Thomas Bedeau

tb@thordecall.com

NFA019 : Compte-rendu de Projet

Projet : “MonsterMayhem”

Merci de prendre le temps de corriger ce travail. Avant toute chose, j’espère que vous avez lancé le jeu et profité d’une petite partie. Si ce n’est pas le cas, je vous suggère de le faire, ce qui rendra la lecture de ce compte-rendu bien plus claire.

Ayant travaillé quelque peu avec le framework Slick2D pour Java, je souhaitais réaliser un projet sans framework autre que ceux de Java, pour solidifier mon savoir et tester mes compétences dans l’utilisation du langage.
Je suis donc parti sur un jeu vidéo, un jeu de tir vu du dessus basé sur Asteroids, un jeu d’ATARI de 1979.

Pour le contexte, j’ai commencé le projet début Mars, en même temps que trois autres matières de cette formation dont deux concernant Java : NFA035 (Java : bibliothèques et patterns) et NFA032 (Java : programmation orientée objet). Je n’avais dans ma formation que NFA031 (Java: notions).

I - Création du moteur graphique

1) Création du moteur et premiers problèmes

Mon premier objectif était donc de créer une forme, de l’afficher, et de la faire bouger avec des instructions reçues du clavier. Pas de problème pour le keyListener et la lecture des touches du clavier.

Cependant, je m’attendais à trouver dans les librairies Swing et AWT d’avantages d’outils que ce que j’y ai trouvé. Premièrement, l’affichage de plusieurs objets dans un JFrame, pour le néophyte, n’est pas chose aisée. C’est seulement il y a quelques semaines, à la fin de NFA035, que j’ai vu en détail le fonctionnement des interfaces graphiques. En Mars, à l’époque, il m’a alors fallu de nombreux essais avant d’arriver à un résultat cohérent.

Après avoir créé une forme, (surpris de n’avoir à ma disposition que Rectangle et Polygon), j’ai vite compris à ce moment que sans framework, j'aurai très peu d’outils simples à utiliser. J’ai malgré tout réussi à sortir un Polygon en forme de triangle pour le vaisseau. Il m’a fallu apprendre et comprendre comment marche le rendu des objets avec Swing (et paint). J'en suis arrivé à une première version qui fonctionnait avec un AffineTransform, pour tourner et faire bouger ma forme. Malheureusement cet outil modifie également les axes d’orientation de ce qu’il transforme, ce qui ne permettait pas de mouvements indépendants de la vélocité (pour un déplacement comme dans l’espace).

A ce niveau-là du projet, j'étais bien embêté, et coincé : je ne trouvais pas de solution ni d'outil qui allait me permettre de faire ce que je voulais. Je m'attendais à avoir plus d'outils que ça à ma disposition, mais ce n'était pas le cas.

2) Trigonométrie

J'ai alors fait ce que j'imagine tout développeur dans le pétrin fait : je suis allé chercher sur Internet, une manière en Java de faire des jeux de tirs avec des mouvements dans l'espace. J'ai fini par trouver un projet similaire au mien : un clone d'Asteroids en Java (fourni avec le projet, dans le dossier vendor), datant de 1998. En étudiant le programme, j'ai trouvé ce qu'il me fallait. La solution n'avait rien d'évident : elle consistait à transformer point par point les formes polygonales, ce que je n'avais aucune chance de pouvoir faire moi-même, ayant pour formation un bac en sciences économiques et sociales et des souvenirs de trigonométrie datant de plus de 20 ans.

Heureusement, j'ai réussi à implémenter la formule dans mon propre code après de nombreux essais, et à force de tests et de petites modifications, avoir des mouvements cohérents pour mes trois objets Shot, Ship et Asteroid.

3) Timer et fonction Update

Evidemment, pour calculer les déplacements en fonction du temps, il me fallait une fonction temporelle. J'ai alors eu l'idée d'implémenter un chronomètre qui appellerait une fonction tous les x temps, ce qui m'a amené à mettre un Timer. Après quelques essais d'implémentation de l'ActionListener (je n'ai vu les Listeners qu'il y a quelques semaines), je suis arrivé à une solution satisfaisante, le Timer appelant la fonction actionPerformed() de l'ActionListener toutes les 20ms, commandant la fonction update() de chaque objet, elle-même envoyant les instructions de rendu et de déplacement des points.

