui sont liées respectivement à *objets* et *entite* ainsi que les variables *etat* pour déterminer si la case possède des murs ou non, quatre directions *haut*, *bas*, *gauche* et *droite* pour les directions que l'entité

peut utiliser pour se déplacer à l'intérieur du labyrinthe.
— les variables de type structure *labyrinthe[N][N]*, *joueur* et *monstre* sont des variables globales représentant les modules pour le jeu.

3.2 Génération du labyrinthe

Dans un second temps, nous allons exposer la manière dont nous avons réalisé la gestion du labyrinthe.

Pour générer le labyrinthe, nous avons réutilisé un algorithme trouvé sur le net. Il consiste à placer une première case vide aléatoirement sur la grille puis de placer des 'ponts'. Chaque pont représente en fait la liaison entre 2 cases.

Pour savoir ou créer ces ponts, on prend une salle au hasard, on regarde l'une de ces voisines. Si celle-ci est creuse (fait partie du labyrinthe) alors on peut créer le 'pont' entre les deux en retirant les murs de l'initialisation et en 'creusant' le mur.

Pour créer des couloirs et un labyrinthe parfait, on alterne entre la création d'un pont vertical et horizontal. Aussi, la génération de pont doit être faite N*N-1 fois, c'est-à-dire 1 par case du labyrinthe -1 pour celle générée aléatoirement au début.

Ainsi, pour proposer un but à notre jeu, on choisit une case au hasard de la première colonne du tableau du labyrinthe pour y placer le joueur en début de partie ainsi qu'une case sur la Nième colonne pour y mettre l'arrivée.

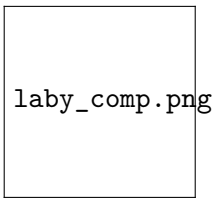

FIGURE 3.1 – Schéma de génération du labyrinthe

Chapitre 4

Codage du jeu

Dans cette partie, nous allons expliciter les fonctions utiles à la programmation du jeu qui dépendent des modules, représentés dans le diagramme suivant :

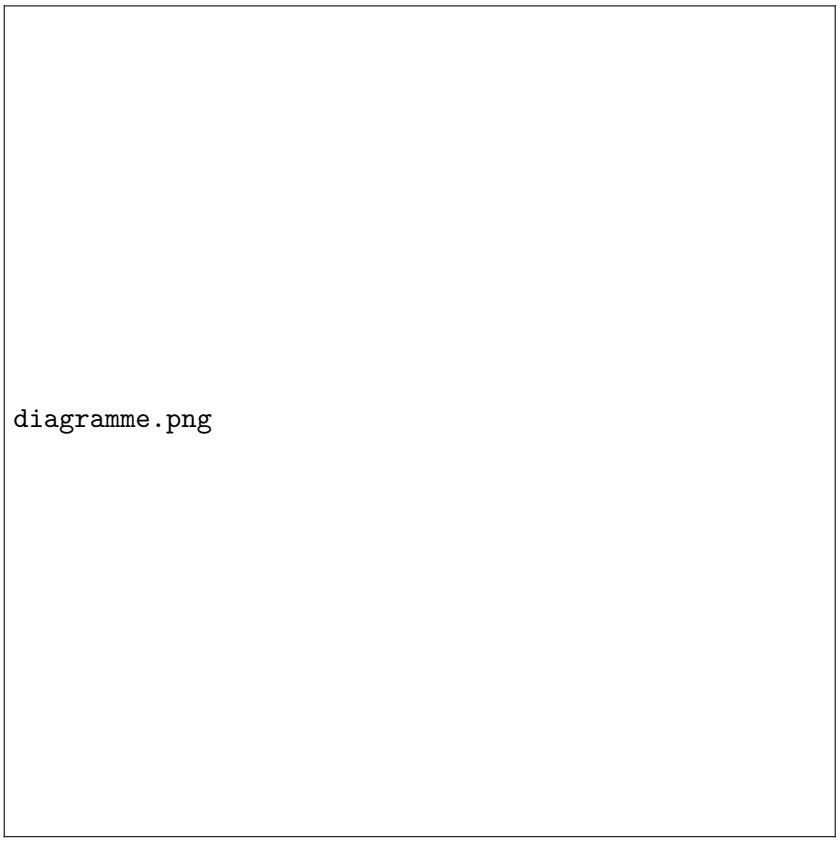

FIGURE 4.1 – Diagramme de représentation des modules avec améliorations (en bleu).

Joueur

— Dans le jeu, le joueur se déplace à l'intérieur du labyrinthe. Pour qu'il puisse se déplacer en fonction des murs ou des chemins disponibles dans le labyrinthe, la fonction *Joueur_deplacer* permet ses déplacements.

```
void Joueur_deplacer()
    Recupere la position (x,y) du joueur;
    Affecte a une variable l'orientation de l'entite du
        labyrinthe;
    Cette variable est comprise entre 1 et 4, pour chaque
         orientation possible;
    Selon l'orientation et l'indication d'un labyrinthe
        ou non, affiche les disponibilites pour se
        deplacer;
    Affiche un message d'erreur si ce que l'on fait n'est
         pas demande
```

— Pendant le tour, le joueur, en fonction des choix de l'utilisateur, peut effectuer différentes actions. La fonction *Joueur_agir* donne sur un menu disposant de plusieurs possibilités de jouer tout en utilisant des points d'actions (PA). Dès que les points d'actions sont terminés, le tour se termine.

```
void Joueur_agir()
    Affiche le labyrinthe;
    Affiche le menu disponible pour le joueur;
    Recupere de la touche correspondant a l'action a
        faire;
    En fonction de celle-ci, l'action donnee est faite;
    Si l'action est incorrecte, affiche un message d'
        erreur;
```

Labyrinthe

— Pour gérer le labyrinthe ainsi que tout ce qui s'y trouve efficacement, les structures décrites en patie 3.1 ont été créées. L'élément central du projet étant le labyrinthe, absolument tout s'y trouve. Il sert d'environement dans lequel le joueur se développe. La fonction qui le définit et qui sert à créer le labyrinthe est Laby_generator qui exécute l'algorithme détaillé dans la partie 3.2.

