Le Mans Université

Institut d'Informatique Claude Chappe Le Mans Université

# Le Mans Université

Licence Informatique 2ème année

Module 174UP02 Rapport de Projet

# LittleRogueNight

Maelig Pesantez, Clément Lelandais, Enzo Desfaudais

24 avril 2024

https ://github.com/Pixis-py/LittleRogueNight

# Table des matières

# 1 Introduction

Ce rapport est une analyse détaillée du projet LittleRogueNight, un **roguelike**, jeu vidéo s'inspirant du célèbre jeu Little Nightmares. Ce projet est réalisé dans le cadre du module projet, en seconde année de licence informatique au Mans. Réalisé par Clément Lelandais, Enzo Desfaudais et Maelig Pesantez, le jeu est codé du 19/01/2024 au 24/04/2024, soit un peu plus de 3 mois. LittleRogueNight est codé en majorité en langage C, car imposé pour ce projet et étudié depuis la première année de licence. C'est le cas du shell linux, du markdown, du LaTeX et d'autres langages présent en minorité, dont nous verrons plus tard l'utilité. LittleRogueNight propose une aventure où le joueur incarne un personnage emblématique naviguant à travers des labyrinthes générés aléatoirement peuplés de monstres. Ce jeu offre une expérience unique à chaque partie car les chemins du labyrinthe menant à l'arrivée changent, de même que l'emplacement des monstres et autres entités, offrant ainsi une rejouabilité infinie.

LittleRogueNight s'inspire des rogue-like traditionnels, il est donc basé sur les fondamentaux d'un rogue-like classique : labyrinthe parfait combiné à une augmentation progressive de la difficulté, ainsi que les concepts de *différents niveaux avec renouvellement* et de *perte permanente*. Ainsi, le jeu se compose des 3 niveaux où chaque labyrinthe est différent à chaque nouvelle partie, et où la difficulté augmente au fur et à mesure, et si le personnage meurt à n'importe quel niveau, il recommence au niveau 1.

Les mécanismes de déplacement offrent une variété d'actions essentielles telles que : les *dash*(sauts continus) et les *drifts*(glissades continues) qui ajoutent une dimension tactique aux affrontements. Les graphismes, réalisés en pixel art, apportent une touche rétro à l'esthétique du jeu, et renforçent son ambiance immersive.

Les combats sont au cœur de l'expérience de jeu, avec un système de barre de vie symbolisée par quatre cœurs qui représentent chacun 25 pourcents de la vie du joueur. Les attaques, qu'elles soient directes (via le drift) ou facilitées par des *items* spéciaux (via le briquet), permettent de combattre les monstres. Chaque *boss* et chaque monstre possèdent leur propre barre de vie et attaque.

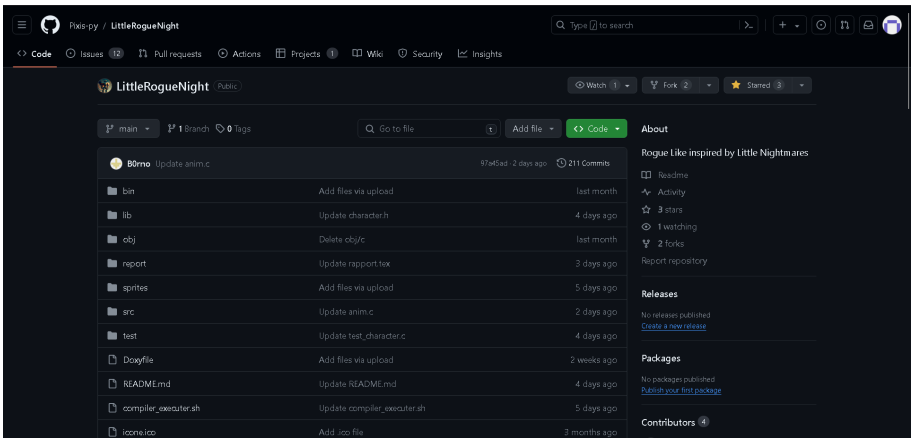
Tout cela dans le but de faire de LittleRogueNight un projet combinant éléments classiques du genre rogue-like avec des inspirations qui enrichissent l'expérience de chaque *gameplay*.

