Projet de Mathématiques et Physique pour le Jeu Vidéo – Phase 2

Laurianna FERRA, Clément BOURLET, Mathis RONZON, Xavier PALIX, Léna HERAU
Groupe I

Déroulement de la Phase 2

La phase 2 a commencé avec un travail de refactoring du code de la phase 1 afin de pouvoir séparer les éléments de notre moteur physique avec les éléments uniques au mini-jeu développé durant la phase 1. En parallèle, la modélisation d'un blob a été réfléchie, conçue et développée dans la partie spécialisée pour la phase 2. De même, le principe du mini-jeu afin de pouvoir tester les nouvelles fonctionnalités du moteur physique a été conçu d'un point de vue gameplay et level design. Par la suite, les collisions ont été implémentées en même temps que les objets nécessaires à la scène du mini-jeu ont été modélisés ou programmés. Enfin, le blob a été peaufiné avec l'ajout de câbles et la scène du mini-jeu a été assemblée à demi par manque de temps.

Résumé des implémentations

Parmi les nouvelles implémentations, nous avons en premier lieu celui du blob. Par volonté de respecter l'aspect 3D du moteur physique, nous avons relevé le défi de concevoir un blob complexe en 3D via un système d'octaèdre dont les faces ont été découpées récursivement en triangles plus petits. Puis en éloignant les points à une distance R du centre, nous obtenons une sphère. En ajoutant un noyau en son centre, nous obtenons notre blob. Bien évidemment, des ressorts ont été implémentés pour relier les particules entre elles et permettre que le blob soit géré comme un tout déformable. Graphiquement, l'équipe est partie pour un rendu qui explicite chaque élément composant le blob, une particule étant rendue par une sphère et un ressort par un cylindre.

Ensuite, nous avons l'implémentation des forces comme calcul de l'accélération qui va permettre d'intégrer la particule en considérant toutes les forces qui s'appliquent sur elle. Pour cela, nous avons un registre des forces manipulé par notre classe windowpart2, soit l'équivalent du classe monde pour le jeu. Il permet de mettre à jour toutes les forces recensées dans le registre sur leur particule cible. Ainsi la gestion des forces est découplée de la gestion des particules ce qui permet une réutilisation des forces pour n'importe quel contexte. Dans la même idée, le registre contient une liste vector d'une structure Registration qui conserve la particule cible et un générateur de force générique qui est une interface abstraite. Ceci permet de pouvoir affecter n'importe quel type de force tant que celle-ci hérite de l'interface abstraite. Enfin, c'est par le biais de ce générateur de force que l'on rajoute la force concernée dans la somme des forces de la particule cible, somme qui est réutilisée dans l'intégration de la particule une fois transformée dans l'accélération résultante. Or, en appelant la mise à jour des forces depuis le registre dans la classe monde, cette méthode parcourt la liste vector et appelle la mise à jour de la force du générateur de force concerné. Ce système garantit une indépendance des différentes fonctionnalités et donc une réutilisation recherchée dans la conception d'un moteur physique.

De plus, nous avons implémenté un système de collision à impulsion. Cela veut dire qu'au lieu d'enregistrer une nouvelle force pour dévier la particule en collision, nous utilisons une impulsion qui va modifier de manière instantanée la vélocité de la particule dans la direction opposée. De plus, nous prenons en considération le principe de collision élastique afin d'appliquer une perte d'énergie à la particule collisionnée. Cela revient au final à modifier directement la vélocité de la particule dans la direction opposée à la normale du point de contact tout en considérant un coefficient entre 0 et 1 qui diminue la norme de la vélocité avant la collision.

Enfin, pour élaborer un mini-jeux servant de démonstration, nous avons aussi implémenté un chrono qui permet de donner un but au joueur ainsi que des éléments graphiques comme des plateformes, des pics, un lac, des pièces à collecter pour ne pas faire mourir le blob etc.