— Les autres fonctions propres au labyrinthe sont celles qui donnent accès aux données stockées dans celui-ci ainsi les fonctions check_salle, presencer, joueur_orienter et d'autres inutilisées. Ces autres fonctions permettent respectivement de récupérer les objets présents dans une salle, de savoir si d'autres entités sont à coté du joueur et les possibilités des déplacements du joueur (ou monstres mais non implémentées). Aussi le labyrinthe est épaulé d'un système d'affichage, le module "menu", qui contient des fonctions pour faire le lien avec l'utilisateur.

Menu

— Le module menu a été créé en suplément tout le long du projet. Chaque étudiant a participé à sa création, de l'affichage du menu principal, de la carte (labyrinthe complet) jusqu'à la barre de vie et d'action du joueur. Ce module complémentaire ne contient que des fonctions d'affichage et fait en sorte d'informer au mieux la situation du joueur à tout instant.

Chapitre 5

Outils de développement et de réalisation du projet

5.1 Gestion des fichiers - Github

Afin de collaborer et de partager nos fichiers, nous avons utilisé le dépôt distant Github. L'adresse url où est situé ce dépôt distant est : `https://github.com/BastienMor/Labyrinthe_SPI`.

Sur Github, nous avons tout d'abord créé trois fichiers analogue aux modules. Au fur et à mesure des mises à jour, nous avons établi des dossiers pour les divers types de fichiers. Ainsi, lorsque l'on a commencé à relier les différents modules entre eux, nous avons constitué des branches (comme *testb*) afin de voir comment le regroupement entre deux modules puis trois pouvait être opérationnel.

5.2 Makefile et son utilisation

Etant donné que la programmation du jeu contient énormément de lignes de code, il est nécessaire de séparer les différentes fonctions utiles en fonction de chaque module. Pour ce faire, il est pratique d'utiliser Makefile. Sur le terminal, la commande à taper est *make*.

Cet outil nous permet tout d'abord de créer des liens entre chaque fichier des différents modules. Il nous permet ensuite de compiler l'ensemble des modules afin de faire fonctionner le programme correctement.

Nous nous sommes heurtés à quelques problèmes lors de l'exécution du makefile, comme compiler les fichiers contenant des libraires accédant à d'autres fichiers de fonctions. Néanmoins, nous avons réussi à régler ce problème.

5.3 Documentation Doxygen

Dans le but de générer de la documentation sur les structures de données employées ainsi que les fonctions des modules, il est aisé d'utiliser Doxygen. Cet outil de documentation technique est concu à partir d'un fichier Doxy, contenant un ensemble de données à remplir pour plus de facilité.

En exécutant le fichier Doxy, deux dossiers html et latex sont générés. Pour pouvoir obtenir la version disponible par un navigateur, il faut ouvrir le fichier index.html. Pour avoir la version pdf de la documentation, il faut effectuer la commande make pdf après avoir été dans le dossier latex.

Chapitre 6

Résultats

Cette partie constitue les résultats que nous a donné le jeu lors de sa programmation.

Globalement, le jeu démarre sur un affichage d'un menu permettant de commencer à jouer, d'effectuer une sauvegarde, de charger une partie, d'afficher les options ou de quitter.

Lors de la partie, un autre menu apparaît et permet de faire plusieurs actions disponibles selon les points d'action. Le joueur peut se déplacer à l'intérieur du labyrinthe. Il voit le labyrinthe à la première personne. Cependant, une carte dynamique l'aide dans sa quête à retrouver la sortie du labyrinthe, indiquée par une croix. Le jeu se termine lorsqu'il atteint son but.

Néanmoins, le module monstre n'étant pas réalisé ccomplètement, nous n'avons pas eu la possibilité de l'inclure dans le programme du jeu. Nous aurons donc le jeu avec les monstres dans la version améliorée du jeu.

Chapitre 7

Conclusion

En prenant du recul, nous pouvons dire que ce projet nous a permis de créer un jeu avec tous les outils mis à disposition.

En effet, ce projet est tout d'abord un projet créé à plusieurs mains donc il était nécessaire d'avoir des moyens de communication pour pouvoir avancer chacun de son côté. Nous avons eu quelques soucis au niveau des modules.

Ce jeu a besoin d'améliorations. Effectivement, nous avons pensé à mettre des objets que le joueur peut récupérer, s'équiper ou bien utiliser contre les monstres présents dans le jeu. Cependant, le module monstre n'ayant pas été réalisé à temps, nous avons laissé de côté.

En outre, nous avons mis une possibilité de modifier les commandes permettant de déplacer le joueur. Mais par manque de temps, elle n'est pas encore prête.

De plus, nous pensons pouvoir continuer le projet en ajoutant les améliorations nécessaires ainsi qu'une éventuelle interface graphique.