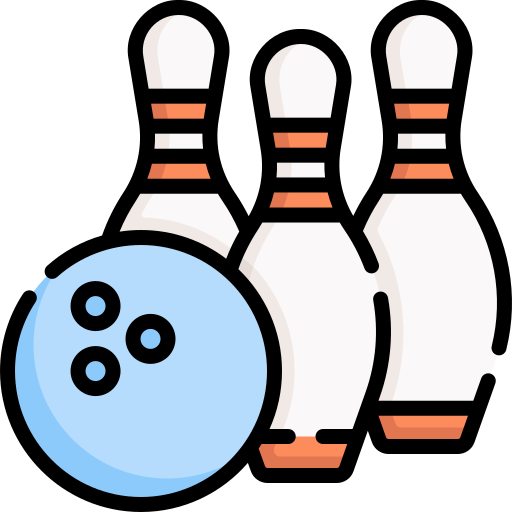
29//11/2022

**Compte rendu**

**PROJET BOWLING**

***INFO3B***



**THOMAS Allan**

**STIZ Romain**

Sommaire :

1. **Fonctionnalités**
2. **Règles**
3. **Élaboration**

**a) Piste**

**b) Boule**

**c) Caméra**

**d) GUI**

**b) Quilles**

**c) Collisions**

**e) Optimisation**

**f) Score**

**g) Reset**

Fonctionnalités :

**Dans ce jeu, le but est simple : être le vainqueur !**

**Pour cela, vous et votre équipe doivent marquer le plus de points afin de battre votre adversaire.**

**Le tout sur un jeu bien connu du grand monde, le ‘*Bowling*’.**

**“Visez, tirez, marquez”**

**Il vous sera possible de jouer deux mènes de bowling, de contrôler la trajectoire de la boule, de changer de points de vue, et d’utiliser un menu pour changer cette fameuse trajectoire !**

**Et le plus important, amusez-vous !**

**( les captures d’écran seront potentiellement plus lumineuses/claires que le rendu** **! )**

Règles :

**Le jeu se déroule en deux mènes par équipes.**

**Chaque mène est composée au plus de 2 lancers.**

**Si l’une des équipes fait un ‘strike’, soit le reversement de toutes les quilles en un seul lancer, alors elle marque 30 points.**

**Si celle-ci parvient à faire tomber les quilles en une mène, alors elle marque 15 points.**

**Sinon le nombre de quilles tombées sera le score de l’équipe.**

**Les équipes ont des couleurs différentes.**

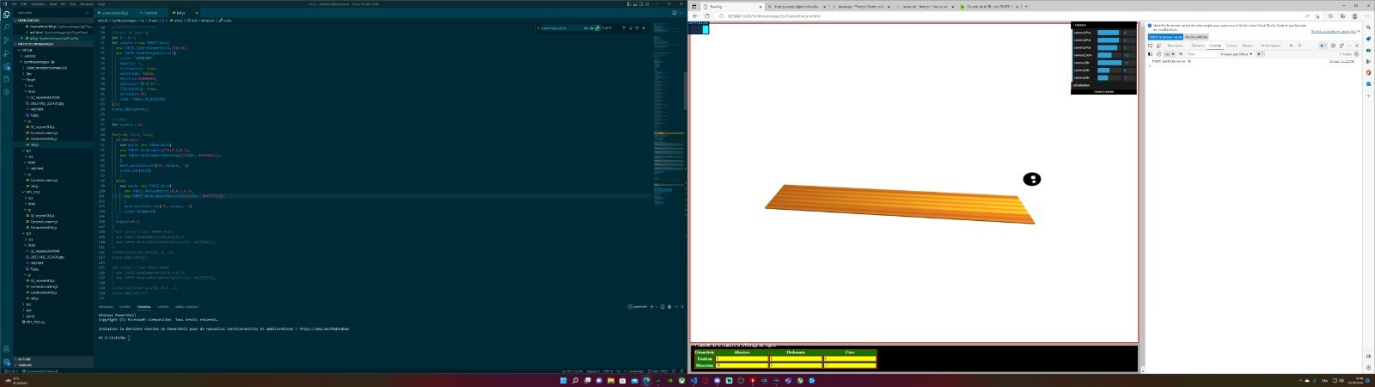
**De plus, nous avons ici ajouté une courbe sur la boule de bowling** **( une clélie ).**

