Rapport IA pour les jeux

Stepan TYURIN, Pierre OUVRARD, Jules CHAIDRON, Malvin CHEVALLIER

Mai 2023

1 Introduction

L'idée de notre projet est de coder une intelligence artificielle capable d'apprendre comment jouer à un jeu de course, du moins capable de terminer un tour de circuit le plus rapidement possible.

2 Environnement de développement

Pour ce projet, nous avons décidé de concevoir le jeu sous l'éditeur Godot Engine, de part sa prise en main simple et les nombreux tutoriels sur comment concevoir un jeu qui y sont intégrés. Pour ce qui est de la programmation de l'IA, nous avons utilisé le langage C# car langage natif des scripts sous Godot Engine.

3 Le jeu

Pour ce qui est du développement du jeu, nous avons repris un projet de jeu de course en voiture en vue du dessus qui existait déjà. Le travail que nous avons entreprit sur ce projet existant est d'avoir rajouté un principe de ligne d'arrivée, de point de contrôles, et de zones ralentissant la voiture. Nous y avons implémenté un chronomètre qui se lance lors du lancement du jeu et défile en continu, ce qui nous permet de mesurer le temps mis pour finir un tour de circuit.

Zone de ralentissement : Implémentées pour valoriser le fait de rester sur la route, il s'agit de zones de collisions faites pour être traversables, auquel un script est attaché. Le script en question vient modifier la vitesse de la voiture s'il détecte qu'elle se trouve en collision avec la zone, et la réinitialise lorsque la voiture en sort.

Points de contrôles : Implémentés pour forcer un joueur à faire le tour du circuit en entier avant de passer la ligne d'arrivée pour confirmer qu'un tour a

été bien fait. Tout comme les zones de ralentissement, il s'agit de zones placées de manières régulières le long du circuit, auxquelles sont attachées un script. Ce script vient détecter la collision entre la voiture et la zone, et vient rendre "Vrai" un booléen dans un tableau de booléen de taille égale au nombre de points de contrôle. Ce booléen représente le fait que le point de contrôle a bien été passé au moins une fois durant la course. Le script change également une variable qui compte le nombre de points de contrôle passés, et vient l'afficher dans la fenêtre du jeu.

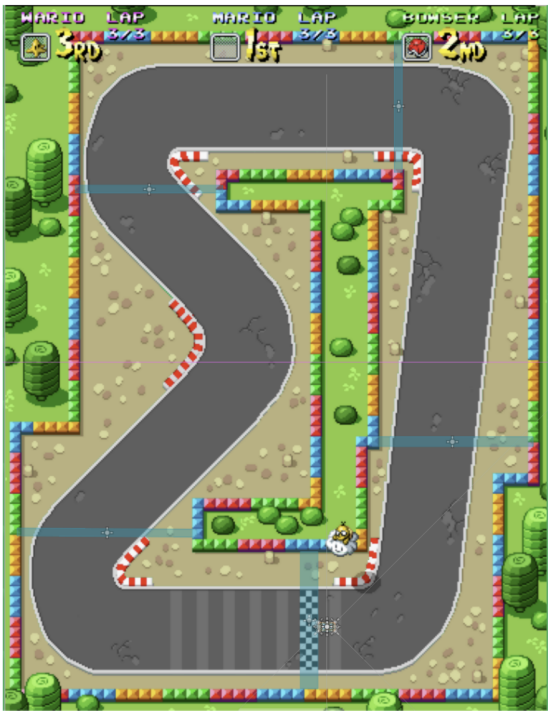

Figure 1: le circuit de course, avec les différents points de contrôle et la ligne d'arrivée (zone bleues transparentes)

Ligne d'arrivée : Ce n'est autre qu'un autre point de contrôle, mais identifié comme ligne d'arrivée par le script. Lors d'une collision entre la voiture et cette zone, le script vient la détecter et vérifie si tous les points de contrôle ont bien étés passés (si tous les booléens du tableau sont à "Vrai"). Si c'est le cas, le script enregistre dans une variable "meilleur temps" le temps mis par la voiture pour finir le circuit si celui-ci est meilleur que celui du tour précédent, ou s'il n'y a pas encore de meilleur temps. Il affiche ensuite ce meilleur temps dans la fenêtre du jeu; remet le chronomètre à zéro et réinitialise le tableau de booléen avec la valeur "Faux" dans chaque case.

La voiture : Il s'agit d'une image de voiture, vue du dessus, auquel nous avons attaché une zone de collision (hitbox), huit "tiges" servant de détecteurs de proximité ainsi qu'un script "Player.cs". Ce script gère tout ce qui aspect déplacements, ainsi que la vélocité et l'inertie de la voiture. Il gère également tout ce qui est génération d'individus pour l'apprentissage par génération, nous couvrirons ce point plus en détails plus tard.

