L2 : projet Info3B (S. I.), année 2022-2023

QAEZE

Noël

Groupe : MI3-ISI 2

Introduction

Le projet d'Info 3B de 2022-2023 a pour consigne d'animer, en utilisant « THREE js », une partie de jeu de bowling entre deux équipes. Les règles du jeu données dans l'énoncé diffèrent un peu des vraies. Le but de cet exercice sera de créer le jeu tout en respectant les contraintes données dans l'énoncé. Dans ce rapport, je vais vous présenter les différents points de construction de mon projet. Nous verrons tous les concepts que j'ai su appliquer dans le projet et ceux que je n'ai pas su appliquer. J'expliquerai aussi comment j'ai appliqué ces concepts. Enfin après vous avoir présenté mon projet dans sa globalité et expliqué les méthodes utilisées, je vous présenterai les différentes difficultés que j'ai rencontré durant la réalisation de mon projet et les réussites que j'ai pu accomplir.

Démarches

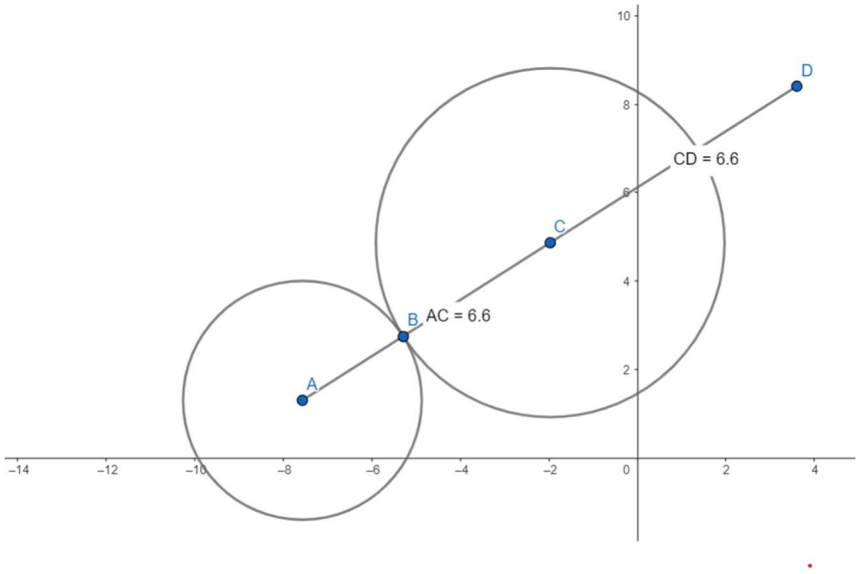
Dans cette partie je vous explique le plus brièvement possible par quel chemin j'ai commencé la création de mon projet. Premièrement j'ai utilisé comme base pour mon projet la même présentation « html » et « javascript » utiliser dans les tps. J'ai réutilisé ces tps pour les constructions de ma boule et de mes quilles. En ce qui concerne le décor et ma piste je me suis inspiré de certain jeu vidéo comme « Minecraft ». Dans un premier temps j'ai commencé par créer et introduire à ma scène tous les objets nécessaires dans le déroulement de mon jeu, c'est-à-dire la piste, mes quilles ma boule et mes courbes de Bézier. Une fois que tous mes objets furent ajoutés à ma scène, je me suis ensuite occupé des interactions entre les objets (exemple : la collision entre mes quilles). Enfin j'ai fini par création et la simulation d'une partie de bowling entre deux équipes.

Les quilles

Les quilles, sont construites à partir de trois courbes de Bézier. Vous pouvez retrouver leurs points de contrôle dans le ficher « init.js » à partir de la ligne 940 à 960.

Dans la construction de mes lathes j'ai commencé par tracer les courbes de Bézier dans un repère orthonormé (Annexe 1), puis j'ai réduit les dimensions de mes lathes en divisant les coordonnées de tous leurs points de contrôles jusqu'à trouver la taille qui me convenais le mieux. Après la construction complète des trois surfaces de révolution de ma quille, je les unis grâce à un « THREE.Group() » qui me permet de traiter les trois surfaces de révolution de ma quille comme un seul et même objet. Enfin je fini par faire une rotation de ma quille d'un angle de $\pi/2$ par rapport à l'axe des abscisses afin de redresser ma quille car la construction de cette dernière se fait dans mon plan le plan orthonormé des abscisses et des ordonnées.