Particularités rencontrées

Lors de la conception du blob, une question de rendu graphique s'est posée. Afin de pouvoir donner un aspect unitaire comme dans le jeu Roco Loco donné comme exemple dans l'énoncé de la phase 2, la volonté première était de réaliser uniquement un blob 2D qui réaliserait un rendu polygonale en utilisant le centre de chaque particule de la couche extérieure comme sommet. Toutefois, l'une des contraintes des phases est de concevoir en 3D. Le problème s'est alors déporté sur un système de wrapping de vertex externes, de la même manière que le logiciel Blender réalise afin de donner un rendu graphique satisfaisant depuis un système de vertex. L'idée principale était de concevoir un arbre qui donne l'ordre des vertex à considérer lors d'un dessin de polygone OpenGL via un parcours de graphe. L'idée étant conséquente, elle a été abandonnée au profit d'un rendu graphique simple : une particule par une sphère et un ressort par un cylindre.

De plus, un autre problème s'est posé lors des tests du blob : n'ayant implémenté que des ressorts, lors du déplacement du blob par le noyau, les particules se retournaient jusqu'à créer une sorte de traînée derrière le noyau. Afin de rendre les ressorts liant les particules au noyau plus rigides, nous avons remplacé ses ressorts par des câbles. Dans la même idée, afin d'éviter que le blob s'étale de tout son long sur les plateformes à cause de la gravité et de la gestion des collisions, nous avons implémenté une particule virtuelle (et donc non rendue à l'écran) à laquelle est reliée un ressort qui permet de garantir que le noyau « flotte » à l'intérieur du blob. Ainsi, ce ressort agit pareil à une béquille et force le blob à ne pas s'écraser entièrement sur la plateforme.

Par ailleurs, lors de la conception du jeu, des éléments de gameplay tels que des pics en forme de cône et des lacs circulaires ont été mis en avant. Toutefois, au moment de l'implémentation des collisions, le système de « hit box » en forme de pavé nous a poussé à revoir la forme graphique : les pics sont devenus des pyramides à base carré et le lac en piscine rectangulaire.

Cependant, pendant la conception des détections des collisions, nous avons exploré plusieurs idées pour finalement retenir une méthode généraliste permettant d'adapter la « hit box » aux transformations rotationnelles des éléments de jeu. En effet, on utilise un système de ray casting. De manière plus intuitive, un lancer de rayon nous informe sur le nombre de faces présentes devant et derrière le blob et si le blob est cerné par deux faces, on considère que le blob a collisionné un objet et que la face arrière est la face par laquelle le blob a pénétré. Ainsi, le principe de « hit box » pavé n'est plus d'actualité. Tant que le volume est concave et composé soit de triangles soit de carrés, le ray casting détecte correctement les bords de chaque élément. Par exemple, le blob peut longer les pics au lieu de rester bloqué au-dessus d'eux comme le faisait les anciens jeux NES.

Enfin, par manque de temps, le principe du mini-jeu n'a pas été entièrement implémenté. Le level design n'a pas été réalisé en accord avec la conception validée du mini-jeu et deux éléments de gameplay n'ont pas été implémentés : les pièces redonnant de la vie au blob et apparaissant de manière aléatoire dans les alentours de la scène et les pics infligeant des dégâts lors d'une collision du blob avec eux. De ce fait, comme la diminution de vie du blob (qui se fait visuellement observer par le rétrécissement du blob) au fur et à mesure que le temps passe est active afin de motiver le joueur à bouger son blob, le jeu se termine de manière fatal quelque soit les actions du joueur. Toutefois, certains éléments de gameplay ont été implémentés comme le déplacement et les sauts du blob par le contrôle du noyau par l'utilisateur ainsi que le principe de chronomètre, de fin de jeu lorsque le blob a perdu trop de vie et le score du joueur qui correspond au temps qu'il a tenu avant que son blob ne meure.