L2 Sdn 2023

**Donjon Crawler**

– Projet C –

Yazid Kilito – Hugo Hebbinckuys

Table des matières

[I. Introduction 3](#_Toc152281697)

[a) Le problème 3](#_Toc152281698)

[b) Les impératifs et les options 3](#_Toc152281699)

[c) Le plan du Rapport 4](#_Toc152281700)

[II. Le développement 4](#_Toc152281701)

[a) les structures 5](#_Toc152281702)

[b) Le programme 6](#_Toc152281703)

[c) Ajout personnel 11](#_Toc152281704)

[d) Problèmes 12](#_Toc152281705)

[III. Conclusion 12](#_Toc152281706)

# Introduction

## Le problème

// intro du sujet lui-meme // Les dungeon crawlers sont des jeux (de société ou vidéo) qui mettent l’accent sur l’exploration de donjons. Dans cette sorte de jeux un donjon se concrétise par une carte sur laquelle figure des informations importantes comme des portes, des monstres et des trésors. On souhaite développer quelques bibliothèques C rudimentaires simplifiant la gestion de donjons. Pour notre application, les donjons seront représentés par des grilles dont chaque case peut contenir au plus un élément.

La mise en forme du créateur de donjon est très libre et des choix ont dû être fait pour proposer ce projet sous la forme qui nous paraissait être la meilleure.

## Les impératifs et les options

Il fallait s’organiser entre les heures de cours, le travail des autres matières et l’avancée du projet. Chose que nous avons su faire en nous retrouvant au début au moins une fois par semaine pour discuter de l’organisation de notre application, et pour rapidement s’attaquer au problème.

Nous travaillons au début souvent ensemble et c’était important, pour s’accorder sur la manière de construire nos éléments. C’était une étape plus contraignante puisqu’il fallait être disponible au même moment, mais rien ne nous empêchait d’y réfléchir seul et avancer dans nos idées. Vers la fin du projet nous étions plus à même de travailler chacun de notre coté en sachant les choses sur lesquelles il fallait avancer.

Nous n’avions pas convenu de tâches pour chacun au début, c’est seulement quand on faisait face à certains problèmes, ou que l’on devait avancer sur différentes choses que nous nous répartissions les rôles. Soit le problème était « majeur » et assez important et nous réfléchissions ensemble pour trouver une solution, un changement d’architecture ou des fonctions pour remédier aux problèmes. Soit le problème était moins « général » et celui qui y faisait face essayait d’y remédier pendant que l’autre avançait sur autre chose.

Au départ, l'objectif principal était de créer un Dungeon crawler efficace tout en limitant considérablement les actions de l'utilisateur pour garantir le bon fonctionnement sans bugs du logiciel. Nous avions envisagé de restreindre au maximum les éléments que l'utilisateur pouvait manipuler. Cependant, après une discussion avec notre professeur, nous avons réalisé que le potentiel créatif était un élément clé dans la réussite du projet.

Nous avons donc dû trouver un équilibre délicat entre fonctionnalité et créativité. Nous avons accepté que, de temps à autre, l'utilisateur puisse entreprendre des actions susceptibles de modifier   
De façon significative, l'utilisateur peut altérer l'aspect et le fonctionnement du donjon, par exemple, en créant des couloirs qui semblent conduire vers un mur. Cette action peut sembler irrationnelle au premier abord et compliquer la programmation, mais elle se révèle être une fonctionnalité potentiellement intéressante par la suite, comme illustré par l'exemple d'un mur invisible qui disparaît lorsque la « Hitbox » du personnage entre en contact avec lui.

## Le plan du Rapport

Nous allons tout d’abord faire part des structures que nous avons mis en place pour parvenir à la création de notre application. Puis nous parleront du développement des fonctions. Nous parlerons également des choix et des problèmes que nous avons rencontrés durant notre travail.

