LES TROPHÉES NSI

Édition 2022

Simon Oudet,
Valentin Plaveret
Rémi Veillerot
Emmanuel Renevier

Music Dungeon

> PRÉSENTATION GÉNÉRALE :

• L'idée principale du projet était d'allier musique et programmation par l'intermédiaire d'un jeu vidéo. De ce fait, on a imaginé un système de combat permettant au joueur de compléter une musique. Nous voulions que le joueur recompose une musique par piste d'instrument, donc la séparation en plusieurs salles d'un même donjon nous a paru la plus adaptée.
Nous avons choisi une représentation à la troisième personne, dû à la simplicité de codage, nous permettant de nous concentrer sur la mécanique du jeu.
Pour diversifier le gameplay, les couloirs sont générés aléatoirement, ainsi que l'orientation des salles.

• L'ambiance du donjon nous est venue de l'influence du jeu vidéo Pixel Dungeon.

• Préférant la fonctionnalité à la quantité, nous nous sommes concentrés sur un seul étage, en facilitant cependant l'addition de nouveaux étages.

• On voulait partager une manière de « créer » une musique pour un non musicien tout en restant ludique.

• Le choix d'un jeu vidéo nous a permis d'utiliser nos connaissances acquises au cours de l'année de terminale en NSI telles que la programmation orienté objet, la modularité, les types abstraits de données et enfin la récursivité.

> ORGANISATION DU TRAVAIL :




• Notre équipe est composée de quatre membres dont chacun avait une tâche bien précise :

- *Simon Oudet, qui avait la charge de chef de projet. Son rôle consistait à assembler le travail des autres membres ainsi que de faire fonctionner le jeu.*
- *Valentin Plaveret, qui avait la charge de musicien. Son rôle consistait à la création de musique ainsi qu'à la préparation des combats avec les boss.*
- *Rémi Veillerot, qui avait la charge de graphiste. Son rôle consistait à la réalisation des images incluses dans le jeu.*
- *Emmanuel Renevier, qui avait la charge de programmeur. Son rôle consistait à la réalisation des codes que le chef de projet lui demandait.*

• Nous avions réparti le travail entre chaque membre du groupe de sorte que chacun puisse travailler à son rythme. Ainsi, nous travaillions dès que l'occasion se présentait : au lycée pendant les heures de pause ou à la maison. De ce fait, il fut important pour nous de constamment nous tenir informés de nos avancées. Pour se faire, nous discutions souvent au CDI de notre lycée et, lorsque nous n'étions pas ensemble, nous avions choisi ***WhatsApp*** *comme application de communication, ainsi que* ***GitHub*** *(https://github.com/SimonOudet/trophees-nsi-2022.git) comme lieu d'échanges pour les fichiers et le code.*

> LES ÉTAPES DU PROJET :

1. *La première étape fut la recherche d'un projet. Nous avons commencé par faire une recherche personnelle d'idées de projet, que nous avons ensuite mises en commun pour choisir celle que l'on affectionnait le plus : allier les jeux vidéo à la musique.*
2. *Une fois l'idée trouvée, nous avons cherché un concept, c'est à partir de là que le concept d'un donjon où les combats sont régis par la musique nous vînt.*
3. *Dès lors, nous nous sommes mis à la réalisation et ainsi nous avons commencé par répartir les tâches en fonction des compétences de chacun :*
   → Simon se mit à gérer l'élaboration du squelette du programme et à l'approfondissement de l'idée.
   → Valentin prît en charge la création d'un morceau de musique ainsi que la gestion des séquences que chaque boss doit réaliser pour empêcher le joueur de l'atteindre.
   → Rémi s'attaqua au design du terrain ainsi que celui du joueur et des monstres.
   → Emmanuel se chargea de la génération aléatoire du terrain ainsi que du comportement et des combats avec les vagabonds qui parcourent le labyrinthe.
4. *Si l'un de nous avait des problèmes de code, nous n'hésitions pas à en discuter entre nous.*
5. *Lorsque l'un de nous avait fini une de ses tâches, nous la partagions sur **GitHub** afin que le chef de projet puisse assembler l'ensemble du code, nous permettant de réaliser les premiers tests concrets du jeu.*
6. *Viennent alors les phases de débogage et de correction d'erreurs, que chacun des membres de l'équipe réalisait lorsqu'il s'agissait de sa partie.*
7. *Enfin, lorsque l'un d'entre nous avait le temps, il se mettait à l'écriture du dossier pour **Les Trophées NSI.***

> FONCTIONNEMENT ET OPÉRATIONNALITÉ :

• Nous avons aujourd'hui réussi à générer un étage qui comporte les boss et leur mécanique de combats avec cependant, les deux boss les plus complexes, qui possèdent encore des problèmes dans leurs derniers mouvements. Lors de la génération aléatoire de l'étage, celui-ci génère des vagabonds qui vont avoir pour rôle de ralentir le joueur.