4) Collisions

Là aussi, je m'attendais à plus de simplicité dans les outils de Java. Il m'a pourtant fallu implémenter des itérations dans des itérations : chaque forme regarde si elle contient une itération dans les points d'une autre forme. Cela m'a valu pas mal de problèmes dans la gestion des collisions, certains éléments étant retirés directement au contact, mais continuant d'être itérés dans les boucles d'autres objets. Cela a été réglé avec un système de nettoyage : les éléments sont marqués à être détruits, et sont retirés à la fin de la boucle lancée par le Timer.

II - Création du système audio

A ce moment-là du projet, j'aurais pu avoir fini : le projet était "jouable" et fonctionnel, bien que sans grand intérêt. J'ai eu alors l'idée de tester une autre idée que j'avais en tête, et d'utiliser mes compétences en musique pour transformer le jeu en une expérience musicale.

1) Le système audio MIDI

A la base, je suis parti sur un projet en MIDI (musique générée par ordinateur). Cela m'a pris du temps, mais j'étais arrivé à un résultat assez amusant, avec plusieurs Timers, correspondant aux temps dans la chanson (un pour les rondes de quatre temps, un pour les blanches de deux temps, un pour les noires d'un temps, un pour les croches d'1/2 temps, et ainsi de suite jusqu'à la triple croche), qui lançaient chacun une note aléatoire dans la gamme de mon choix. Les notes, jouées par un synthétiseur dont je pouvais choisir le son, variaient en fonction du nombre d'astéroïdes.

Le système était parfaitement fonctionnel mais a dû être abandonné : pour qu'il fonctionne, l'utilisateur du programme devait avoir installé une interface MIDI et l'avoir activée, ce qui était le cas sur mon ordinateur, me servant pour la musique ; mais ce qui ne serait cependant malheureusement pas le cas sur la plupart des ordinateurs sur lequel ce projet allait passer, et par conséquent, j'ai du l'abandonner. Cela ne sera pas perdu, ayant appris l'implémentation d'un système MIDI en Java, qui pourra me servir dans le futur.

2) Le système audio "analogique"

Sans le MIDI, je savais que je pourrais toujours faire jouer des fichiers audio. J'ai donc enregistré 44 notes de guitare, 22 en son clair et 22 en son overdrive, pour reproduire un peu le principe du MIDI. Enregistrées dans un tableau, je pouvais faire appel à l'index du tableau pour jouer la note dont qu'il fallait. Et donc donner un index au hasard dans une gamme donnée pour que chaque tir joue une note aléatoire tout en restant harmonique.

Mon problème principal est venu du fait que c'était très lourd, et que je n'ai pas trouvé de documentation efficace sur l'audio dans Java. La plupart de ce que je j'ai trouvé était cohérent pour lire un fichier audio, mais pas pour lire à répétition de très petits fichiers.

Cela m'a valu plusieurs implémentations différentes. La plus efficace fut celle où chaque tir créait son propre AudioInputStream, son propre Clip et le jouait et puis fermait tout. Mais lors de longues parties, j'ai eu quelques crashs dus à la mémoire, et en étudiant le problème j'ai vu que mon application Java utilisation 4 à 5 fois plus de Threads que mon système d'exploitation. Chaque tir ouvrait un Thread qui ne se fermait visiblement jamais, malgré toutes mes tentatives.

Au final, je suis resté sur une solution qui "boucle" les notes. Les AudioInputStream sont ouverts au début du jeu, et fermés à la fin : il y en a un pour chaque note, qui est rembobinée après sa lecture.

III – Optimisation et finalisation

A ce niveau-là du projet, j'étais bien avancé, et les deux autres matières Java qui prenaient déjà beaucoup de temps se sont vues agrémentées de leur propre projet. J'ai donc travaillé sur ceux-ci qui à la base étaient à rendre plus tôt que celui-ci. J'ai également appris beaucoup sur les collections, la lecture et l'écriture des fichiers, et les interfaces graphiques.

1) Les évènements (AbstractEvent)

A la reprise du projet, j'avais plusieurs problèmes. Mon code était affreusement long : pour gérer les évènements qui arrivaient au fur et à mesure de la musique et dans le jeu, il y avait des conditions if-else et les instructions à chaque fois. J'en était qu'à 3 minutes / 10 de la chanson : ce n'était pas possible.