Élaboration du jeu :

Piste :

**Pour commencer ce projet, nous avons tout d’abord créé une piste.**

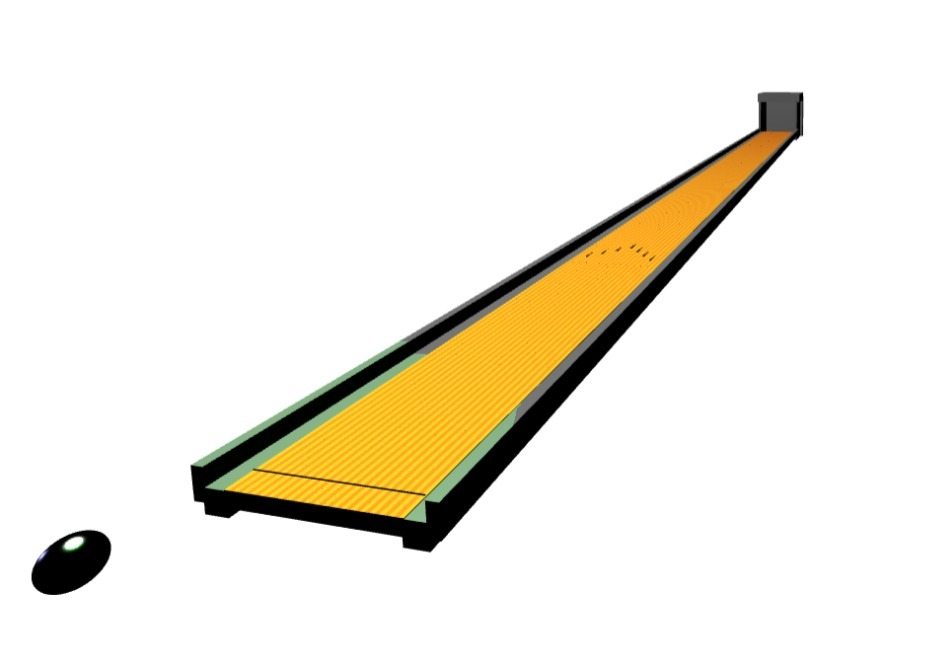
**Elle était au début composée de deux lattes ( planches ) de couleurs différentes.**



**Ensuite, nous avons ajouté des formes autour de la piste,**

**Le fond de la piste, les gouttières, les petits triangles de marquages, la ligne de faute, …**

**Ce qui au bout de 7 heures, a donné ceci :**



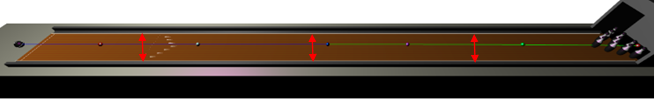
Boule :

**Une fois la piste créée, il a fallu s’attaquer au mouvement de la boule**

**Pour la boule, le tout s’est fait dans un fichier test afin de pouvoir expérimenter.**

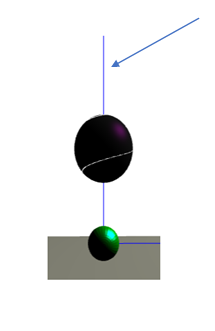
**Nous avons commencé par dessiner la boule, mais aussi une clélie sur la boule. Ensuite, nous avons commencé à faire les fonctions pour animer la boule et la clélie : on fait suivre la boule le long d’une ligne. On dit que le début de la courbe vaut 0 et la fin 1, on prend un point sur la courbe ( par exemple d’indice 0.001 ) ( ces “points” sont des proportions sur la courbe ! ), et on positionne la boule sur ce point, puis on incrémente l’indice... Tout cela nous a pris de nombreuses heures de recherche et développement ! De plus, nous avons pris le noir pour l’équipe 1 et le blanc pour l’équipe 2.**

