UFR Sciences et techniques
Université du Maine

RAPPORT DE PROJET

RogueLike

14 décembre 2016

Emeric Mottier
Valentin Pelloin
Titouan Teyssier

L2 Sciences pour l'ingénieur

Table des matières

Chapitre 1

Introduction

Nous avons choisi le jeu *Roguelike* car c'est un jeu que nous trouvons intéressant, puisqu'il est complet, et que c'est un jeu aux possibilités infinies : il est toujours possible d'ajouter de nouvelles actions que le joueur pourra effectuer.

Notre jeu se déroule dans le bâtiment IC^2. Nous sommes un étudiant, nous partons du rez-de-chaussée, et nous devons aller chercher QUELQUE CHOSE tout en haut, pour le ramener.
Nous devons cependant faire attention aux monstres : des L1, L2, L3, des masters, des doctorants, et certains fantômes : Claude et Chappe.
Sur notre chemin, nous pouvons trouver quelques pièges : des flaques d'eau laissées par les femmes de ménages qui nous font glisser, des trous entre les étages qui nous font tomber d'un étage à un autre inférieur, ou des cartes à jouer qui nous sont jetées dessus par des L1.
Durant notre parcours, nous devons aussi tenir compte de notre faim. Nous possédons une barre de vie, lorsqu'elle est à zéro, nous mourrons. Pour régénérer de la vie, il y a deux possibilités : ne plus avoir faim (en mangeant de la nourriture, attention, certaines sont empoisonnées), et des seringues de soin à s'injecter directement. Ces objets peuvent être consommés directement sur place quand le joueur le trouve, ou plus tard, en les gardant dans son inventaire.
Enfin, lorsque le joueur apparait, il ne voit pas entièrement la carte, il doit la découvrir pour cela. Lorsque le joueur a trop faim, en plus de perdre de la vie, il s'évanouit : il se déplace plus difficilement, et perd connaissance de ce qu'il a découvert.

Pour le projet, nous devions au minimum effectuer un jeu qui génère des niveaux (ici, des étages dans notre bâtiment) aléatoires, avec une taille variant en fonction de l'étage où se trouve le joueur. Il était aussi demandé, en fonction de l'avancement du projet, d'ajouter des fonctionnalités supplémentaires : des armes, des monstres, des pièges, ou autres.

Chapitre 2

Organisation

ici mettre un petit mot <tout le monde>

2.1 Répartition des tâches

Mettre ici notre répartition des tâches. <tout le monde>

2.1.1 Pourquoi cette répartition

<tout le monde>

2.2 Utilisation d'un gestionnaire de versions

Pour gérer les versions du projet nous avons utilisé Git et Github. Git pour toute la partie locale à chacune de nos machine. Quand nous codons, nous pouvons ainsi faire des versions du projet régulièrement et revenir en arrière si besoin. Github pour la mise en commun des modifications apportées, ce qui nous permet de travailler ensemble sans nécessairement coder en même temps ou au même endroit. Afin que chacun puisse librement ajouter ses modifications au projet, un seul dépôt Github a été créé et chaque membre de l'équipe a reçus le droit en écriture sur le dépot.

2.3 Utilisation des projets Github

<emeric>

2.4 Notre boîte à outils

Notre projet a été réalisé en plusieurs modules différents, et l'un d'entre eux est notre boite à outil. Dedans se trouve de nombreuses fonctions qui nous sont utiles, mais qui ne sont pas pour autant liées à notre projet en particulier : des fonctions de comparaison d'intervalles, d'aléatoires, de caractères, de log d'erreurs, de fichiers, ...
Nous avons aussi les fonctions essentielles pour l'accès à des listes et des files.

2.5 Doxygen, CUnit, GDB

Durant la réalisation de notre projet, nous avons utilisé divers outils d'aide à la programmation et au débogage.

La première chose que nous avons mis en place est la documentation à l'aide de *Doxygen*. C'est un programme qui génère une documentation automatiquement, en fonction des fichiers d'en-têtes et sources. La documentation peut être générée de plusieurs formats, nous avons choisi au format HTML car il est plus facile de s'en servir. Celle-ci est sur internet, à l'adresse suivante : `https://roguelike.vlntn.pw/`. Elle se met à jour automatiquement en fonction de notre code (via un webhook mis en place sur Github).

Ensuite, nous avons utilisé *CUnit*, un framework de tests unitaires pour le C. Toutes nos fonctions de notre boîte à outil ont été testés, avec des assertions que nous jugeons pertinentes (sur des valeurs qui pourraient poser problème dans certaines fonctions, comme des valeurs nulles, négatives, sur des fichiers inexistants, ...).

