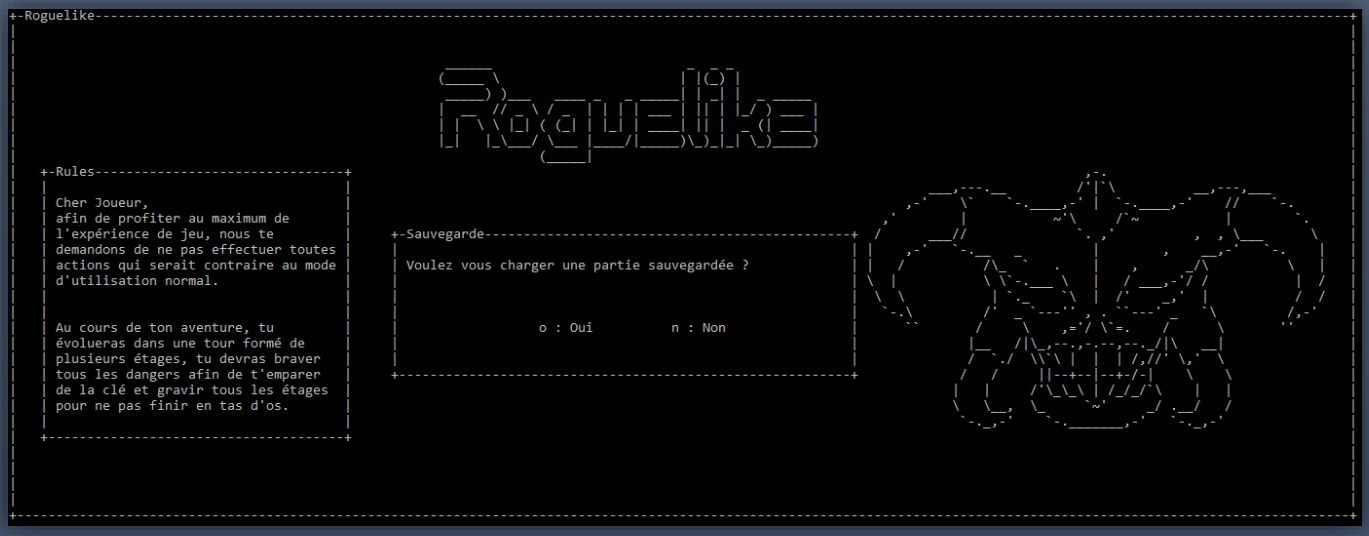
2016-2017



ROGUE LIKE

DANS LE CADRE DU PROJET L2

ANTHONY LARDY, ARTHUR TROTTIER ET FLORIAN SEBILLE

https://github.com/kirozz/ExtremRoguelike

Institut Informatique Claude Chappe

Université du Maine

SOMMAIRE

1. Introduction

Dans le cadre de notre projet du semestre 3 de la deuxième année de licence science pour l'ingénieur, nous cherchons à développer un jeu parmi une liste de sujets prédéfinis. Nous avons choisi parmi cette liste 3 sujets ordonnés par ordre de préférence afin que les professeurs nous en attribuent un, dans notre cas le sujet qui nous a été attribué était de mettre en place un jeu « Roguelike » de découverte d'un labyrinthe. Le projet présente différents objectifs :

-Créer et mettre en œuvre des algorithmes

-Gérer un projet (Dépôt/gestion de versions)

-Mettre en place les outils pour le développement (makefile, débogueur, documentation Doxygen)

-Présenter le résultat du travail par écrit et à l'oral (rapport, soutenance)

-Juger de nos capacités d'initiatives

-Nous constituer une boîte à outils de fonctions utiles

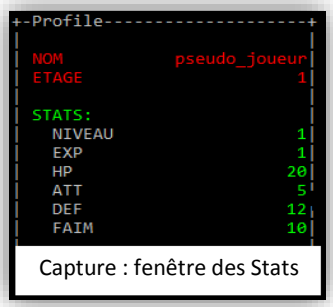
Le rapport se décomposera en 3 parties, premièrement l'organisation du travail avec la répartition et les outils et méthodes de travail utilisées puis deuxièmement la conception avec les règles du jeu et les choix de composition du jeu et pour finir troisièmement le développement qui regroupe le codage ainsi que les outils et méthodes de programmation.

2. Organisation Du Travail

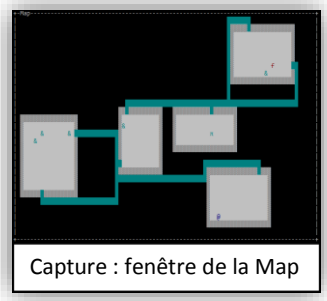
a) Répartition du travail

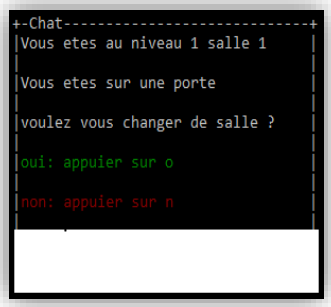
La répartition du travail, c'est fait au fur et à mesure que le projet avançait les séances de TP nous ont servi à mettre en commun notre travail, discuter des différents ajouts potentiels aux jeux comme l'ajout de médikit pour que le joueur puisse régénérer sa vie ou de monstre pour augmenter la difficulté. Et à se répartir les différentes tâches même si en dehors des séances de TP, nous continuons de se répartir ces différentes tâches et à réfléchir aussi sur d'éventuels ajouts.