Toutes les étapes de construction de ma quille sont exécutées dans ma fonction « Quille() » que l'on retrouve dans mon fichier « init.js » à partir de la ligne 933. J'utilise ensuite cette fonction dans ma fonction « init() » dans une boucle « for » pour créer toutes les quilles de mon jeu.



Le déplacement des quilles :

On observe dans le schéma ci-dessus une représentation d'une collision entre une quilles et un autre objet, c'est-à-dire soit une boule ou soit une quille. Ici le cercle de centre « C » représente une quille vue du haut dans le plan orthonormé des abscisses et des ordonnées. Le cercle de centre « A » représente le deuxième objet qui lui est en mouvement et entre en collision avec ma quille. J'ai décider afin de simuler une projection de ma quille suite à une collision, de générer dans une fonction « SegQuille() » un segment qui est calculer en fonction de la collision des deux objets. On utilisera ensuite le segment généré par la fonction pour déplacer ma quille touchée en fonction de ce segment. Sur l'image du dessus mon segment est représenté par le segment [CD]. A l'aide du segment [AC] qui est la distance entre les centres des deux objets au moment de la collision, j'en déduis les cordonnées du point D. Les calculs sont les suivant :

$$Dx = (Cx – Ax)/d + Cx$$

$$Dy = (Cy – Ay)/d + Cy$$

On remarque dans mes calculs la présence d'une division par un nombre « d », ici il s'agit d'un paramètre que l'on initialise l'or de l'appel de ma fonction. Ce paramètre changera en fonction de si ma quille est soit en collision avec ma boule ou soit avec une autre quille en déplacement. Autrement dit quand elle sera en collision avec ma boule on divisera la distance par 1 et si c'est une quille on la divisera par 2. Enfin j'utilise ce segment pour déplacer ma quille en fonction de lui. Pour simuler un déplacement de ma quille je commence par la supprimer de ma scène, puis je récupère tous les points du segment que je parcours dans un ordre croissant un à un, puis je remplace les coordonnées du centre de ma quille par ceux des points de mon segment, et à chaque changement de position je réinsère ma quille à ma scène avec ces nouvelles positions.

La collision

La collision entre deux objets est traitée par une fonction « Collision() » (début de la fonction à la ligne 594 du fichier « init.js »). Cette fonction prend en paramètre deux objets et elle vérifie si un des points qui compose le premier objet touche un de ceux du second objet. Pour détecter si une intersection entre deux points à lieu, la fonction stocke chaque objet en paramètre dans des « Box3 » qui sont des boites englobantes. Ensuite elle utilise une fonction de « Box3 » qui est « intersectbox() » et qui permet de détecter si deux boxes sont en collision ou pas. Ainsi siles deux boxes sont en collision alors les objets quelles comportent le sont aussi. La fonction finira par renvoyer un booléen « true » si elle détecte une collision et « false » sinon.

La fonction « Collision » est ensuite utilisée dans mes fonctions « Avance_Quille() » et « Avance ». Le principe de la fonction « Avance() » est de déplacer la boule par rapport à la courbe de Bézier tracée sur ma scène. Durant sont déplacement on utilise la fonction Collision() pour vérifier si à chaque déplacement la boule est en collision avec une des quilles présentes sur la piste. Enfin la fonction « Avance_Quille() » est utilisée après une collision entre une quille et soit une autre quille ou soit la boule. Cette fonction permet de déplacer la quille sur le segment donné par la fonction « Seg_Quille() » vue précédemment. Et durant ce déplace, comme avec la boule, la fonction Collision() vérifie à chaque déplacement de la quille si cette dernière est en collision avec une autre quille présente sur la piste est qui ne bouge pas.

La boule et les courbes

Il m'a été demandé dans l'énoncé du projet, d'intégrer à ma boule un courbe comme celle de tennis vu en TP. La courbe de ma boule est construite dans une fonction « Nadal() », cette fonction reprend le code de la correction du TP2 qui nous a été donné. Une fois ma courbe créée et ma boule aussi, j'opère de la même manière qu'avec mes quilles, c'est-à-dire je les unis grâce à un « Group ». Ensuite, dans le but de rendre le déplacement de la boule plus réaliste je lui applique une rotation à l'axe des abscisse avec « rotateX() » à chaque déplacement de ma boule.

