GameCodeur – Gaming Campus

Soutenance LUA – Löve2D

Simon Foucher

« Gromokkg Escape »

COMMENT JOUER ?

LE JEU

« Gromokkg escape » est le prototype d'un jeu top-bottom dans lequel le joueur incarne Gromokkg, un grand orc vert ayant été kidnappé et enfermé par « les héros ». Prisonnier d'un donjon obscur où il a subi les atroces expériences d'un mage, il est parvenu à s'emparer du bâton magique de ce dernier et compte bien s'évader. Notamment à l'aide du nouveau pouvoir que les tortures ont fait naître chez lui…

PRINCIPE DE GAMEPLAY

Le gameplay repose sur une loop de jeu très simple :

1. Le joueur entre dans une nouvelle pièce du donjon et découvre une quantité d'ennemis (chevaliers, mages, nains ou princesses).
2. Le joueur doit éliminer l'intégralité des ennemis en leur lançant des sorts l'aide de son bâton de mage.
3. Lorsqu'il a tué suffisamment d'ennemi, le joueur peut « booster » Gromokkg et le transformer en monstre sanguinaire et surpuissant pendant 5 secondes.
4. Lorsque tous les ennemis sont morts, une porte s'ouvre et le joueur peut passer à la pièce suivante.

Lorsque le joueur a remporté les 7 pièces du jeu, il remporte la partie.

COMMANDES

Les commandes de Gromokkg Escape sont les suivantes :

- Déplacement par **touche AZERTY** : *Z haut, S Bas, Q gauche, D droite*. Le jeu permet également via un menu une configuration en mode QWERTY.
- Visée à la souris à l'aide du curseur et **clic gauche** pour lancer un sortilège dans cette direction.
- **Touche Espace** pour activer la transformation en monstre lorsque la barre de boost est pleine.
- **Touche Echap** pour ouvrir le menu ou sauter l'introduction.

COMMENT GAGNER ?

Pour gagner le jeu, le joueur doit éliminer l'intégralité des ennemis présents dans les 7 pièces (ou niveaux) du jeu.

COMMENT PERDRE ?

Le joueur possède une barre de vie. Lorsque les ennemis l'attaquent, il perd de la vie. Le joueur n'a aucune possibilité de regagner de la vie, aussi lorsque sa vie tombe à zéro, il est game over.

CODE SOURCE

STRUCTURE DU CODE

Les différentes scènes

Comme tout code löve2D, mon projet commence par un main.lua. Ce dernier initialise un **UIManager**, qui va choisir quel UI afficher selon l'état actuel du jeu, un **SoundManager**, qui choisit la musique à jouer selon l'état actuel du jeu, et un **GameManager**, qui choisit quelle « scène » utiliser selon l'état actuel du jeu. Cet état actuel du jeu est enregistré dans un fichier de constantes, GAMESTATE.LUA.

Selon le GAMESTATE, le GameManager peut choisir d'afficher les scènes suivantes :

- **Start** : écran de démarrage du jeu invitant le joueur à commencer le jeu.
- **Narrative** : un écran d'introduction narrant au joueur l'histoire du personnage qu'il va incarner.
- **Game** : le jeu à proprement parler
- **Menu** : l'écran correspondant au menu du jeu, avec les settings, restart et exit.
- **GameOver** : l'écran indiquant au joueur qu'il a perdu le jeu.
- **Win** : l'écran indiquant au joueur qu'il a gagné la partie.

Certaines de ces scènes sont très simples : *Start*, *Narrative* et *Win* sont essentiellement des images backgrounds ou du texte avec une interactivité limité. *Menu* comporte des boutons qui déclenchent une fonction spécifique comme relancer le jeu.

Game, le jeu à proprement parler, a un fonctionnement forcément plus complexe.