Pendant la première semaine de travail, nous nous sommes concentrés sur la partie cahier des charges et étude du projet. Une fois cette partie finie, nous avons commencé à réfléchir sur la partie code. La problématique de l'affichage de l'interface du jeu s'est posée rapidement à nous. Nous avons donc choisi d'utiliser Ncurses pour gérer l'affichage. Une fois l'adaptation à Ncurses fait, nous avons commencé à nous séparer les taches. Dans un premier temps. Arthur s'est concentré sur la matrice de la map (création des salles et des couloirs). Quant à lui Anthony c'est occuper de la fenêtre des statistiques qui affiche les renseignements relatifs au joueur tel que son niveau, son nombre de points de vue ou encore sa nourriture. Quant à Florian, il s'est occupé de la fenêtre du chat qui renseigne le joueur sur sa position et les différents choix qu'il puisse faire (se déplacer, changer de salle ou encore changer de stage). Une fois ces taches bien commencées nous nous sommes réparti sur d'autres taches comme le déplacement du personnage sur la map la mise en place des escaliers pour changer de stage le lien entre les salles, l'interaction avec l'utilisateur via le chat en posant des questions au joueur comme s'il voulait changer de stage ou de salle. Une fois la partie de la génération de la map finit, Arthur a pu se lancer dans l'ajout des pièges, Anthony c'est ensuite occupé de l'écran d'accueil du jeu, ensuite il s'est occupé de la sauvegarde et pour finir il s'est occupé des montres et de leurs interactions avec le joueur, le principe des déplacements automatique et des combats. Florian quant à lui s'est occupé de la mise en place des soins pour le joueur (la nourriture et le médikit) ensuite je me suis occupé de l'écran de fin qui varie en fonction du résultat de fin (victoire ou défaite).



Capture : fenêtre des Stats


Capture : fenêtre de la Map



Vous pouvez donc voir que le travail s'est réparti au fur et à mesure de l'avancée du projet, tout en tenant compte de l'avancer de chacun dans leurs tâches et du temps estimé pour faire chaque tache. Cependant, grâce à plusieurs outils de communication et aux séances de TP, il a été possible de demander de l'aide aux personnes du groupe pour pouvoir mener à bien nos différentes parties.

b) Outils de travail

Pour travailler sur les mêmes fichiers de code tous en ne perdant aucune version des fichiers, nous avons utilisé GitHub. GitHub nous a permis de pouvoir modifier les mêmes fichiers en même temps tous en perdant aucune modification. GitHub nous a aussi permis de sauvegarder ces modifications en salle de TP et les reprendre à d'autres endroits. Il nous a servi aussi à tester des modifications en créant des branches tous en ne modifiant pas le code sur lequel les autres membres du groupe travaille et ensuite si la modification sur laquelle nous travaillons est fiable, la mettre à disposition de tout le groupe. Dans notre Projet, nous avons eu besoin d'utiliser dans branches à plusieurs moments par exemple, nous avons commencé par créer les trois fenêtres puis chacun à créer une branche pour travailler sur nos premières fonctions tous de notre côté puis une fois ces fonctions finies, nous avons pu mettre en commun sur la branche master.

Ci-dessus vous pouvez voir les différentes branches à un moment donné. Par exemple vous pouvez voir en noir la branche principale master ensuite en bleu la branche créée par Florian, puis envers la branche créée par Arthur et enfin la branche violette par Anthony. Pour la communication dans le projet, nous avons choisi d'utiliser Messenger pour des conversations instantanées comme tous les membres du groupe l'utilisaient auparavant. Ensuite, nous avons aussi travaillé avec Gitter

c) Méthode de travail

En ce qui concerne la méthode de travail, nous avons décidé dans un premier temps de se séparer complètement les taches pour ne pas se gêner même si nous étions tous à l'écoute sur les différents moyens de communication (Gitter et Messenger) ou en séance de TP si un membre du projet avait un problème à un moment. Dans un deuxième temps, nous avons travaillé sur les ajouts (comme les monstres la nourriture ou encore les médikit) du jeu donc, nous avons travail un commun sur les dernières semaines.