Durant la construction de mes courbes de Bézier, j'ai eu beaucoup de difficulté à récupérer les tableaux de points de mes courbes. Au début de mon projet la fonction « TraceBezierQuadratique() » qui me permet de créer mes courbes se situait en dehors de ma fonction « init() ». Et de ce fait la fonction « getPoints » qui me permet de récupérer les points de ma courbe dans un tableau ne fonctionnait uniquement que dans ma fonction « TraceBezierQuadratique ». Parmi les solutions que j'avais, j'ai décidé d'intégrer la majorité de mes fonctions dans ma fonction « init() ». Cela m'a permit d'accéder plus facilement à certaines informations telle que les tableaux points et ça m'a permis de réduire mon code de moitié. Suite à cela j'ai pu utiliser les tableaux de points de mes courbes pour déplacer ma boule en fonction de ces dernières comme expliquer précédemment avec le déplacement des quilles.

Enfin il nous a été demandé dans l'énoncé que les deux courbes présentent sur la piste réalisent une jointure G^1. Pour cela j'ai intégré à mon menu GUI un calcul qui aligne constamment mon premier point de contrôle, ma jointure et mon deuxième point de contrôle entre eux. Plus précisément quand je fais varier la position de l'un de mes points de contrôle, le second voit sa position changer aussi. Je fais varier uniquement leurs positions sur l'axe des ordonnées, ce qui fait que si on ajoute une valeur « y » à l'ordonnées de l'un de mes points, on ajoute automatique l'opposé de cette valeur au second, c'est-à-

dire « -y ». Cette astuce me permet de toujours avoir deux courbes et une jointure qui respecte la définition d'une jointure vue en cours.

Conclusion

Je considère avoir respecter une majorité des contraintes données dans l'énoncé. J'ai eu énormément de mal intégrer les jointures demandées dans le projet, par exemple il me manque une jointure pour mes quilles mais à cause d'un manque de temps et un manque d'attention je n'ai pas pu traiter la question correctement. De même pour la jointure des courbes de ma piste, je considère mon code incomplet car je ne peux pas générer toutes les variations possibles que ma courbe peut prendre normalement selon moi, par exemple je ne fais varie mes points de contrôle que sur l'axe des ordonnés et en aucun cas sur celui des abscisses. Et je trouve personnellement que je n'utilise pas assez ou tout simplement pas des fonctions mathématiques utiliser en TP comme « addScaledVector() » ou encore « subVectors() » . Mise à part cela la réalisation du projet fut très instructive et surtout amusante. Je regrette de m'y être mis trop tard. Le jeu fonctionne correctement, le comptage des scores est respecté et le rendu de l'animation et assez correcte par rapport à ce qui a été demander. Pour finir vous remarquerez pendant l'exécution de mon projet que je me suis permis d'ajouter un décor fantaisiste à ma scène et une animation d'introduction. Même si cette partie du projet n'étais pas demander, je voulais tout de même vous faire part de la difficulté que j'ai eu à insérer une texture de à ma scène sans passer part un serveur. Au final la texture de lave que vous trouverez dans mon projet et générer grâce à des fonctions présentent dans mon fichier « Lave.js ». Le principe de ma méthode est d'appliquer à une matrice de 10x10 que j'ai créée, une palette de couleur que j'ai aussi choisi. En conclusion le projet s'est bien déroulé, cependant même s'il semble être correcte je pense qu'il peut encore être optimisé et améliorer et je pense sincèrement avoir manqué la partie mathématique du projet.

