Projet de Programmation Fonctionnelle Avancée

VaniaCastle

Rapport

April 28, 2025

Èvelyne CHEVAILLIER et Gayathiri RAVENDIRANE

Pour ce projet, nous avons décidé de nous approprier une recette simple et efficace : celle des action-platformer 'Castlevania', en gameplay comme en ambiance. On y retrouve, comme le genre l'indique, de la plateforme, ainsi que des mécaniques de combat, que l'on à voulu rendre les plus accessibles et confortables possibles. Petit détail 1 : le fichier "PrototypesVisuels" contient - comme son nom l'indique - divers prototypes des visuels que nous utilisons dans la version finale, en plus de quelques brouillons de game design (structure des niveaux par exemple). S'y trouvent aussi (ainsi que possiblement quelques uns dans le fichier ressources/images) une myriade de fichiers dont l'extension est ".ase". Il s'agit de fichiers lisible par le logiciel Aseprite, utilisé pour la créations des visuels du jeu, et contenant les différents layers des visuels. Petit détail 2 : la vidéo "demoFinal.mp4" est une vidéo de 2min qui montre comment est censée se dérouler une partie si le boss ne fait pas crasher le jeu.

1 Contexte du jeu

Lors de la nuit de 30 avril - La nuit de Walpurgis - une jeune femme maudite s'élance dans une quête de vengeance et de liberté. Guidée par une carte, a traversé des ruines et des forêts maudites pour atteindre la forteresse de son nemesis, le vampire qui l'a maudite. Pour y atteindre la chambre où il se terre, elle doit d'abord défaire le gadre du Hall, Cyclotor l'Affreux, puis collecter les 4 clef élémentaires qui scellent l'antre du vampire. Si elle échoue avant l'aube, elle disparaîtra avec la nuit. Le projet s'arrête à la mort du personnage où après sa victoire contre Cyclotor.

2 Fonctionnalités

En lançant le jeu, un écran titre se présente à vous. Pressez entrer pour démarrer. Il y a 2 écrans à parcourir et un boss à affronter dans sa salle dédiée. Chaque zone est parsemée de 2 types de monstres : les Squelettes Sanglants et les Jonhsons. Les premiers sont des squelettes qui attaquent devant eux s'ils détectent le player. Les seconds sont des têtes de roches immobiles, à l'apparence familière, qui tirent des boules de feu dans la direction du player. Tuer les monstres est facultatif mais récompensant : en mourant, ils ont 50% chances de laisser tomber une croix inversée qui va augmenter l'attaque du player, 25% de chances de laisser tomber un calice de sang, qui restaure 10 points de santé, et 25% de chances de laisser tomber une orbe de mana, qui resautre 10 points de mana. Le Boss, cyclotor l'affreux, ce trouve dans une salle accessible après avoir parcouru les 2 la zone 1 (plaines) et la zone 2 (entrée du château). Ce cyclope zombie jette de puissantes haches en cloche dans le but d'abbatre le player, tout en se déplaçant selon un pattern un peu plus complexe que celui des squelettes. Il a également beaucoup plus de PV que les autres ennemis, mais, l'abbatre vous montrera l'écran de victoire (fin du jeu). Si, par malheur, un ennemi à la curieuse idée de mettre fin à vos jours (vos PV tombent à 0), c'est un écran de Game Over qui se présentera à vous.

Pour ce qui est du player, il est possible d'attaquer, de lancer des boules de feu, d'aller à gauche et à droite, et de sauter. Les informations relatives au player, à savoir sa santé et sa jauge de mana, correspondent aux respectivement à la barre rouge et à la barre bleue dans le coin supérieur gauche de l'écran. La barre rouge dans le coin supérieur droit de l'écran correspond à la jauge de vie de l'ennemi courant (le dernier ennemi frappé)

3 Utilisation

Voici, la liste des commandes possibles :

Déplacement

- **'Q'** : Déplacement vers la gauche
- **'Z'** : Saut
- **'D'** : Déplacement vers la droite

Combat

- **'Espace'** : Donne un coup d'épée
- **'E'** : Lance une boule de feu.

Autres

- **'R'** : Recharge la zone actuelle (debug)
- **'S'** : Active/désactive la pause.
- **'P'** : Affiche des informations de debug du joueur
- **'Enter'** : Sur l'écran titre, démarre le jeu. Ne fais rien ailleurs

4 Organisation des classes et des méthodes

On a repris la structure de base du TP2 pour organiser notre projet selon le modèle ECS (Entity-Component-System). Ensuite, nous avons intégré l'algorithme de collision du TP3, puis étendu le tout pour créer un jeu complet avec 3 salles, un système de combat, des animations, des décors et des collectibles.
Nous conservé la séparation du code en trois grands dossiers : components, systems et core.