3. Conception

Nous allons voir dans cette partie, ce qui touche à la conception du jeu en tant que tel en passant des règles du jeu, aux éléments de composition du jeu et l'explication de ces choix

a) Règles du jeu

Afin d'avoir des informations sur le style de jeu du Roguelike et les différentes règles du jeu, nous avons été sur Internet pour trouver cela. Il est en ressorti que le Roguelike contrairement à la plupart des autres genres de jeux vidéo possède une définition qui lui est propre, cette définition a été explicitement définie lors de la première « International Roguelike Développement Conférence », en 2008, à Berlin. La liste des caractéristiques d'un Roguelike bien que certains s'en éloignent sur certains points est :

-Génération aléatoire de l'environnement

L'environnement du jeu est généré de façon aléatoire d'une manière que la jouabilité soit augmentée. L'apparition et le placement des différents éléments sont aléatoire, en ce qui concerne les monstres, leur apparition est fixe cependant leur placement est lui aléatoire.

-Mort permanente

Le joueur n'est pas censé gagner le jeu avec son premier personnage. L'échec doit mener à la mort permanente du personnage, c'est-à-dire que le joueur ne peut pas retourner en arrière lorsqu'il perd, néanmoins les sauvegardes sont possibles au moment de quitter le jeu.

-Tour par tour

Chaque commande correspond à une seule action / mouvement. Le jeu n'est pas sensible au temps, il est possible de prendre son temps pour choisir ses actions.

-Basé sur une grille

Le monde est représenté par une grille uniforme. Le personnage ainsi que les autres éléments font tous la même taille (un carreau de la grille).

-Non modal

Mouvement, bataille et autres actions, ont lieu dans le même mode. Chaque action doit être disponible à n'importe quel point du jeu.

-Complexité

Le jeu a suffisamment de complexité pour permettre plusieurs solutions aux objectifs communs. Cela est obtenu en fournissant assez d'élément / monstre et objet / objet interactif et est fortement connecté au fait d'avoir seulement un mode.

-La gestion des ressources

Le joueur devra gérer ses ressources qui sont limitées (par exemple, la nourriture, les potions de soins).

-Hack 'n' slash

Le hach 'n' slash est un genre de jeu vidéo où les combats en temps réel sont très présents. Dans un Roguelike aussi les combats contre les monstres sont importants. Le joueur joue contre le jeu, il n'existe pas de relations entre les monstres (comme les inimitiés ou la diplomatie).

-Exploration et découverte

Le jeu nécessite une exploration minutieuse des niveaux des donjons et la découverte de l'utilisation des objets non identifiés. Cela doit être fait à chaque fois que le joueur commence une nouvelle partie.

Voici les règles les plus importantes du Roguelike cependant, il existe d'autres règles qui elles sont moins influentes :

-Un joueur

Le joueur contre un seul personnage, le jeu est centré sur celui-ci, le monde est vu à travers lui et le mort de celui-ci mène à la fin du jeu.

-Les monstres sont semblables aux joueurs

Les règles qui s'appliquent au joueur s'appliquent également aux monstres. Ils ont un inventaire, de l'équipement, des objets utilisables, des sorts, etc.

-Défi tactique

Le joueur doit apprendre des tactiques avant de pouvoir faire des progrès significatifs. Ce processus se répète, c'est-à-dire la connaissance au début du jeu n'est pas suffisante pour battre les derniers niveaux. (En raison d'environnements aléatoires et de décès permanent, les Roguelikes sont un défi pour les nouveaux joueurs).

-Affichage ASCII

L'affichage traditionnel d'un Roguelike est de représenter le monde et ses éléments par des caractères ASCII.

-Donjons

Les Roguelikes contiennent des donjons tels que des niveaux composés de salles et de couloirs.

-Nombres

Les numéros utilisés pour décrire le personnage (points de vie, attributs, etc) sont délibérément affichés.

Toutes ces règles ne font pas l'unanimité en effet Darren Gray accuse dans son article « Screw the Berlin Interprétation ! » les règles proposées par cette conférence d'être inexact, dépassé et non représentatif d'un genre vibrant et ouvert. Il n'est donc pas possible de définir des règles du jeu qui font l'unanimité.

b) Élément / Composition du jeu

Après avoir vu les différentes règles que compose un Roguelike, nous allons nous intéresser aux différents éléments qui composent notre jeu.

Lorsqu'on lance le jeu, on tombe sur l'écran d'accueil (voir annexe n°2) qui nous demande sur l'on souhaite charger une partie sauvegardée ou en relancer une nouvelle. Cet écran est composé d'une fenêtre de règle, du titre du jeu et d'un ASCII art qui est la représentation d'une tête de monstre. Si le joueur saisit non à la question voulez-vous charger une partie une nouvelle fenêtre s'ouvre laissant la possibilité au joueur de saisir son pseudo pour la partie, dans le cas où le joueur voudrait charger une partie, le jeu soit repris là où la dernière partie a été sauvegardée soit s'il n'existe pas de sauvegarde la fenêtre demandant le pseudo apparaît et le joueur doit saisir celui-ci.

