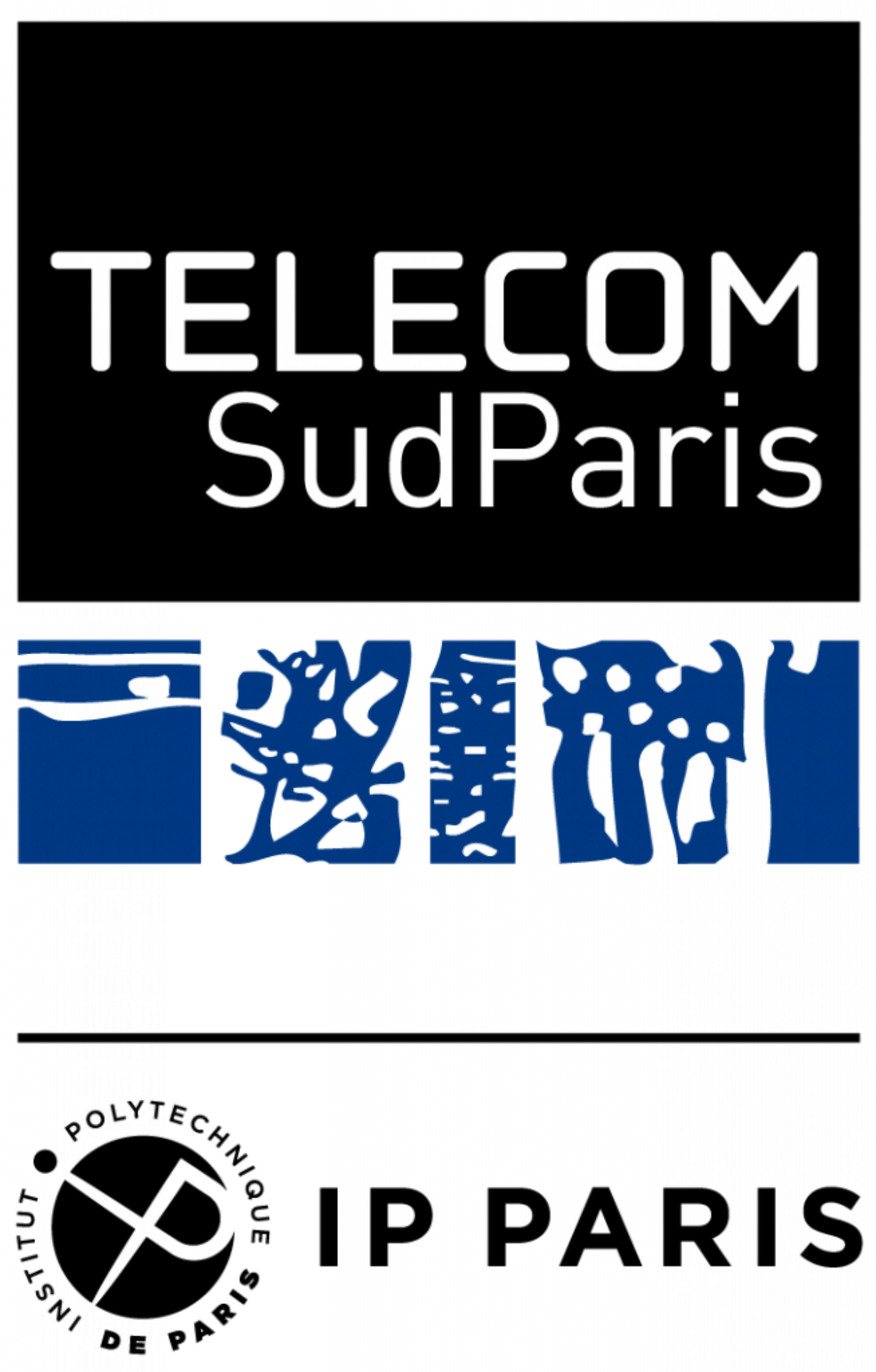
Guillaume Macquart de Terline

Pierre Chambet

**Pré-rapport PRO 3600**



Tuteur de projet : Éric Lallet

17/09/2023

Table des matières

[I) Présentation générale du projet 3](#_Toc157166096)

[1) Idée générale 3](#_Toc157166097)

[2) Objectif final 3](#_Toc157166098)

[II) Objectifs 4](#_Toc157166099)

[III) Cahier des charges 5](#_Toc157166100)

[1) Prototype 5](#_Toc157166101)

[2) Améliorations envisageables 5](#_Toc157166102)