# Le développement

Comme nous l’avons dit dans l’introduction, au cours du développement de notre application, les objectifs, contraintes et envies ont évolué tout au long de la réalisation. En effet, l’arrivée de nouvelles idées, les changements d’architecture de certaines structures pour faciliter la suite du travail, et aussi la rencontre de certains problèmes nous ont poussé à constamment remettre en question la manière dont nous avions décidé de prendre les choses.

Le développement s’est articulé en plusieurs parties.

La première était la base, la manière dont nous imaginions les choses la plus simplement. Au tout début nous testions les choses pour afficher tout simplement un rectangle vide pour avoir une idée de comment symboliser les donjons. Et étonnamment la première idée qui nous est venu en tête était de plutôt organiser notre donjon sous forme de liste chainée. Chaque élément pointerait vers l’élément suivant pour, au fur et à mesure afficher une ligne de ‘#’ puis des lignes vides et ensuite une nouvelle ligne de ‘#’, pour finalement avoir un rectangle de ‘#’.

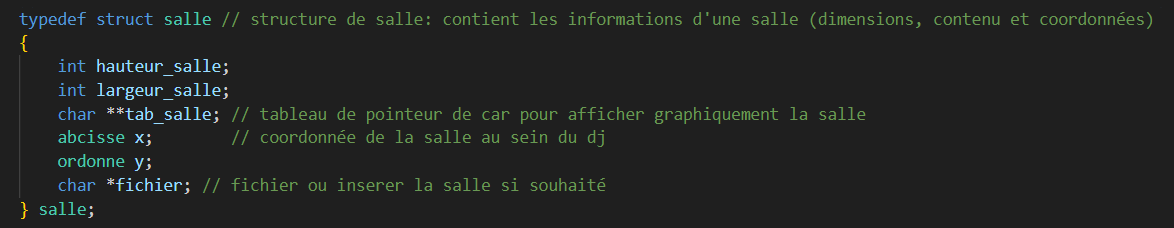
Puis, une manière de faire bien plus simple nous est venue en tête : faire des tableaux de tableaux. Le tableau du donjon aurait alors des tableaux de chaines de caractères, chaque tableau de chaine de caractères symbolisant une ligne du donjon. Cette solution nous est apparu comme bien plus simple et plus logique. Nous avons donc décidé de choisir cette méthode pour construire nos donjons.

A ce stade, nous travaillions donc ensemble, mais essayant de trouver les bonnes méthodes chacun de notre côté, pour ensuite mettre en commun et voir quelle solution était la plus simple et optimale des deux méthodes proposées.

Une fois la solution claire dans nos têtes pour la construction de nos donjons, nous nous sommes mis réellement au travail pour avancer sur le reste, conscient qu’il y avait encore beaucoup de choses à réaliser. Ainsi on s’est mis à travailler sur les structures de salles, là aussi c’était encore le début donc nous avancions pas à pas. Après les donjons et salles nous avons avancé plus rapidement étant donné que ce qui était fait jusqu’ici était cohérent et nous poussait à faire évoluer notre travail dans de bonnes condition puisque la suite était conditionnée par la cohérence de nos choix jusqu’ici. Il a donc été largement possible de continuer à travailler sur nos structures existantes et sur des nouvelles fonctions, étants persuadés que nos choix étaient judicieux et qu’ils étaient réfléchis.