# 2 Organisation

## 2.1 Outils utilisés

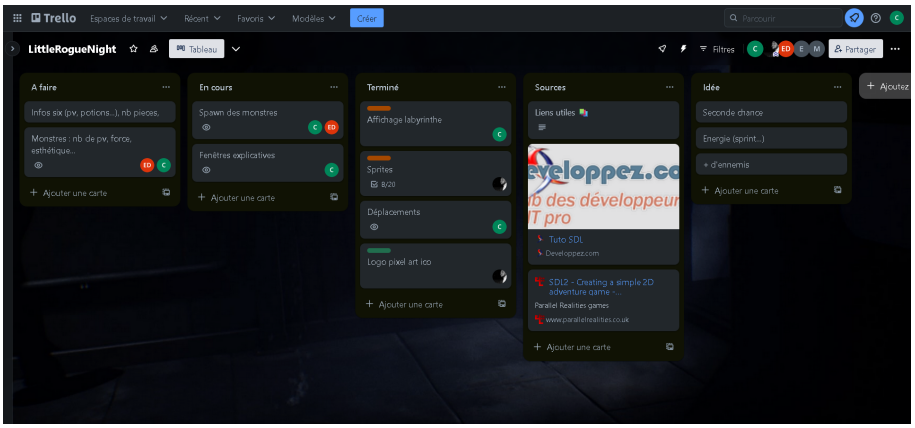
Pour organiser notre travail et permettre une collaboration efficace entre les membres du projet, nous avons mis en place des outils. Nous utilisons GitHub, un Trello et un Drive Google. Cette approche nous a permis de suivre l'évolution du projet, de partager les ressources nécessaires et de coordonner nos efforts de manière efficace.

Un GitHub a été créé pour héberger notre code source et faciliter la gestion des versions. Dès qu'un fichier atteint un niveau de maturité suffisant et qu'il est fonctionnel, il est alors *push* sur le dépôt distant. Cette façon de gestion de versions nous a permis de maintenir un historique complet des modifications apportées au code, ainsi qu'une structure de répertoires cohérente. En effet, notre organisation des fichiers sur GitHub reflète celle de nos sessions de travail, avec plusieurs dossiers contenant différents types de fichiers tels que les librairies, les fichiers sources, les exécutables, les objets, les *sprites*, etc. Cette cohérence dans la structure facilite la compréhension et la navigation au sein du projet pour l'ensemble de l'équipe.




Répartition des dossiers comme sur Visual Studio Code mais sous Github

Parallèlement, un Trello est utilisé pour représenter le cahier des charges et suivre l'avancement des fonctionnalités. Des checklists ont été créées pour chaque tâche, permettant à chaque membre de l'équipe de suivre sa progression et de s'assurer que les objectifs soient atteints en temps voulu.



Répartition des tâches entre les membres de l'équipe

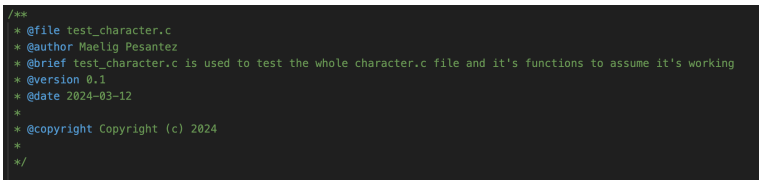
En outre, au début du projet, nous avons également utilisé un Drive Google pour partager des documents et des ressources nécessaires à la planification et au démarrage du projet. Ce Drive a notamment été utile pour stocker des documents de référence, des captures d'écran de l'avancée du projet pour les premiers affichages du labyrinthe et d'autres fichiers partagés entre les membres de l'équipe, offrant ainsi un espace de stockage centralisé et accessible à tous.

L'utilisation combinée de ces outils a grandement contribué à la gestion efficace du projet, en facilitant la collaboration, le suivi de l'avancement et la gestion des ressources, tout en favorisant une communication efficace entre les membres de l'équipe.

Nous avons également utilisé une gamme d'outils spécifiquement sélectionnés pour faciliter le processus de création et garantir la qualité du produit final.

**Environnement de développement intégré (IDE) : Visual Studio Code** : L'IDE joue un rôle crucial dans notre processus de développement. Nous avons choisi Visual Studio Code pour sa polyvalence et sa facilité d'utilisation. L'interface intuitive de Visual Studio Code nous a permis d'écrire, de modifier et de déboguer notre code de manière efficace. De plus, l'intégration d'extensions a considérablement amélioré notre flux de travail. Parmi ces extensions, l'outil *doxygen* revêt une importance particulière.

**Extension Doxygen pour la documentation** : L'extension Doxygen pour Visual Studio Code a été un atout majeur dans la production de la documentation de notre projet. Cette extension simplifie le processus de documentation en permettant la génération de commentaires Doxygen quasiment automatiquement. Grâce à des balises prédéfinies telles que "/// @brief" ou "/// @param", nous avons pu documenter efficacement notre code, facilitant ainsi sa compréhension et sa maintenance ultérieure.




