



# Moto (A)GP(X) 2023



Description of the project

# Introduction

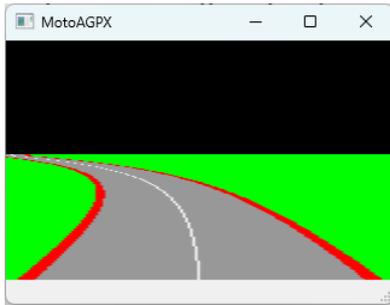
Have you ever wondered how games like 'Pitstop II' draw and animate that beautiful road so quickly and fluidly? I have and I wanted to formulate my own version. In this document, I will explain my idea and implementation, which I later saw echoed many of the ideas of the aforementioned game.

The idea is to memorise a sequence of screens representing the various streets (from the one with the leftmost curvature, to the one with the rightmost curvature). We are in multi-colour text mode. The various screens are pre-calculated by a programme written in C++, which will export the sequence of bytes to be inserted into the BASIC programme. To avoid the excessive growth of the number of roads, the kerbs and the central strip are drawn in a continuous manner (without 'zebra striping') and then a raster interrupt is entrusted with the task of subdividing and animating them.

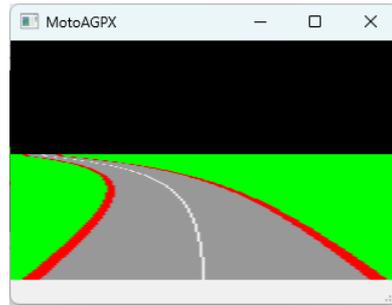
Initially I stored all the roads in RLE format, but I found that the frame rate often dropped below 50 fps (despite hand-written assembly routines), but then I found a solution which you will see in the following slides.

So a compiled BASIC program but, more than ever in this case, we also have quite a bit of code written in assembly because the timing is too tight (assembly is not dead, in the C64 it is a MUST, however compiled BASIC is quite a 'conductor')!

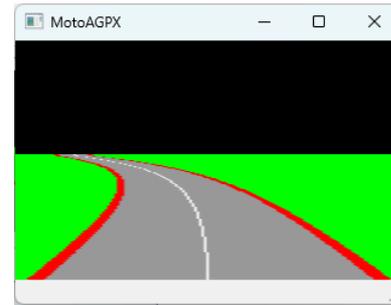
The number of roads I have decided to create is 29, divided as follows: 14 left-hand curves, 1 centre and 14 right-hand curves (in that order). The number 14 corresponds to the constant NUM\_ROADS in BASIC. Each road therefore has an index ('roadlx') from 0 to 28 (in general it is  $2 * \text{NUM\_ROADS} + 1$ ). The road starts at line #12 on the screen and ends at #24 (zero-based).



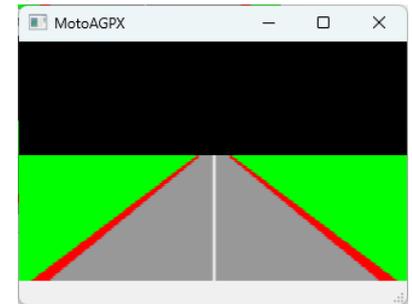
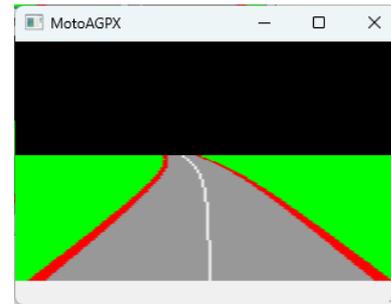
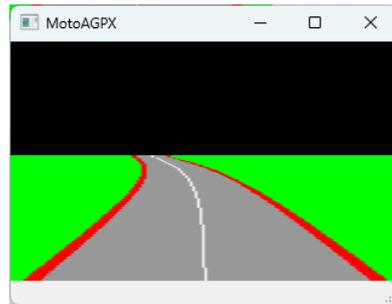
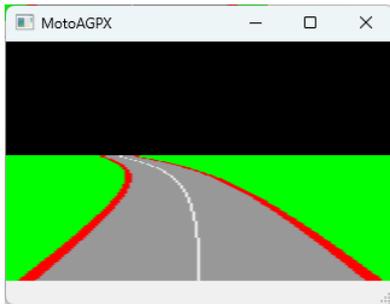
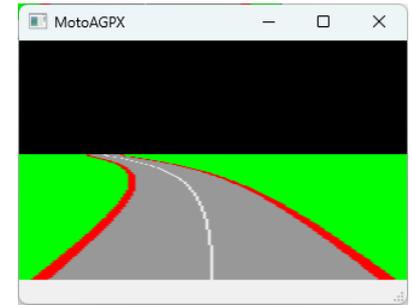
roadlx = 0