• Afin de ne pas passer notre temps au débogage, nous faisions des tests indépendants réalisé au sein des fonctions que chacun de nous réalisait. Lorsqu'une fonction était créée, il était important que nous prenions le temps de la documenter à l'aide de docstring et de commentaires que nous avons parsemé dans le code.

• Lors de l'implémentation de notre jeu, nous avons rencontré trois problèmes majeurs qui nous ont ralenti :

→ Comment générer semi-aléatoirement des chemins entre les salles afin de relier toutes les salles de boss, tout en créant un labyrinthe sans se retrouver coincé entre trois murs ?

La réponse se réalisa en deux étapes : la première fut de générer les salles de boss avec un périmètre de sécurité autour d'elles, afin de ne jamais rencontrer une case bloquée entre trois murs. La seconde fut non pas de créer un chemin allant d'une salle à une autre, mais plutôt de faire démarrer de chaque salle le chemin afin de relier celui-ci au milieu.

→ Comment forcer le joueur à faire une suite prédéfinie de mouvements avec un temps prédéfini pour pouvoir compléter une séquence de notes ?

Pour se faire, nous avons décidé de ne faire apparaître qu'une seule partie du parcours, laissant peu de choix au joueur. De plus, pour la gestion d'un temps prédéfini, nous avons joué avec le temps d'exécution des sorts du boss pour faire apparaître ou disparaître le terrain.

→ Enfin le dernier problème rencontré fut : comment gérer le changement de séquence de mouvements dû à un changement d'orientation de la salle ?

Pour pallier ce problème, nous devions au préalable récupérer l'orientation que le programme générait aléatoirement pour chaque salle, puis nous avons adapté la séquence en fonction.

> OUVERTURE :

• Pour améliorer le jeu nous souhaiterions amener une interface plus aboutie, avec par exemple une barre de vie, ajouter des animations ainsi que du butin aléatoire permettant une amélioration et une diversification des stratégies de jeu. Lorsque le premier étage sera complétement abouti, nous en ajouterons de nouveaux, avec des mécaniques de combat de boss différentes, et une diversité de monstres dans le labyrinthe.

• Notre volonté de vulgariser la composition musicale nous a mené à présenter ce projet qui, ludiquement et pédagogiquement, permet d'en comprendre les procédés. C'est ainsi que par l'intermédiaire de l'exploitation de la simplicité du jeu vidéo, ce projet se présente comme novateur, car le joueur se retrouve acteur et musicien de fortune, ce sont ses actions qui produisent les sons, qui produisent un morceau de musique.

• Nous sommes très fiers du résultat. Cependant, le manque de temps nous a empêché de rendre le jeu diversifié dû à son unique étage, qui possède seulement cinq boss avec un niveau de difficulté inégal. Le manque de butin rend le jeu assez répétitif, avec un parcours de couloir très morne.

DOCUMENTATION

• ***Manuel de l'aventurier :***

Le but de la partie est de compléter la musique en vainquant les cinq boss présents dans l'étage, généré aléatoirement à chaque essai. Les salles de boss sont fixes mais peuvent changer d'orientation. Les parties des instruments s'empilent à chaque combat de boss réussi, et la musique est complète lors du dernier combat.

Pour se déplacer :
→ Z, Q, S et D sont respectivement haut, gauche, bas et droite.
→ Les déplacements en diagonale sont possibles uniquement dans les salles de boss, et sont gérés avec A, E, W et X pour haut-gauche, haut-droit, bas-gauche et bas-droit.
→ La touche P permet d'attendre un tour, le joueur ne bouge pas mais l'environnement joue. Dans chaque cas il est possible de maintenir la touche enfoncée afin de fluidifier le jeu. Dans les combats avec les boss cela reviens à prévoir son prochain coup.

