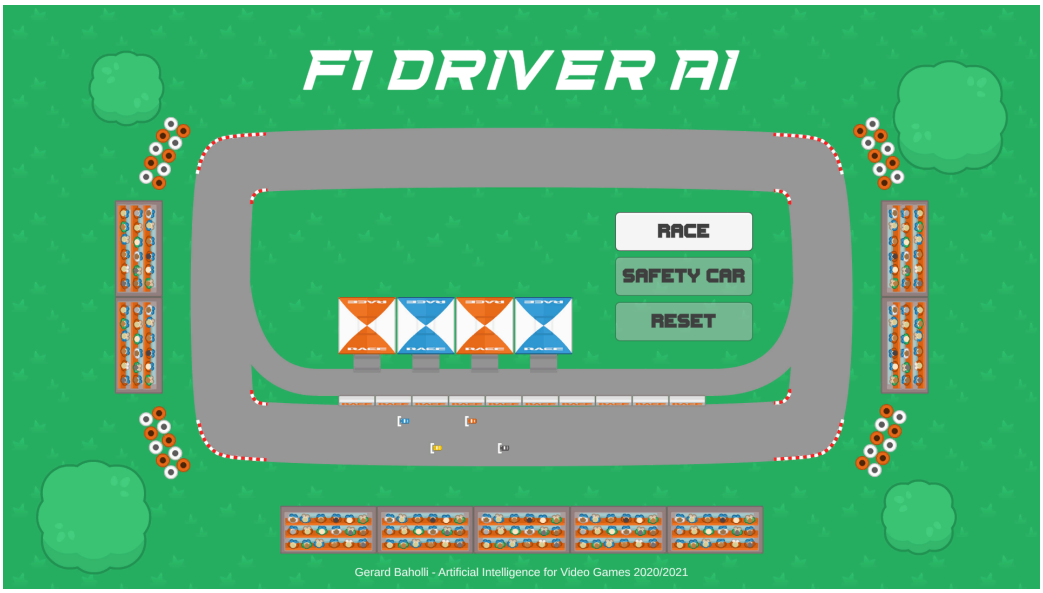
UNIVERSITÀ
DEGLI STUDI
DI MILANO

PON9
Playlab fOr inNovation in Games

Formula 1 driver AI

Gerard Baholli

September 2021



1 Introduction

The following project was carried out for the Artificial Intelligence for Video Games course, in the academic year 2020/2021. The project aims to implement the AI of a Formula 1 driver. The driver is governed by a Final State Machine for decisions to be made on the track. Behavior Trees are associated with each state of the FSM that will do the action based on the cars around him, the state of the tires, and the conditions of the track. The purpose of this document is to explain how the agent's AI was created, the choices made, the problems encountered and the techniques used. This project was made in Unity (2021.1.15f1). The link for the project repository: `https://github.com/gerardbaholli/ai4vg-project`

2 Overview

We want to create the AI for an F1 driver who can move along the track independently. The movement of the driver is a single individual. It must reach a degree of realism that allows the player to play against it without fear of being hit (unless in special cases of course). The AI must be able to move and/or overtake taking into account the cars around it, so it is influenced by the environment. No pathfinding is required. Physical simulation is required, with a medium/high degree of realism.

This project can be the basis for the implementation of racing games in which you want to be able to insert bots to play against. It can also be used in management games, a concrete example of this type is Motorsport Manager. In the first case, the target audience can range from younger to older audiences. The second will have more impact on a very specific kind of audience. Also, if you try to add a more refined level of simulation the target audience will shift from casual to hardcore gamers.

3 AI Design

In this section we will talk about how the AI was designed and what actions were implemented

3.1 Finite State Machine

Starting to design the driver's behavior, the first step we wanted to face is the Finite State Machine. In this case it was used as a way to expresses the states in which the driver can be found during the race. The high-level idea is based on three simple states: stop, race, and pitstop.

Referring to Figure 1, we have designed the FSM with the following conditions and the following groups of actions:

- The first state is the stop state, where the driver will simply stay stationary without exerting any type of acceleration. This state was thought of as a starting state. Once the traffic light is turned on (started directly from the user with the race button) from this state the car will pass to the next one (race state).
- The second state is the race state, where the driver will drive according to the track and according to the cars that he will have around him. Remaining in this state will cause the condition of the tires to deteriorate over time. Once, a certain threshold value for the condition of the tires has been reached, the driver will be called to the pits to change the tires, from here there will be the transition to the pitstop state.
- The third state is the pitstop state, where the driver will follow the path that will lead him to the pits by managing things like the entry speed and the release speed. In this state, there will be the tire change of the car, which will bring them back to perfect condition.

