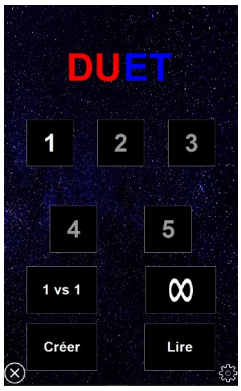
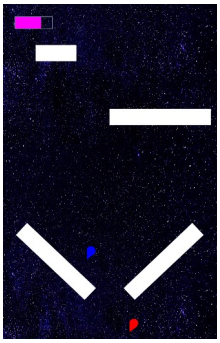
Projet Y2Ja : DUET

I : Introduction



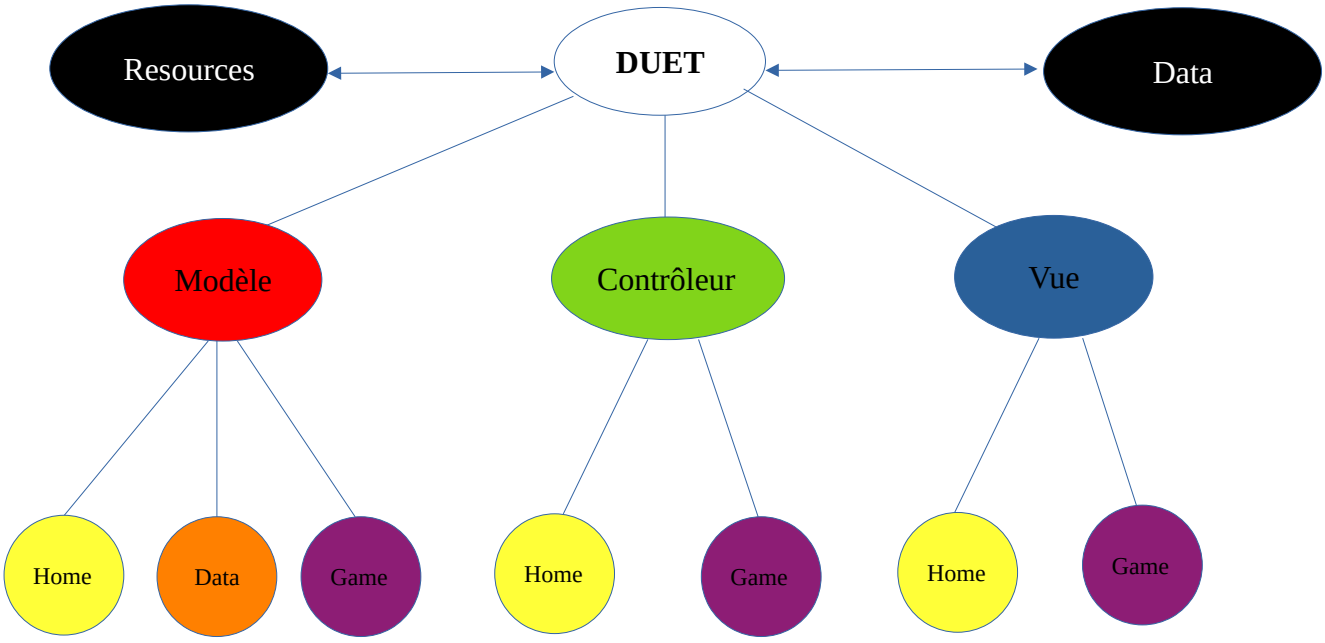
DUET est une recréation du jeu vidéo homonyme développé par Kumobius. Le joueur prend le contrôle d'un volant de deux boules colorées et diamétralement opposées afin d'esquiver les rectangles blancs se dirigeant vers le volant. L'objectif est simple : il s'agit d'une épreuve d'endurance, de parvenir à persister jusqu'à la fin du niveau… si elle existe.
Le jeu est en effet découpé en 5 niveaux prédéfinis, au-cours desquels le joueur doit tourner et déplacer horizontalement le volant pour survivre à la marée d'obstacles. Mais comme mentionné précédemment, il existe également un niveau infini, prenant des morceaux des 5 niveaux et les mélangeant pour créer une session prenant fin à la moindre erreur du joueur.
Ce dernier peut également paramétrer DUET à ses goûts, en déterminant l'usage d'effets sonores et de musique spécialement composée pour le jeu, des effets d'arrière-plan, et de désactiver l'inertie du volant si celle-ci s'avère trop gênante. La possibilité de modifier les contrôles du clavier est également mise à disposition.