Enfin, lorsque nous avions certains bogues que nous n'arrivions pas à résoudre, nous avons utilisé le logiciel de debogage *GDB* (*GNU DeBugger*). Nous n'avions pas réussi à le faire fonctionner dès le début, car nous utilisons la libraire d'affichage *ncurses*[1], qui utilise déjà le terminal pour afficher notre jeu. En le combinant avec *GDB*, le terminal n'était plus utilisable.
La solution à été d'utiliser deux téléscripteurs (TTY) différents : un pour le jeu, et un pour le débogueur.
Dans l'annexe, vous pouvez retrouver 3 exemples de cas où nous nous sommes servis du débogueur.

1. Voir la section interactions et déplacements, page 9.

Chapitre 3

Analyse et Conception

ici mettre un petit mot

3.1 Notre cahier des charges

Avant de commencer à programmer notre jeu, nous nous sommes concertés autour d'un même fichier pour mettre en commun nos idées à propos du jeu. Notre jeu étant très libre dans son fonctionnement, c'était une étape cruciale.

Notre *rogueLike* prend place dans le bâtiment IC^2. Le joueur démarre le jeu au premier étage. Il doit se rendre tout en haut, aller chercher l'objet, puis revenir en bas.
Chaque étage de ce bâtiment est généré aléatoirement, il doit comporter des pièces et des couloirs. Entre ceux-ci, peut se trouver des portes, ou juste un trou. Les portes sont aléatoirement ouvertes ou fermées, et le joueur peut essayer des les ouvrir manuellement (il a une probabilité de ne pas réussir). Le nombre de pièces par étage est défini en fonction de l'étage : les étages supérieurs sont plus compliqués car ils possèdent plus de pièces. Pour changer d'étages, le joueur doit emprunter des escaliers (un escalier pour monter, et un pour descendre par étage, ceux-ci ne peuvent pas se trouver dans la même pièce).
Lorsque le joueur apparait, il ne voit que la pièces où il se situe. Dans les couloirs, lorsqu'il progresse, les zones s'éclairent petit à petit, en revanche, dans dès qu'il entre dans une pièce, celle-ci s'éclaire entièrement.
Le joueur possède un certain nombre de point de vie (défini à 10) au maximum, lorsque celle-ci est à 0, il meurt. Au fur et à mesure de ses déplacements, il perd de la nourriture (une barre allant de 0 à 100). Lorsqu'il a trop faim (nourriture inférieure à 10), il ne se déplace plus correctement, il perd certaines zones de sa mémoire de la carte (elle ne sont plus éclairées), et il perd de la vie. Le joueur peut trouver de la nourriture aléatoirement par terre (un total de 2 objets de nourriture par pièce, par étage), tout comme il peut trouver des soins de santé pour lui faire régénérer sa vie. L'autre façon de re-gagner de la vie est de manger, ainsi certains point de nourritures seront utilisées pour la vie.
Lorsque le joueur mange de la nourriture, il a une faible probabilité de chance que celle-ci était empoisonnée. Le joueur perd de la vie à chaque déplacement lorsqu'il est empoisonné. Pour ne plus l'être, il doit soit attendre quelques déplacements de ne plus l'être, soit consommer un kit de santé.
Durant son chemin dans le bâtiment, le joueur peut se faire avoir par des pièges. Ceux-ci peuvent, au choix, enlever de la vie, nous faire glisser à travers la pièce, ou nous faire tomber d'un ou

plusieurs étages (une chance sur 3 pour chaque).
Les objets tels que la nourriture, les soins de santé, ou les pièges peuvent être récupérés par le joueur dans son inventaire, pour les déposer ou les utiliser plus tard. L'inventaire du joueur possède 5 *slots*, et il ne peut y déposer qu'un seul objet par *slot*.
Des monstres se trouvent dans le jeu, ceux-ci nous attaquent, en se rapprochant de nous. Nous pouvons aussi les attaquer. Ils possèdent, tout comme nous, une agilité, qui est plus importante pour un monstre évolué. Cette agilité permet de faire plus de dégâts au combat, et de moins en subir. Ces monstres doivent avoir une mini-intelligence artificielle pour qu'ils puissent se déplacer de façon similaire à ce que un véritable joueur pourrait faire.