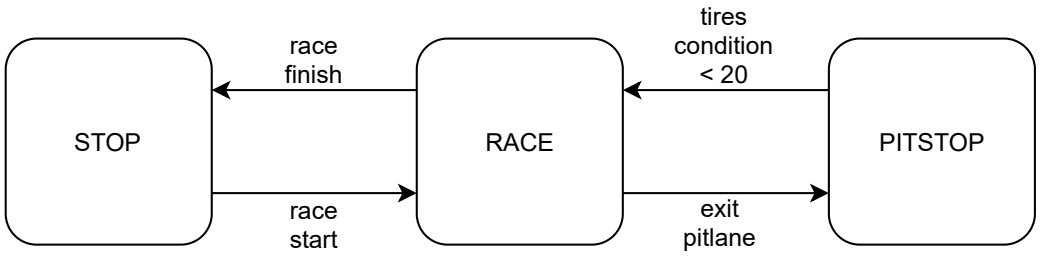


Figure 1: Final State Machine of the car.

3.2 Behavior Tree

Subsequently, Behavior Trees were used to manage the driver's behaviors within each state of the Finite State Machine. In this way, it is possible to keep in memory the current state of the pilot thanks to the FSM.

3.2.1 Behavior Tree of the race state

The behavior of the car within the race state is based on the path following of the waypoints positioned along the track (explained later in Section 4.1). For this reason, the Race action is the first action of the sequence (see Figure 2). A new sequence is then started to check whether the safety car is on the track or not (in the implementation of the project there is no safety car that physically enters the track, for this reason, it could be seen more like a virtual safety car). If returns true simply lowers the speed, in this way the cars will stand behind each other and follow the path in line. Finally, there is a final sequence that checks whether the safety car is still in operation. In this case, if the safety car is not running, the speed is set to the initial values. The purpose of the latter sub-tree is to reset the values that were imposed by the presence of the safety car, if this was never present during the simulation, the action would still be performed but without any consequence.

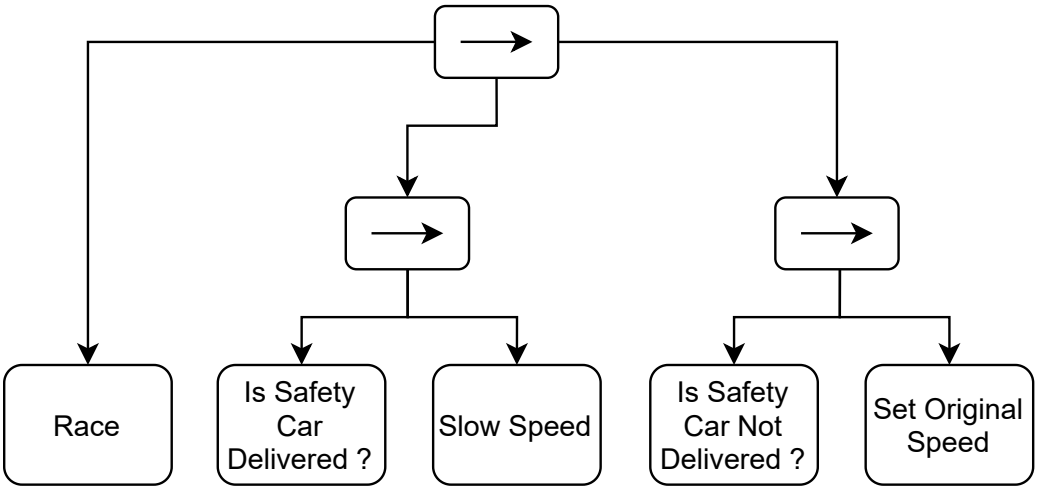


Figure 2: Behavior Tree of the race state.