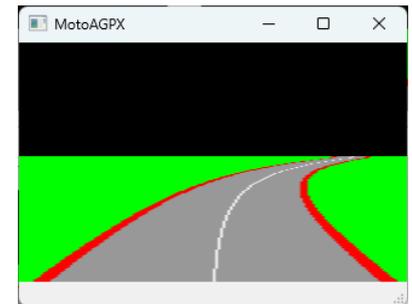
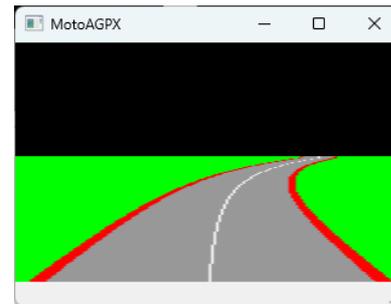
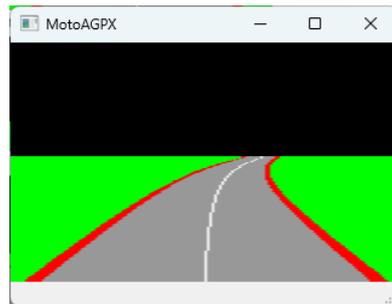
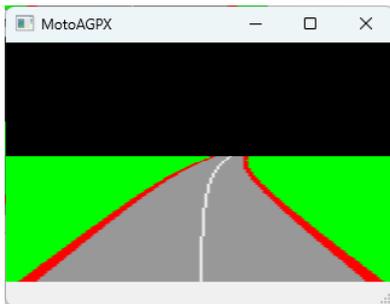
roadlx = 1



roadlx = 2

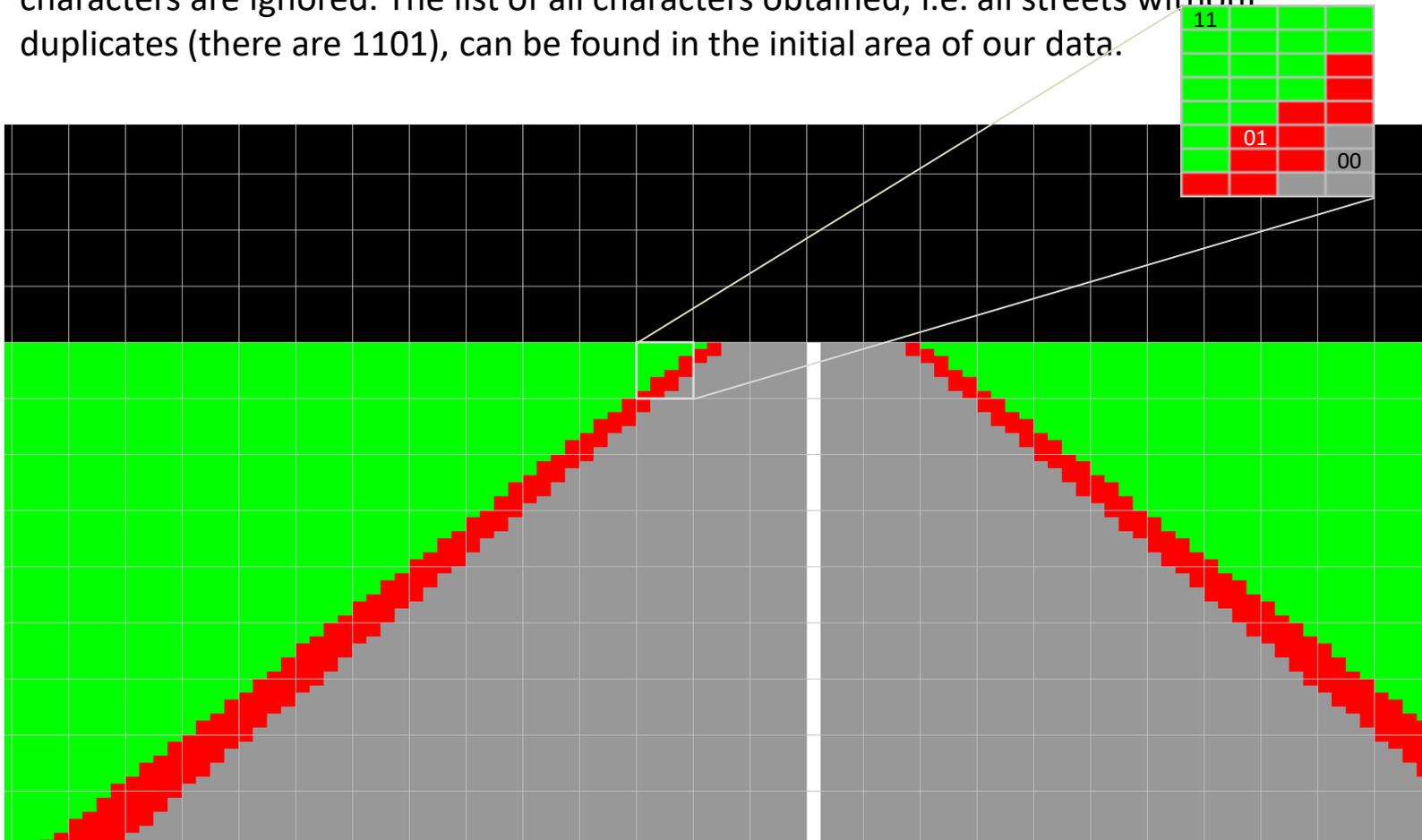


roadlx = 14 (NUM\_ROADS)