**Nous avons ensuite transformé la ligne droite en 2 courbes de Bézier cubiques, avec une jointure G1 pour relier les deux. Les points de contrôle des courbes de Bézier ont été choisis de manière “linéaire” par rapport aux courbes :**



**Les points de contrôle sont alignés, seul le paramètre Y ( pour changer la position en largeur ) va changer grâce au gui. Il est possible de modifier la position Y de tous les points de contrôle.**

**Les deux courbes de Bézier forment une trajectoire plane non rectiligne. Mais il nous faut une trajectoire rectiligne ! C’est pour cela que l’on fait apparaitre la boule en hauteur : on la fait tomber via une ligne droite, donc rectiligne ! Le dernier point de la ligne droite est le premier point de la première courbe de Bézier.**



**Ensuite, nous avons ajouté les “frottements de l’air” : chaque instant, si la boule franchit un certain point de la ligne, alors elle va décélérer afin de créer une sorte de réalisme ! Bien que la piste soit huilée pour empêcher les frottements, nous ne sommes pas dans un exercice de physique : on ne néglige pas les frottements de l’air !**

Caméra :

**Ce n’était pas compliqué ( si nous avions su... ), mais nous voulions absolument une caméra qui suit la boule.**

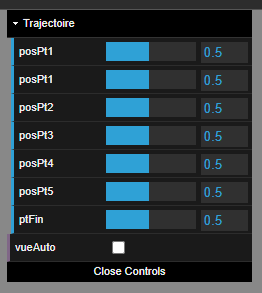
**Cela nous a pris du temps, entre les bugs de OrbitalsCamera, l’actualisation du centre de gravité…Au final, nous avons juste gardé le centre de gravité, qui est amplement suffisant. Pour cela, une recherche Google nous amène dans un forum qui parle de OrbitalsCamera, mais ne fonctionnait pas pour nous, en effet les différents centres de gravité se ‘disputaient’ entre eux...**

**Donc nous avons décidé de prendre une grosse fonction ( que nous avons mis dans controlsCamera.js ) faisant la même chose mais en plus compliqué, ET qui fonctionne très bien...**

GUI :

**Parlons maintenant de notre menu GUI : Grâce à lui, il est possible de modifier le paramètre Y de chaque point de contrôle des deux courbes de Bézier. A chaque fois qu’un tour commence, tous les paramètres du GUI sont réinitialisés :**

**c’est à dire que tous les points des courbes de Bézier sont remis à zéro, les courbes sont remises à l’origine** **( au centre de la** **piste ). Voir photo ci-après.**

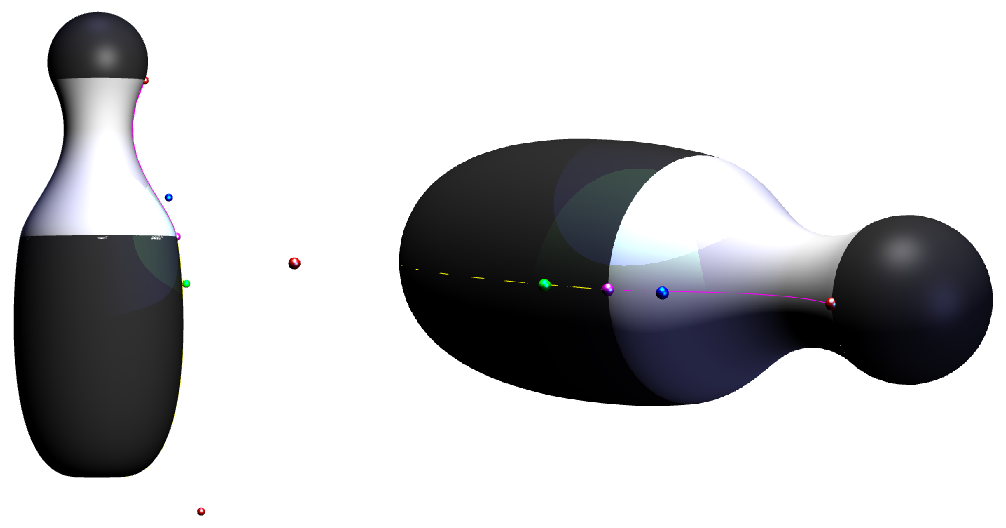


**Nous avons pensé qu’il serait plus agréable et simple pour l’utilisateur de pouvoir utiliser plusieurs points de vue. Nous avons donc créé 3 points de vue différents : le premier est celui que l’on voit en chargeant la page web, le deuxième se trouve au centre de la piste et le dernier est un point de vue proche des quilles, pour voir quelles quilles ont été renversées.**