*Exemple d'un commentaire doxygen généré automatiquement*

**Tests unitaires avec CUnit** : Pour garantir la fiabilité et la robustesse de notre code, nous avons intégré des tests unitaires à notre processus de développement. Pour cela, nous avons utilisé la bibliothèque *CUnit*, largement reconnue dans l'écosystème du langage C pour sa capacité à effectuer des tests unitaires efficaces. Ces tests ont été conçus pour évaluer le bon fonctionnement de nos fichiers sources contenant des structures cruciales, assurant ainsi la qualité du code avant son intégration.

```c
void test_create() {
    character_t * character;
    create(&character, 100, 25);
    CU_ASSERT_PTR_NOT_NULL(character);
    CU_ASSERT_EQUAL(character->pv, 100);
    CU_ASSERT_EQUAL(character->damage, 25);
    CU_ASSERT_EQUAL(character->x, 0);
    CU_ASSERT_EQUAL(character->y, 0);
    destruct(character);
}
```

*Exemple d'une fonction de test : creation d'un character*

```c
int main() {
   CU_initialize_registry();

   CU_pSuite suite = CU_add_suite("Character Tests", NULL, NULL);
   CU_add_test(suite, "Test Creation", test_create);
   CU_add_test(suite, "Test Pv Loss", test_pv_loss);
   CU_add_test(suite, "Test Attack", test_attack);
   CU_add_test(suite, "Test Pv Gain", test_pv_gain);

   CU_basic_run_tests();
   CU_cleanup_registry();

   return 0;
}
```

*Fonction main d'un test unitaire : appel des fonctions de test de la structure character*

```
     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/


Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      2      2      2      0        0
             asserts      6      6      6      0      n/a

Elapsed time =    0.000 seconds
```

*Sortie du test unitaire de la structure $character_t$ et de ses fonctions*

**Production de sprites avec Piskel** : Bien que cela ne soit pas directement lié au développement du jeu en lui-même, il est important de mentionner l'outil Piskel que nous avons utilisé pour produire les sprites de LittleRogueNight. Piskel est un outil de pixel art en ligne gratuit qui nous a permis de créer des sprites uniques et personnalisés pour notre jeu, contribuant ainsi à son esthétique et à son identité visuelle.

*Exemple de spritesheets avec l'animation de marche de Six*

## 2.2 Rôles

Concernant les tâches, Maëlig est en charge de la création des *spritesheets*, qui représentent toutes les images nécessaires à l'animation des entités du jeu. Pour ce faire, il utilise un logiciel en ligne puis intègre ces animations dans le code. Outre cette responsabilité, Maëlig prend en charge la gestion des combats entre le héros Six et les concierges (monstres provenant du jeu originel) présents dans le labyrinthe. Il est également responsable du spawn (apparition) des entités au sein du labyrinthe.

Ensuite, Enzo se concentre sur l'affichage du menu principal ainsi que des menus annexes, en mettant en place les boutons et en assurant la gestion de la souris pour l'interaction utilisateur. Il contribue également à l'apparition aléatoire des entités dans le labyrinthe, ajoutant ainsi une dimension aléatoire au jeu qui enrichit l'expérience du joueur.

Enfin, Clément prend en charge la création et l'affichage du labyrinthe, une tâche cruciale pour le fonctionnement du jeu. Il développe aussi les mécanismes de déplacements et de collisions des entités, assurant ainsi le réalisme et la cohérence du monde virtuel. Clément apporte également son aide au codage de la partie combat avec Maëlig et Enzo pour garantir son bon fonctionnement.