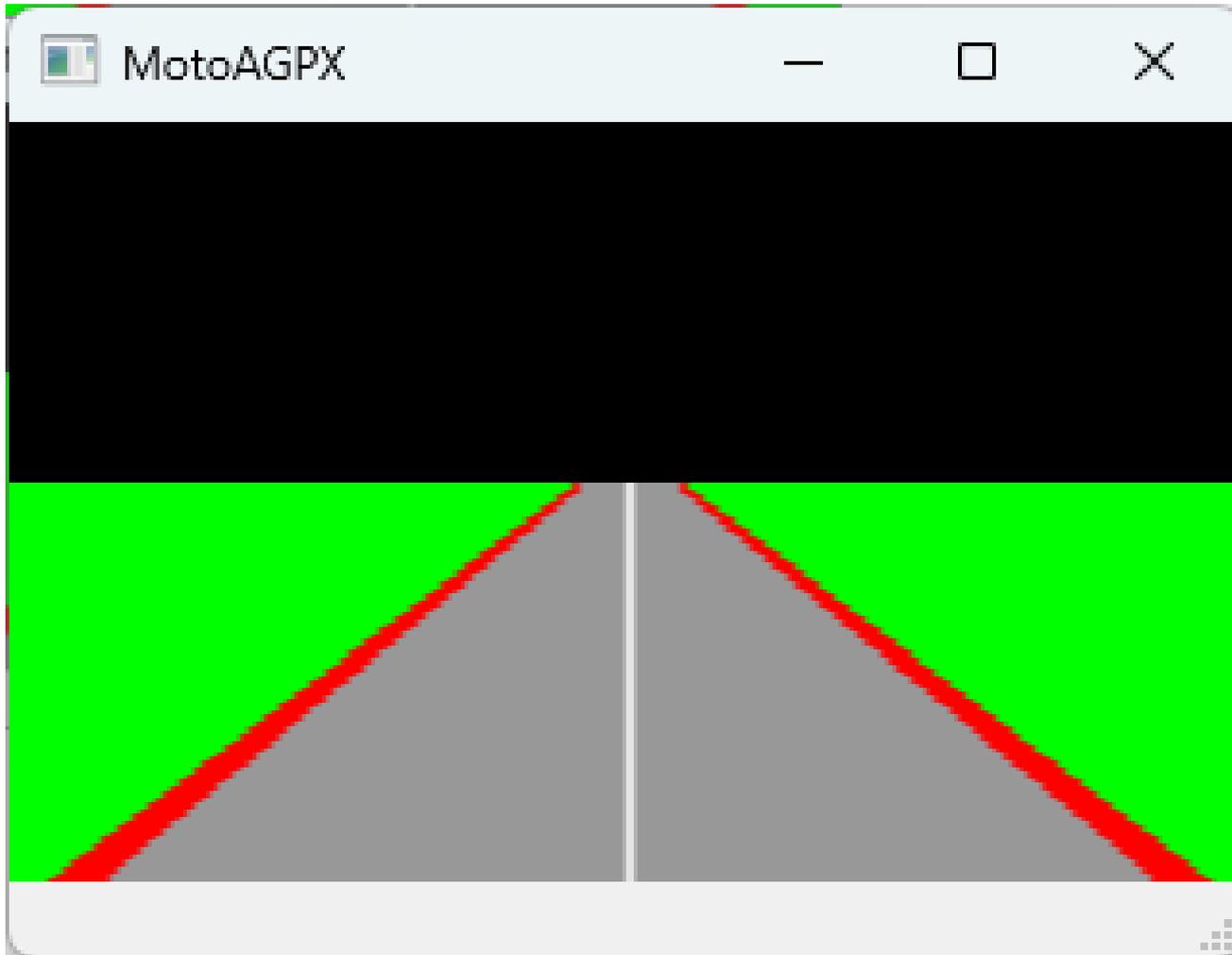
roadlx = 28 ( $2 * \text{NUM\_ROADS}$ )

The C++ programme divides each frame into 4 x 8 pixel blocks and generates the bytes of the relevant redefined character (00 = grey, 01 = red, 10 = white, 11 = green). Duplicate characters are ignored. The list of all characters obtained, i.e. all streets without duplicates (there are 1101), can be found in the initial area of our data.