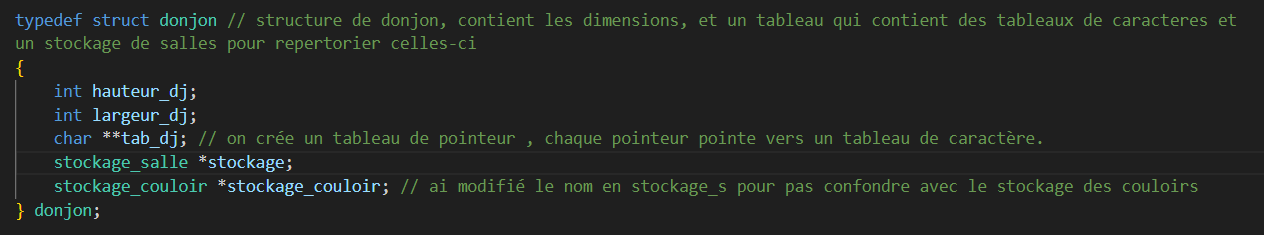
Nos structures, bien que judicieuses dans nos esprits, ont quand même également évolué. Notamment la structure de salle qui à la base n’était qu’un simple renommage du donjon. Nous étions partis du principe qu’une salle était un « donjon » que nous irions insérer à l’intérieur même du donjon. Finalement, cette structure à évoluée et sa version finale et assez différente d’un simple donjon.

## a) les structures

En effet notre structure de salle est la suivante :

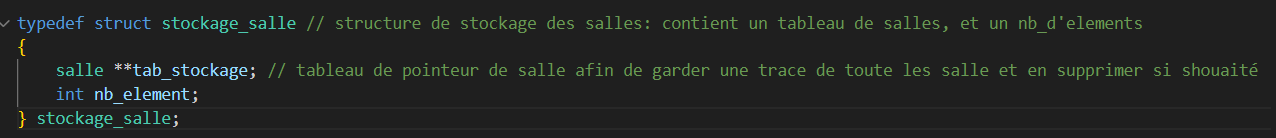
Là où jusqu’ici la structure de donjon comporte certaines informations différentes des salles, et où la structure de salle comporte elle aussi des éléments qui ne sont pas dans un donjon comme les coordonnées de la salle ou encore le fichier dont nous parlerons plus tard.

On peut également voir dans notre structure de salle que des nouveaux types ont été recréés, pour une question de simplicité et surtout de bonne lisibilité du code, nous avon renommé le type `int` en abscisse et également en ordonne. D’autant plus que ces types de coordonnées ne sont pas utiles uniquement pour les salles.

Voici notre structure de donjon :

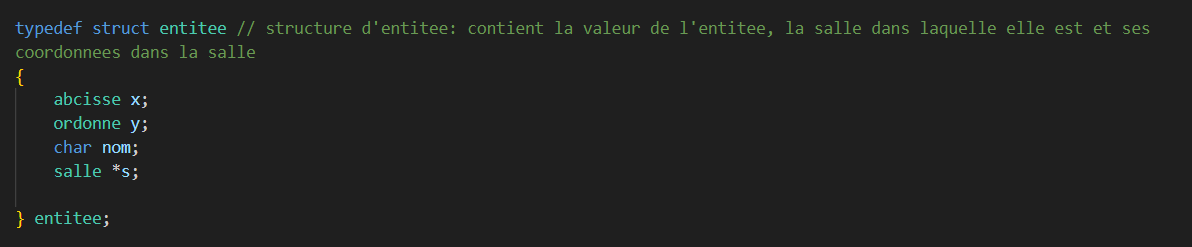
On peut voir ici que notre donjon à une structure de stockage de salles, et de stockage de couloirs pour pouvoir utiliser ces derniers (notamment les supprimer).

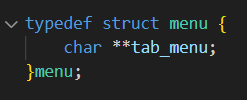
Voici notre structure de stockage de salle :



Cette dernière se compose simplement d’un tableau de salles, ainsi on aura accès à toutes les salles créées dans la construction d’un donjon. Ainsi que le nombre d’éléments (le nombre de salles) qui sera incrémenté à chaque nouvelle salle insérée dans le stockage. Ce nb\_element nous permettra d’indiquer un numero de fin de boucle lorsqu’on voudra parcourir les salles du stockage. C’est la même chose pour le stockage des couloirs.

Nous avons aussi une structure entitee qui représente les éléments qu’on voudra placer dans une salle. Ces entitees sont composées d’un nom (par exemple ‘M’ pour Monstre), de coordonnées x et y, et d’une salle dans laquelle ils sont inséré.