**Il est possible, via un bouton en bas à droite, de changer entre les 3 points de vue. Mais, via le GUI, il est également possible de passer en mode automatique ! Il suffit de cocher vueAuto : cela va appeler une fonction pour automatiquement mettre à jour le point de vue.**

Quilles :

**Pour la construction des quilles, nous avons utilisé deux courbes de Bézier : une pour la lathe du bas ( noire ), et l’autre pour celle du milieu ( blanche ) ; il a donc été possible de faire une jointure G1 entre elles. Le haut de la quille est en fait une sphère tronquée, placée au-dessus de la lathe blanche.**



**Il est probable que construire les quilles a été une des choses les plus difficiles de ce projet. Nous avions commencé par construire la quille grâce à deux lathe avec des sinus, cosinus... mais pas avec des courbes de Bézier ! Nous avons donc dû tout refaire quand un camarade nous a indiqué la marche à suivre, ce qui nous a fait perdre beaucoup de temps.**

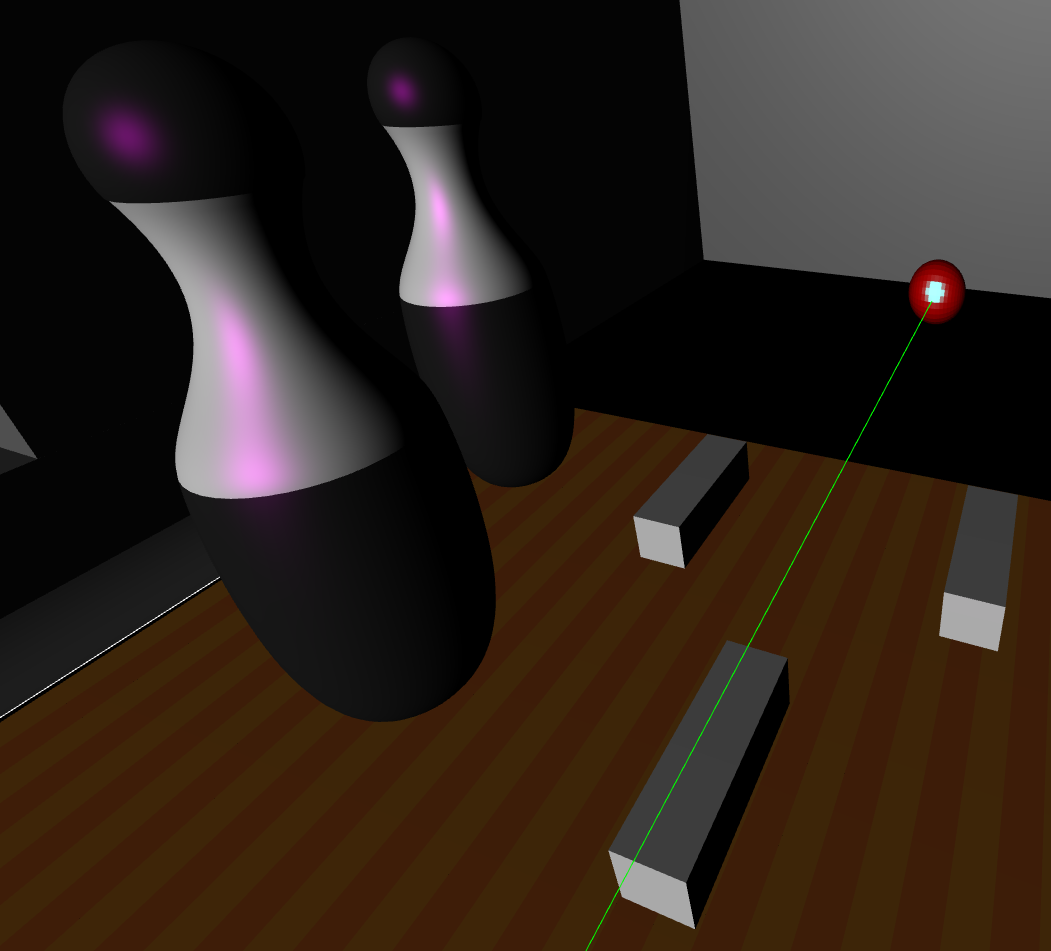
**Ensuite, au lieu de créer dix fois deux lathes plus une sphère tronquée, nous avons simplement créé un groupe ( Object3D ) avec ces trois objets géométriques, que nous avons cloné. Il suffisait ensuite de placer correctement le groupe pour faire le triangle de 10 quilles. Les placer a pris du temps...**