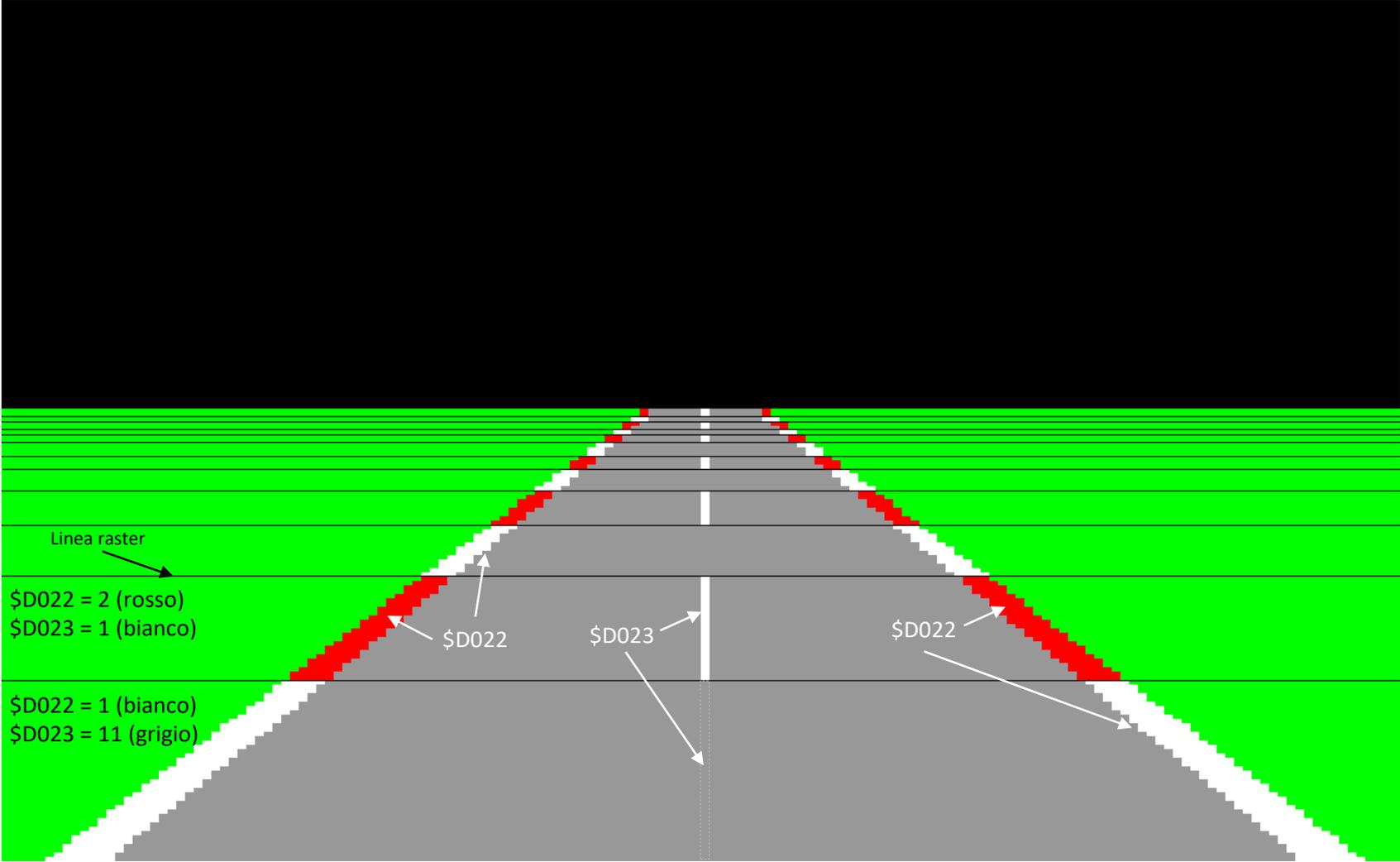


A character is made up of 8 bytes, so a 64-bit unsigned integer (`uint64_t`) can be used to store a character. In C++, the complete list is stored in a hash table, where the key is the character (64-bit) and the value is the index of the character in our final array. In this way, very quickly, when scanning the screen, it can be determined whether a character is duplicated (existing key).

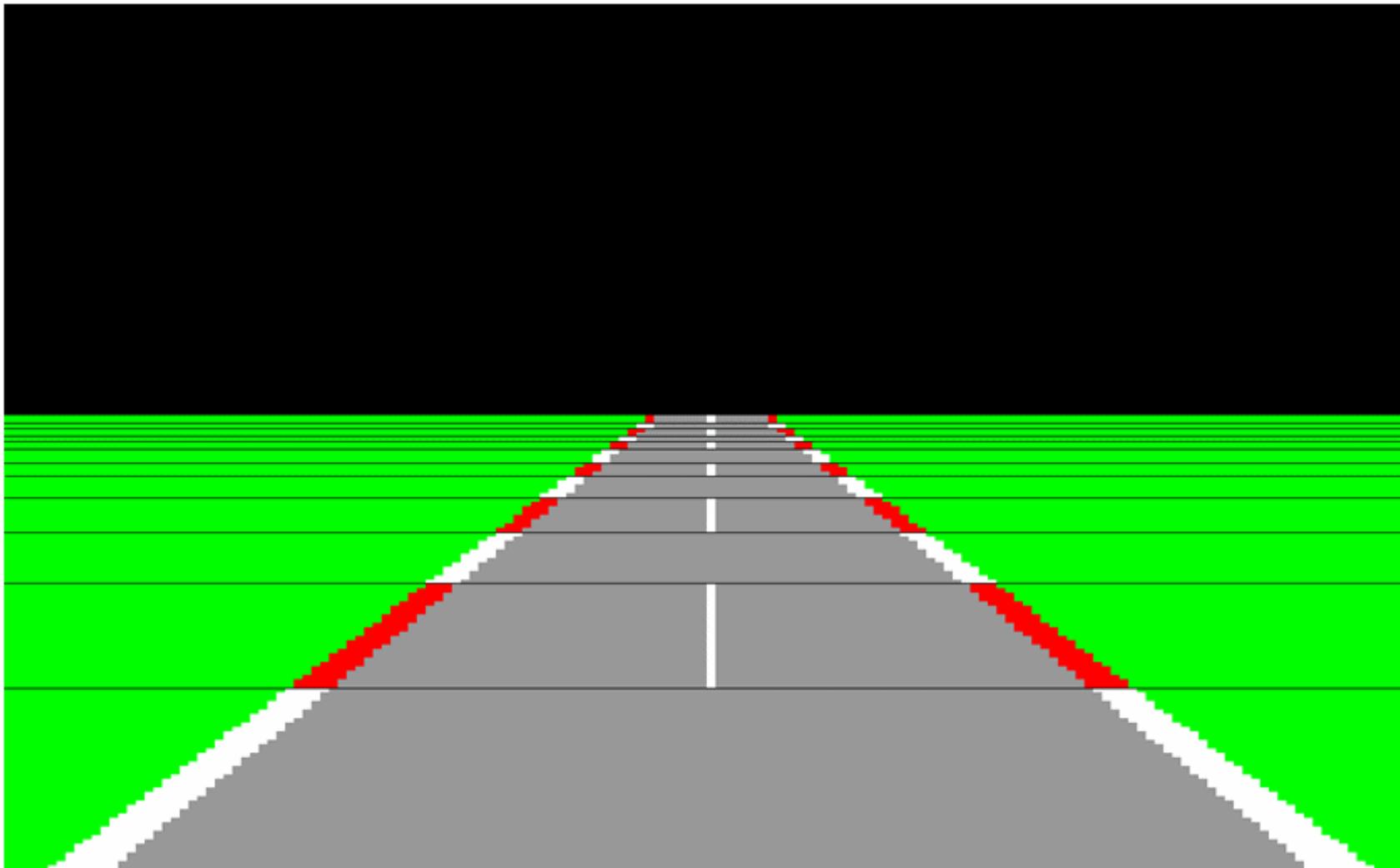
The side kerbs (in red) and the white central strip are continuous. The kerbs use the colour multi 1 (\$D022), while the central strip uses multi 2 (\$D023). The foreground colour (\$D800 - \$DBE7) is green (for grass), while the road is transparent (\$D021).