Comparé au jeu original, cette version conserve le principe et l'esthétique, tout en rajoutant quelques nouveautés ; tout d'abord, le mouvement du volant selon un axe horizontal et l'habilité de faire disparaître un obstacle sont des nuances de jeu absentes du prédécesseur, en plus de deux nouveaux modes de jeu : un mode un-contre-un où deux joueurs s'affrontent pour déterminer lequel des deux durera le plus longtemps face à un niveau infini joué simultanément, et un mode de création de niveaux, où les joueurs pourront créer leurs propres niveaux, y jouer, et se les partager grâce au stockage sous forme de fichier.

II : Le fonctionnement du jeu et son architecture

La programmation de DUET suit l'architecture Modèle-Vue-Contrôleur, également divisée en une section 'home' pour les menus et 'game' pour le jeu en soi. Dans les grandes lignes, l'architecture peut se résumer au graphe suivant :



On remarque également la présence de données externes. Il s'agit, pour data, de sauvegarder la progression du joueur à travers les niveaux et les paramètres entrés par ce dernier. Quant aux ressources, ces dernières comprennent les images utilisées pour l'affichage du jeu, les effets sonores et la musique, ainsi que les niveaux prédéfinis.

a) Lecture des ressources

Afin de pouvoir se servir des ressources, il a fallu se servir de classes spécifiques pour assurer la bonne lecture et écriture des fichiers. Le son est géré par une classe Sound présente à la racine de l'arborescence, là où les images sont chargées et affichées par la classe Graphics2D de Swing. Quant aux niveaux et à la progression, on se sert des méthodes de sérialisation de Java. L'on a également programmé un parseur pour convertir des fichiers texte en niveaux. La possibilité d'utiliser le format JSON a été évoquée, mais l'on a eu d'autres priorités lors du développement, sachant que le parseur n'a pas encore montré de vulnérabilité jusqu'à présent, ce n'était pas une fonctionnalité urgente.

b) Modèle

i) Éléments de jeu

Le jeu est modélisé principalement par un volant constitué d'un centre et de deux boules, doté d'une vitesse et état de rotation et si activée, d'une inertie. Les obstacles, eux sont des polygones (en particulier des rectangles dans le cas présent) décrits par leurs points dans le plan, un centre et une vitesse de rotation, ainsi qu'une vélocité et d'une direction de chute.

ii)Défilement

Pour animer le jeu, l'on s'est servi d'une classe Timer de java ; cette classe a la particularité de pouvoir appeler une méthode avec un temps différé et une périodicité définie en paramètre, tous deux basés sur le temps réel. Grâce à cette dernière, l'on peut appeler une fonction de rafraîchissement à des intervalles très courts pour animer le jeu, c'est à dire faire tourner le volant, tomber les obstacles, vérifier les collisions, etc.

iii)Stockage des niveaux

Ici, les niveaux ne sont pas codés en dur dans le programme. Il s'agit plutôt de fichiers sérialisés par une utilité développeur non-utilisée par le logiciel, mais toujours présente dans le code par la classe TxtToLvl, qui donne une interface graphique simpliste pour parser un fichier texte en fichier niveau. La classe sérialisée est PatternData, qui se résume à une table de hachage associant à un obstacle un instant d'apparition en millisecondes. Ainsi, pour charger les niveaux juste avant d'y jouer, l'on charge le PatternData correspondant depuis son fichier dans une implémentation du Timer de java ObstacleQueue, qui vise à faire apparaître un obstacle au moment qui lui est associé dans le PatternData.

c) Contrôleur

i) Gestion des entrées clavier

L'écoute du clavier par le programme est gérée par l'interface KeyListener de Swing. Étant donné la nature modifiable des contrôles, l'on a opté pour le stockage des commandes dans un fichier à part, qui est chargé lorsque l'on entre dans un niveau. En comparant l'entrée détectée aux commandes associées, l'on parvient à contrôler le jeu au clavier.