Fonctionnement de Game

L'élément central de Game est le **LevelManager** qui est chargé de déterminer le contenu pour chaque niveau (ou pièce), notamment les ennemis à afficher et la carte à charger. Pour cela, il s'appuie sur un fichier de configuration nommé **LEVELSCONFIGURATION,** qui décrit pour chaque niveau les ennemis voulus. Il envoie également un *ID* de niveau, *currentLevel*, à un **MapManager** qui sera chargé d'afficher la bonne carte en allant la chercher dans un fichier de configuration nommé **MAPLIST**. En parallèle, le levelManager initie une **MainCamera**, qui est un script simulant une caméra en effectuant des translates sur les éléments draw, et un **CinematicManager** qui permet d'animer automatiquement le personnage joueur quand il entre dans une nouvelle pièce pendant quelques secondes.

Fonctionnement détaillé du LevelManager

Lorsqu'il va piocher dans le fichier **LEVELSCONFIGURATION**, le LevelManager fait ensuite appelle à deux spawners, un spawner pour créer le joueur (**PlayerManager**) et un spawner pour créer les ennemis (**EnnemiManager**). Ces spawners font appel à leur tour à deux factories : une factorie « **Character Factory** », chargée de créer les personnages en piochant dans un dossier de configuration de personnages (un fichier par type de personnage avec les caractéristiques)., et une factory « **Weapon Factory** », chargée de créer une arme en piochant dans le dossier de configuration des armes (un fichier par arme avec les caractéristiques). Ainsi, si le LevelManager demande un chevalier avec une épée, le EnnemiManager va demander à la Character Factory de faire un chevalier, à la Weapon Factory de faire une épée, puis va équiper le chevalier avec l'épée et le renvoyer sous la forme d'une entité **Ennemy**. Le même processus est utilisé pour faire l'entité **Player**, mais le joueur n'est spawn qu'une seule fois.