Une fois la partie lancée le jeu se décompose en trois parties :

-Une fenêtre « profile » (voir annexe n°3) qui est composé elle aussi de trois parties, une partie statistique, qui sont les statistiques du joueur (nom, étage, niveau, points de vie, attaque, faim), une partie que l'on pourrait appeler légende, car elle donne des informations que différents code ASCII afin d'aider de le joueur à comprendre l'environnement qui l'entoure et une partie commande qui résume les différentes commandes qu'il lui est possible d'effectuer (quitter, sauvegarder, se déplacer)

-Une fenêtre « Map » (voir annexe n°3), c'est ici qu'est affichée comme son nom l'indique la carte où se trouve le joueur.

-Une fenêtre « Chat » (voir annexe n°3), qui donne des informations pour le joueur telles que l'étage et la salle où il se trouve, et la procédure à suivre lorsqu'il se trouve sur une porte où un escalier.

Il existe deux fins possibles au jeu (voir annexe n°4), le joueur réussit à prendre la clé et arrive jusqu'à la porte finale à l'étage 5 et alors il gagne ou alors il meurt soit de faim soit dans un piège soit par un monstre et alors la partie est terminée. Dans ces deux cas le joueur a devant lui l'écran de fin. Dans les deux cas la fenêtre « Profile » affiche les statistiques du

joueur, la fenêtre « Map » est supprimée et deux nouvelles fenêtres sont créées « FIN » et « REJOUER » (demande au joueur s'il souhaite rejouer ou non) et la fenêtre « Chat » est toujours présente. En cas de victoire la fenêtre « FIN » affiche « Well Done » et félicite le joueur de sa victoire et un smiley est affiché dans la fenêtre « Chat ». En cas de mort, la fenêtre « FIN » affiche « Game Over » et donne les raisons de la mort de la personne, quant à la fenêtre « Chat » elle affiche une tête de mort.

c) Choix de composition

Après avoir vu les différents éléments qui composent notre Roguelike, nous allons nous expliquer pourquoi nous avons effectué ses choix plutôt que d'autres.

Les inspirations dans le choix des éléments ont été diverses étant donné que les règles de constructions d'un Roguelike ne sont pas strictement définies. En ce qui concerne l'écran d'accueil, nous avons pensé qu'il était important que le joueur rentre dans l'esprit du jeu dès qu'il le lance au lieu qu'il se retrouve qu'avec une fenêtre pour lui demander s'il veut charger une sauvegarde, il en est de même pour l'écran de fin. La fenêtre « Profile » vient du jeu Angband(voir annexe n°5) qui utilisait une même fenêtre pour afficher les statistiques du personnage, cependant nous avons fait le choix d'ajouter les commandes ainsi que certaines légendes afin que le joueur puisse avoir une prise en main du jeu plus rapide. La fenêtre « Chat » est quant à elle une idée que nous avons retrouvé dans aucun autre jeu de ce type, cette fenêtre est avant tout là pour afficher au joueur les informations contextuelles comme, lorsqu'il est sur une porte ou un escalier et plus tard peut-être les points de vie infligés aux monstres et vice versa. Pour ce qui est de l'affichage de la carte nous avons choisi quelque chose dans le style de NetHack(voir annexe n°5) avec un sol et des murs qui ne sont pas des caractères ASCII, puis pour les différents éléments tels que la nourriture ou le personnage, les caractères ont un rapport avec ce qu'ils sont par exemple la nourriture est un « f » (comme food) ou la clé un « K » (comme key), cependant le personnage est lui un « @ » afin de faire un clin d'œil au genre du Roguelike. Nous avons fait le choix que le Game Play ainsi que l'esthétique du jeu soient des points importants.

4. Développement

a) Initialisation, structures, constantes

Le code est structuré en 2 aspects : un aspect purement matriciel qui nous permet de manipuler et de mettre à jour notre jeu et un aspect graphique travaillé à l'aide de la bibliothèque ncurses.

En ce qui concerne les constantes, la taille de la matrice est préétablie et ne sera plus modifiée par la suite. C'est également le cas des dimensions des différentes fenêtres d'affichage dont la fenêtre principale qui contient le jeu (adaptée à la matrice).

Pour créer et modifier notre jeu, nous utilisons trois structures :

- une structure t_cellule qui contient différents indicateurs tels que les coordonnées (la matrice étant de ce type) qui permettent de caractériser chaque cellule et de construire la carte.

- une structure t_joueur qui contient les caractéristiques d'un joueur tel que son nom, ses points de vie etc.

- une structure t_monstre qui contient les caractéristiques d'un monstre.

Ainsi qu'un type énuméré qui regroupe les différentes possibilités qu'une case de la carte peut prendre (par exemple : porte ou mur).