A tout moment, l'utilisateur peut sauvegarder sa partie sur l'un des trois emplacements de sauvegarde prévus. Il peut aussi quitter le jeu à n'importe quelle étape. Au lancement, l'utilisateur doit avoir un écran lui indiquant les sauvegardes, il peut en charger une, ou en supprimer une. Si il lance le jeu sur un emplacement vide, une nouvelle partie est alors crée.
A la fin du jeu, lorsque le joueur perd, ou gagne, il se retrouve sur l'écran initial des sauvegardes.
Au lancement du jeu, un texte explicatif avec l'objectif de celui-ci doit être affiché. Le jeu est décomposé en trois parties :

- la zone de jeu
- la zone de logs, où se trouve des informations sur ce que le joueur peut effectuer
- la zone de statistiques, qui indique toutes les propriétés du joueur : étage, vie, faim, nombre de déplacements, si il est empoisonné, si il a trouvé l'objet, ...

3.2 Comment jouer ?

- Lancement du jeu :
  Pour commencer à jouer, vous devez télécharger le jeu à partir de l'adresse suivante : rogueLike ; avec la commande suivante : `git clone git@github.com:TitouanT/rogueLike.git`
  Vous faîtes : `cd rogueLike`.
  Puis vous compilez grâce au makefile : «make install».
  Le jeu commence dès que vous faîtes : «./rogueLike»
- Les déplacement :
  Nous pouvons gérer nos déplacements sur la carte grâce aux flèches de direction.
- Interactions avec des objets :
  Les interactions avec un objet (serringues de soins, nourriture, escalier) se font avec la touche «entrée».
- Ouvrir et fermer une porte :
  Si vous souhaitez ouvrir une porte, déplacez-vous devant la porte, appuyer sur la touche «o» et marqué la direction de la porte avec les flèches de direction. Mais pour fermer, c'est le même principe que pour ouvrir une porte sauf que la touche est «c» au lieu de «o».
- Gestion de l'inventaire :
  Pour voir votre inventaire, vous devez appuyer sur la touche «i». Pour prendre un objet (serringues de soins, nourriture), vous devez appuyez sur la touche «g» mais pour poser votre inventaire, vous devez appuyez sur la touche «d» et indiquer la case se trouve l'objet.
- Sauvegarder sa partie :
  Vous pouvez sauvegarder la partie à tout moment avec la touche «s», cette manoeuvre n'arrêtera pas votre expérience de jeu.

Chapitre 4

Codage, méthode et outil

ici mettre un petit mot

4.1 Structures et énumérations

Nous avons utilisé un certain nombre de structure et d'énumération pour modéliser le rogueLike.

4.1.1 Une case de la carte

La structure la plus importante est celle qui représente une case de la carte. Cette structure contient un champ qui définit sont type, un autre qui défini son état ainsi qu'un tableau listant les objet présent sur la case et deux entiers pour savoir si le joueur a découvert cette case et pour connaitre le nombre d'objets présent.

Le type est une énumération des différents type de case possible : un vide, un mur, un encadrement de porte, une piece ou un couloir.

L'état est une énumération utilisé pour les encadrements de portes afin de savoir si il y a une porte, et si elle est ouverte ou fermée. L'état est aussi utilisé pour les pieces afin de savoir si il y a de la lumière ou non dans la pièce.

Un objet est décrit par une structure qui est composé d'un type et d'un entier pour savoir si l'objet a été découvert par le joueur. Le type d'un objet est lui décrit par une énumération et définit les escaliers ascendants et descendants, la nourriture, les kits de santé et les pièges.

4.1.2 Les être vivants

Le joueur controle un personnage que l'on représente à l'aide d'une structure. Elle permet d'enregistrer son nom, sa position (ligne, colonne et niveau), ses points de vie et d'attaque, son agilité, son expérience, son appétit, sa santé, le nombre de mouvement effectué, un booléen pour savoir si il a trouvé le QUELQUE CHOSE ainsi qu'un tableau d'objet qui représente son inventaire.

Les monstres ont eux aussi une structure qui enregistre leur nom, leur types, leur position (ligne, colonne et niveau), leur points de vie et d'attaque, leur champ de vision, leur agilité et quelque champs supplémentaire qui est utilisé comme mémoire pour leur IA. Les différents types de monstres sont L1, L2, L3, master, doctorant et fantome.

4.1.3 Et le reste

La représentation en matrice des niveau n'étant pas dans tous les cas la plus efficace, nous avons créé une structure de niveau qui contient un tableau de pièces et le nombre de pieces que contient l'étage.

Une pieces est elle-même représentée par une structure qui contient la position de son angle supérieur gauche (ligne et colonne) ainsi que ses dimensions (largeur et hauteur).