Collisions :

**Au départ, nous nous posions des questions sur les collisions, mais surtout, comment les coder ?**

**Eh bien... cela a été plus simple que prévu ! Il nous a suffi de créer une “boite” ( Box3 , autrement dit une hitbox ) autour des deux objets passés en paramètre de notre fonction collision, puis de regarder si les deux hitboxes se croisent ( = collision ).**

**SI la fonction collision renvoie true, on appelle notre fonction disparition, qui va remplacer la quille touchée par une petite boite blanche :**



Optimisation :

**Tout le long du projet, nous avons fait notre maximum pour optimiser le programme, pour avoir le plus de FPS possible, et pour économiser les ressources de l’ordinateur. Pour cela, au lieu de** **définir** **à chaque fois des** **matériaux, on en** **définit quelques un que l’on met dans un fichier** **à part.**

**Nous avons ajouté une condition selon laquelle on ne vérifie les collisions seulement si la boule arrive à un certain point de la piste : on économise encore ! Beaucoup de conditions ont été créées dans le but d’éviter les calculs inutiles.**

**Même chose pour les quilles, tout se fait dans un fichier a part** **( pour ne pas se** **mélanger****... ).**

**Quand se fût possible, nous avons utilisés l’opérateur ternaire, car légèrement plus rapide qu’un if/else...**

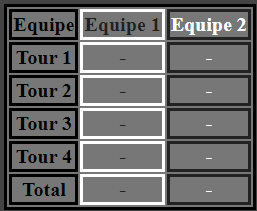
**Le code dupliqué, inutile... tout a été supprimé.**

Score :

**Si un joueur fait un strike, il gagne 30 points, un spare : 15 points, sinon il marque autant de points que de quilles tombées. A chaque fois qu’il y a collision entre la boule et une quille, on incrémente un compteur qui détermine les points du tour. Les points sont directement ajoutés dans le HTML, et non dans une variable javascript, ce qui permet un suivi en direct des points.**

**Le tableau des points possede une colonne pour l’équipe 1 et une autre pour l’équipe 2.**

**Les bordures des colonnes sont de la couleur de l’autre équipe, mais le texte est de la couleur de l’équipe.**



**Au-dessus du tableau se trouve une zone de texte, qui dès la fin du jeu affiche quelle équipe est gagnante.**



Reset :

**Il y a plusieurs reset dans notre programme : celui des quilles est une fonction qui supprime les boites rectangulaires et fait** **réapparaitre les** **quilles ; celui du GUI remet à 0** **tous les** **paramètres du GUI et reset également les courbes de** **Bézier ; celui de la boule permet de replacer la boule et les** **paramètres d’animation par** **défaut.**

**La fonction de fin remet le score à 0 et reset tous les** **paramètres en appelant** **les fonctions nécessaires. Elle permet aussi de demander au joueur** **s'il veut rejouer ou non.**

**Nous avons eu un** **problème : on utilise un prompt pour cela, mais on l’appelle** **après l’affichage du** **résultat de la partie. Cependant, le prompt s’affichait avant et bloquait l’affichage.**

**Après** **réflexion, nous avons utilisé la fonction setTimeOut(), pour ajouter un délai sur le prompt.**

C'est tout ! Amusez-vous bien !