d) Vue

i) Géométrie

Les éléments de jeu n'étant que des formes géométriques simples et monochromes comme des cercles et des polygones, les outils de la bibliothèque Swing sont suffisants pour générer leur apparence, puis les afficher dans la fenêtre de jeu. Quant aux balles du volant, elles présentent une traînée colorée, qui est en fait une file de cercles plus petits qui suivent la trajectoire de la balle.

ii) Tâches et arrière plan

Étant des éléments graphiques plus complexes, les tâches et l'arrière plan ont dû être importés depuis des fichiers. Les tâches sont des images partiellement affichées selon un point d'ancrage déterminé par une précédente collision entre un obstacle et une boule. Seule la partie de l'image étant dans l'obstacle est affichée. Quant à l'arrière-plan, il s'agit en fait d'une image fixe, que l'on fait défiler verticalement en boucle, revenant à l'extrémité opposée une fois que l'on a dépassé la limite de l'image.

III: Difficultés de développement

a) Faire tourner le jeu en temps réel

L'un des défis majeurs posés par ce projet a été la programmation d'un jeu en temps réel sans se servir d'outils externes comme Gradle ou Maven. Au départ, nous étions simplement parti sur une boucle infinie dans laquelle l'on mettait constamment à jour les données selon une variable incrémentée à chaque tour de boucle servant de chronomètre depuis le lancement du jeu. Bien sûr, ce modèle posait un gros lot de problèmes, comme le fait que le jeu soit beaucoup plus susceptible aux ralentissements processeur, le fait qu'il ne puis tourner qu'un temps limité à cause de la limite supérieure des entiers, entre autres. On a donc opté à la place pour un Timer de la bibliothèque Java, qui crée un Thread séparé du jeu principal aligné sur le temps réel chargé d'exécuter les fonctions de mise à jour toutes les millisecondes.

b) Détecter les collisions

Un autre obstacle rencontré fût… la détection des obstacles. Lors de la création des premiers prototypes du jeu, l'une des principales difficultés a été les collisions entre les obstacles et les balles du volant. Au départ, les collisions étaient assez chaotiques quand elles n'étaient pas entre le bas d'un obstacle et le haut d'une balle : quelques fois, la balle pouvait entrer à l'intérieur d'un obstacle par le côté ou le haut, mais ne pas ressortir à cause d'un décalage dans les calculs. Après avoir cherché longuement, on a trouvé que l'on ne testait la collision que pour le sommet supérieur des boules et non autour du centre. On a donc réparé le décalage et les collisions ont fonctionné correctement

