

# Rapport

# Projet OpenGL

ROCKETLEAGUE

IMAC 1

# Sommaire

- I. Fiche Travaux
- II. Présentation du projet
- III. Difficultés rencontrées
- IV. Architecture du programme

# I. Fiche Travaux

## Éléments demandés et codés qui fonctionnent :

- Checkpoints présents sous forme de cercle
- L'aéroglesseur a une physique avec rotation, accélération et collision
- Les deux aéroglesseurs sont contrôlables aux claviers (flèches directionnelles et Z,Q,D)
- Collision avec le checkpoint visible grâce à sa disparition et à un changement des propriétés du véhicule (boost ou freeze).
- Le terrain est généré à partir d'un fichier texte où l'on définit les murs, la position des checkpoints et sa dimension.
- Le terrain possède des murs rigides faisant rebondir véhicules et ballon.

## Éléments demandés mais pas codés :

- Le jeu devant permettre de voir toute la carte, nous n'avons pas voulu donner la possibilité au joueur de zoomer sur son véhicule car cela n'apportait pas grand intérêt dans notre type de jeu. Cependant, lorsqu'il y a un but, nous zoomons sur la balle et faisons en sorte que la caméra ne montre que l'intérieur du terrain.
- Nous n'avons pas non plus mis en place de système d'indication de prochains checkpoints car ils restent visibles toute la durée de la partie et à des positions fixées par le terrain.

## Éléments non demandés et codés qui fonctionnent :

- En contrepartie, nous avons décidé d'afficher une mini-map permettant aux joueurs de voir leur position sur le terrain par rapport à celle de la balle.
- Un menu a été créé, il permet de créer une nouvelle partie ou de quitter le jeu. Le menu de pause a été également créé. Il met en pause le temps et fixe la position des véhicules et ballons.
- Du son a été rajouté avec SDL\_mixer.
- Une identité visuelle a été créée.
- Collision entre deux véhicules.

- Rajout d'une balle avec sa physique permettant de marquer des buts. Il y a cependant quelques problèmes dans les collisions véhicules - ballon. Le ballon peut traverser le véhicule.
- Système de score.
- Timers pour les parties et les checkpoints.
- Ralentis des véhicules lorsqu'un but est marqué
- Contrôle des deux aéroglisseurs avec une seule manette (1 joystick par véhicule)

### Éléments non demandés mais pas codés ou qui ne fonctionnent pas

- Mise en place d'un son lors d'un but
- Mise en place d'autres bonus
- Création de nouveaux types de terrain (ex : circulaire)



## II. Présentation du projet

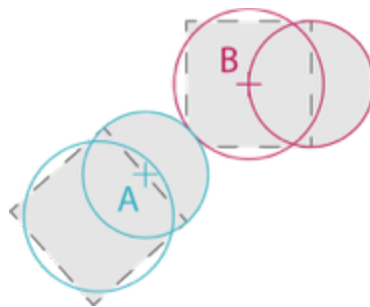
Nous avons décidé de modifier le jeu initial du projet. Ce ne sera pas une course contre la montre mais un jeu de football entre deux hovercrafts, à la manière de Rocket League, le but étant de marquer plus de buts que son adversaire dans un temps imparti. C'est donc un jeu multijoueurs jouable à 2 sur un clavier (Z-Q-D pour le joueur 1 et les flèches directionnelles pour le joueur 2).

Il a alors été nécessaire de gérer les collisions entre les véhicules, le ballon, le terrain, le but ainsi que les checkpoints. Ces derniers ont été gardés du sujet initial et correspondent à des bonus (accélération de son propre véhicule) ou des malus (« freeze » du véhicule adverse).

De plus, le jeu est compatible sur les trois systèmes d'exploitation : Linux, Windows et Mac OSX.