A raster routine changes \$D022 and \$D023 in order to colour the kerbs and create breaks in the centre line (which is still a colour change). The screen was divided into 12 strips (ROAD\_STRIP constant in the basic code). Making it stable, however, is by no means a simple matter!



Queste 12 linee raster vengono shiftate in basso (sempre tenendo conto della prospettiva), per un totale di 32 insiemi (costante STRIP\_STEPS). Il passo con cui si muovono è studiato affinché al 32-esimo step, i cordoli ritornino come al passo 0: in questo modo si avrà un effetto di movimento continuo. Avremo quindi una tabella di  $12 * 32$  byte (all'inizio dei nostri dati), che rappresentano tutte le possibili posizioni delle linee raster e ci serviranno appunto per creare l'effetto di scorrimento di tutte le strade. L'immagine seguente mostra soltanto 2 delle 32 posizioni possibili:



With the C64 you have to go easy on overdraw, that is, plotting the same area several times, especially when it has not changed. For performance issues, it is best that, when the road must curve, we avoid plotting things that do not change (like many areas of grass or the road). To achieve this, we only store the differences between the road of index  $i$  and its next  $i + 1$  (and also with the previous  $i - 1$ ). For example, to store the differences with the next road, I used the following scheme:

```
06                ; 6 different characters
E8 01            ; offset from which to draw ($01E8 = 488 = 12 * 40 + 8, i.e. from the 8th character of line 12)
02 01 04 05 08 03 ; the 6 characters to be written

03                ; 3 different characters
F5 01            ; offset from which to draw ($01F5 = 501 = 12 * 40 + 21)
05 0B 08          ; the 3 characters to be written

...

00                ; end of screen
```

A 0 as the number of different characters will indicate the end of the differences. One can find more compact patterns, but this is very fast in the decompression phase. Note that with this trick, I have the disadvantage that in order to go from the  $i$  to the  $i + 2$  route, I will necessarily first have to go through the  $i + 1$ , as I am only able to evolve the current picture to the next or previous one.

In order to implement the previous trick, it is necessary to store at least the first frame (straight road, roadx = NUM\_ROADS) in full, since the screen is obviously empty initially. To do this, I compressed the screen using the RLE (Run Length Encode) format, which stores the characters as follows:

**09** ; 9 equal characters  
**01** ; character to be repeated (0 to 255)  
  
**8A** ; 1 character only (8th bit on), whose value is 0A (the 7 least significant bits)  
**03** ; 3 identical characters  
**61** ; character to be repeated (0 to 255)  
  
...  
  
**FF** ; indicates the end of the image

The example breaks down to: **01 01 01 01 01 01 0A 61 61** (6 bytes instead of 13). Fortunately, the number of redefined characters for a single screen is less than 128, so I can eliminate the repetition byte when I have a single character, using the 8th bit as a flag. We will therefore have 8A instead of 01 0A, in our example.

The ingredients for drawing the road are all there! This data is exported by the C++ programme in a single block (see label 'road:' in the BASIC source) and amounts to the beauty of 25714 bytes, which is not a few, but the compiler can handle it. The exact format of the data is as follows:

Offset	Descrizione	Dimensione (in byte)
0	Table of rasterline positions (12 positions for 32 variations)	$(ROAD\_STRIPS * STRIP\_STEPS) = (12 * 32) = 384$
384	Character table for all road variants (1101 characters)	$CHARS\_TABLE\_SIZE = 1101 * 8 = 8808$
9192	Road offset table. It contains an address (2 bytes) for each of the possible roads, pointing to the memory area containing the road description (which characters it uses of the 1101, characters of the previous and next road, possible RLE compression). For example for road 0, it would be 9250.	$(2 * NUM\_ROADS + 1) * 2 = (2 * 14 + 1) * 2 = 58$
9250	Number of redefined characters (nc) of index road 0	1
9251	Offset table (in the 1101-character table) of nc characters	$nc * 2$
9251 + nc * 2	Size in bytes (drle) of the frame in RLE format (if missing, value 0)	1
9252 + nc * 2	RLE frame data, if any	drle (< 256)
9252 + nc * 2 + drle	Size (dprev) of 'previous frame'	1
9253 + nc * 2 + drle	Previous frame data	dprev (< 256)
9253 + nc * 2 + drle + dprev	Size (dnext) of 'next frame'	1
9254 + nc * 2 + drle + dprev	Next frame data	dnext (< 256)
9254 + nc * 2 + drle + dprev + dnext	Number of redefined characters (nc) of index road 1	1
<b>and so on for all other (2 * NUM_ROADS) road variants</b>		