Il y a également une structure de menu pour simplifier les choses pour notre menu d’options :

## Le programme

Notre programme comporte beaucoup de fonctions, elles sont parfois utilisées à l’intérieur d’autres fonctions. Nous avons articulé notre code de la manière la plus « optimisée » possible, en prenant en compte l’échéance qui arrivait à grands pas, et nos compétences. Nous estimons que notre programme est loin d’être à tous les endroits le plus optimale possible mais avons fait de notre mieux pour produire quelque chose de cohérent à rendre dans le temps imparti.

Nous allons ici décrire les fonctionnements des grands principes de notre code. Pour que l’appréhension de nos choix soit compréhensible à travers le code lui-même.

* Les donjons.

Ayants déjà expliqué les donjon dans les grandes lignes avec la définition de sa structure, nous allons un peu plus rentrer dans les détails.

La fonction initialiser\_donjon(donjon \*d, int hauteur, int largeur) prend en paramètre le pointeur d’un *donjon* déclaré auparavant, qui par le biais de cette fonction va être modifié. On va allouer la mémoire pour le stockage des salles et des couloirs. Et assigner les paramètres *hauteur* et *largeur* aux elements *hauteur\_dj* et *largeur\_dj* du donjon. Ensuite, une première boucle qui symbolisera chaque lignes, et donc chaque tableau qui contient les tableau de caracteres, sera intialisé, dans laquelle on allouera la mémoire nécessaire, c’est-à-dire la hauteur du donjon multiplié par la taille d’un char. Puis directement une autre boucle est initialisé pour parcourir cette fois chaque tableau de caracteres dans lequel on se situe d’après la première boucle. A partir de la on assignera à chaque élément du tableau de caractere le bon caractere. A savoir pour la première et dernière ligne, chaque caractere devra être ‘#’ et pour les lignes intermediaires le premier et dernier caractere de chaque ligne seront des ‘#’ là où le reste sera un espace ‘ ‘.

Une fonction afficher\_donjon(donjon \*d) va afficher d’abord le menu interactif que nous avons créé, puis caractere par caractere le contenu du donjon.

Une fonction liberer\_donjon(donjon \*d) est destiné à libérer l’espace alloué pour le donjon pour éviter les fuites de mémoire.

Dans l’affichage du donjon, nous avons décidé de faire apparaître en marge du donjon des coordonnées pour les lignes et les colonnes pour faciliter l’utilisation de nos fonctionnalités.

* Les salles

Le fonctionnement des salles, leur affichage et la libération de la mémoire est exactement le même que pour les donjons. D’où au premier abord l’envie de faire un simple renommage de type pour le donjon et la salle.

Finalement nous avions besoin de plus d’informations sur les salles pour ne faire que ce renommage. La différence réside dans le fait de stocker les coordonnées de la salle dans le donjon.

Pour ce qui concerne l’insertion des salles dans le donjon, une fonction chevaucher permet de savoir si une salle est déjà insérée dans le donjon à l’emplacement de la nouvelle salle, l’emplacement étant connu puisque ses coordonnées sont connues. Cette fonction retourne l’indice de la salle qui est chevauchée s’il y en a une, ce qui permettra de directement mettre à nul la salle de l’indice en question dans le stockage de salle. On ne supprime pas les salles du stockage mais les mettons à nul pour ne pas bouleverser les indices de salles antérieures et futures. Une fonction supprimer\_salle est dédiée à cette suppression. Elle réassignes les caractères vides ‘ ‘ aux endroits ou les caractères sont ceux de la salle qu’on veut supprimer.

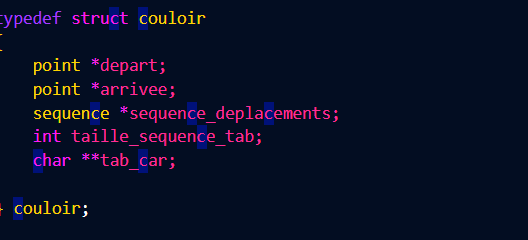
Aussi, nous avons mis en place une sécurité pour qu’il soit impossible d’insérer une salle si elle dépasse les dimensions du donjon.