[IV) Points d’intérêt 6](#_Toc157166103)

[V. Développement 7](#_Toc157166104)

[1) Structure initiale 7](#_Toc157166105)

[2) Structure actuelle du code 7](#_Toc157166106)

[3) Prochaines étapes du développement 8](#_Toc157166107)

[VI. Problèmes rencontrés lors du développement 9](#_Toc157166108)

# I) Présentation générale du projet

## Idée générale

L’idée de notre projet PRO 3600 est de créer un jeu vidéo de courses de voitures. Nous avons comme idée de rendu final un jeu multijoueur (2 voitures) dans lequel les deux voitures font la course sur un circuit. Nous incluons la gestion des collisions avec les murs, avec les voitures entre elles, la gestion de la vitesse, de l'accélération, du freinage. Nous incluons la gestion du meilleur temps pour connaître le gagnant.

Cependant, nous allons procéder à une première version simple du projet pour aborder les premières difficultés de façon croissante.

## Objectif final

Le projet final suppose plusieurs choses :

* Proposer une interface contenant au moins un circuit de course,
* Y intégrer au moins une voiture de course qui puisse répondre aux commandes exercées par le joueur.
* Intégrer la possibilité de jouer à deux, et/ou contre un ordinateur, selon le temps restant.

Le jeu sera en 2D, et le joueur aura une vue du dessus du circuit. La voiture qu’il dirigera pourra accélérer, freiner, tourner à gauche et à droite de façon fluide. Une gestion des collisions entre les murs et les voitures permettra éventuellement d’infliger des pénalités au(x) joueur(s) concerné(s).

# II) Objectifs

L’objectif principal de ce projet est bien sûr d’en venir à bout, afin de proposer un jeu qui soit fonctionnel.

C’est également l’occasion de nous entraîner à programmer. Nous pensons dans un premier temps coder en Java (ce choix peut changer, cf. partie IV), et en ce sens le projet est un moyen pour nous d’approfondir notre connaissance de ce langage étudié en 1A.

De plus, le projet étant un jeu vidéo en 2D, cela suppose que nous devions déterminer puis nous familiariser avec des bibliothèques adaptées.

C’est l’occasion de travailler quelques notions de physique (en fonction des bibliothèques trouvées) afin de rendre un jeu où les mouvements de la voiture sont fluides.

# III) Cahier des charges

Le jeu doit comprendre au minimum un prototype fonctionnel, qui pourra être amélioré par la suite.

## Prototype

* Circuit rond basique,
* Une voiture sur le circuit qui se déplace fluidement suivant les touches exercées par le joueur,
* Gestion des collisions (dans un premier temps avec les bords du circuit) Une collision entraîne un crash (rebonds, etc… pas encore pris en compte).

## Améliorations envisageables

* Ajout d’autres circuits plus complexes
* Ajout d’une deuxième voiture, pilotée par l’ordinateur ou un deuxième joueur (Nous traiterons le cas du deuxième joueur d’abord),
* Gestion des collisions :
* entre voitures,
* gestion des rebonds éventuels contre le bord du circuit.
* Choix des voitures par le/les joueurs, choix du circuit.

# IV) Points d’intérêt

Cette Partie est consacrée aux points importants qui nécessitent plus d'investissement. Ce sont des parties du projet que nous pensons plus complexes à réaliser, et qui peuvent potentiellement être plus longues à solutionner.

Nous pensons notamment que les collisions seront sûrement complexes à traiter. En effet, il y a deux types de collisions. Les collisions avec les murs dans un premier temps (en joueur solo), et les collisions entre les voitures (multijoueur). Pour les collisions avec les murs, l’aspect intéressant est de comprendre comment la voiture interprète l’image png du mur, et comment coder la réaction de la voiture après la collision. Existe-t-il des libraires pour traiter les collisions, faudra-t-il coder de manière brute la collision ?

Dans un second temps, il sera intéressant d'appréhender la collision entre voitures. Même si la conséquence reste la même, la nature de la collision change drastiquement. Il faudra interpréter le fait que la collision n’est pas “statique”. En effet, les limites du circuit sont connues et ne bougent pas. Ici, la collision peut survenir à tout moment par un élément extérieur au circuit. Pourrions-nous traiter de la même manière les collisions ? Faudra-t-il séparer les deux cas ?