Quelque fois, nous avons besoin de manipuler des position (ligne, colonne). Nous avons donc créé une structure de position qui contient une ligne et une colonne.

4.2 Séparation du code en modules

Afin d'avoir un code source clair et maintenable, nous l'avons séparé en plusieurs modules. Nous avons un module dédié pour :

- les affichages,
- les interactions (interprète les commandes du joueur),
- les lectures/écritures dans des fichiers,
- la boîte à outils,
- la génération des niveaux,
- et la prise en charge des monstres.

4.3 Détail des modules

On met du code comme ça :

Listing 4.1 – programme.c

```
#include <stdio.h>
#include "maFonction.h"

int main(){

        int b;
        b = test(10);
        return 0;
}
```

4.3.1 La génération des niveaux

La génération d'un niveau ce fait en quelques étapes :

1. Le choix du nombre de pièce à placer en fonction de l'étage qu'il représente (de 0 à 5).
2. Le placement des pièces sur la map.
3. La création des couloirs pour lier les pièces.
4. Le placement des objets (nourriture, escaliers, pièges).

placement des pièces

Pour placer les pièces, on les places les unes après les autres. Pour chaque pièces, nous commençons par lui donner une position (la position de son angle supérieur gauche). Ensuite nous lui choisissons une dimension (hauteur, largeur) afin qu'elle entre la plus petite et la plus grande pièce possible. Pour finir, nous vérifions qu'elle n'est pas en collision avec une autre pièce. Dans le cas ou elle l'est, on recommence (pas plus de 100 fois).

création des couloirs

Le but est de créer un réseau de pièces ou aucune pièce ou groupe de pièces ne soit isolé des autres. Pour cela nous connectons les pièces les unes après les autres dans leur ordre de création. L'algorithme qui créé les liens commence par initialiser la première pièces simplement en y ouvrant une porte. Ensuite, pour chaque pièces il procède ainsi :

— Création d'une ouverture dans la pièce.
— Création d'un couloir entre cette ouverture et la case de type couloir ou encadrement de porte la plus proche à l'aide d'un algorithme de recherche de chemin.

4.3.2 La sauvegarde

<emeric>

4.3.3 Les changements d'étages

<emeric>

4.3.4 Les interactions et déplacements

Notre jeu devait se jouer de façon simple, le joueur devait pouvoir déplacer son personnage d'un simple appui sur une touche. L'utilisation de simples `scanf()` pour récupérer l'appui sur une touche aurait été difficile à gérer, car ces fonctions sont exécutées uniquement lorsque la personne appuie sur la touche *entrée*. Pour palier ce problème, nous avons utilisé la libraire *ncurses*, qui est une TUI, une interface utilisateur par terminal. Celle-ci nous permet, entre autres, de récupérer les touches (ou combinaisons de touches) que l'utilisateur effectue sans avoir à attendre qu'il appuie sur entrée.

4.3.5 L'affichage

<valentin>

4.3.6 Les monstres

Le nombre de monstre dans une partie est choisi aléatoirement entre deux bornes définient en constantes. Les monstres sont équitablement répartient entre les étages proportionnellement au nombres de pièces qu'ils contiennent. Chaque étages a une proportion différentes de chaque monstres, plus l'étage est haut, plus les types de monstre présent sont difficile à battre.

Pour animer les monstres, nous marquons toutes les cases de la matrice accessible par le joueur avec la distance de celle-ci par rapport au joueur. Ensuite chaque monstre fait un mouvement en direction du joueur si le joueur est dans son champ de vision. Sinon, ils ne bougent pas.

Nous faisons attention à ce que les monstres n'aille pas sur le joueur et si le joueur est à porter alors ils l'attaquent. Et si le joueur fait un déplacement vers une case ou se trouve un monstre alors le joueur ne bouge pas et attaque. Ainsi, il n'y a jamais deux être vivant sur la même case.

Les fantomes sont particuliés car :

— ils ne visent jamais le joueur,
— ils peuvent se déplacer partout sur la map,
— ils ont un halo lumineux autours d'eux, permettant au joueur d'apercevoir des zones qu'il n'a pas encore découvertes.

4.3.7 La nourriture et la vie

<valentin>

4.3.8 Les pièges

<emeric>

Chapitre 5

Résultat et conclusion

Faire une comparaison avec les objectifs. <titouan>

5.1 Améliorations possibles

<tout le monde>

5.2 Apport personnel du projet

<tout le monde>

Chapitre 6

Annexe