Les librairies utilisées sont au nombre de 8 parmi lesquels il y a :

- les indispensables (stdio.h, string.h, math.h, stdlib.h, unistd.h)

- les librairies d'affichage (ncurses.h, curses.h)

Pour finir, il est nécessaire d'initialiser les couleurs souhaitées lors de l'affichage (comme ci-contre) avec ncurses pour par la suite les utiliser.

```
init_pair(1, COLOR_RED, -1);
init_pair(2, COLOR_GREEN, -1);
init_pair(3, COLOR_BLUE, COLOR_WHITE);
init_pair(4, COLOR_BLACK, COLOR_WHITE);
init_pair(5, COLOR_CYAN, COLOR_WHITE);
init_pair(6, COLOR_WHITE, COLOR_CYAN);
init_pair(7, COLOR_RED, COLOR_WHITE);
init_pair(8, COLOR_GREEN, COLOR_WHITE);
init_pair(9, COLOR_YELLOW, -1);
```

b) Fichiers et fonctions principales

Notre code est divisé en plusieurs fichiers qui ont, chacun rôle bien précis ce qui facilite la compréhension et par la suite les modifications.

Une première partie comporte les fichiers s'occupant de la génération, que ce soit des fenêtres ou de la matrice. On y retrouve dans un premier lieu le fichier *GenWindow.c* qui a pour but de créer un espace de jeu agréable pour un Roguelike dans le terminal. Il fait appel à des fonctions prédéfinies de la librairie ncurses qui permettent de créer des fenêtres et donc de définir des limites.

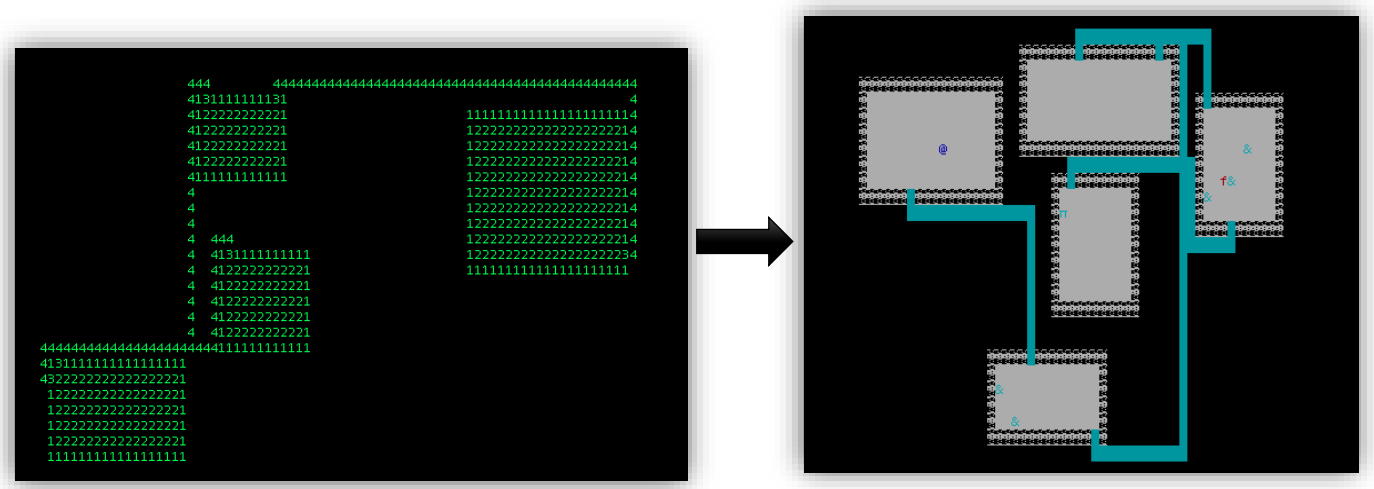
Les fenêtres créées, il faut y ajouter du contenu. Le cahier des charges nous indique que le jeu doit être complètement aléatoire et différent à chaque nouvelle partie. La seule zone impactée par cela est la zone de jeu, c'est-à-dire la carte. Le fichier *RandomRoom.c* modifie donc la matrice pour créer une carte contenant un nombre aléatoire de salles de taille aléatoire relié entre elles par des couloirs.

Pour finir, il serait trop simple pour le joueur de se déplacer librement dans la carte. Le fichier *Placer_element.c* s'occupe donc de générer des pièges, de la nourriture, des monstres, des médikit… C'est-à-dire tout élément susceptible d'améliorer le gameplay et de rendre le jeu plus intéressant.

Une seconde partie comporte les fichiers plus axés sur la gestion de l'affichage. Ici nous avons décidé de regrouper toutes les fonctions dans un même fichier : *Affichage.c*.

On y retrouve une fonction s'occupant de transcrire la matrice en une carte composée de différents caractères pour permettre à l'utilisateur de s'immerger au mieux dans le jeu. Ce fichier permet aussi d'indiquer au joueur dans une fenêtre annexe (qui pourrait s'apparenter à un menu) ses caractéristiques actuelles telles que son nom, ses points de vie ou sa barre de faim. Dans cette même fenêtre nous avons décidé d'y ajouter une légende pour que le joueur comprenne à quoi correspond chaque caractère, car ce n'est pas si intuitif que cela au premier regard. Pour finir, les commandes utiles et possibles à tout moment de la partie sont aussi affichées ici.