c) L'affichage des tâches

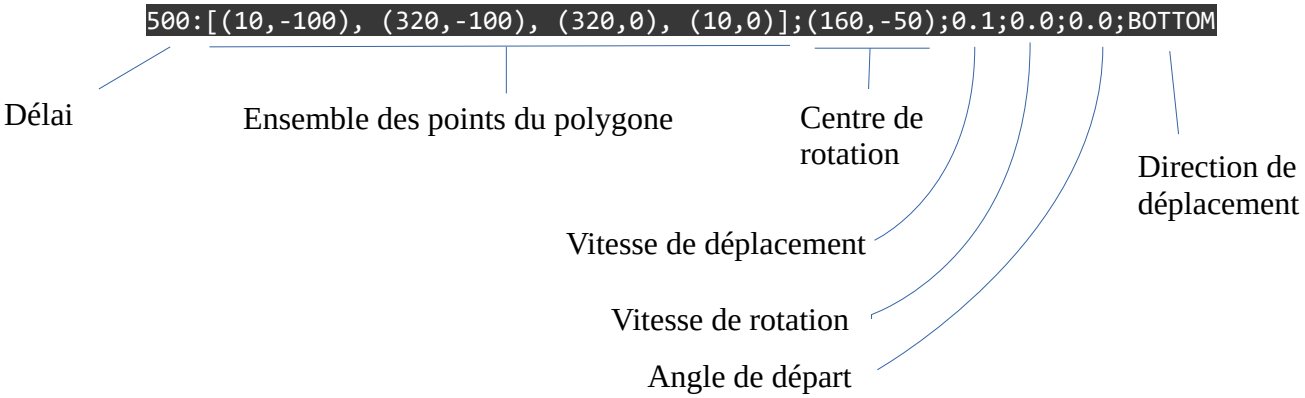
Une autre difficulté surprenante a été d'afficher les tâches après la collision d'une boule sur un obstacle. Au départ, tout semblait fonctionner : l'image de la tâche était limitée à l'obstacle et suivait le mouvement de l'obstacle. Cependant, lorsqu'on a commencé à donner une rotation aux obstacles, les problèmes ont commencé à se montrer de manière chaotique : les tâches échangeaient leurs positions et ne s'appliquaient pas les mêmes rotations entre elles. Il se trouve en fait que les rotations étaient gardées en mémoire et s'appliquaient plusieurs fois aux tâches au fur et à mesure qu'elles s'accumulaient ; la tâche n°1 tourne 1 fois, la tâche n°2, 2 fois, etc.… Seulement, ce n'était pas le seul problème lié aux tâches. A chaque fois que l'on ajoute une tâche, l'image de la tâche était chargée une nouvelle fois en mémoire, ce qui avec le temps épuise la mémoire disponible. Pour y remédier, on les a chargées une seule fois et on utilise la référence chargée pour afficher la tâche

IV : Choix de design

Au cours du développement du jeu, on a dû faire de nombreux choix concernant le design du code et les outils que l'on utilise pour créer notre jeu. Allant des modes de jeu au stockage et à la création de niveaux, en passant par certaines utilités développeurs, l'on comparera les différentes options qui se sont présentées lors de la fabrication du jeu, et qu'est-ce qui nous a décidé à choisir la direction empruntée.

a) La création de niveaux par fichiers texte

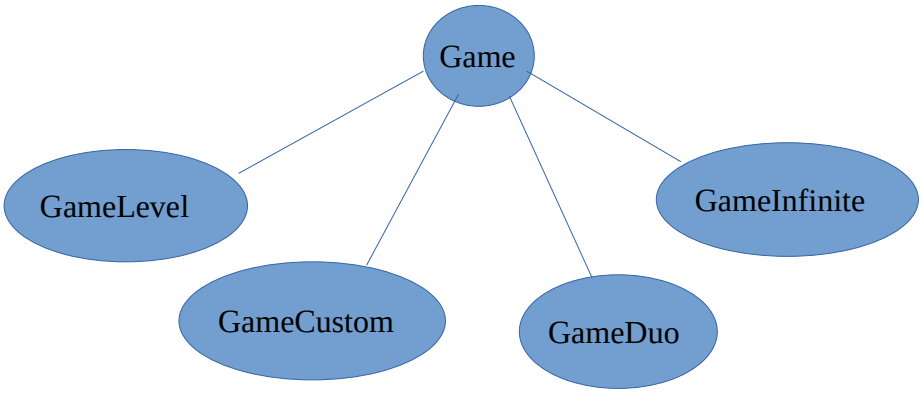
Assez tôt dans le développement, la nécessité d'avoir un niveau dans lequel on puisse tester les différents aspects implémentés s'est vite faite sentir. Afin de pouvoir créer des niveaux relativement facilement côté développeur, l'on a mis au point un parseur de niveaux, censé lire un fichier texte avec un formatage spécifique et en créer un niveau à partir des données trouvées. Le formatage est comme-suit :
À chaque ligne, correspond un couple délai:obstacle, ressemblant à ceci :



Au final, un tel stockage aurait pu tout aussi facilement pu être implémenté sous le format JSON, spécialisé dans le stockage de classes pour divers besoins et facilement lisibles par n'importe-quel utilisateur. La raison pour laquelle on a gardé ce système, comme énoncé précédemment, est parce que d'autres priorités se sont faites sentir, et tenter de modifier un système qui a bien fonctionné jusqu'alors pendant que d'autres fonctionnalités sont en perspective est rarement une bonne idée. De plus, il aurait fallu prendre le temps d'apprendre à utiliser les librairies mettant Java et JSON en relation, tandis que RegEx et la lecture de fichiers texte avec Java sont des acquis du semestre précédent. C'est cet argument qui a poussé ce choix initial, et c'est le bon fonctionnement du système qui l'a fait durer.

