

PROGRAMMATION ORIENTÉE OBJET AVANCÉE

**Rapport de conception :
Genetic Driver**

Auteurs :
Guilhem CICHOCKI
Vincent PERA
Matthieu LE BOUCHER

Table des matières

1	Description du projet par fonctionnalités	3
1.1	Le jeu de course	3
1.1.1	Caméra	3
1.1.2	Règles	3
1.1.3	Les collisions	3
1.2	Architecture modèle-vue-contrôleur	3
1.3	Les contrôleurs	4
1.3.1	Le joueur humain	4
1.3.2	L'intelligence artificielle	4
1.4	Le réseau	4
2	Diagramme de <i>packages</i> et diagramme de classe	5
2.1	Diagramme de <i>packages</i>	5
2.2	Diagramme de classes	6

Introduction

Dans le cadre du cours de programmation orientée objet avancée, nous avons dû élaborer le dossier de conception suivant. Le logiciel à concevoir consiste dans notre cas d'un jeu de course type *Micro Machines* (voir plus loin) en réseau avec une intelligence artificielle. Dans le but de faciliter l'implémentation d'un logiciel de cette envergure, réaliser un dossier de conception est important et crucial.

Le logiciel doit permettre à des utilisateurs de se connecter à un hôte pour lancer une course. Une intelligence artificielle sera aussi développée grâce à des algorithmes génétiques et réseaux de neurones. Le joueur hôte pourra choisir de lancer l'intelligence artificielle ou de jouer.

L'objectif principal du projet est la réalisation d'un produit en appliquant de bonnes méthodes issues de la programmation orientée objet. Ainsi, certains choix concernant le jeu seront faits de façon à faciliter l'implémentation de ce dernier.

Dans la suite de ce dossier, nous présentons notre modélisation UML de l'application et décrivons le processus de conception et la réflexion que nous avons menés pour en arriver à cette réalisation finale. Notre objectif est de concevoir une application flexible. À la fin du projet, la conception et l'implémentation doivent permettre l'ajout de nouvelles fonctionnalités très facilement.

1 Description du projet par fonctionnalités

1.1 Le jeu de course

1.1.1 Caméra

Notre jeu s'inspirera principalement du jeu de course **Micro Machines** (cf. figure 1), inspiré des jouets portant le même nom datant du début des années 1990.



FIGURE 1 – Capture d'écran du premier Micro Machines

Ainsi, nous proposerons un jeu en 2D avec une caméra en vue de dessus.

1.1.2 Règles

Les joueurs connectés à la partie auront pour but de remporter la course. Ils devront donc suivre le circuit le plus rapidement possible. Pour commencer, un seul circuit sera proposé.

1.1.3 Les collisions

Pour les collisions, nous avons choisi d'utiliser la librairie de simulation physique Java *open-sourceDyn4j*. Elle nous permettra notamment de simplifier l'implémentation de la gestion des collisions. Nous pourrions donc nous concentrer sur le développement complet de l'intelligence artificielle.

1.2 Architecture modèle-vue-contrôleur





Nous justifions le choix d'un *design pattern* de type MVC par la volonté d'un découplage complet du projet entre l'interface homme-machine et le noyau dur du système. Ainsi, nous serons en mesure d'avoir un projet fonctionnel sans interface graphique, ou encore de changer notre interface graphique sans difficultés. Ainsi, la moindre

interaction de l'utilisateur avec l'interface graphique sera signalée au contrôleur, noyau dur de notre jeu. Le contrôleur répondra à la requête de l'utilisateur par l'appel de méthodes spécifiques, ou en communiquant la requête au package réseau si nécessaire.

1.3 Les contrôleurs

1.3.1 Le joueur humain

Le joueur humain est contrôlé par le clavier. L'utilisateur va appuyer sur les flèches de son clavier pour effectuer les actions suivantes :

- flèche du haut  : **accélérer** ;
- flèche du bas  : **reculer** ;
- flèche du gauche  : **tourner à gauche** ;
- flèche du droite  : **tourner à droite**.

1.3.2 L'intelligence artificielle

L'intelligence artificielle sera développée avec un algorithme génétique. Le script sera entièrement créé de notre main dans le but de découvrir le fonctionnement charnel de ces algorithmes.