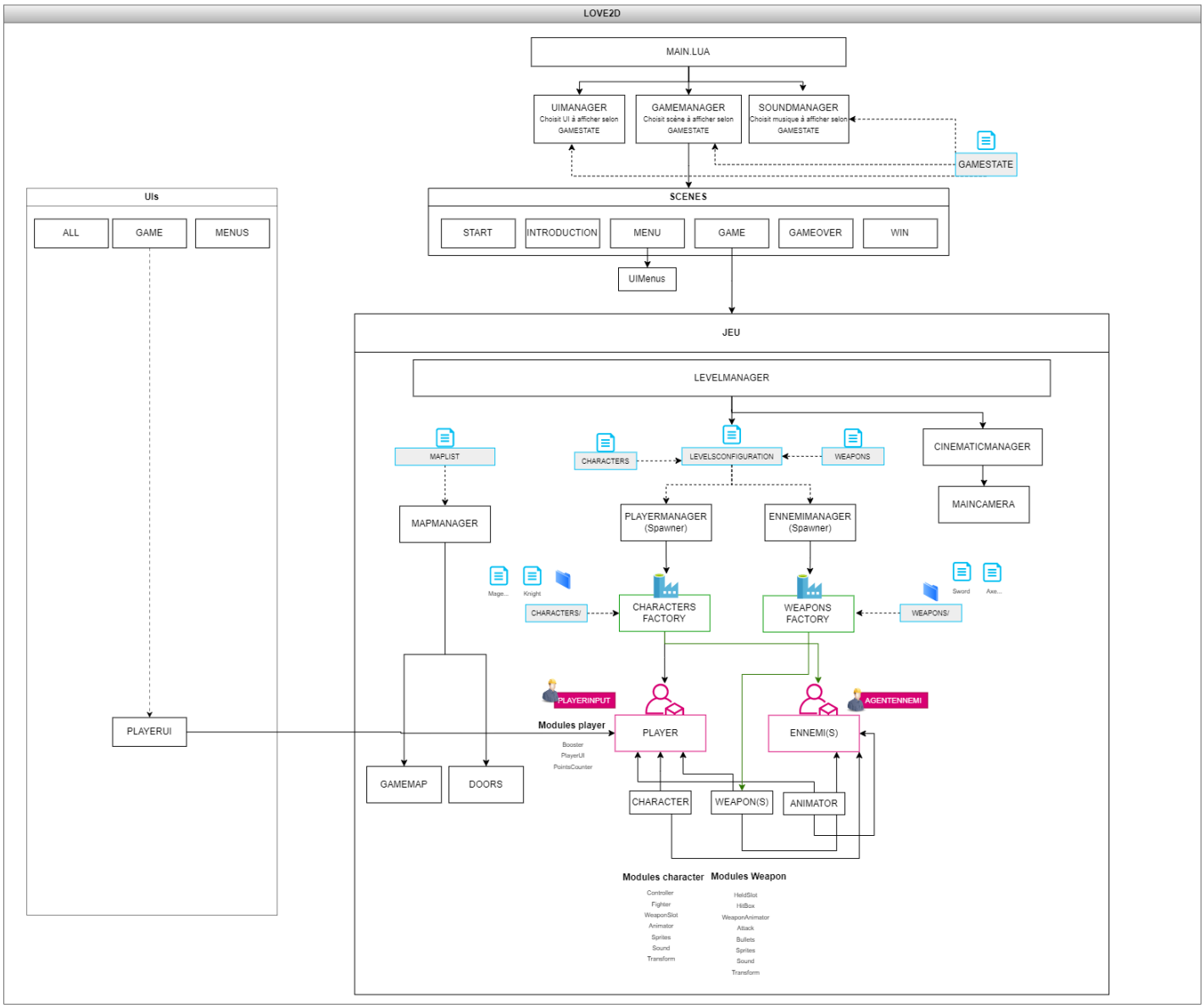
Les entités Player et Ennemy

Player et Ennemy sont chacune des entités qui font appel à un certain nombre de modules : character, qui contient les données « personnage », collider, qui permet de savoir si le personnage est en collision avec la map ou pas, animator, qui permet leur animation automatique (tout le temps pour les ennemis, seulement en cinématique pour le joueur), un fighter qui est le module de combat, qui contient quant à lui un weaponSlot qui peut accueillir l'arme qui a été associé au personnage et bien sûr un module pour les sprites et un pour le son. L'arme, Weapon, a également quelques modules qui lui sont propres comme Attack, pour gérer les tentatives de dégâts, et Bullets, pour gérer l'affichage des tirs s'il s'agit d'une arme à distance.

Néanmoins, la différence majeure entre Player et Ennemy réside dans le fait que le controller de Player va recevoir un module **PlayerInput**, qui permet de contrôler le personnage en appuyant sur des touches, tandis que le controller des ennemis vont recevoir un module **EnnemiAgent**, une machine à état, qui correspond en quelque sorte à l'IA de mes ennemis (Patrouiller, alerté, attaque…).

Le Player a également une UI qui lui est spécifique : barre de vie, barre de points en tuant les adversaires, et icônes des attaquent qu'il peut faire.

SCHEMA DE LA STRUCTURE



DECOUPAGE

Mon idée générale, lors de la conception de ce projet, était de permettre la création de nouveaux niveaux avec une grande facilité. C'est pourquoi, j'ai fait en sorte que le code repose en partie sur les fichiers de configurations.

Si je devais faire un découpage grossier du projet je dirais que les éléments principaux sont les suivants :

1. Un GameManager qui choisit les scènes à afficher.
2. Un LevelManager qui s'appuie sur un fichier de configuration de niveau, pour demander aux factories de créer les personnages et à la map d'afficher la bonne carte.
3. Deux factories qui génèrent les personnages et leurs armes en allant chercher chacune les fichiers qui correspondent à l'arme et au personnage demandé (dossier characters/ et weapons/ dans la partie config).
4. Une entité player qui reçoit des inputs et à une ui propre

5. Une entité ennemie qui a une IA cherchant un joueur à attaquer.
6. Des entités weapons associées soit à une entité ennemie soit une entité joueur, qui peuvent mettre des dégâts au corps à corps ou à distance.