Pour vaincre un boss, il faut suivre l'enchaînement de déplacements forcés par l'apparition et la disparition de cases de sol. Il peut y avoir plusieurs possibilités, mais une seule est la bonne. Le joueur a un temps limité pour effectuer son action, et le boss a un moment de réflexion, représenté par une bulle, avant de changer le sol, ce qui permet à la note d'avoir la bonne durée. Le combat démarre au moment où le joueur marche sur la case la plus avancée par rapport à la porte de la salle.

Pour varier les parties, des vagabonds sont placés aléatoirement autour de la carte, et marchent dans des directions aléatoires jusqu'à ce qu'ils apercoivent le joueur, et dans ce cas vont dans sa direction. Pour attaquer un vagabond, il faut être à côté de lui et avancer dans sa direction. Les vagabonds sont tués lorsque leur vie tombe en dessous de 0.

• ***Voici les réponses pour chaque salle dans leur orientation initiale dans l'ordre de difficultée :***

→Trompette :
D, Z, D, Z
→Batterie :
S, D, S, D, S, D, S, D, S, D, S, D, S, D, S, D, S, D, S, D, S, D, S, D, S, D, S, D, S, D
→Basse :
Z, Q, Z, S, D, Z, D, Z, Z, Q, Z, S, Q, Z, Q, Q
→Clarinette :
Q, A, Q, X, Q, S, D, Z, Z, W, S, S, W, S, Q, X, Q, A, Q, X, Q, S, D, Z, Z, S, Q, Z, Z, D, S, W
→Saxophone :
Q, S, A, Z, Q, S, E, Z, D, Z, E, W, D, Z, X, E, Q, S, A, D, Q, S, E, Z, D, S, Z, D, D, A, S, Q

• *Pour pouvoir lancer une partie il faut exécuter le fichier du nom de « main.py ».*

• *Il faut noter que le programme peut avoir des problèmes de synchronisations si l'ordinateur exécute beaucoup de tâches même temps.*

• *Voici l'architecture des fichiers du projet :*

Main :
main.py :
Le fichier général qui initialise tout et qui contient la main loop.

Structures de données :
general.py :

Contient des constantes générales à tout le programme, ainsi que la structure de données Room qui représente une salle

level.py :

Représente le niveau, avec notamment la map, le joueur, les boss et les vagabonds

music.py (Pulsable) :

Le gestionnaire de musique du jeu, notamment lors des combats de boss

sequence.py :

Contient deux classes, Sequence qui représente une séquence d'Action jouée par un boss et Action qui correspond à la suppression ou à l'ajout d'une tuile

video.py :

Interface pour tout ce qui est affichage

Algo :

algo.py :

Contient différents algorithmes, notamment A, is_in_map(), la vision et search_drawable()*

stage.py :

Génération aléatoire de la map, avec l'orientation aléatoire, le placement, la création des chemins et des murs

Composants :

pulsable.py :

Classe contenant la méthode pulsable() appelée à chaque tour de main loop et pouvant être redéfinie de n'importe quel façon par les descendants

drawable.py (Pulsable) :

Classe contenant notamment la méthode draw() qui calcul l'image actuel de l'objet et la retourne

entity.py (Drawable) :

Représente une entité, ajoute notamment les méthodes move() et go_to().

player.py (Entity) :

Représente le joueur

monster.py (Entity) :

Représente un monstre, c'est-à-dire soit un boss ou un vagabond

boss.py (Monster) :

Représente un boss, implémente les combats avec l'exécution des séquences

stray.py (Monster) :

Représente un vagabond, notamment son déplacement "aléatoire", l'agro, le déplacement vers le joueur et l'attaque.

*• On a utilisé le langage Python dans sa version 3.9.2 en utilisant le module **Pygame** pour de l'affichage simple. Il serait possible d'utiliser un module plus bas car on n'utilise que des fonctions simples de **Pygame**.*

*• Nous avons effectué les tests sous **Linux**. Le jeu devrait cependant fonctionner sous d'autres systèmes d'exploitation (pour **Windows** voir les commentaires de music.py).*

• Pour faciliter les améliorations et les ajouts de contenu, le programme récupère les informations dans des fichiers .txt.