- **components** : contient toutes les définitions des composants du jeu (player, monstre, item, etc...).
- **systems** : regroupe les systèmes, c'est-à-dire les logiques qui agissent sur les divers composants.
- **core** : contient les utilitaires et définitions globales.

5 Résumé du travail effectué

- **Visuels :** Pour avoir des visuels cohérents ainsi que des animations, tout ce qui relève des visuels repose sur 2 systèmes : celui de dessin et celui d'animation. Le système de dessin, on l'a légèrement modifié afin de permettre aux objets d'avoir une texture qui n'est pas de la même taille que sa boîte de collisions. En faisant cela, on peut faire 2 choses : avoir la même taille de boîte de dessin pour toutes les animations des entités (en effet, sans changer la taille d'un sprite, donner un coup d'épée donne des résultats ridicules), sans changer leur boîte de collison; et donner de la perspective au sol (en mettant sa texture quelques pixels plus hauts que sa boîte de collisions). On a aussi ajouté une notion de priorités aux diverses textures, afin d'avoir la garantie que tout soit correctement dessinée dans l'ordre. Le système va par ailleurs calculer le coordonnées réelle de l'image à dessiner, si l'objet est animé, en se basant sur 2 facteurs. Le premiers sont les coordonnées du sprite courant dans son tileset, calculées par le système d'animation, le second est l'attribut booléen "left", qui indique si l'entité regarde, ou non, à gauche (nécessaire pour choisir le bon sprite à dessiner). En effet, on a construits les tileset de telle sorte à ce que leur moitié gauche contienne les sprites pour quand l'entité regarde à droite, et sa moitié droite contienne les sprites pour quand l'entité regarde à gauche (ça peut sembler contre intuitif, en effet). En utilisant ainsi l'attribut "file_width", le draw système va pouvoir, si l'entité regarde à gauche, calculer le sprite à partir de la droite du tileset plutôt qu'à partir de sa gauche, et ainsi dessiner le bon sprite. Le système d'animation va, quant à lui, s'occuper de calculer les coordonnées de la prochaine frame d'animation d'une entité en se basant sur plusieurs facteurs : L'animation courante et l'animation précédente. Si cette dernière est identique à la courante, on poursuit l'animation. Si l'animation courante est différente de la précédente, on débute la nouvelle animation. Il y a une exception pour l'animation d'attaque : Il est possible qu'une entité se déplace, et change d'animation, alors qu'elle n'a pas finit son animation d'attaque. Pour palier à ça, on force à ce que l'animation d'attaque se finisse avant d'en débuter une nouvelle. Les changements d'animations sont effectués dans le Move_system et le Fight_system.

- **Déplacements :** Les déplacements sont eux aussi gérés par 2 systèmes. En premier le Gravity_system, qui s'occupe simplement de faire chuter les entités qui y sont enregistrées. Ensuite, et surtout, le Move_system, qui fait 3 choses : Pour les projectiles, il les fait avancer selon leur vitesse. Pour les ennemis, d'abord il applique leur pattern (attribut "pattern", fonction () -> ()), puis il les déplace dans la bonne direction selon leur vitesse. Enfin et surtout, le player. Ses déplacements sont plus complexes car ils s'accompagnet d'un déplacement de littéralement tout le reste : on veut que la caméra soit centrée sur lui. Et c'est ce que fait le Move_system dans son cas : s'il n'est pas sur un bord de la carte, le système déplace tout les éléments de cette dernière afin de laisser la caméra centrée sur le player.

- **Collisions :** L'essentiels des collisions sont gérées par l'algo du TP3. Il y a néanmoins 4 exceptions notables : Quand un projectile touche une entité, selon les tags, soit le projectiles fait des dégâts à l'entité (exemple : attaque du player sur un ennemi), soit le projectile disparaît (exemple : attaque du player sur une boule de feu ennemie). Quand un ennemi touche un mur ou atteint le bord d'une plateforme, il fait demi-tour (fonction turnaround). Quand le player touche le sol, il peut à nou-

veau sauter (fonction reset_if_ground). Quand le player touche une warpzone, il la zone de destination est chargée, la zone courante est déchargée, et les coordonées du player sont mises à jour. Les warpzones sont en effet construites à partir d'une zone de destination et de 2 fonctions : une pour calculer la nouvelle coordonnée en x, une autre pour la nouvelle coordonée en y. Ce choix permet de conserver l'ordonnée lors du passage d'une warpzone.