* Les entitees

L’initialisation des entitees se fait par le biais de la salle dans laquelle elle se trouve, son nom (ex : ‘M’ pour Monstre) et de ses coordonnées. Le principe est la aussi semblable a l’initialisation des salles, en modifiant le caractère qu’il faut modifier de par ses coordonnées dans ses salles. Une sécurité est également mise en place pour ne pas initialiser une entitee qui n’existe pas.

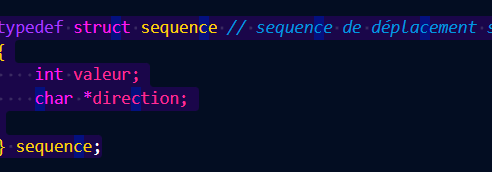
* Les couloirs

Dans la première version, nous avions envisagé de créer des couloirs à partir de deux points d'accès de salle, en calculant la différence de distance en termes de coordonnées x et y entre la salle de destination et la salle de départ. Cependant, cette approche restreignait l'utilisateur à créer des couloirs uniquement entre deux salles, et ceux-ci auraient été très similaires les uns aux autres, manquant ainsi de créativité.

Par conséquent, nous avons opté pour une approche plus libre dans la création des couloirs. Pour ce faire, nous avons mis en place une nouvelle structure de couloir.

Cette structure repose sur deux autres structures et un attribut que nous expliquerons ultérieurement (point, séquence, taille\_sequence\_tab) et sur un tableau de pointeurs de caractères qui servira à stocker le couloir sous une forme "graphique".

La structure "point" est définie avec deux entiers renommés respectivement en "abscisse" et "ordonnée" pour une meilleure compréhension du sujet.



La structure "séquence" est composée d'une valeur numérique et d'une chaîne de caractères. La valeur numérique représente le nombre de cases que le couloir effectuera dans la direction indiquée par la chaîne de caractères. Une direction peut être l'une des suivantes : [NORD, OUEST, SUD, EST]. Nous n'avons pas implémenté les déplacements en diagonale, car cela n'apportait pas significativement plus de valeur ajoutée compte tenu de la grande liberté déjà offerte par nos couloirs, et cela aurait complexifié davantage le processus de création en termes de temps.

La structure "couloir" est ainsi constituée de deux points, un point de départ et un point d'arrivée, qui sont des paires de coordonnées (x, y). Elle comporte également un tableau de séquences, où chaque séquence est un couple clé-valeur, la clé étant un entier et la valeur étant une chaîne de caractères. De plus, elle possède un attribut "taille\_sequence\_tab" qui représente la taille du tableau de séquences, facilitant ainsi son parcours. Enfin, la structure comprend un tableau de chaînes de caractères qui permet de donner une identité graphique au couloir.

Pour élaborer ces couloirs, nous avons divisé le processus de création en plusieurs étapes. Tout d'abord, nous avons la création de séquences de déplacement, puis le traçage d'un parcours en utilisant ces séquences de déplacement. Ensuite, nous entourons ce tracé avec des caractères "#" symbolisant des murs, et enfin, nous supprimons le tracé initial, ne laissant que les murs formant le couloir.

En ce qui concerne l'interaction avec l'utilisateur, nous avons tout d'abord demandé à ce dernier de fournir deux points pour son couloir, à savoir un point de départ et un point d'arrivée. Ensuite, nous avons calculé la distance à parcourir en termes de coordonnées x et y du point de départ vers le point d'arrivée. Nous avons ensuite présenté à l'utilisateur la distance à parcourir dans chaque axe et dans quelle direction il souhaite se rendre, ainsi que le nombre de "pas" qu'il souhaite effectuer dans cette direction. La distance en x et y se met à jour en fonction de ses choix.