# 3 Conception

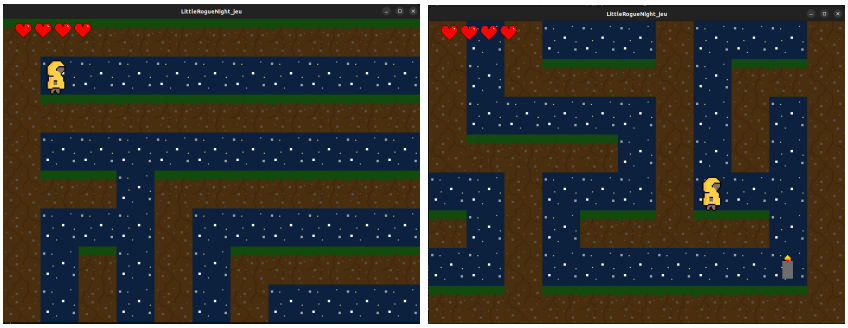
## 3.1 Analyse et cahier des charges

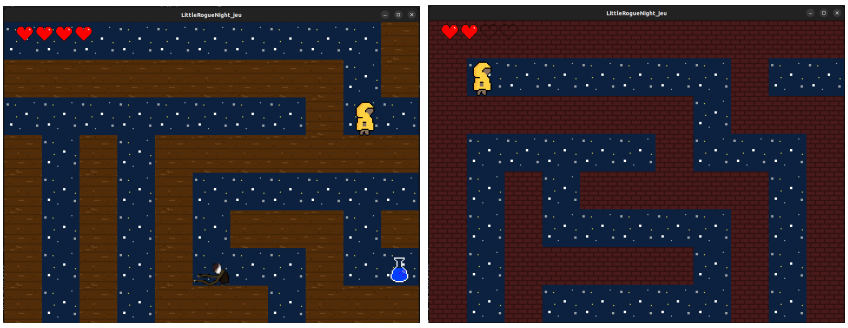
Les règles du jeu s'articulent autour du personnage principal, Six, qui doit naviguer à travers un labyrinthe complexe en quête de la sortie, située en bas à droite de la structure. Au cours de cette progression, Six est confronté à une multitude d'ennemis qu'il doit soit éviter, soit affronter.

Pour se défendre, Six dispose d'attaques. Au 1er niveau, il a recours aux drifts. Aux niveaux 2 et 3, il obtient un briquet, une arme plus efficace pour tuer ses adversaires.

La survie du joueur dépend de sa capacité à gérer ses vies limitées. Il peut régénérer ces dernières en collectant des morceaux de viande et des potions dispersés dans le labyrinthe. Cette gestion prudente des ressources est cruciale pour affronter les dangers croissants qui ponctuent sa progression. Ainsi, si la barre de vie du personnage est trop basse, celui-ci peut retourner sur ses pas pour utiliser des potions ou morceaux de viande repérés.

La difficulté s'intensifie au fil des niveaux, notamment avec les concierges, les monstres des niveaux. Leur puissance augmentent progressivement, accentuant le défi pour le joueur. Parallèlement, l'architecture du labyrinthe évolue, passant d'une structure en terre, puis en bois et pour finir des murs de briques aux niveaux supérieurs. Cette transition visuelle reflète la complexité croissante du défi et contribue à l'immersion du joueur dans l'univers du jeu.




LittleRogueNight_jeu
LittleRogueNight_jeu

# 4 Développement

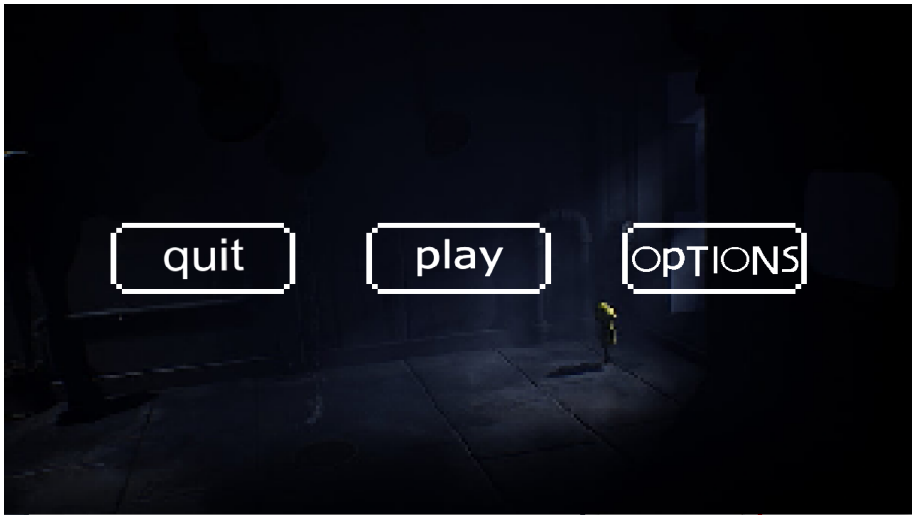
## 4.1 Fonctionnemeent des algorithmes

Pour la création du menu, Enzo utilise des boutons pour faciliter le choix à l'utilisateur, pour créer ces boutons il a défini des structures boutons_s contenant leur positions à travers des rectangles (SDl_Rect), leur texture classique et une texture grisée permettant de le différencier quand le curseur du joueur passe sur la position du bouton, et un entier ETAT valant 0 quand le bouton est relaché et 1 quand le bouton est enfoncé.

```c
typedef struct bouton_s
{
    SDL_Rect *position; // contient la position (.x et .y) et les dimensions (.w et .h) du bouton
    int etat; // contient l'état du bouton (BOUTON_RELACHE / BOUTON_APPUYE // BOUTON_DESSUS )
    SDL_Texture *dessus; // image du bouton lorsqu'il est relaché
    SDL_Texture *classique; // image du bouton lorsqu'il est appuyé
}bouton;
```

Structure du bouton

Le menu permet donc au joueur de choisir entre le bouton "play", "options", et "quit". Quand le joueur appuie sur le bouton "play", la fonction menu.c retourne 1 ce qui permet de lancer la creation du personnage (Six de base) et le lancement du jeu. Quand le joueur appuie sur le bouton "options" cela lui ouvre le menu du choix des personnages et en fonction de son choix cela modifie un pointeur dans la fonction main(). Enfin le joueur a aussi la possibilité de quitter le jeu via le bouton "quit" qui retourne 2 dans le main, ce qui enclenche la fermeture de l'instance SDL et la libération de la mémoire.