Puis, pour dans notre troisième fenêtre qui s'apparente à un chat, un fil d'information, il nous fallait des fonctions permettant d'inscrire du texte ainsi que de l'effacer en fonction des mouvements du joueur. Toutes ces fonctions sont aussi comprises dans ce fichier.



Carte aléatoire avant gestion de l'affichage

Carte aléatoire après gestion de l'affichage

La troisième partie s'occupe quant à elle des fichiers permettant une sauvegarde totale à n'importe quel moment de la partie. On y retrouve deux fichiers.

Le premier fichier *sauvegarde.c* contient la fonction qui permet au joueur de conserver l'état actuel de sa partie. Toutes les données importantes seront inscrites dans un fichier texte. Puis si l'utilisateur décide de repartir de l'endroit où il s'était arrêté, il peut recharger sa partie grâce au fichier *Charger_Sauvegarde.c* qui lui assure de se retrouver dans les mêmes conditions qu'auparavant. Actuellement, il n'y a qu'un seul espace disponible qui s'écrase à chaque nouvelle sauvegarde.

La dernière partie aborde l'interaction avec le joueur et la mise à jour du jeu en fonction de ses déplacements, actions, etc.

Le premier fichier *Ask_Load_Save.c* a pour but d'afficher le menu d'introduction du jeu, et de rentrer en contact avec le joueur en lui demandant s'il souhaite reprendre une partie ou recommencer ainsi que son pseudo.

Dans le même style, le fichier *fin_de_la_partie.c* va déterminer le menu de fin, c'est-à-dire de victoire ou de mort et choisir un affichage conséquent.

Le reste des fichiers s'occupe plus précisément du jeu. En effet, le joueur peut se déplacer librement dans la carte, c'est le fichier *Depl_perso.c* qui le permet et qui s'occupe de mettre à jour les différentes caractéristiques du joueur en fonction de tel ou tel déplacement (Exemple : Si le joueur effectue 7 déplacements, alors il perd 1 de faim). Puis pour progresser dans le jeu, le joueur doit emprunter des portes (pour changer de salle) et des escaliers (pour monter de niveau). C'est le rôle du fichier *porte_ou_escalier.c* qui proposera au joueur à chaque fois s'il souhaite passer la porte ou monter l'escalier, il y a donc un système d'interactions avec le joueur qui n'est forcé à rien et qui n'est pas pris en traitre. Pour finir 2 fichiers s'occupent des monstres, le premier de ses déplacements (*monstre.c*) qui sont complètement aléatoire. À noter qu'un monstre ne peut pas sortir de la salle où il a été généré. Le second fichier (*combat.c*) s'occupe quant à lui du système de combat qui est à l'état actuel assez punitif puisque si le joueur se situe à 1 de distance du monstre, il est directement tué.

c) Tests

Durant l'écriture du code, il était obligatoire de tester nos fonctions et plus globalement notre jeu assez régulièrement. Pour la matrice et la génération d'une carte, il était assez simple de l'afficher dans le terminal pour en voir le résultat. Malgré tout, pour les couloirs, nous avons utilisé un fichier texte annexe dans lequel on écrivait à chaque étape l'état du *flood & fill* (technique utilisée pour la recherche de couloirs), ce qui permettait de comprendre à quel moment et pourquoi celui-ci se stoppait. Pour le reste, c'est à dire les fonctions d'affichage et d'interaction, celles-ci influençant uniquement lorsque ncurses rentrait en compte, nous étions obligés de faire tourner le jeu en boucle et d'y jouer pour déceler bugs et problèmes. Les seuls et uniques tests unitaires furent donc effectués sur des fonctions simples tel que *aleat* qui retourne un nombre aléatoire entre 2 bornes ou encore *fillmap* qui place la carte à vide (Voir annexe N°1).

d) Programmation modulaire

Pour faciliter la compilation et relier les fonctions entre elles, nous avons décidé de créer un fichier *total.h* qui regroupe tous les prototypes des fonctions utiles au jeu, les structures nécessaires ainsi que certaines variables globales utiles dans plusieurs fichiers. Dans un premier temps, nous avions commencé à créer un fichier .h par fichier créé, mais du fait du grand nombre de fichiers, nous avons opté pour cette méthode. Le makefile s'organise simplement avec une compilation séparée de chaque fichier puis une compilation globale pour obtenir l'exécutable. A la fin de chaque make, tous les fichiers .o sont supprimés pour ne pas surcharger le dossier ainsi que le git.