Nous avons également ajouté la possibilité de supprimer un couloir pour permettre à l'utilisateur de modeler au maximum son donjon sans jamais le compromettre. Cependant, étant donné que la création de couloirs peut s'avérer complexe et source de nombreuses incohérences, la suppression peut, avec des couloirs complexes, être partiellement fonctionnelle. Par conséquent, nous avons introduit deux fonctions supplémentaires, "ajouter\_mur" et "supprimer\_mur", qui permettent d'ajouter et de supprimer un mur à des coordonnées spécifiées afin d'ajuster la suppression de certains couloirs ou d'autres objets au sein du donjon.

* Le menu

Pour le menu, une structure est prévue comme nous l’avons dit tout a l’heure. Seulement ici il y a deux « types » de menu. Le premier est le menu du « début » c’est-à-dire quand il n’y a pas encore de donjon créé. En effet nous avons voulu afficher le menu à coté du donjon pour que les informations soient explicitement présentes constamment dans le programme. C’est pour cela que nous avons besoin de la fonction afficher\_menu\_debut() pour afficher quand aucun donjon n’est encore créé. La fonction créer\_menu() quant à elle, sera utiliser pour l’affichage du menu à coté du donjon. On se servira de la fonction afficher\_donjon pour qu’une fois que l’affichage du donjon est fait, à droite du donjon il y ait le menu.

* Les murs

Les couloirs sont des structures complexes. Ils sont également difficiles à manipuler de par le caracteres très libre que nous avons voulu donner à notre application. La liberté peut conduire à faire des choses basées sur l’imagination et peut se rapprocher d’autant plus de ce que l’on veut faire, c’est donc un point positif. Il en découle qu’il faut bien connaître les mécanismes de construction du donjon pour arriver à ses fins. C’est comme si l’on utilisait des logiciels très particuliers qui sont, avant de produire quelque chose de propre et sérieux, assez compliqués à s’approprier. Et nous avons voulu offrir la possibilité que si un couloir était mal placé de supprimer ce couloir. Il arrive que cette suppression entraine la suppression de caracteres indésirables du fait de la complexité des structures et des fonctions. Cela nous a donné une idée, celle d’ajouter des murs ou de supprimer des murs à la guise de l’utilisateur, s’il veut compléter un mur mal supprimé ou s’il veut mettre comme des « pierres » au milieu du donjon s’il le veut. D’où les fonctions ajouter\_mur et supprimer\_mur.

* Les fonctions de création

Les fonctions de création sont des prototypes de fonctions qui généralisent et assemblent d’autres fonctions entre elles pour former quelque chose d’important.

En effet par exemple *creation\_salle* regroupe toutes les fonctions pour assembler une fois pour toute une salle. Il y a dans ces fonctions une interaction avec l’utilisateur pour qu’il puisse définir les paramètres qu’il veut pour pouvoir pleinement developper son donjon.

* La fonction finale

La fonction finale se base sur le même principe que les fonctions de création c’est-à-dire qu’elle regroupe les fonctions importantes pour créer cette fois non pas des edifices important mais l’entiereté du donjon. Elle repose sur un do {} while () ; qui ne s’arrete pas ava,t que l’utilisateur n’ai spécifié ‘q’ pour quitter le constructeur du donjon. Avant cela, il a toute une possibilité de choix à faire pour personnaliser son donjon. S’il veut créer un donjon il peut presser la touche ‘d’ par exemple, s’il veut créer une salle il peut apppuyer sur ‘s’ etc.

* Le Main

Le Main ne fait qu’un simple appel à notre fonction\_finale pour créer le donjon.