Affichage du menu au lancement du jeu

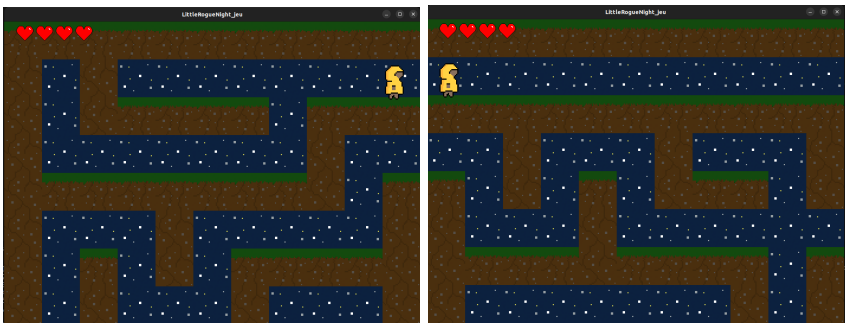
Pour la création du labyrinthe parfait parfait, Clément utilise la génération aléatoire, avec des piles et des files, rencontrée au semestre 3 en Algorithmique

et Programmation 2. Bien sûr, les fonctions ont été légèrement modifiées pour s'adapter à nos fins : la gestion des différentes tuiles grâce à des constantes comme NUIT, TERRE, TERREVERTE etc, les paramètres de la fonction d'affichage changent pour permettre d'afficher le labyrinthe directement sur le rendu grâce à des rectangles (SDL_Rect) qui se "posent" sur le rendu.

```
9   #define VRAI 1
10  #define FAUX 0
11  // Constantes representant le contenu des cases
12  #define NUIT 0
13  #define MUR 1
14  #define TERRE 2
15  #define TERREVERTE 3
16  #define DUR TERRE || TERREVERTE
17  //labyrinthe
18  #define FORMATPIXEL 32
19  #define ZOOM 3
20  #define FORMATPIXELZOOM FORMATPIXEL * ZOOM
21  #define LARGEUR 7
22  #define LONGUEUR 10
23  #define N LARGEUR * 5 + 7
24  #define M LONGUEUR * 5 + 9
```

Constantes du labyrinthe

La fonction permet également de rendre le labyrinthe plus esthétique en mettant un bloc de terre avec de l'herbe au dessus des autres blocs de terre dans le niveau 1 et le labyrinthe est réparti en plusieurs parties qui sont décalées en fonction des variables coeffX et coeffY : si le joueur arrive à une certaine distance près d'un bord de l'écran, alors le coeff concerné est incrémenté ou décrémenté selon les circonstances et donc tout est décalé dans l'affichage du labyrinthe.



Défilement du labyrinthe et décalage de Six

Le niveau 2 est constitué de tuiles de bois et le niveau 3 affiche des briques. Maintenant, la partie animation du jeu aborde plusieurs fonctionnalités :

— Les monstres deviennent de plus en plus puissants quand Six avance en terme de niveau de labyrinthe.

— Pour la détection des touches, il faut simplement déterminer si le joueur appuie sur une touche nécessaire au jeu, ce qui met à 1 la variable concernée et idem si la touche est relâchée. Cependant, une distinction se fait quant aux directions : si la dernière touche appuyée puis relâchée est 'd' par exemple, alors Six regarde vers la droite et idem pour la gauche.

— Chaque touche est reliée à un affichage différent du sprite et des combinaisons peuvent être utilisées, ainsi si Six avance à droite grâce à la touche 'd' et qu'il saute en même temps par la touche "espace", alors cela fait appel à 2 animations. Pour coder cela, il faut créer une boucle for avec autant de répétitions qu'il y a d'images dans le spritesheet d'animation, par exemple, pour courir, 11 images ont été créées par Maëlig et chacune d'entre elles possède une position en x différente. Pour appliquer un zoom sur chaque image, la constante FORMATPIXELZOOM est utilisée car elle représente le format 32 pixels des images multiplié par le zoom, ici 3, pour obtenir des images de 96 pixels.