5. Résultats et conclusion

a) Bilan

La création du jeu achevé, nous en tirons un bilan positif. En effet, après avoir reçu le sujet voulu, nous ne savions pas quelles seraient nos limites bien que motivées à l'idée de le faire. Cependant, notre jeu n'est pas parfait ni complétement fini, car pour un jeu de ce type il est toujours possible d'enrichir le scénario en y ajoutant de nouveaux éléments. Il persiste encore un léger « bug » capable de faire « crash » notre jeu mais que l'on compte bien régler dans les jours à venir (ayant passé déjà pas mal de temps dessus). Ce projet est malgré tout une réussite, car nous avons réussi à intégrer la plupart des éléments souhaités au départ mais aussi des éléments qui nous ont apparus comme nécessaires par la suite et qu'il ne l'était pas lors de l'élaboration du cahier des charges.

b) Améliorations

Le jeu est fonctionnel est intéressant à jouer, mais comme dit précédemment, il est toujours possible de rajouter de nouveaux éléments pour améliorer encore plus le « gameplay ». Parmi ces éléments, nous aurions aimé intégrer un système plus poussé de combat contre les monstres en faisant entrer en compte les points d'attaque et de défense et qui se serait déroulé tour par tour avec une probabilité pour chaque combattant (le joueur et le monstre) de rater ou de réussir son attaque. La deuxième chose que nous aimerions rajouter est un chronomètre pour chaque carte rendant le jeu encore plus extrême et ne permettant pas au joueur de se reposer sur ses lauriers pour par la suite établir un classement en y répertoriant les différents temps de chacun. Puis pour finir, quelque chose de plus utilitaire : mettre à disposition plus d'un espace de sauvegarde comme c'est le cas actuellement pour le(s) joueur(s) et ainsi ne pas avoir à écraser le même fichier de sauvegarde (surtout s'il y a plusieurs joueurs).

c) Gain personnel

Dans un premier temps, le projet fut pour nous trois l'occasion de travailler en équipe pendant un mois et donc de s'organiser au mieux pour travailler efficacement. Ce fut le cas, personne ne refusa de travailler et tout le monde s'entraidait dès qu'il était nécessaire. Dans un second temps, d'un point de vue plus technique, cela nous a permis de nous perfectionner dans l'utilisation du langage C, mais aussi de découvrir la librairie Ncurses qui fut la base de l'affichage de notre jeu. Pour conclure, la création de ce jeu fut un projet vraiment intéressant et qui a permis de nous conforter grandement sur le choix de la branche informatique.

6. Annexes

Annexe N°1 :

Test N°1 : aleat(int min, int max)

```c
#include <stdlib.h>
#include <time.h>
#include <stdio.h>



int aleat(int min, int max){
    return (rand() % (max - min + 1)) + min;
}

int main(){
  int i;
  int nb_tour = 10;
  int result;

  for(i = 1; i <= nb_tour; i++){

    result = aleat(1,10);

    if(result >= 1 && result <= 10){
      printf("Test %i réussi \n",i);
    }
  }
}
```

```
essaie_ncurses|master⚡⇒ gcc -o Test test.c
essaie_ncurses|master⚡⇒ ./Test
Test 1 réussi
Test 2 réussi
Test 3 réussi
Test 4 réussi
Test 5 réussi
Test 6 réussi
Test 7 réussi
Test 8 réussi
Test 9 réussi
Test 10 réussi
```

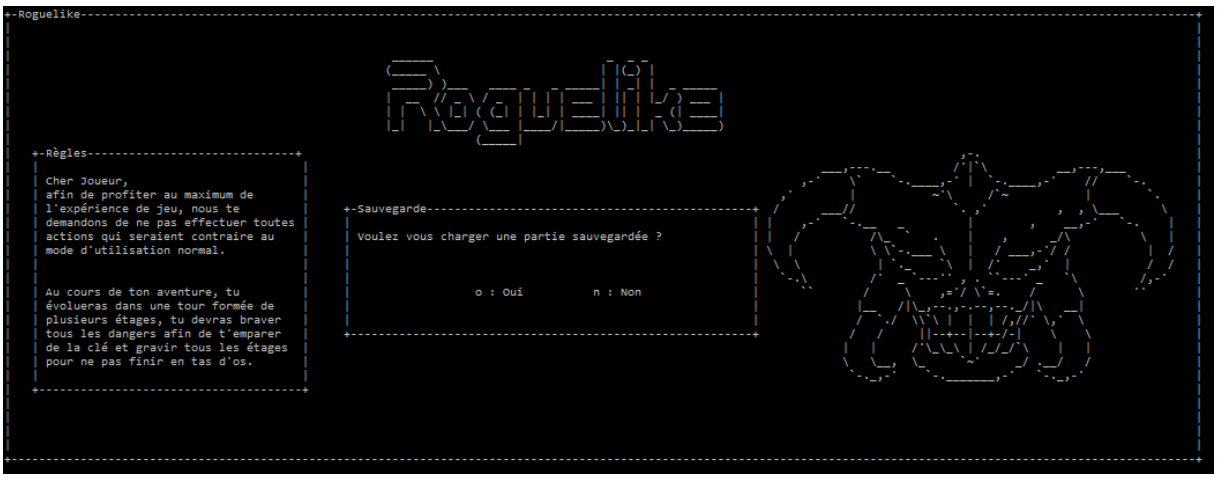
Test N°2 : fillmap()

```c
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#define x 30
#define y 89

typedef enum {vide, mur, sol, porte, couloir, personnage, uplevel, arriver, cle, food, piege, medikit, mechant} t_element;
typedef struct {t_element lieu; int position; int relie; int xb;int yb;int num_salle;int presence;} t_cellule;

t_cellule MAP[x][y];

void fillmap(){

  int i, j;
  for(i = 0; i < x; i++){
    for(j = 0; j < y; j++){
      MAP[i][j].lieu = vide;
      MAP[i][j].position = 0;
      MAP[i][j].relie = 0;
    }
  }
}

int main(){
  int vide = 0, position = 0, relie = 0;
  int i, j;
  fillmap();
  for(i = 0; i < x; i++){
    for(j = 0; j < y; j++){
      if(MAP[i][j].lieu != vide) vide = 1;
      if(MAP[i][j].position != 0) position = 1;
      if(MAP[i][j].relie != 0) relie = 1;
    }
  }
  if(vide == 0)printf("Test lieu réussi \n");
  else printf("Test lieu echec \n");
  if(position == 0)printf("Test position réussi \n");
  else printf("Test position echec \n");
  if(relie == 0)printf("Test relie réussi \n");
  else printf("Test relie echec \n");
}
```

```
[essaie_ncurses|master⚡⇒ gcc -o Test test.c
[essaie_ncurses|master⚡⇒ ./Test
Test lieu réussi
Test position réussi
Test relie réussi
```