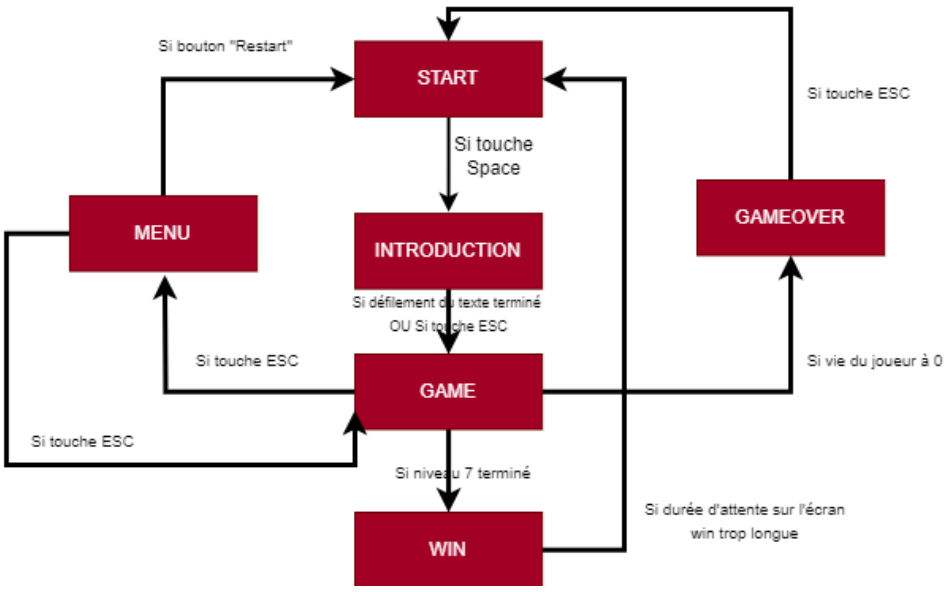
J'ai essayé au maximum de penser les choses comme des objets : je voulais que les personnages aient leur vitesse, leur force, mais les armes également. Donner une autonomie aux armes afin de garder l'idée qu'un personnage « tient » une arme, mais qu'il ne s'agit pas d'un « personnage-arme », me semblait important pour éventuellement pouvoir les utiliser différemment dans le futur. En faisant ainsi, cela permet d'imaginer toutes les combinaisons possibles (un chevalier avec une épée, une princesse avec une fleur, une princesse avec une hache, une princesse avec un bâton de mage, un chevalier avec rien…).

MACHINE A ETAT

Le jeu comporte plusieurs machines à état (il y a par exemple une machine à état dans le levelManager pour gérer si on est au début, au milieu ou à la fin d'un niveau), mais il y en a deux qui sont essentielles :

Machine à état du jeu :

La machine a état du jeu consiste simplement à dire où le joueur en est dans le jeu. Elle est située dans le document GAMESTATE et comporte les états suivants : Start, Narrative, Menu, GameOver, Game, Win. C'est grâce à cela que le GameManager peut choisir la scène à afficher au joueur.