Although the global character table (offset 384) contains 1101 characters, only those needed (nc) for the particular path are copied at run-time into the character memory. They are of course re-indexed starting at 0. Finally, the above table is exported taking 0 as the starting offset, but in the C64's memory it will correspond to a specific address assigned by the compiler. For this reason, the routine 'relocate Data', adjusts the necessary offsets (e.g. the table of offsets starting at 9192) in order to transform them into physical addresses and avoid having to do the sum at run-time each time.

Tracking takes time and inevitably causes flickering and various artefacts, considering also that we have to redefine the characters each time. To avoid this, two video pages are used: one visible and one hidden (double buffering). "VIC-II Calculator' in hand, I decided to use the 3rd bank as follows: Page 0, has character memory in \$E800-\$EFFF and video memory in \$F000-\$F3FF  
Page 1, has character memory in \$F800-\$FFFF and video memory in \$F400-\$F7FF

The 4 areas do not overlap and in total occupy 6Kb (the last 6 of the 64Kb), so we leave the lower area free for BASIC.

When page 0 is visible (via register \$D018), I can safely write my characters in \$F800 and draw my way in \$F400. As soon as I am finished, page 1 becomes visible (again via register \$D018) and page 0 becomes hidden. At this point I will write in the 0 and this is repeated by continuously swapping the role of page 0 and 1 (from visible to hidden).

In practice, I always write in the hidden and as soon as it is ready I display it lightning fast.

The algorithm for tracing the road of index  $i$ , the previous of which was e.g.  $i - 1$ , performs the following steps:

1. It accesses the road offset table (9192) to obtain the memory address describing the  $i$ -th road;
2. From here it copies the  $nc$  characters from the global table to the character table of the hidden page (starting with character 0);
3. If you have been asked to draw the road using the RLE format, it does so;
4. Otherwise, just draw the differences by taking the information from the table for the previous road ( $i - 1$ );

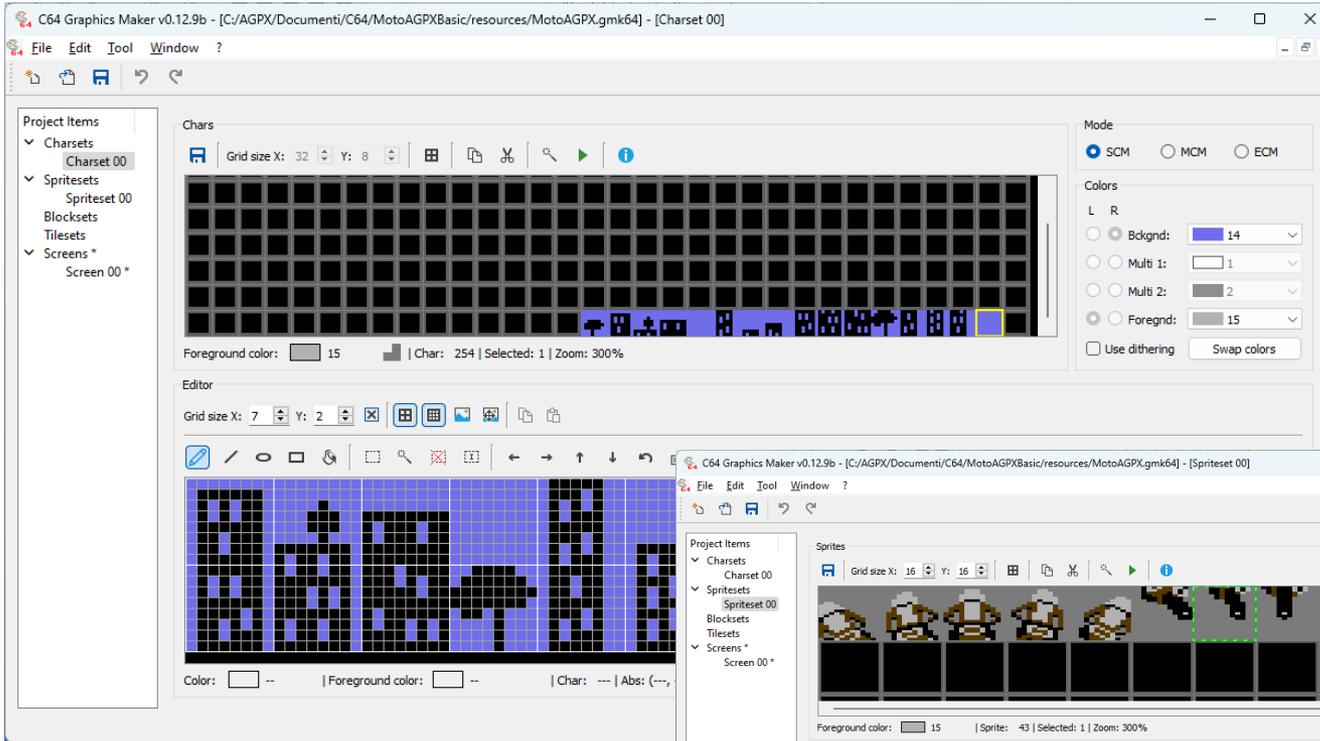
Each time we draw on a page, we note down the index of the road because we will need it for the next trace. These indices are stored in variables named `roadIxPage0` and `roadIx Page1`.

