IN104 : Compétition d'IA pour un jeux de course 2D

Thomas Verrecchia et François Derrida

27/05/2022

1 Introduction

L'objectif de ce projet est de développer un jeux de course de vaisseau en 2D en utilisant la librairie SFML. Il s'agissait ensuite d'implémenter des IA pour permettre au joueur de faire la course contre ces dernières.
Ce document à pour objectif d'expliquer les fonctionnalités que nous avons ajoutés à cette base ainsi que les modifications de l'architecture initiale.

2 IA Visant le prochain checkpoint sans le rater

Nous avons commencer par implémenter une IA simple qui se contente, une fois un checkpoint passée, de viser le prochain en ligne droite et avec la valeur de poussée maximale. Cette IA, loin de proposer des solutions optimales, nous assure néanmoins qu'elle finira la course. Le code correspondant à été implémenter directement dans la fonction *getDecision* de la classe *pod*.

3 Menu et course contre l'IA

Nous nous sommes ensuite attaché à implémenter un écran d'accueil. Pour ce faire nous avons créer une nouvelle classe *menu* pour pouvoir afficher l'image de l'écran d'accueil tant que le joueur n'a pas cliqué sur le bouton "jouer". Une fois que le joueur à cliqué, l'interface graphique bascule sur le jeux et lance la course de vaisseau.

Deux vaisseau apparaissent à l'écran, un des deux vaisseau est contrôlé exclusivement par l'IA décrite dans la section précédente. Le second vaisseau suit les trajectoires de l'IA mais son accélération est contrôlé par le joueur grâce aux flèches. La flèche du haut multiplie par deux la vitesse initiale et la flèche du bas la divise par deux. Pour se faire, nous avons avons ajouté une variable *"p"* comme variable de *updatePhysics* pour faire passer l'information de puissance entre la classe *pod* et la classe *game*.

4 Amélioration de l'IA : Algorithme génétique

Nous avons ensuite implémenter la structure d'un algorithme génétique sans pour autant réussir à l'intégrer au reste du code. L'algorithme génétique se décompose comme suit :

- classe "**gene**" : Cette classe représente les valeur de *power* et *target* à un instant t de la course. Elle ne comporte aucune autre méthode.

- classe "**chromosome**" : Un *chromosome* est composé d'un nombre n de gènes avec n le nombre de pas de temps total nécessaire pour finir la course. On peut voir un *chromosome* comme un élément de l'espace des solutions. Il comporte plusieurs méthodes pour l'évaluer et fixer les gènes qui le composent.

- classe "**population**" : Une *population* est composée d'un ensemble de *chromosomes* et représente un sous ensemble de l'espace des solutions. Les méthodes contenues dans cette classe servent à classer les *chromosomes* du plus au moins efficace, à réaliser des croisements et des mutations entre les *chromosomes* et à passer à la génération suivante.

Nous n'avons pas réussi à intégrer l'algorithme génétique au reste du code principalement à cause de la fonction *evaluate*. Nous n'avons pas trouver de moyen viable pour déterminer la fitness d'un chromosome ce qui nous a bloqué sur le reste de l'implémentation.

5 Conclusion

Nous avons ajouté au programme de base 2 fonctionnalités en parfaite état de marche, l'IA visant le prochain checkpoint et le menu. De plus nous avons développer un algorithme génétique que nous n'avons pas réussi à intégrer au reste du programme mais dont les classes sont fonctionnels.