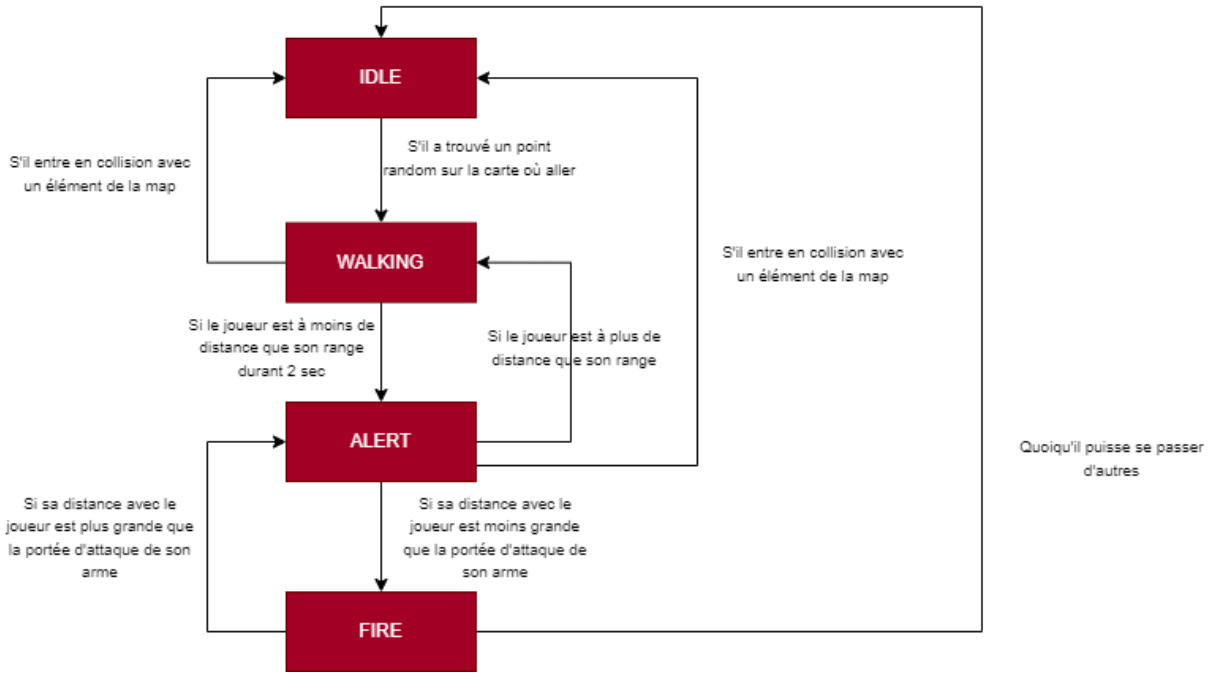


Machine à état du jeu

Machine à état des ennemis (Character) :

Le fonctionnement des ennemis est basé sur une machine à état qui va effectuer différentes actions selon ce qui se passe.

Un ennemi peut être : IDLE, WALKING (marcher ou patrouiller), ALERT (en alerte, il a repéré un joueur), FIRE (attaque). Les conditions pour passer de l'un à l'autre sont assez simples : l'ennemi est IDLE par défaut. Il cherche un point random sur la carte puis passe en WALKING pour s'y rendre. Si jamais le joueur a une position plus proche que le range de l'ennemi pendant plus de 2 secondes, l'ennemi passe en ALERT. Il se met alors à pourchasser le joueur. Si l'ennemi et le joueur ont une distance moins élevée que la distance nécessaire à l'arme que porte l'ennemi pour taper, alors l'ennemi passe en FIRE (attaque). A l'inverse, si le joueur se retrouve hors de son range, il repasse en mode WALKING. Enfin, à chaque fois que l'ennemi entre en collision, il repasse en IDLE pour chercher un nouveau point.



Machine à états d'un ennemi

MON EXPERIENCE

Temps consacré au projet : Durant un peu moins d'un mois (environ 24 jours), j'ai consacré environ 4 soirs par semaines et mes week-ends à ce projet. Je dirais donc une moyenne de 120 heures.

Qu'ai-je appris ?

J’ai appris la vraie utilité des fonctions de trigonométrie comme les calculs d’angle et de distance, que j’avais déjà utilisé par le passé dans des projets Unity, mais sur des problèmes donnés résolus grâce à Google, sans comprendre vraiment pourquoi ça marchait.

Etant habitué à Unity j’ai également appris des choses spécifiques à la 2D comme tout ce qui est affichage de tilemap, création de quad etc. qui est très intéressant.

Quelles sont les difficultés que j’ai rencontrées ?

Mes difficultés sont principalement organisationnelles au niveau du code : savoir où mettre quoi, qui doit communiquer avec qui comment… Je sais faire « fonctionner » les choses, mais j’ai souvent l’impression de complexifier des choses pour rien. Je n’arrive souvent pas à savoir si mon organisation est la bonne et si elle l’était, le doute est trop présent et m’amène parfois à tout réorganiser différemment, pas forcément pour le mieux.

De même puisqu’il ne s’agit pas d’un langage orienté POO, j’ai eu un peu de mal à savoir dans quels sens faire les injections de dépendance parfois. J’avais souvent l’impression de passer des informations que je ne devrais pas faire passer de cette façon par « facilité ».

Quels sont les domaines dans lesquels je pense devoir m’améliorer ?

En raison du point évoqué juste au-dessus, je pense qu’il me faut étudier les notions de design pattern pour comprendre comment les choses doivent s’emboiter.