As for the circuit, it is stored in an array containing two pieces of information per entry: the distance to the start and the road index. An extract of the test map is made as follows:

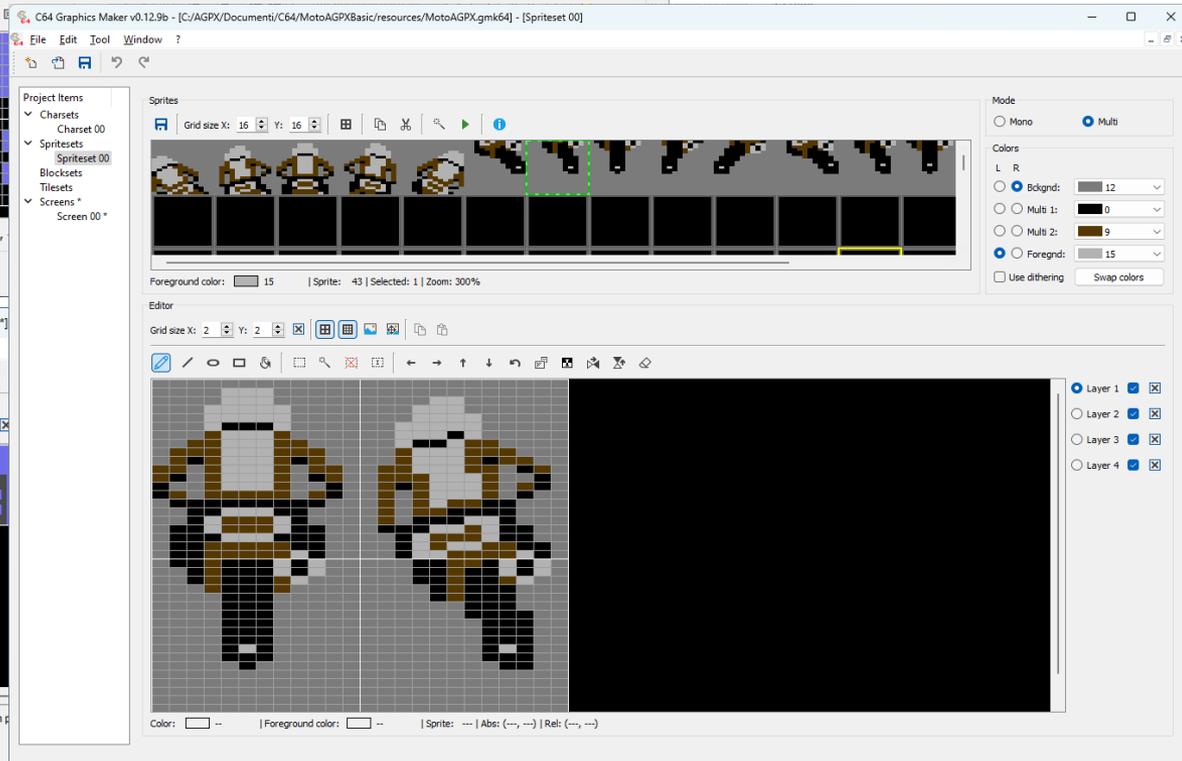
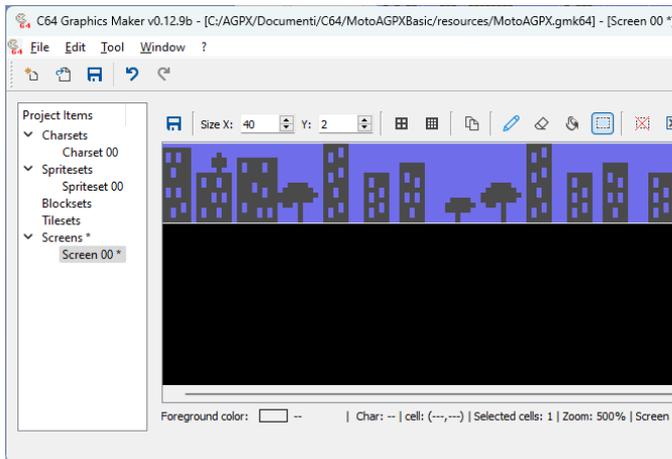
Distance in cm from the start (dcm)	Road index(roadIx)	Description
3000	NUM_ROADS	30 m of straight
9000	NUM_ROADS + (NUM_ROADS / 2)	Slight right turn into 60 m
15000	NUM_ROADS	Straight back into 60 m
24000	NUM_ROADS	Straightening of 90 m
30000	NUM_ROADS + (NUM_ROADS / 2)	Slight right turn in 60 m
36000	NUM_ROADS - (2 * NUM_ROADS / 3)	Sharp left turn in 60 m
42000	NUM_ROADS	Straight back in 60 m
45000	NUM_ROADS	Straightening of 30 m
48000	NUM_ROADS * 2	Super right turn in 30 m
51000	NUM_ROADS	Straight back in 30 m

Essentially, the circuit is made up of 'segments' (MapSegment type) of road, arranged one after the other. Note that by playing with the road index and the length of the segment, more or less difficult turns can be created.

