**P1-Steering Behaviors in F-1 Spirit**

The goal of Project 1 is that you learn how to implement **Steering Behaviors**, which are very commonly used in computer games for controling cars, motorbikes, space-ships, etc.  
  
In this project you will work with [F-1 Spirit](http://www.braingames.getput.com/f1spirit/), an indie car-racing game. In this website you can find a specially modified version of the game to complete this project.  
  
**Specific Project Tasks:**

* Set up F-1 Spirit in your favorite C++ IDE (X-Code, Visual Studio, CodeLite, etc.), or just set it up so you can compile it and run it if you want to use the GNU tool-chain. In order to compile F-1 Spirit, you will need to download a collection of libraries (SDL, SDL\_mixer, SDL\_image, etc.). You can get all of them from the SDL website [here](http://www.libsdl.org/index.php) (<http://www.libsdl.org/index.php> ) (download the 1.2 version, not the 2.x version). Please do not hesitate to contact the instructor if you need help with this!
* To test that you have it properly setup, compile and run. If a race starts, with lots of cars, and the main one (the one the camera is following) has three lines coming out of it (white, blue and green), then you are good. Everything is working.
* Read the documentation at the bottom of this page to familiarize yourself with the code. In this project you will just have to create a new function that overrides the "PlayerCCar::PlayerAICS480(...)" function. You can just comment out the example one, or delete it altogether.
* Implement a controller that uses steering behaviors for controlling the car and complete at least a lap without crashing. The example controller does not use steering behaviors. Replace it with one that: uses "seek" to stay in the road, and also uses ray casts to detect if it's going to collide with other vehicles or with the side of the road, and avoids collisions accordingly (similarly as with the Java version of this project)

**Notes:**  
F-1 Spirit is a real game (you can download the complete game from [website](http://www.braingames.getput.com/f1spirit/)  if you want to try it out). Therefore, it has lots of complicated details that you will have to work around.  
  
The source code distributed to you is prepared to stat in a "stock car" race, which is the easiest and slowest in the game. If you want to try your controller in other races, just go to the line 217 of the file "sources/F1SpiritApp.cpp". The line should look like this:  
  
menu\_selected\_track = 0;  
  
If you change the value to another number (valid numbers are from 0 to 20, races 21,22,23 and 24 are special tracks, and should not be used, numbers above 24 are not allowed)  
  
The controller I give you as an example seems to work very good. But it is actually very bad, it can win the "stock car" race, but it wont even pass the first curve in the "rally race" (menu\_selected\_track = 1), and will crash continuously in any of the "Formula 1" races (menu\_selected\_track = 5 or higher). This is because the controller doesn't take into account the other cars in the road (doesn't know how to overtake), nor takes into account the speed at which different curves should be attempted.  
  
I have included a special controller, called "PlayerAICS480\_keyboard". If you replace the regular controller by this, you will be able to control the car with the keyboard (arrow keys to turn and shift gears, space to accelerate and 'm' to brake).  
  
I highly recommend reading and understanding the "PlayerCCar::PlayerAICS480(...)" method I provided, since designed it to contain most of the calls that you will need to complete your project. Things to observe when reading that function:

* All the physical variables (position, speed, engine status, etc.) can be accessed from the "car" variable. You can see for instance, how it is used at the bottom of the controller to determine whether we have reached the maximum RPM (revolutions per minute) and we need to shift gears.
* The track in F-1 Spirit is specified as a spline. Each piece of the spline is stored as an instance of the CRoadPiece class. The variable "road\_position" is the list of pieces that constitute the track. If you call "road\_position.GetObj()", it will return the current piece over which the car is. This road piece is where all the geometry of the car is stored.
* The best way to access the geometry of the track is with the function "get\_path\_position" that, given a point P1, returns the closest point P2 in the spline, the angle at which the car should be when driving in P2, the width of the road in that point, and the distance from P2 to P1. See how I used it in the source code to determine the angle at which the car should be driving.
* You might want to detect whether there are other cars around. You can do that by calling the "car->cars\_collision(...)" function. See how it was used in the example code.
* The only one thing that is not illustrated in the provided controller is how to test collisions with the track. You can do that by accessing the "track" variable. But unless you want to code a very advanced controller, I don't think it is required. The one use you might have for it is the following: you might have observed that the current controller is preprogrammed to drive always in the center of the road, when driving by the pit-stop area, the road widens, and sometimes there is an obstacle separating the right and the left hand side. The current controller always crashes there. You can check the "track" variable to see if you are approaching a pit-stop. But there is a smart way to avoid this just by checking at the track information (including the width of the track), I'll leave it to you to figure out (although it is not required to complete the project).

**Best!!**