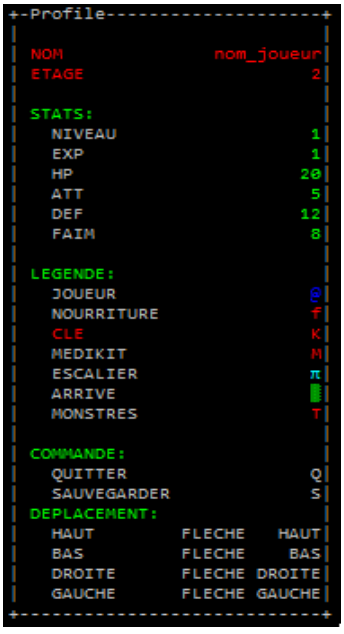
Annexe N°2 :

Ecran d'accueil du jeu.



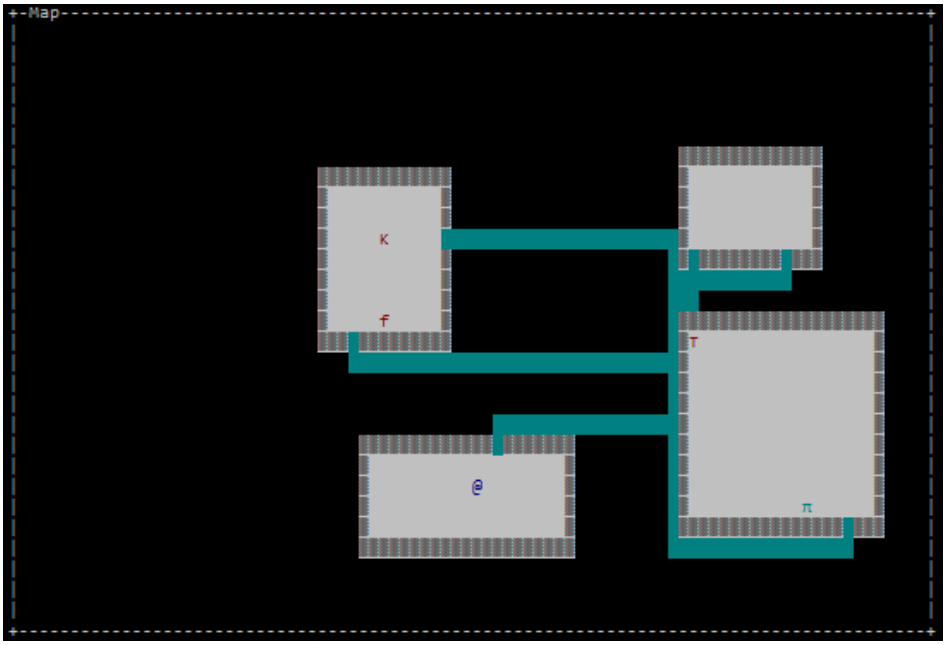
Annexe N°3 :

Fenêtre « Profile »



Fenêtre « Chat »

Fenêtre « Map ».



Annexe N°4 :

Ecran de fin dans le cas où le joueur a réussi à sortir.

Ecran de fin dans le cas où le joueur est mort.

Annexe N°5 :

Angband, un Roguelike de 1990.

NetHack un Roguelike de 1987 après une révision du jeu modifiant les graphismes.