Nous appliquerons les techniques de génétique à des réseaux de neurones qui constitueront notre population. De même nous avons choisi d'implémenter entièrement la logique liée aux réseaux de neurones pour en découvrir le fonctionnement précis.

Si ces choix peuvent paraître aberrants du fait de l'existence de nombreuses librairies dans ce domaine, il est très peu probable que dans notre travail futur nous ayons l'occasion de découvrir d'aussi près le fonctionnement de tels algorithmes car certaines librairies sont relativement fermées (comme *TensorFlow* de Google).

1.4 Le réseau

Le jeu sera proposé en réseau. Chaque joueur contrôlera son véhicule par son clavier. L'un des joueurs devra créer une partie — *host* — pour que les autres puissent rejoindre. Il sera notre serveur pour héberger le jeu. La voiture contrôlée par l'intelligence artificielle sera basée sur l'ordinateur qui héberge le serveur de jeu. Pour transmettre les positions des joueurs sur le réseau nous avons décidé d'envoyer des objets sérialisés¹. Ainsi nous disposons d'un paquet abstrait qui est dérivé en fonction du message que nous souhaitons envoyer à savoir une position ou une collision d'un joueur avec un mur.

Les communications sont réalisées à l'aide du protocole UDP, qui permet de s'affranchir des contraintes temporelles inhérentes au système de gestion et de récupération d'erreurs du protocole TCP, qui s'avère en pratique trop lent pour un système critique temporel tel qu'un jeu en réseau. Nous prévoyons d'implémenter un système de récupération d'erreurs beaucoup plus léger et qui sera davantage adapté à nos besoins.

Pour que la liste des joueurs de la vue soit mise à jour en même temps que la liste du modèle, nous mettrons en place un design pattern observer ; la vue observe la liste du modèle, et à chaque insertion, suppression ou modification, le modèle notifie la vue qui applique les changements en conséquence.

1. Processus qui permet de transformer un objet dans une chaîne de caractère. Le processus inverse appelé désérialisation permet de reconstruire l'objet à partir de la chaîne.

2 Diagramme de *packages* et diagramme de classe

2.1 Diagramme de *packages*

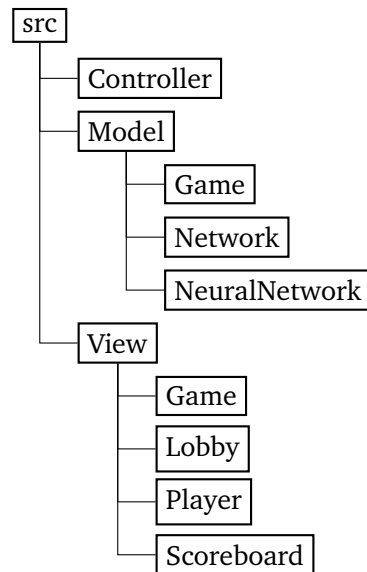


FIGURE 2 – Diagramme de *packages* de notre application.

2.2 Diagramme de classes

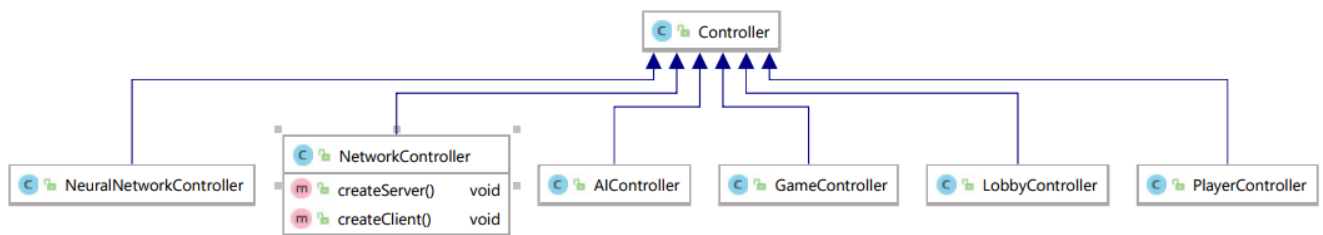


FIGURE 3 – Diagramme de classes du *package* Controller.