3.2.2 Behavior Tree of the pitstop state

The BT of the pitstop state is more complex. We start the BT sequence with a condition that is the Need To Pit which will return true if the tires' conditions are equal to or less than 20%. Immediately after the condition, we will face two possible situations. The first is the one in which the car is near the pitlane, in this case, the car will follow the entry waypoint of the pitlane to enter and head towards the box. The second case is when the car has just passed the waypoint that leads to the entrance to the pitlane. In this case, the problem lies in the fact that he will have to continue his race always at the highest levels taking into consideration the fact that he will have to enter it later when he arrives near the pitstop. *EnterPitlane* action takes care of this, its purpose is to follow the race waypoints until the car is near the pitlane. Then we have another situation, namely the one in which the car will have to arrive at the garage for the tire change. In this case, he will have to follow all the waypoints of the pitlane with the same principle as the race waypoints, but at a lower running speed. Once near the garage, the car will make a sudden movement towards it via *TeleportToBox* and then the condition of its tires will be restored via the *ChangeTires* action. The last sub-tree is responsible for sending the car back to the track by requiring it to follow the pitstop waypoints as long as it is inside the pitlane. To help us in this part, we have added two game objects that are responsible for detecting

the moment in which the car is inside and outside the pitlane.

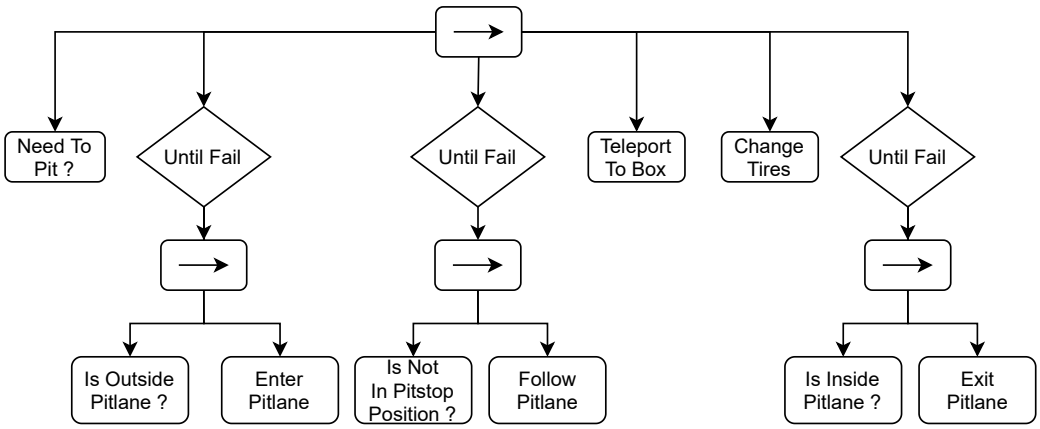


Figure 3: Behavior Tree of the pitstop state.

4 Implementation

The project was conceived as a 2D representation with an overhead view of the track and the cars. There are 4 machines in total and each of them uses the same AI, the only difference is in the statistics (tire consumption impact and maximum speed) in the color and in the box assigned for the tire change.

The FSM will take care of the decision making and the actions will be managed and represented by the leaves of the BT. The use of BTs integrated within the states of an FSM greatly simplifies the design work done to define the AI. The states of the FSM indicate at a high level what the car is doing at a given moment. In this way, it is possible to keep in memory the current status of the driver and it is possible to represent him through a BT. The BT is not used to make decisions but for the execution of actions. An example can be the transition from the race state to the pitstop state. Being near the entrance to the pitlane at the start of the condition is an action, having just passed it, is another. There is no type of group strategy even if it could be added as a future implementation if we want to implement teams.

4.1 Path Following

The basic idea for the implementation of the movement lies in the fact that the cars, to move along the track, use points called waypoints. The waypoints are joined to each other forming a ring of interconnected points (in reference the Figure 4). The machine tries to follow the path formed by the union of these points. Once you get close to a waypoint (the level of proximity depends on the track and the effect you want on the steering of the car) the car will change direction towards the next waypoint. These waypoints are divided into two groups. In Figure 4, you can see the green waypoints, which represent the points that the car will follow during the race, and the yellow waypoints which are the points that the car will follow during the pitstop phase.