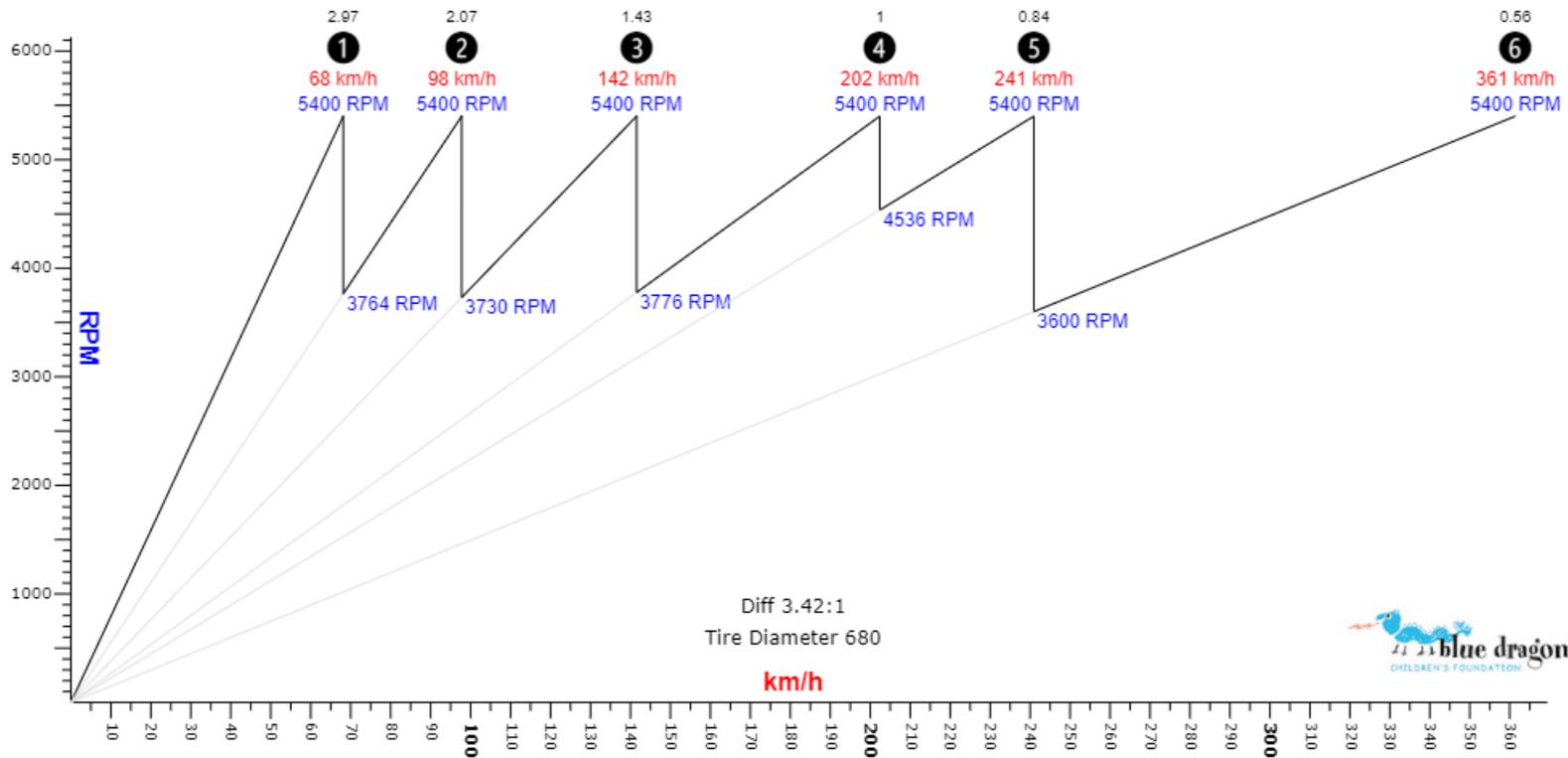
Sprites, fonts and backgrounds are made using the usual C64 Graphics Maker (OK, the bike sucks, you do better!):



Characters start at index 254 (blank space) up to 128, as those from 0 to 127 are reserved for street characters. Letters and numbers are also copied (once) at run-time (to save space) from the character ROM.



The sound of the motorbike was made with the help of a website (<https://www.blocklayer.com/rpm-gear>) that shows a graph of the number of revolutions as a function of speed in km/h and gear change (I went up to 5th gear):



The emitted frequency (square waveform or sawtooth) is proportional to the engine speed. This also simulates gear changes during acceleration.

Finally, a list of possible improvements:

- Add an initial counter to kick off
- Create more circuits
- Add a map display
- Add a starting line (with sprites)
- Improve sprites
- Improve background graphics
- Oil stains on asphalt (sprites) with falls
- Roadside signs (sprites)
- Adding opponents
- Adding fuel and pit stops

**That's All Folks!**

AGPX