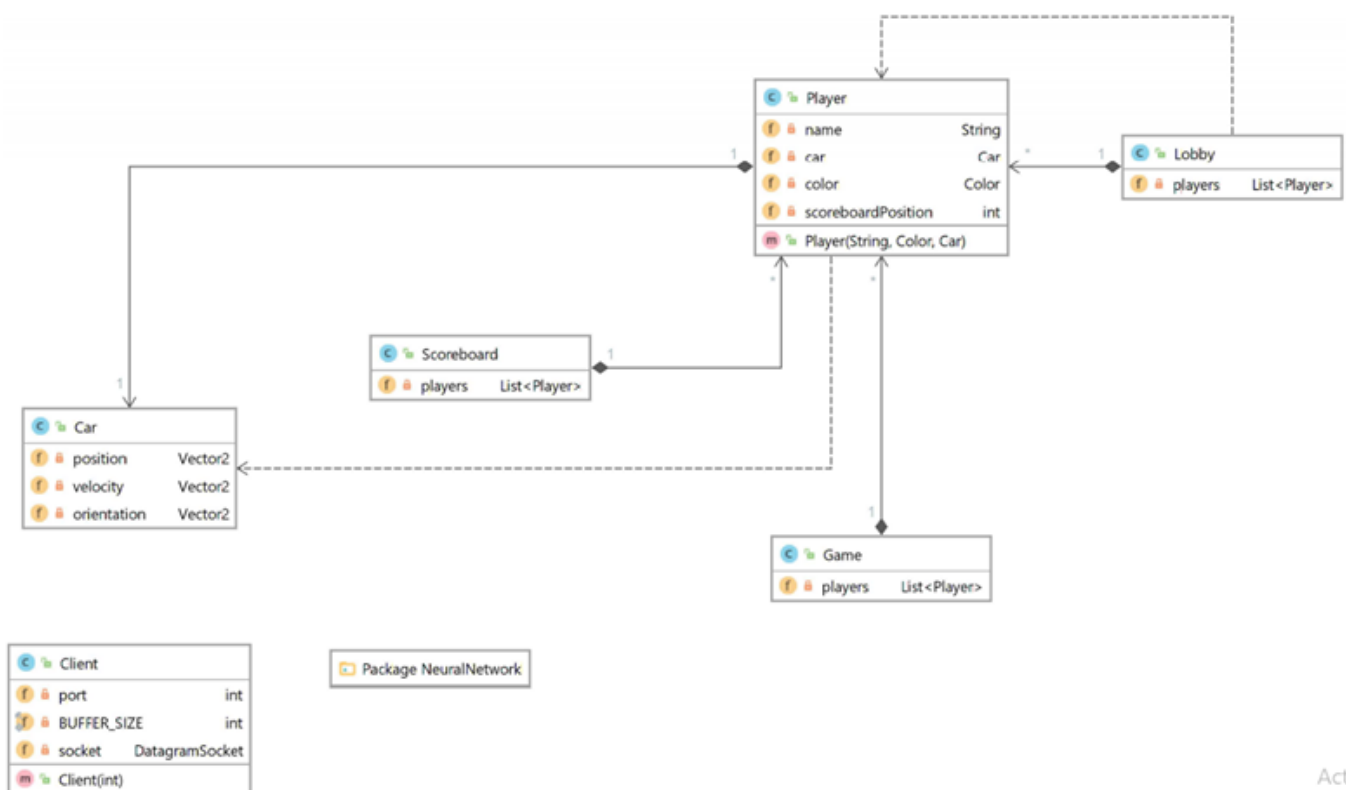


FIGURE 4 – Diagramme de classes du *package* Model.



FIGURE 5 – Diagramme de classes du *package* NeuralNetwork.

Conclusion

Si l'entière conception de *Genetic Driver* nous permettra d'implémenter relativement rapidement l'application, elle nous permet de prendre du recul sur l'implémentation particulière du réseau neuronal. Commencer par coder dans un projet massif ne nous permettrait pas d'atteindre nos objectifs de souplesse initiaux.