## III. Difficultés rencontrées

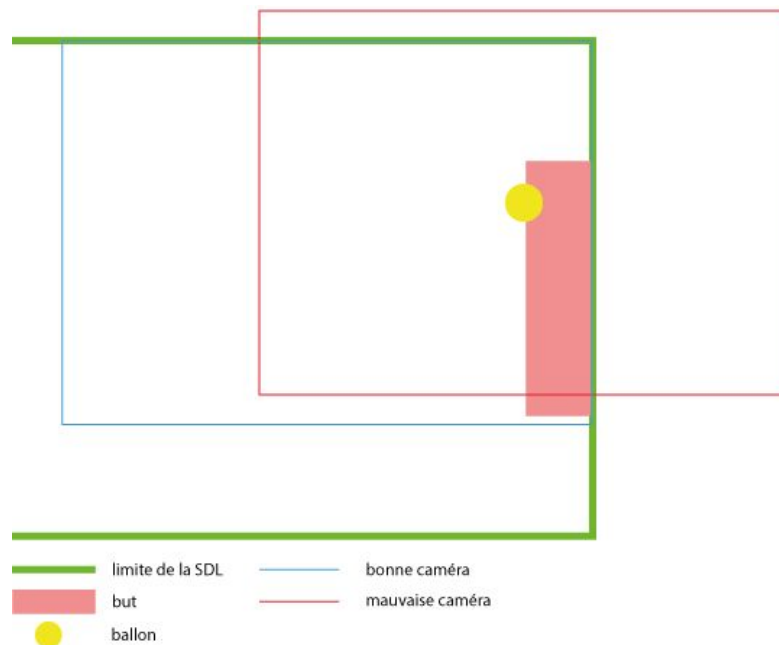
Une des difficultés principales rencontrée dans le développement du jeu a été le système de collision entre les véhicules. Nos véhicules sont composés d'un carré et d'un cercle à l'avant. Nous n'en avons jamais fait auparavant et ne voyions pas comment faire pour gérer les collisions entre un cercle et un carré par exemple. Nous avons fait différentes tentatives mais avons vite vu qu'elles étaient trop compliquées. Nous avons remarqué cependant que les collisions entre deux cercles étaient beaucoup plus simples à coder et moins gourmandes en calcul. Nous avons donc décidé de créer un faux cercle autour du carré du véhicule qui nous servirait à effectuer les collisions.



Ainsi, si la distance entre le point A et le point B (les centres des deux cercles qui vont entrer en collision) est inférieure ou égale à la somme des rayons des deux cercles, alors il y a un contact. Cette méthode permet un nombre minime de calculs. Au lieu de prendre la vraie distance entre A et B, on prend la distance au carré, cela permet de faire un calcul en moins

inutile. Pour faire la bonne comparaison, on met également les rayons au carré. Les collisions sont cependant moins précises.

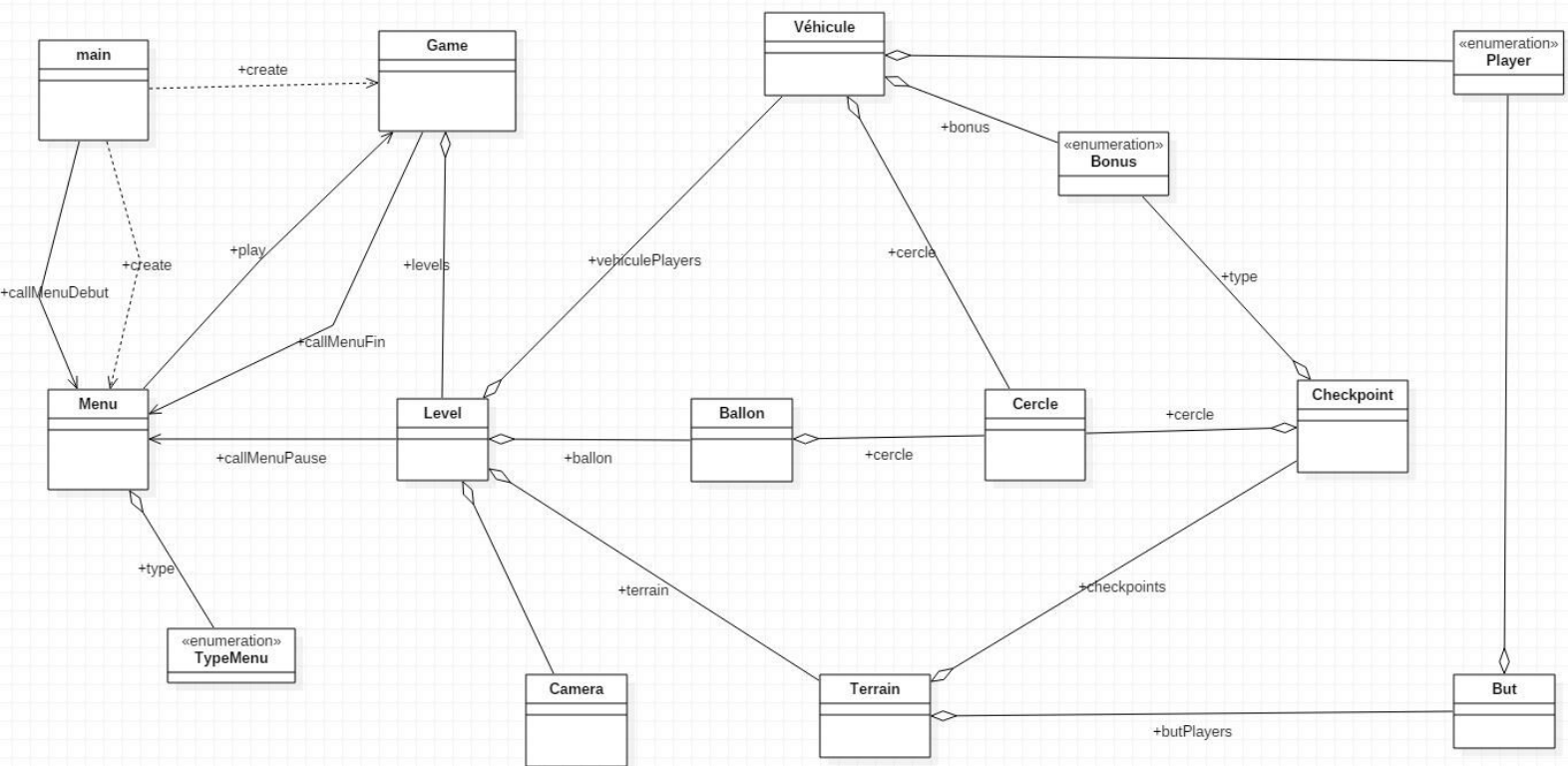
Une autre difficulté a été de faire en sorte que la caméra ne dépasse pas le terrain lorsque l'on marque un but.



