1) Initialisation, structures, constantes

Le code est structuré en 2 aspects : un aspect purement matriciel qui nous permet de manipuler et 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 préétabli et ne sera plus modifié 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 tel que les coordonnées (la matrice étant de ce type) qui permet de caractériser chaque cellule et 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.

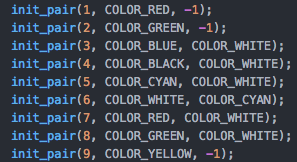
Ainsi qu’un type énuméré qui regroupe les différentes possibilités qu’une case de la carte peut prendre (Par exemple : porte ou mur).

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-dessous) avec ncurses pour par la suite les utiliser.



2) 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 prédéfinis de la librairies ncurses qui permettent de créer des fenêtres et donc de définir des limites.

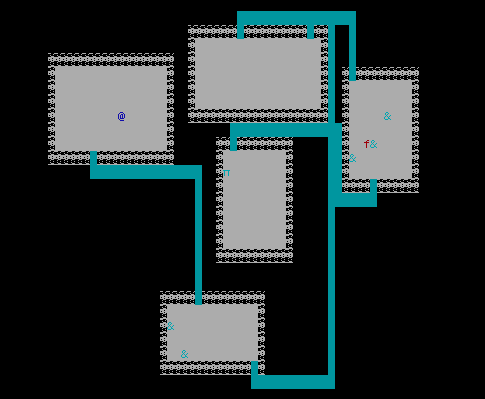
Les fenêtres créer, il faut y ajouter du contenu. Le cahier des charges nous indique que le jeu doit être complètement aléatoire et différent à chaque nouvelle partie. La seule zone impacté par cela est la zone de jeu, c’est à dire la carte. Le fichier *RandomRoom.c* modifie donc la matrice pour créer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

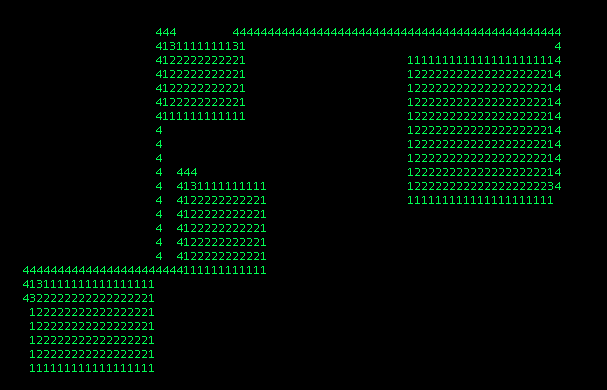
Pour finir, il serait trop simple pour le joueur de se déplacer librement dans la carte. Le fichier *Placer\_element.c* s’occupe donc de générer des pièges, de la nourriture, des monstres, des médikit… C’est à dire tout élément susceptible d’améliorer le gameplay et de rendre le jeu plus intéressant.

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

On y retrouve une fonctions 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) ces caractéristiques actuels tel que son nom, ses points de vie ou sa barre de faim. Dans cette même fenêtre nous avons décidés d’y ajouter une légende pour que le joueur comprenne à quoi correspondent chaque caractère car ce n’est pas si intuitif que ça 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 après gestion de l’affichage

Carte aléatoire avant 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 si il souhaite passer la porte ou monter l’escalier, il y a donc un système d’interaction avec le joueur qui n’est forcé à rien et qui n’est pas pris en traitre. Pour finir 2 fichiers s’occupe des monstres, le premier de ses déplacements (*monstre.c*) qui sont complétement 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.

3) Tests

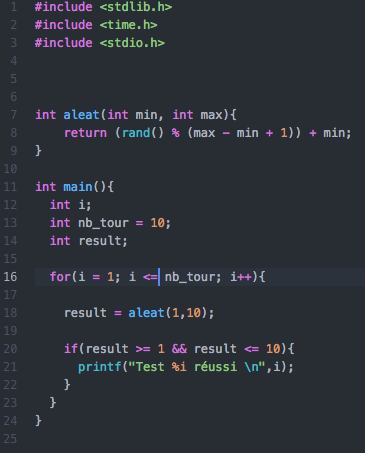
Durant l’écriture du code, il était obligatoire de tester nos fonctions et plus globalement notre jeu assez régulièrement. Pour la matrice et la génération d’une carte, il était assez simple de l’afficher dans le terminal pour en voir le résultat. Malgré tout, pour les couloirs nous avons utilisés un 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 beugs et problèmes. Les seuls et uniques test unitaire furent donc effectuer sur des fonctions simple tel que *aleat* qui retourne un nombre aléatoire entre 2 bornes ou encore *fillmap* qui place la carte à vide (Voir annexe).

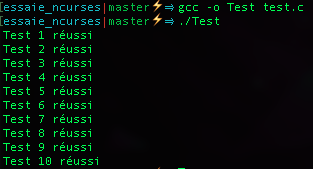
4) Programmation modulaire

Pour faciliter la compilation et relier les fonctions entre elles, nous avons décidés de créer un fichier *total.h* qui regroupe tous les prototypes des fonctions utiles au jeu, les structures nécessaires ainsi que certains variables globales utiles dans plusieurs fichiers. Dans un premier temps, nous avions commencés à créer un fichier .h par fichier créé mais du fait du grand nombre de fichier, nous avons optés 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.

Annexe 1

Test N°1 : aleat(int min, int max)





Test N°2 : fillmap()