ANNEXE

ANNEXE 1

Les illustrations suivantes illustrent les différentes courbes de Bézier utilisées dans la construction des surfaces de révolutions de mes quilles. Pour chaque image on a :

- Le point A désigne le point de départ de ma courbe
- Le point B désigne le premier point de contrôle de ma courbe
- Le point C désigne le deuxième point de contrôle de ma courbe
- Le point D désigne le point d'arriver de ma courbe

Illustration 1 : Représente la courbe permettant de construire le haut de ma quille

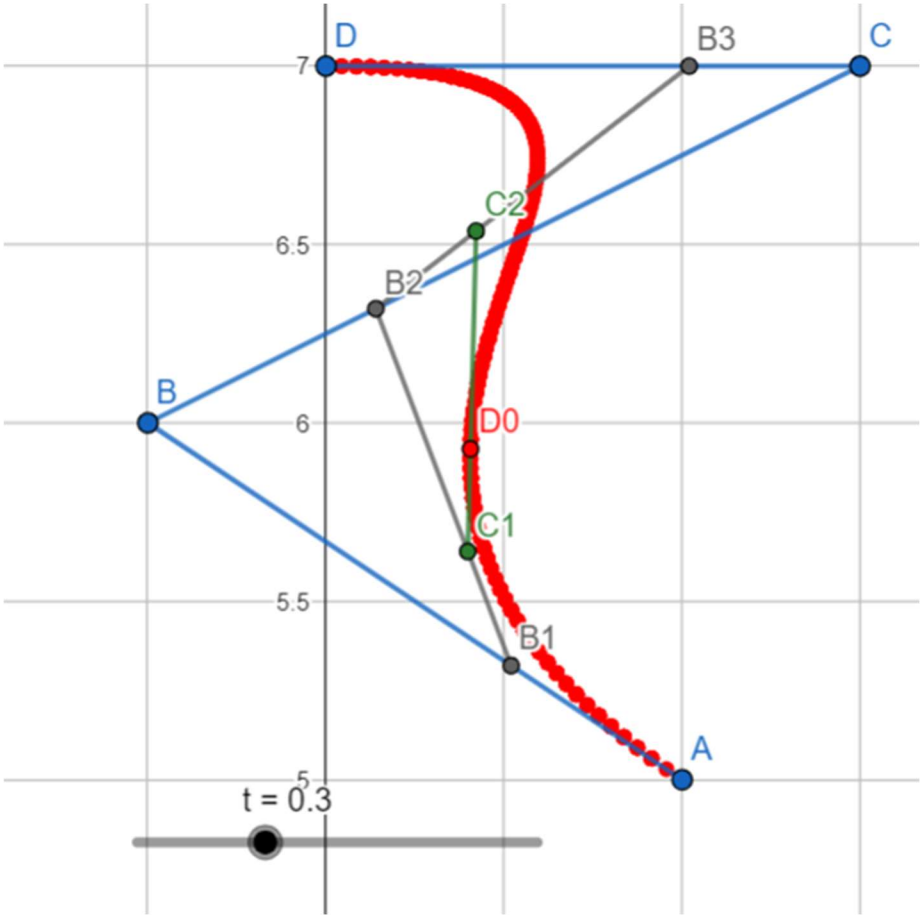


Illustration 2 : Représente la courbe permettant de construire le milieu de ma quille

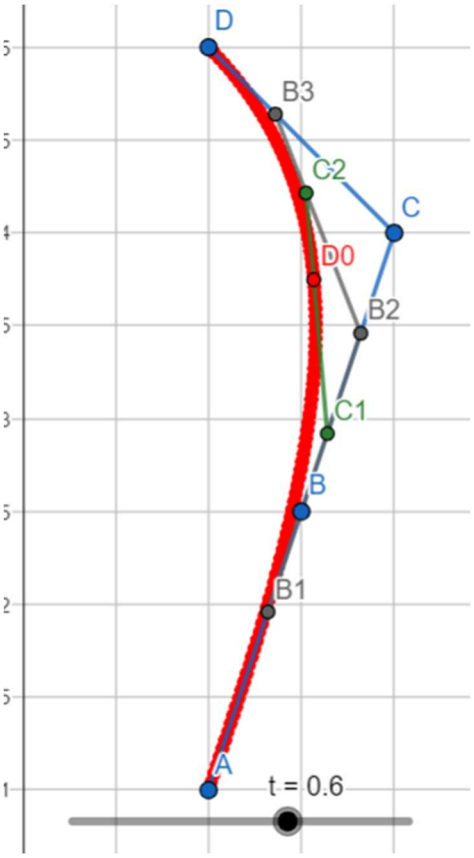


Illustration 3 : Représente la courbe permettant de construire le bas de ma quille