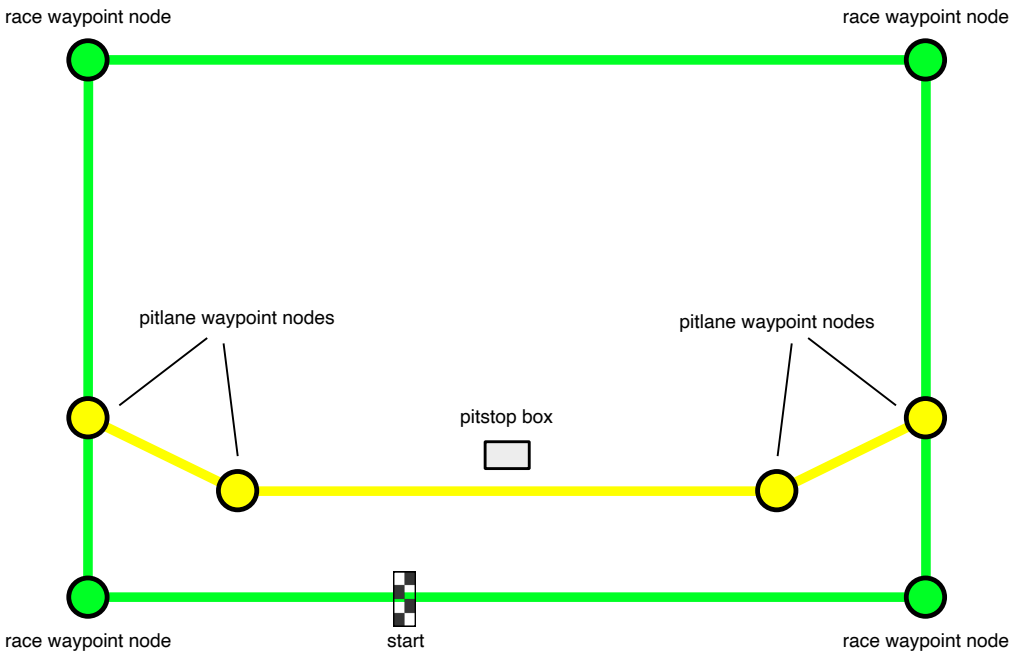


Figure 4: Waypoint scheme.

4.2 Scene

CarAI For the representation of the car has been chosen a sprite depicting it from a top view. Each car is associated with a Rigidbody2D in which the gravity scale is set to zero, to avoid gravity being applied to the y axis and is managed through the CarController script. Then several scripts are associated which handle different things. This choice was made to apply a separation of duties to more easily locate the components associated with the machine game object. In CarFSM there is the logic that manages the states of the Finite State Machine and in PitstopBT and RaceBT the two Behavior Trees are associated with two states of the FSM. A CarAIHandler script has been associated with the management of some actions that the AI must perform. Finally, the CarStatus script has been associated, which maintains status information relating to the car.

Figure 5: CarAI sprites.

AI Path This game object takes care of separating waypoints into two distinct groups: race nodes and pitstop nodes. Each of these groups has been associated with the DrawPathHandler script which takes care of showing the connection that each node has with its next. The main purpose of this script is to simplify the creation of the project by showing some Gizmos.

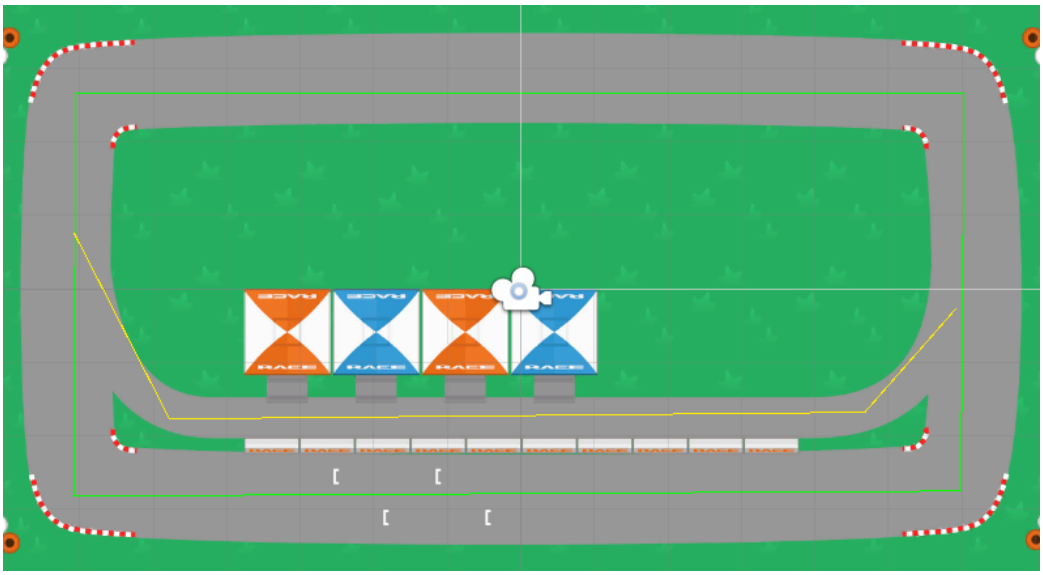


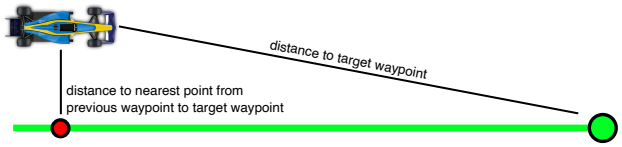
Figure 6: Path created from the waypoints.