— Concernant les positions des entités présentes dans le labyrinthe, chacune d'entre elles possède des coordonnées et un nouveau SDL_Rect associé au nom (souvent utilisé "position_nom_sprite") permettant de manipuler leurs coordonnées en contrôlant qu'elles respectent certaines conditions : elles ne doivent pas dépasser la fenêtre et apparaître quand il le faut grâce aux décalages par les coeffX et coeffY. Cela se fait par un calcul de la position dans le labyrinthe par rapport aux pixels : on divise cette valeur par FORMATPIXELZOOM et encore par 9, qui correspond à $\text{ZOOM}^2$. Cependant, il y a un petit décalage selon la direction de Six : s'il regarde à droite alors il faut ajouter 8 à la position initiale et 70 s'il regarde à gauche. Pour l'opétation inverse, pour effectuer l'apparition

aléatoire des entités dans le labyrinthe, on multiplie la valeur par FORMATPIXELZOOM et on effectue le décalage en soustrayant par le coeff correspond multiplié par FORMATPIXELZOOM.

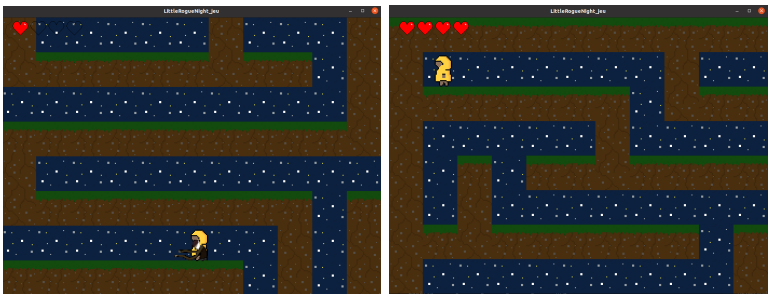
— Pour appliquer la gravité, il faut simplement vérifier que la case sur laquelle se trouve Six est de type NUIT auquel cas il tombe jusqu'à ce que ce ne soit plus le cas. C'est presque le même système pour les sauts sauf que le y manipulé sur la position de l'entité est décrémenté à la place d'être incrémenté pour la gravité.

— A la fin d'un niveau, Six est téléporté au début du niveau suivant s'il y en a un autre. Sinon une fenêtre apparait lui demandant s'il veut retourner au menu, arrêter de jouer ou encore poursuivre le jeu en débutant du niveau souhaité. Il obtient alors, bien sûr, tous ses points de vie en début de partie.

— Maintenant, s'il finit par mourir au cours d'un niveau, il revient au début du niveau 1 avec tous ses points de vie et perdant ainsi toute son avancée dans le jeu ! Toutes les entités sont de nouveau placés aléatoirement.

— Pour tuer un monstre ou consommer une potion de points de vie ou de la viande, 3 variables ont été créées : si l'une d'entre elle vaut 1 alors l'entité concernée disparaît de l'écran.

— Enfin, une détection de Six par les entités est nécessaire au bon fonctionnement du jeu : s'il se trouve à moins de 2 blocs du monstre et qu'il n'attaque pas, donc il ne drift pas, alors il reçoit des dégâts toutes les 0.75 secondes (1 seconde est trop long), s'il attaque par contre et qu'il est toujours à la même distance alors c'est le monstre qui reçoit des dégâts. Idem pour la potion, seulement, s'il a tous ses points de vie, alors la potion reste dans le labyrinthe.



## 4.2 Structures

En langage C, diverses structures sont utilisées dans les fichiers du jeu, notamment les files et les piles. Ces structures jouent un rôle important en matière de stockage des données, offrant des solutions efficaces pour gérer la manipulation des informations. Les piles et les files, couramment employées en programmation, présentent des caractéristiques distinctes avec des avantages et des inconvénients spécifiques. Tant les piles que les files possèdent une tête (représentant le premier élément de la liste) ainsi qu'une queue (le dernier élément de la liste).

L'utilisation de piles et de files en langage C facilite la gestion efficace des données, notamment pour la manipulation des matrices. Dans le contexte du jeu, cela simplifie la création et la manipulation du labyrinthe, permettant un fonctionnement fluide et optimisé des mécanismes de jeu.

La structure *character* prend en charge les différentes entités présentes dans le jeu telles que le personnage contrôlé par le joueur, les monstres, ainsi que les objets comme les potions. Cette structure permet d'assigner des points de vie, des dégats et des coordonées à chaque entité.

```
typedef struct character_s{
    int pv, damage, x, y;
}character_t;
```

*Structure character*

## 4.3 Fichiers

Les fichiers ont été organisés de la même manière que sur le Github, c'est à dire en gèrant chaque aspect du jeu de manière efficace, en les rangeant dans différents dossiers en fonction de leur type et leur fonctionnalité. Cette structuration nous a permis de maintenir un code propre et bien organisé tout au long du développement.

Dans le dossier *src*, nous avons regroupé tous les fichiers contenant du code source écrit en langage C. Cette section constitue le cœur du projet, abritant les implémentations des fonctionnalités principales du jeu, telles que la logique de jeu, la gestion des personnages et des niveaux.