- **Combat :** Pour le combat, on est parties d'un principe simple : toute attaque est un projectile. Les attaques de mélées deviennent alors des projectiles transparents et immobiles, tandis que les boules de feu et haches se déplacent et ont une texture. Comme indiqué dans la section précédente, une interaction entre un projectile et ça cible fait baisser les points de vie de la cible. Ainsi, afin de différencier les projectiles alliés et ennemis, nous avons 4 tag différents : 1 pour les projectiles lancés ennemis, 1 pour les projectiles lancés alliés, 1 pour les attaques de mélée alliées, 1 pour les attaques de mélée ennemies. Pour pouvoir créer ces projectiles, les ennemis possèdent un attribut "create_projectile", une fonctions () -> () qui leur permet de créer le projectile associé à leur attaque. Cela permet d'effectuer les attaques de manière simple, mais empêche les ennemis d'avoir plusieurs attaques. Pour ce qui est du player, il possède 2 méthodes similaires, une pour la magie et une pour la mélée, afin d'attaquer. La mort du joueur est détectée par la boucle principale du jeu, qui agit en conséquence (déchargement de tout puis affichage de l'écran de game over), tandis que la mort d'un ennemi est détectée par le Death_system, qui s'occupe simplement de supprimer les ennemis morts. Les attaques des ennemis sont décidées par le Fight_system, qui, pour résumer, fait en sorte que l'ennemi déclenche une attaque s'il y a le droit (préparation terminée, et cooldown terminé aussi) et si le player se trouve dans son rayon d'attaque. Enfin, il a été décidé que les ennemis n'auraient pas de mouvement de recul lors de la prise de dégâts contrairement au player.

- **Items :** La classe Interactable représente les objects avec lesquels le player peut intéragir par simple contact. Seuls les OtherItem sont utilisés dans cette version du projet, afin de faire des loots lachés par les ennemis lors de leur mort. Ramasser un tel object exécute la fonction () -> () de son attibut "action".

- **Temps :** Le Time_system s'occupe de faire disparaître les objets temporaires (items et projectiles) après un certain temps. On remarquera que ce système est le seul à continuer de tourner pendant que le jeu est en pause : c'est nécessaire pour décaler l'instant de mort des entités qui ont une durée de vie et qui sont, comme le jeu est en pause, inactifs.

- **Zones :** Les zones sont en quelques sortes des wrapper qui contiennent un ensemble d'objets de différentes classes. Elles sont représentées par une origine, des dimensions, et surtout une liste d'ennemis, une liste d'items, et une liste de boîtes. Le contenu de ces listes est chargé/déchargé à l'entrée/sortie d'une zone.

6 Prise de recul

- Dans un premier temps, l'implémentation de l'algorithme du TP 3 nous a rajoutés quelques bugs plutôt déplaisants :

1. Il arrivait fréquemment que les collisions entre l'entité player et les diverses plateformes n'aient pas le comportement attendu. 2 principaux bugs survinrent au même moment. Le premier faisait que le player pouvait passer à travers un plateforme particulière, ontologiquement identique aux autres. Celui-ci a miraculeusement disparu lorsqu'on est passées de notre première à notre seconde carte de tests. Le second était bien plus embêtant : lors d'un contact avec la partie inférieure d'une plateforme trop fine (verticalement) la vitesse verticale du player explosait (elle passait de 0 à 7000). On pense que c'était parce que le player passait légèrement au dessus de la moitié de la plateforme (lors de la collision), et donc que l'entité était propulsée vers le haut de manière très violente. Nous avons réglé le problème en empêchant toute entité d'avoir une vitesse verticale supérieure à la vitesse de saut.
2. Après la correction du second bug susmentionné, un nouveau, bien moins gênant, survint. Il fait que, lors du contact avec la partie inférieure d'une plateforme, le player est catapulté vers le bas. Cela est dû à la façon dont le système de collision fait rebondire une entité qui a une masse finie (ici, player) lors du contact avec une entité de masse infinie. Ce bug n'a pas été corrigé.
3. L'écran "tremble" quand l'entité player fonce dans un mur. Ce bug ne génant que très peu le gameplay, nous avons préféré implémenter d'autres fonctionnalités plutôt que de le corriger.

- Initialement, le jeu plantait quand le player changeait d'écran. Le problème est survenue 2 fois. La première fois, il y avait simplement des entités qui étaient chargées plusieurs fois dans certains systèmes, ce qui causait le plantage, la seconde fois, impossible de tracer la source du problème. Après un certain temps passé à essayer de trouver d'où venait le problème, dans une tentative supplémentaire d'en trouver l'origine, on a essayé de charger la zone de destination avant de décharger la zone courante (au lieu de faire les choses dans l'ordre inverse) ce qui a juste corrigé le bug. Il n'est ensuite jamais revenu.

- Il y a de très très gros problèmes avec la version SDL, en particulier au niveau de l'affichage des éléments (ainsi que peut-être du lag, mais c'est difficile à dire).