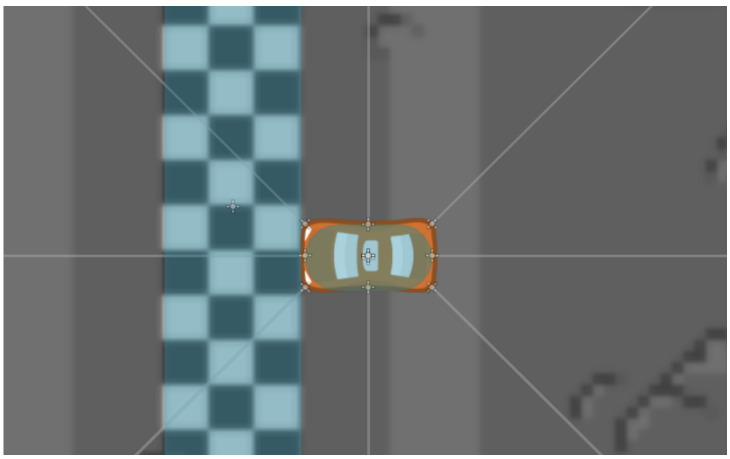

Figure 2: la voiture et ses détecteurs de collisions

4 L'IA

Nous avons longuement hésité sur le type d'IA à mettre en place, au vu de la complexité du problème nous n'arrivons pas à modéliser l'étape qui permet de donner la direction à suivre pour la voiture. Comment faire pour que notre IA s'améliore et comment calculer la fonction qui attribue le score. Nous sommes donc parties sur la base d'un algorithme génétique, cependant nous voulions prendre trop de paramètres en considération comme la distance au mur de chaque côté de la voiture ce qui nous a fait réfléchir sur la manière d'implémenter cet algorithme. Au lieu de prédire le mouvement adéquat, il suffit de créer des vecteurs de mouvements tels que pour chaque frame, la voiture avance dans la case suivante de ce vecteur. Voici les explications de notre méthode d'implémentation de l'algorithme génétique dans le fichier population.cs.

4.1 Individu

La classe Individu est imbriquée dans la classe population, elle comporte les attributs comme le score, le vecteur de déplacement ou bien la taille de ce dernier. A la création d'un nouvel Individu nous générons un nouveau vecteur aléatoire de mouvement. Les choix des mouvements ne sont pas que les 4 directions car cela limiterait la probabilité d'avoir plein de fois de suite la direction tout

droit qui est la direction majoritaire. Ainsi pour économiser du temps et des générations nous avons décomposé les types de mouvements de cette manière:

- 3 direction : top, right, left
- 1 non-direction: b (none)
- 4 variantes pour chaque direction et non-direction (1,2,4 et 8)

Les variantes augmentent le nombre d'apparitions consécutives d'une direction, si j'ai 8t, cela signifie que j'irais au top (tout droit) pendant 8 frames. De plus comme précisé plus haut nous avons mis une non-direction, cette dernière devait à la base être la direction pour reculer mais ayant remarqué que la voiture utilisait trop souvent cette action qui ne doit pas être appelé beaucoup de fois sur le circuit couplé au fait que l'on peut faire demi-tour sans celle-ci nous as convaincu pour la remplacer par une action qui ne fait rien. Cela a été bénéfique car la voiture s'arrête parfois comme pour prendre le temps de changer de direction.

4.2 Population

La classe Population contient 2 attributs importants. Le premier sert à stocker tous les individus et le deuxième stocke le meilleur individu actuel afin de ne pas avoir de régression dans notre population. Cette classe suit les principes d'un algorithme génétique, une fonction pour calculer le score (fitness) dont le détails du calcul sera précisé plus bas, une fonction de cross-over et une dernière pour faire évoluer notre population. Pour le cross-over nous sommes partis sur une sélection par tournois et l'ajout de mutation est présent dans cette fonction. Au niveau de l'évolution, nous calculons qui est le meilleur individu et ajoutons les nouveaux individus qui ont été hybridés et mutés (la mutation n'est pas systématique) auxquels on ajoute le meilleur individu afin de conserver ses gênes.

Pour le calcul de la fonction de fitness nous combinons combinons 2 tableaux, le premier est celui des poids et le second celui des valeurs de l'individu. Nous jugeons un individu sur 4 critères:

- son nombre de checkpoints parcouru
- la distance au prochain checkpoints
- le temps qu'il a mis dans une zone lente
- le nombre de collisions avec un mur

Les poids attribués sont les suivants:

- 2500 pour chaque checkpoints parcouru

- 10 pour la distance au prochain checkpoints
- -15 pour le temps qu'il à mis dans une zone lente
- -200 pour chaque mûr cogné

Au final même si ces poids semblent arbitraires, ils ont pour but de guider la voiture sur un vecteur de déplacement plus optimisé, les poids tiennent aussi compte de l'unité de chaque valeurs (le nombre de checkpoints est exprimé en unité alors que la distance est en centaine d'unités). Le poids le plus important est bien entendu le nombre de checkpoints parcouru car cela met en avant les individus qui sont allés le plus loin en respectant le circuit. De même pour la distance au prochain checkpoint. Les 2 autres poids sont des malus pour éviter que notre individu n'aille trop se taper dans les murs et qu'il évite également les zones lentes pour rester sur la route. Néanmoins cela ne l'empêche pas de passer par des zones lentes pour gagner du temps en passant par un raccourci.

4.3 Points d'amélioration et limites

Le nombre de valeurs pour calculer le score peut être augmenté, mais cela est assez difficile de trouver d'autre critère qui puisse qualifier un comportement meilleurs qu'un autre et la gestion des poids attribués doit aussi être mieux réfléchie. L'une des difficultés c'est aussi l'amélioration de la population au fil des générations. En effet le cross-over fonctionne mais nous sommes dans un jeu de course, une simple mutation pour tourner à droite et la suite du vecteur de déplacement ne s'effectuera pas dans la même direction et donc on peut avoir cette impression de non-évolution de la population, qui ne l'est pas car il faudrait entraîner sur des centaines de générations pour remarquer que la génétique à bien un impact sur la performance. Enfin nous n'avons pas réussi à entraîner plusieurs individus au même moment cela est dû aux collisions intra-individu qui aurait lieu et qui fausserait l'algorithme.

Voici le lien de notre github: https://github.com/KartIAProject/Kart_Project