Le dossier *bin* contient l'exécutable du jeu ainsi que les fichiers de test associés. Ces fichiers de test sont essentiels pour vérifier la robustesse et la fiabilité

de notre code, en s'assurant que toutes les fonctionnalités répondent aux exigences spécifiées.

Le répertoire *lib* héberge les fichiers d'en-tête contenant les prototypes de fonctions, les définitions de structures et les constantes utilisées dans notre code source. Cette section est utile dans l'organisation et la documentation de notre projet, en fournissant une référence claire pour toutes les fonctions et structures implémentées et elle est primordiale pour l'édition de liens de la compilation du programme par le makefile.

Le dossier *obj* contient les fichiers objets générés lors de la compilation du code source. Ces fichiers objets sont essentiels pour créer l'exécutable final du jeu, en regroupant toutes les parties du code source en un seul programme exécutable.

Le répertoire *sprites* est divisé en plusieurs dossiers, chacun contenant les différents sprites nécessaires à l'affichage du jeu. Les *sprites*, qui représentent des images en deux dimensions pouvant être déplacées indépendamment du décor de l'affichage, sont essentiels pour créer une expérience visuelle pour les joueurs.

Enfin, le dossier *test* abrite tous les fichiers de test des fonctions, garantissant qu'elles fonctionnent correctement et ne produisent pas d'erreurs inattendues. Ces tests sont nécessaires dans la validation et la vérification de la qualité du code, en s'assurant que toutes les fonctionnalités du jeu répondent aux normes spécifiées.

# 5 Conclusion

Le projet LittleRogueNight a atteint ses objectifs fixés en termes de fonctionnalités de base. Le jeu est fluide, ce qui peut être difficile à mettre en place au vu de la gestion de la mémoire de notre moteur-graphique SDL et du nombre de sprites présents sur l'écran. La partie recherche du chemin vers la fin du labyrinthe fait son effet grâce au principe du labyrinthe parfait. La détection du héros par les ennemis est fonctionnelle : les ennemis sont capables d'attaquer le personnage et inversement, et sont placés aléatoirement dans le labyrinthe. Les collisions et la gravité ont également été ajoutées comme prévu et fonctionnent bien. Les menus ont été implémentés conformément à notre cahier des charges, offrant une navigation intuitive en dehors du jeu. Cependant, des fonctionnalités supplémentaires, telles que des spécificités pour les ennemis et les niveaux comme des pièges, le fait de détruire certains blocs, d'autres ennemis ayant des attaques différentes, n'ont pas pu être implémentées en raison de contraintes de temps et par l'abandon d'un des membres de l'équipe.

Les éléments prévus mais non réalisés, tels que des passages secrets, des salles spéciales et un magasin et donc un système d'argent, auraient pu ajouter de la variété et de l'intérêt au jeu. Cependant, ils auraient demandé de trop grosses modifications de l'algorithme de génération du labyrinthe et nous avons décidé de nous concentrer sur des fonctionnalités de base pour obtenir un jeu fonctionnel. D'autre part, l'ajout de niveaux supplémentaires aurait renforcé la durée de vie et la richesse du gameplay. Enfin, offrir au joueur des "pouvoirs" spéciaux tels que l'invisibilité ou la téléportation, avec un système de cooldown, aurait permis d'ajouter une dimension stratégique au jeu. L'idée d'un système de récompenses à la fin de chaque niveau aurait pu apporter un côté intéressant au jeu et le joueur serait plus enjoué à vouloir continuer à progresser.

Le projet LittleRogueNight, malgré quelques lacunes, se distingue comme une réalisation robuste et fonctionnelle, offrant une expérience de jeu gratifiante. Nous gardons une expérience positive de ce projet qui a représenté l'un des premiers efforts collaboratifs autour d'un projet de développement pour nous tous. Contrairement à nos habitudes de gestion de projet que nous avons pu experimenter au cours des tp individuels par exemple, la mise en place de ce projet a exigé de nouvelles manières de travailler et des outils tels que GitHub et des solutions de gestion de projet. En outre, celui-ci a permis de mettre en œuvre les compétences informatiques acquises au cours des 2 premières années d'études universitaires en informatique, favorisant ainsi le développement de compétences en programmation et en conception de jeux vidéos au sein de l'équipe.