Pour calculer la bonne distance, il a fallu alors faire attention au zoom qui augmentait au fur et à mesure sur le ballon ainsi qu'à la position du ballon et des extrémités de la caméra. Après différents tests, nous avons finalement trouvé la bonne formule qui permet de délimiter la position de la caméra au sein du jeu.

# IV. Architecture du programme

## Architecture principale :

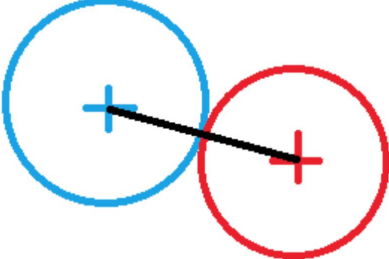




## Architecture des Véhicules : (Vehicule.c)

<pre>typedef struct {     float hauteur;     float largeur;     Cercle* cercle;     Cercle* facticeCercle;     float angle;     Point2D position;     Vector2D direction;     Vector2D vitesse;     Vector2D acceleration;     int avance, tourne;     Player player;     Bonus bonus;     int timerBonus; }Vehicule;</pre>	<p><b>Automate :</b></p> <p><b>TOURNE</b>  Renvoie 0 si aucune touche n'est pressée  Renvoie -1 si touche pour aller à gauche pressée  Renvoie 1 si touche pour aller à droite pressée</p> <p><b>AVANCE</b>  Renvoie 0 si aucune touche n'est pressée  Renvoie 1 si touche pour avancer pressée</p> <p><b>Physique du véhicule</b>  <b>Direction :</b> vecteur unitaire initialisé selon l'orientation du véhicule  <b>Rotation :</b> modifie la direction du vecteur « direction » du véhicule  <b>Accélération :</b> Vecteur colinéaire à la direction, de norme plus grande  (Vérifie la position de l'automate (Avance = 1 ou Avance = 0))  <b>Vitesse :</b> Vecteur additionnel de lui-même avec le vecteur accélération.  On lui enlève ensuite une friction pour se rapprocher du modèle physique.  <b>Position :</b> point qui permet de faire avancer de manière visuelle le véhicule suivant les paramètres de rotation, accélération et vitesse.</p>
---	---

## Architecture du Ballon : (Ballon.c)

<pre>typedef struct {     GLuint texture;     Vector2D direction;     Vector2D vitesse;     Vector2D acceleration;     Cercle* cercle; }Ballon;</pre>	<p><b>Physique du ballon</b>  Basé sur le même principe que celle du Véhicule, à l'exception près que la direction du ballon se fait en fonction des collisions avec le véhicule ou le terrain. Dès qu'une collision a lieu, un vecteur unitaire est créé entre les centres des deux cercles intersectés. (en noir sur le schéma)</p>  <p>Une accélération colinéaire à la direction est créée dès que la collision a lieu. Elle influe ensuite sur la vitesse du ballon et enfin sa position qui sont mis à jour grâce à la boucle du programme.</p>
---	---