De ce fait, nous devons trouver une bibliothèque appropriée pour coder ce jeu. A priori, pour coder un jeu vidéo, le mieux est d’opter pour un langage compilé type C++ ou Java. L’idée initiale était de coder en Java, néanmoins nous n’avons trouvé de bibliothèque particulièrement adaptée à notre cas. En revanche, nous avons trouvé deux bibliothèques Python, PyGame et NEAT, qui le sont. PyGame nous permettra de travailler sur la jouabilité en elle-même, et NEAT devrait nous permettre de coder un joueur piloté par l’ordinateur dans un second temps. Notre choix de langage se porte donc sur Python.

Ensuite, l’idée est de rendre un jeu dont la physique soit relativement réaliste. Cela suppose que les virages de la voiture, l’accélération et le freinage ne soient pas instantanés. En d’autres termes, lorsque le joueur accélère, la voiture ne doit pas passer de 0 km/h à 50 km/h instantanément mais progressivement. De même pour le freinage et les virages. Ceci suppose de se référer aux lois de la physique.

# V. Développement

## Structure initiale

Le développement s’articule en différentes parties. En effet, nous avions choisi de structurer le code selon une conception modulaire.

L’architecture retenue initialement comportait deux composantes majeures :

* Une classe Car régissant l’image de la voiture choisie ainsi que son contrôle (le pilotage de la voiture grâce au clavier),
* Un fichier principal, main.py, régissant la boucle de jeu, ainsi que l’importation et l’ajustement de l’image du circuit.

Lors de ce développement, nous avons rencontré certains problèmes, qui seront évoqués plus précisément plus tard dans ce rapport, notamment un problème lié aux virages de la voiture.

En effet, lors l’appui des touches Gauche et Droite du clavier, notre code ne permettait que de faire une translation de la voiture et non une rotation. Nous avons profité de ce problème pour restructurer notre code, en créant une nouvelle classe Driving.

## Structure actuelle du code

Désormais, notre code est donc articulé de la façon suivante :

* Une classe Car régissant le choix de l’image de la voiture, ainsi que ses propriétés :
  + Les propriétés physiques de la voiture (vecteur position, vitesse, son accélération, son rayon de braquage…)
* Le masque de la voiture (utile pour gérer les collisions de la voiture avec d’autres éléments du jeu)
* Une classe Driving, régissant la gestion des touches du clavier. Cette classe comprend deux fonctions :
  + drive(.) permettant de faire avancer et reculer la voiture (selon la touche UP ou DOWN),
  + steer(.) permettant de faire tourner la voiture (selon les touches LEFT et RIGHT)
* Un fichier main.py, qui régit la boucle de jeu principale.

En l’état actuel des choses, l’avancement est le suivant. Lorsque nous lançons le jeu, le joueur peut appuyer sur les touches UP, DOWN, LEFT, RIGHT, SPACE afin de faire bouger la voiture, ainsi que de la faire freiner. Les collisions sont gérées à une chose près : les masques de la voiture et du circuit sont générés et lorsqu’il y a recouvrement entre ceux-ci, le jeu s’arrête en raison d’une collision. Néanmoins, le masque du circuit est mal placé, et donc la collision n’a pas lieu au bon endroit. Le problème est probablement lié à l’ordre dans lequel nous créons le masque et nous mettons le circuit à l’échelle.

## Prochaines étapes du développement

Néanmoins, nous pouvons nous attendre à devoir changer quelque peu la structure du code. En effet, nous comptons ajouter une deuxième voiture, pilotée par un deuxième joueur. Mais en l’état actuel du code, nous devrons modifier/adapter la classe Driving qui est adaptée à un seul joueur.

Les collisions entre les deux voitures devraient être régies de la même façon que pour une unique voiture et le circuit.

# VI. Problèmes rencontrés lors du développement

Au début nous voulions travailler sur l'image directe de la voiture. C'est à dire effectuer des opérations comme *rotozoom* ou *transform.rotate*. Cependant, nous avions des problèmes avec ces méthodes. La voiture ne tournait que sur elle-même, ou bien celle-ci semblait avoir des "sauts" de rotations ou de déplacements. Nous avons été bloqués quelque temps sur cela. Nous avons commencé par travailler sur les vecteurs pour pallier ce problème, mais l'image associée à la voiture ne s'affichait toujours pas correctement. C'est pour cela que nous avons abandonné le travail sur les images directe de la voiture. Au lieu de cela, nous avons travaillé directement sur les vecteurs de la voiture en associant dans la classe voiture les champs *pos* pour la position et *vel* pour la vitesse. Ainsi, nous changeons la vitesse et les positions avec les vecteurs, et nous changeons le *rect* de la voiture en fonction de ces changements.