# Glossaire

**augmentation progressive de la difficulté** Principe de conception de jeu dans lequel la complexité et le niveau de défi augmentent graduellement au fur et à mesure que le joueur progresse.. 3

**boss** Dans le domaine du jeu vidéo,un ennemi plus puissant et complexe que les autres, souvent rencontré à la fin d'un niveau ou d'une étape importante du jeu.. 3

**briquet** Objet utilisé comme arme par le personnage principal dans le jeu LittleRogueNight, particulièrement efficace aux niveaux 2 et 3.. 8

**character** Dans le jeu LittleRogueNight, désigne les différentes entités présentes, telles que le personnage principal, les monstres et les objets.. 14

**concierges** Monstres redoutables présents dans le jeu LittleRogueNight, notamment aux niveaux 2 et 3, avec une résistance et une puissance accrues.. 8

**constantes** Valeurs immuables utilisées dans le code source d'un programme pour représenter des éléments tels que des dimensions, des limites ou des paramètres spécifiques.. 11

**cooldown** Temps que doit patienter un joueur avant de pouvoir être capable de refaire une certaine opération.. 16

**CUnit** Bibliothèque de tests unitaires en langage C, utilisée pour évaluer la fiabilité et la robustesse du code source en effectuant des tests automatisés.. 6

**dash** Action permettant au personnage de se déplacer rapidement sur de courtes distances dans le jeu, souvent utilisée pour esquiver les attaques ou franchir des obstacles.. 3

**différents niveaux avec renouvellement** Structure de jeu où les niveaux se succèdent, mais où le contenu ou les défis sont renouvelés pour maintenir l'intérêt du joueur.. 3

**doxygen** Outils permettant de créer de la documention à partir des commentaires présent dans le code sources.. 5

**drifts** Action permettant au personnage de glisser sur de courtes distances dans le jeu, souvent utilisée pour se déplacer rapidement ou éviter les obstacles.. 3, 8

**ennemis** Adversaires ou obstacles présents dans un jeu vidéo et qui entravent la progression du joueur.. 8

**files** Structure de données dans laquelle les éléments sont ajoutés à l'arrière et retirés à l'avant, suivant le principe du premier entré, premier sorti (FIFO).. 10

**gameplay** Ensemble des caractéristiques et des mécanismes d'un jeu vidéo, notamment l'intrigue, les interactions et les mécaniques de jeu, qui définissent l'expérience globale du joueur.. 3, 16

**GitHub** Plateforme de développement collaboratif de logiciels, permettant aux développeurs de travailler ensemble sur des projets, de suivre les versions du code source, de gérer les problèmes et de coordonner les efforts de développement.. 4

**items** Objets que le joueur peut collecter ou utiliser dans le jeu pour obtenir des avantages ou progresser dans l'aventure.. 3

**labyrinthe parfait** Type de labyrinthe généré de manière algorithmique, dans lequel il existe un unique chemin reliant chaque point à un autre.. 3, 10

**moteur-graphique** Logiciel qui regroupe et gère en temps réel les fonctionnalités principales d'un jeu vidéo liées au graphisme.. 16

**perte permanente** Concept de jeu où les conséquences des actions du joueur sont irréversibles, ce qui signifie que les erreurs commises ne peuvent pas être annulées ou corrigées.. 3

**piles** Structure de données dans laquelle les éléments sont ajoutés et retirés selon le principe du dernier entré, premier sorti (LIFO).. 10

**push** Opération qui consiste à envoyer des données d'un répertoire local vers un répertoire distant dans un système de gestion de version comme Git.. 4

**rendu** Processus dans lequel les éléments graphiques sont affichés à l'écran, permettant aux joueurs de visualiser le jeu et d'interagir avec lui.. 11

**rogue-like** Genre de jeu vidéo qui s'inspire du jeu original "Rogue", caractérisé par des environnements générés de façon procédurale, une difficulté élevée et la mort permanente du personnage du joueur.. 3

**spawn** Verbe qui signifie apparaître, ou un nom qui signifie apparition.. 7

**sprites** Dans le contexte des jeux vidéo, images en deux dimensions représentant des éléments tels que les personnages, les objets ou les décors, utilisées pour l'affichage à l'écran.. 4, 15, 16

**spritesheets** Feuilles d'images regroupant plusieurs sprites utilisés pour animer les entités du jeu.. 7

**structures** Dans le domaine de la programmation, ensemble organisé de données de types différents, regroupées sous un même nom pour une manipulation plus aisée.. 14

**Trello** Outil de gestion de projet en ligne basé sur le concept de tableaux Kanban, permettant aux équipes de collaborer et d'organiser leurs tâches en listes et en cartes.. 4

# 6 Annexes

## 6.1 Tests

Les test unitaires sont trouvables sur github avec leur fichier texte explicatifs, dans le dossier test ainsi que dans le dossier doc. Ceux-ci permmetent les test des structures character, file et pile.

## 6.2 Diagramme de Gantt

Le lien du diagramme de Gantt est trouvable dans le fichier README de notre github.

## 6.3 Documentation

La documentation en html et en LaTeX est disponible sur github dans le dossier doc. Plus d'informations sur la manière de créer la documentation se trouvent dans le fichier README de notre github.

## 6.4 Débogage

Un exemple de débogage est disponible dans le dossier doc de notre github. Celui-ci concerne un test de variables.