* La sauvegarde

Nous avons décider d’offrir la possibilité à notre utilisateur desuavegarder les informations d’une salle qu’il vient de créer dans un fichier externe. En effet dans la création de salle, après lui avoir demandé ce qu’il voulait faire de sa salle (s’il voulait y insérer des entitees par exemple), on lui demande également s’il veut sauvegarder sa salle dans un fichier externe, du nom de son choix. Ainsi quand il sauvegardera les données un fichier se créera sous le nom qu’il a décider de choisir lui-même. Ce qui lui permettra de mettre des noms personnalisés pour éventuellement se souvenir des salles qu’il a sauvegardé en fonction du nom qu’il lui a assigné.

Il y a également un fichier de sauvegarde de donjon qui enregistre le donjon en cours de création.

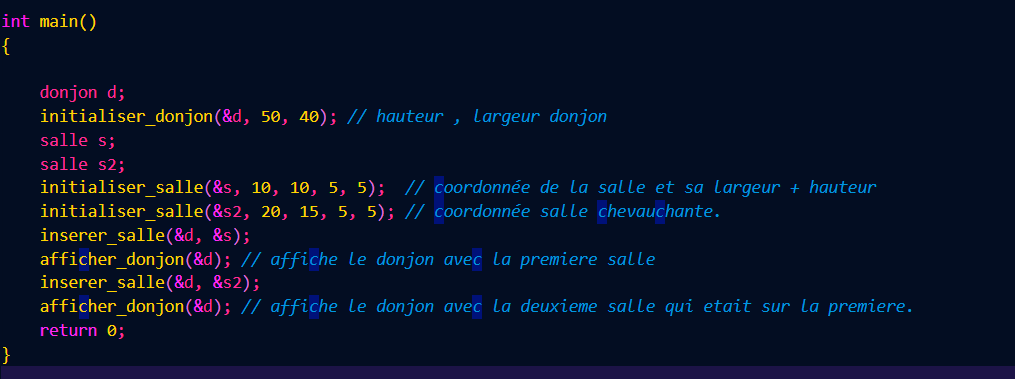
## Ajout personnel

Nous avons accordé une grande attention à l'interface utilisateur, cherchant à garantir que l'utilisateur ne se sente jamais perdu. Dans un souci esthétique, nous avons intégré un titre en ASCII, "DONJON CRAWLER", pour ajouter une touche rétro rappelant les jeux des années 80. Nous avons également créé un menu de démarrage comprenant les options "Start" et "Manuel". "Start" lance le concepteur, tandis que "Manuel" permet à l'utilisateur de consulter un petit manuel, offrant un aperçu du logiciel.

Une fois dans le concepteur, après avoir sélectionné "Start", un menu détaillé avec de nombreuses options légendées est présenté pour faciliter la compréhension. Lors de la création d'un donjon, des coordonnées verticales et horizontales sont affichées. Pour des raisons de clarté, les coordonnées horizontales sont écrites sur une ligne, car chaque "#" représente un caractère, nécessitant l'écriture de chaque nombre à deux chiffres l'un au-dessus de l'autre. Cependant, une fois cette considération prise en compte, le processus de création devient plus simple.

## Problèmes

Il semble y avoir un problème lors de la suppression d'une salle chevauchante, même si cela fonctionnait correctement la veille. Cela peut être frustrant, mais examinons la routine que vous suivez pour l'insertion de salle et la suppression manuelle des salles chevauchantes. Peut-être que cela nous permettra de cerner le problème.

N'hésitez pas à partager la routine ou le code spécifique que vous utilisez pour l'insertion et la suppression des salles, et je ferai de mon mieux pour vous aider à résoudre ce problème.

# Conclusion

Pour conclure le projet nous a beaucoup fait apprendre. Dans le langage C forcément. Il nous a confronté à la logique générale du langage. La définition de structures, de types, les différentes fonctions que l’on a dû faire. Mais également de gérer les interactions avec l’utilisateur ou encore l’utilisation de fichiers externes, par exemple les fichiers de sauvegarde ici.

Il nous a également permis d’apprendre à appréhender un projet en commun. Il faut savoir s’accorder, se trouver du temps en équipe pour discuter des choix, être disponible et travailleur pour le groupe.