- Un autre bug plutôt amusant nous a hantées pendant la quasi intégralité de la réalisation du projet. Il faisait simplement que, aléatoirement, certaines textures ne s'affichaient pas. C'est un peu par hasard qu'on la corrigé, et c'était dû principalement à une grosse erreur d'inattention. Avant de passer aux textures actuelles, où chaque plateforme possède sa propre texture, nous avions décidé d'attribuer une très grosse texture à la zone courante, pour limiter le nombre de fichiers à charger. À ce moment, nous faisions nos tests dans la zone 1. Sa texture était donc le fichier testZone1 (toujours présent dans le dossier resources/images). A première vu, rien dans ce fichier ne semble poser problème. un détail a cependant changé depuis l'utilisation de ce fichier : Avant, toute la partie transparente était colorée en blanc. En conséquence, comme on ne peut pas trop savoir dans quel ordre les choses sont affichées par le système de dessin, il arrivait que la texture de zone soit dessinée après d'autres textures, la partie blancheles recouvrant en conséquence.

- Si un player est trop proche d'un ennemi, il va avoir un mouvement de recul dans le mauvais sens en se faisant attaquer.

- Le dernier bug rencontré était lui aussi un problème de plantage. Il était provoqué par une surabondance d'entités chargées dans les différents systèmes, ce qui faisait crasher le jeu. Cela survenait souvent dans la 2nde zone, à cause notamment des haches lancées par le boss. En séparant la salle du boss du reste, le problème peut encore survenir, mais beaucoup moins souvent

- Un dernier problème, qui n'en est qu'à moité un, est un mauvais alignement de la boîte visuelle du player avec sa boîte de collision quand ce dernier ce dirige vers la gauche. Ça n'est pas vraiment un problème de code : Pour le corriger, il faudrait réaligner chacune des images de chacune des frames du tileset du charachter. Ça n'est par particulièrement dur, mais c'est quelque chose de long et peu intéressant à faire, pour régler un problème pas si génant.

7 Idées non finies

À travers le code subsistent des vestiges d'idées qui n'ont pas eu le temps d'aboutir. La principale d'entre elles est l'inventaire, dont il ne manquait que plus ou moins que l'affichage. C'est ce pour quoi étaient prévus les CollectableItems et les ClickBoxes. C'est aussi le cas du temps de préparation d'attaques. À la base, il était prévu que les squelettes aient une animation caractéristique avant leur animation d'attaque, afin de signaler au joueur de s'écarter pour éviter l'attaque, mais nous n'avons pas eu le temps de finir ce point. Enfin, c'est le cas du 4ème ennemi, la BatFrog, qui était censée sautiller partout dans la première zone, mais dont nous n'avons pas terminée l'implémentation. Un game over était aussi censé survenir si l'on ne finissait pas assez vite le jeu, mais ça non plus ne s'est pas fait.

8 Organisation du travail

Pour bien mener ce projet, nous avons utilisé diverses méthodes de collaboration. Le "pair programming" a été notre approche principale, où chaque membre du groupe a contribué à la rédaction du code. Même lorsque nous avons écrit des classes séparément, nous avons systématiquement passé en revue le code de l'autre pour le déboguer ou l'améliorer.

Concernant la gestion du code source avec Git, nous avons travaillé sur des branches distinctes pour éviter les conflits de versions.

Gayathiri s'est occupée de la gestion des items (tout ce qui relève des Interactable), du Time_system, de l'Animation_system, du Level Design et de la structure de base d'inventaire.

Èvelyne a réalisé les systèmes de collision et de combat, la gestion de la boucle principale, les inputs, les textures (et visuels de manière générale), ainsi que les ennemis et leurs comportements.

9 Conclusion

En conclusion, ce projet nous a permis de découvrir concrètement la programmation fonctionnelle avancée avec OCaml à travers la création d'un vrai jeu. On a appris à utiliser le modèle ECS pour organiser notre code, à gérer les collisions avec l'AABB, et à intégrer des éléments comme les attaques, l'inventaire ou les objets interactifs.

10 Crédits

Le tableau apparaîssant dans la salle du boss est une version retouchée (pour avoir le filtre duochrome rouge) de l'illustration de la jaquette du jeu Castlevania Symphony Of The Night. Les mots de l'écran titre proviennent de l'écran titre de ce même jeu (et on été réorganisés pour former notre écran titre)

Structure du Projet

```
src
| components
|   | area.ml
|   | area.mli
|   | box.ml
|   | box.mli
|   | component_defs.ml
|   | enemy.ml
|   | enemy.mli
|   | interactable.ml
|   | interactable.mli
|   | player.ml
|   | player.mli
|   | projectile.ml
|   | projectile.mli
| core
|   | cst.ml
|   | global.ml
|   | global.mli
|   | input.ml
|   | rect.ml
|   | rect.mli
|   | texture.ml
|   | vector.ml
|   | vector.mli
| systems
|   | collision.ml
|   | death.ml
|   | draw.ml
|   | fight.ml
|   | move.ml
|   | system_defs.ml
|   | timeSystem.ml
| game.ml
```