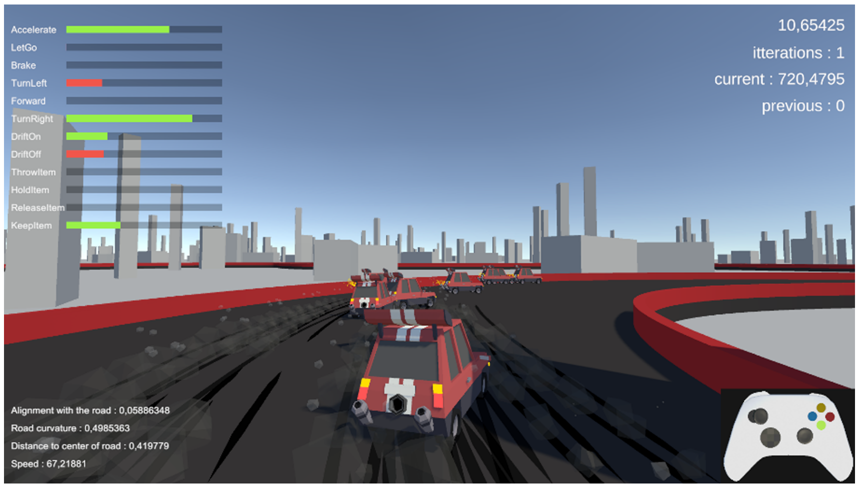
Evolutionary multi-actions UtilityAI using Bézier curves for racing games

Antoine Pavy, Quentin Pamart, Céline Tran

ESGI, Paris 2022



10,65425
itterations : 1
current : 720,4795
previous : 0

Alignment with the road : 0,05886348
Road curvature : 0,4985363
Distance to center of road : 0,419779
Speed : 67,21881

1 Introduction

This paper provides a usage of *UtiltyAI for 3D racing games.*

After one year of development for our game Kart Mania, a 3D racing game, we felt the need to add some challenge for the players. The best way was to add an AI that the player could race against. However, since we expect to let the players create their own circuit and share it to the community, we couldn't rely on classic neural network like a MLP. Indeed, allowing the player such creative possibilities, would mean that we would need a huge dataset in order to train our AI to run smoothly on every level created. Moreover, a MLP is quite opaque and doesn't allow for manual tweakings for the development team.

After some of research, we found an alternative AI algorithm called UtilityAI, mainly known for being created for the Sims. This algorithm could allow us to make our AI choose its behaviour based on the usefulness of an action at a given state in the race, no matter what the shape of the circuit is. We also noticed UtilityAI has never been used for a driving AI. That was our opportunity to create something new in the field of research on Artificial Intelligence. The main topic of our SIGGRAPH poster is about how we've adapted this algorithm to work for our game and how we've combined it with a evolutionary algorithm to improve its performance.

2 Approach

Racing games generally use classical Machine Learning Algorithm for AI training, but it takes a long time to learn, which is complicated for a short term project. UtilityAI combined with an evolutionary algorithm could be a solution for having an independent AI learning by itself. This algorithm is based on determining values of usefulness. Therefore, instead of making complex matrix products that could be quite heavy to compute, we only need to calculate the sums of values given to each possible action which makes the complexity much smaller and more suited for a real time applications. In our case, we've decided that we would need to allow the AI to pick multiple actions at the same time, which is why we've altered UtilityAI to fit our needs.

3 Implementation

3.1 Bézier Curves for Circuit

Our circuits are generated using a model based on Bézier curves. We developed a tool allowing us to quickly and efficiently edit cubic Bézier curves. With this tool, we can get local position and rotation of any object projected on the circuit. We actually generate a tangent - the local forward - and two normals for a position on the Bézier curve - the local right and the local up with cross product. With this, we can get the exact rotation of any given position. Furthermore, we generate a distance for a position, meaning we have a way to easily know the position - or rank - of a car in the race. The tool also has a way to compute a closest position on a curve from any world position, allowing to compare for example the car rotation with the road direction (using normals and tangent of the computed position).

3.2 Curvature of the Road

The main data used for our AI's behaviour is the curvature of the road. Indeed, it needs to know in which direction the road is going in order to turn properly.

In order to get this data, we need to get the position of the vehicle on the curve. The AI looks for the orientation of the circuit a certain amount of meters ahead, in order to anticipate the curve and turn a little bit earlier. After getting this position, we project it a few meters ahead once more in order to get two separate positions on the circuit.

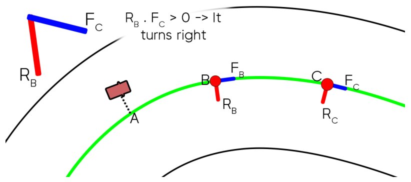


Figure 1: Two points on the circuit used to get the curvature

The two red points represents the previously described positions on curve, and their arrows are their forward vector. By doing the dot product between the right vector of the first point, and the forward vector of the second point, we obtain a value between [-1;1], negative when the road is turning left, positive when it's turning to the right.

4 Evaluating Functions

In order to make our behaviour tend to a certain value based on the data that it receives, we are using custom functions to make the AI's decisions more precise. An action could tend to be useful only when a given value is very high (exponential function) or even when it is still small (logarithmic function). Using these with a coefficient is the core mechanism of UtilityAI.

5 Improve with Evolutionary Algorithm

Even if the AI can already do a decent race on its own, in order to get better results, you need to use a method to make your parameters more precise. In our case, we used an evolutionary algorithm to tweak the coefficients of our UtilityAI. As the AI is already partially trained by us, the learning phase isn't too long, and results are visible quickly.

In order to link UtilityAI with the evolutionary algorithm, we created a genome. This genome contains the coefficients applied to all the data of each action.

Then, the genome is modified by crossing the genomes of two parents picked among the previous AIs that performed the best and then mutated, in order to create a brand new population of AIs, all different from each other. In our case, we rank each generation of a population by sorting them by the distance it has traveled.

6 Future works

While our algorithm is quite robust and effective, we have some improvements tracks we will work on:

- Put our evaluation functions or other parameters (distance check for example as described in 3.2) in the genome, in order to let the evolutionary algorithm tweak it for us.
- As our algorithm is made with Unity, we have some ideas for improving performances by using their Jobs System and Burst Compiler. In fact our simulation for evolutionary algorithm is quite slow, with a quite small amount of karts, so this would allow us to simulate faster and with a bigger population to have a more homogeneous result.
- Currently our IAs try to follow the center of the road as much as possible. An improvement would be to try to stay on the apex of the turn to have the shortest possible path.

References

[Nor99] John Norstad. "An introduction to utility theory". In: *Unpublished manuscript at http://homepage. mac. com/j. norstad* (1999).

[WL12] Jung-Ying Wang and Yong-Bin Lin. "Game ai: Simulating car racing game by applying pathfinding algorithms". In: *International Journal of Machine Learning and Computing* 2.1 (2012), p. 13.

[TC14] Blair Peter Trusler and Christopher Child. "Implementing Racing AI using Q-Learning and Steering Behaviours". In: *Conference on Simulation and AI in Computer Games*. Vol. 11. 2014, pp. 09–2014.