J'ai donc eu l'idée de créer un objet contenant les informations nécessaires aux évènements survenant au long du jeu, et de faire de même pour les évènements audio (changement de gammes). Ce qui m'a amené à faire une classe abstraite AbstractEvent, avec un temps d'activation, un nom et une identification. De cette classe découle deux types d'évènements, les GameEvent, qui définissent une action à être effectuée par la classe principale, et les MusicalEvent, qui définissent la gamme et les effets à être appliqués par le système sonore.

2) Le temps dans la musique

Pour pouvoir marquer et lancer efficacement les évènements dans le jeu, je devais récupérer le temps dans la chanson principale. Malheureusement, l'appel de la fonction qui gère ceci renvoie un long et est affreusement lourd – impossible de l'appeler dans l'update, cela faisait saccader le jeu (un jeu de 58ko, qui plus est). Heureusement, je n'avais pas besoin d'être si précis, et j'ai pu trouver une solution en appelant cette fonction seulement une fois toutes les quatre updates, et rester fluide.

3) Optimisation du code : GameElement

Je me suis aperçu que mes objets Ship, Shot et Asteroid possédaient de nombreuses variables et fonctions en commun. J'ai donc créé une classe abstraite, GameElement, qui regroupe tout ça. Par contre, je trouve le code un peu moins lisible avec des getters et des setters partout que ce qu'il était avant cette modification ; mais sûrement bien plus efficace.

4) Système de nettoyage

J'avais également un problème avec mes collisions : bien que pas fatales, j'avais souvent des exceptions levées lors du contact entre plusieurs objets. J'ai fini par comprendre que les évènements lors des collisions, à savoir le retrait de certains éléments au milieu de leur boucle d'itération, créait de nombreux problèmes d'index. J'ai solutionné ce problème en remplaçant le retrait par un marquage "à détruire", appelé en fin de boucle Update.

5) Tableau des scores

Le jeu étant basé autour d'une chanson de dix minutes, je ne pouvais pas permettre de système de vies, sans quoi le joueur risquait de ne pas profiter de l'étendue de la chanson. J'ai donc implémenté un système de score, ce qui signifiait également un tableau des scores, et donc une sauvegarde des scores. Heureusement mes derniers cours de Java concernaient les entrées-sorties, j'ai donc réussi à sortir un fichier des scores, "encodé", lu à chaque appel du tableau et écrit à chaque fin de partie à l'entrée des scores.

IV – Auto-critique : forces et problèmes

1) Problèmes

Il reste plusieurs problèmes dans le projet.

Premièrement, mon implémentation de Swing est un peu hasardeuse. J'ai mis bien du temps à comprendre comment cela marchait, et encore maintenant, c'est assez flou (sur ce projet). Notamment dans le cadre de superposition d'objets. Vous trouverez dans le code un bon nombre de frame.validate() et frame.repaint(), qui ne sont pas en théorie nécessaires mais qui s'avèrent l'être dans certains cas. Tout marche, mais je pense qu'il y a une manière beaucoup plus optimale de gérer les superpositions d'objets.

Par exemple, il y a dans l'update de la classe principale une fonction dont le seul but est de maintenir le fond étoilé derrière les autres éléments. Pourquoi est-ce le seul à passer devant ? On ne sait pas. Je pense que j'aurais du utiliser le LayeredPane, malheureusement à l'époque je ne connaissais pas, et c'est trop tard pour tout changer.

Il existe un autre bug aussi, implémenté avec le tableau des scores : pour une raison obscure, après avoir entré votre nom dans les scores, le jeu s'arrête après avoir affiché le tableau des scores. Ce qui n'est pas tellement un problème, le jeu étant fini. Mais en vrai, les processus ont l'air tous suspendus. Il faut changer d'application et revenir sur la fenêtre pour reprendre le cours du jeu (même si il n'y a plus rien à faire, le jeu n'ayant pas de fonction de redémarrage). Je pense que c'est dû au retrait du ScoreEntry, le petit tableau d'entrée de nom, qui retire le MouseListener, qui doit perturber les processus.

2) Forces

C'est fini, c'est fonctionnel, c'est jouable, c'est fluide, et ça le reste !

Les collisions sont bien gérées, les évènements sont bien placés, les déplacements sont cohérents. L'aléatoirité fonctionne plutôt bien même si les astéroïdes se retrouvent parfois avec des formes un peu étranges. On peut même partager les scores avec ses amis !

D'une manière générale, les objectifs ont été atteints : j'ai réalisé un petit jeu vidéo sans framework, que je pourrai même distribuer !