Box A box (game object) has been associated with each car in which it will enter to change tires.

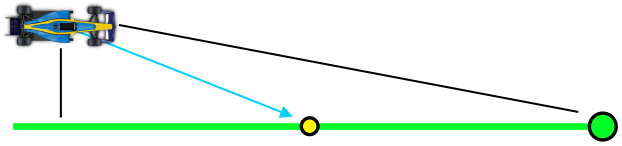
Pitstop Entrance and Exit Two game objects were placed respectively at the beginning and the end of the pitlane. They work as sensors that trigger their effect when a car enters its collider. Their purpose is to change the state of the car from on the track to in the pitlane and vice versa.

4.3 Scripts

CarAIHandler This script is used to apply all the movement actions associated with our AI. The *FollowRaceWaypoints* and *FollowPitlaneWaypoints* methods are the basis of the application. They are placed side by side with the *FindClosestRaceWaypoint* and *FindClosestPitlaneWaypoint* methods which take care of finding the waypoint closest to the car. The follow methods deal with obtaining the target position on which the movement will subsequently be applied. The choice has been made to implement path following behavior which means that the direction of movement of the machine will be conditioned by the line that connects the current waypoint to the previous waypoint (see Figure 7a). This choice was made to prevent the car, once off the track, from traveling through the movement that brings it to the current waypoint, staying out of it for most of the time. It is known that there may be terrain that slows down the movement of the car once off the track. In this way, the car will try to move in the direction of the runway (to ensure that it returns as soon as possible) and also of the current waypoint (to ensure that it still meets the objective to be achieved)(see Figure 7b).



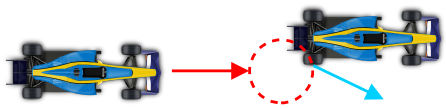
(a) The left vector is found with the method *FindNearestPointOnLine* and the right vector indicates the distance to the target position of the car.



(b) In blue the vector to reach the target and to stay near the line (is computed from the previous two).

Figure 7: Path following implementation.

This script also includes controls to manage the avoidance component of the other machines. *TurnAroungTarget* method indicates how much the car must steer based on the position of the target and whether or not it must avoid other machines. *ApplyMovement* takes care of applying the inputVector generated through TurnTowardTarget and ApplyThrottleOrBrake. *ApplyThrottleOrBrake* apply forward acceleration based on how much the car wants to turn, if it is a short turn the car will apply less speed forward. *IsCarInFrontOfAICar* launch a CircleCast that hits the car in front (see Figure 8a, LayerMask is used), if it collides with something it returns the position, and the right vector of the car hitten, if not, return false. Thanks to that, if there is a car in front, *AvoidCars* calculate the vector to avoid the car and still aim to reach the waypoint (see Figure 8b).



(a) Circle cast in red and reflect vector computed throughth the right vector of the car in blue.

(b) The new vector to reach the target and to avoid the car in front.

Figure 8: Avoid Car method explanation.

CarFSM Within this script, there is all the logic for the execution of the Finite State Machine, including the conditions that lead from one state to another and the actions to start the Behavior Tree.

RaceBT and PitstopBT All the logic of the respective Behavior Tree is contained within these two scripts.

DrawPathHandler The purpose of this script is to draw a line from one waypoint to the next. The reason why it was used is to simplify the implementation of the AI by also facilitating the debugging phase.

5 Final consideration and future upgrades

The actual entry onto the track of the safety car with the cars lining up behind it waiting for the restart. In the project, it is seen as an action of the race state, but in the future, it could be separated into a state of its own. The addition of the red flag and yellow flag status based on the sectors of the track is another possible future implementation. For a more refined implementation of the movement, the TeleportBox behavior could be transformed into an arrive behavior in such a way as to make the car enter the pits in a precise and slower way. Switching to the implementation of path following through splines, following a path through spline manages to make a greater degree of realism, but especially it makes the design phase of the ideal path easier than the approach that was used for the project (more simple to implement but more difficult in the design phase). Bringing the teams into the races leaves room for the AI strategy.

List of Figures