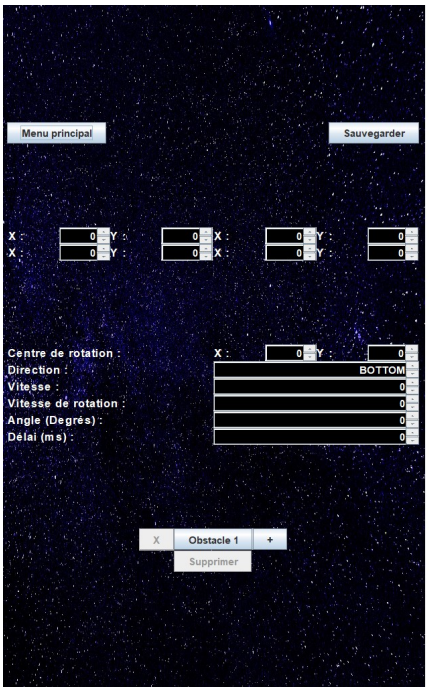
b) Les modes de jeu orientés objet

Duet dispose d'une variété de modes de jeu, entre le mode standard, le mode 1vs1, infini, et les niveaux crées par l'utilisateur, toutes les classes responsables de ces modes de jeu, que ce soit le modèle, la vue, ou le contrôleur, héritent d'une classe abstraite qui leur donne une forme générale d'équerre comme suit :



Java étant un langage de programmation orienté objet, il aurait été bête de se priver de cet outil puissant permettant d'éviter de répéter du code entre les classes. Bien qu'il était possible de garder ces classes entièrement séparées, ce qui était le cas au début du développement lors des premiers prototypes présentant ces modes de jeu, une telle architecture permet d'exploiter au maximum une des principales forces de Java, en plus de donner un sens au grand nombre de classes en réduisant leur contenu à leurs spécificités par rapport au jeu général.

c) Les divers menus et options de jeu



En observant l'implémentation de différents menus, l'on peut remarquer que le style de code est susceptible de changer d'un menu à l'autre ; cela est dû au fait que différents développeurs ont travaillé séparément sur des classes distinctes, faisant que certains menus placent leurs boutons manuellement tandis que d'autres se servent d'un LayoutManager. Cela n'a aucun impact sur le bon fonctionnement de l'interface graphique, car l'on a réussi à accorder les apparences de façon à ce que les changement de style de programmation n'aient aucun impact sur la fluidité de la navigation.
Quant à l'éditeur de niveaux, il aurait été bien plus ergonomique de présenter une toile et un slider que l'utilisateur peut visuellement utiliser pour dessiner les obstacles au moment souhaité. Il y avait cependant deux obstacles majeurs à une telle fonctionnalité : l'on était dans les phases finales du développement quand la fonctionnalité a été commencée, et le manque de temps et d'expérience technique avec les toiles interactives a fait que l'on a préféré une interface moins intuitive, mais plus simple à implémenter et à tester.
Enfin, pour proposer une expérience plus proche du jeu original, l'on laisse au joueur la possibilité de désactiver certaines nouveautés de notre version, comme l'inertie du volant et l'arrière-plan mobile.

V) Conclusion

Pour résumer, Duet a été une recréation au développement mouvementé, avançant quelques fois à tâtons et non sans fautes, mais suivant fidèlement une architecture MVC, faisant usage de nombreux outils mis à disposition par Java en se servant au maximum de leur pleine puissance pour créer un jeu agréable à jouer et modulaire par nature. L'on s'est servi pour programmer ce jeu de toutes les notions abordées les semestres précédents et même plus. Cela aura été une expérience qui nous sera certainement utile à l'avenir, ne serait-ce que pour le travail d'équipe fourni uniformément par tous les membres du groupe.