
MANU & KIT
Travel Through Time
in Detail

MANU & KIT

Travel Through Time

in Detail

The book about how we made the game *Travel Through Time Volume 1: Northern Lights* for the ZX Spectrum computer.

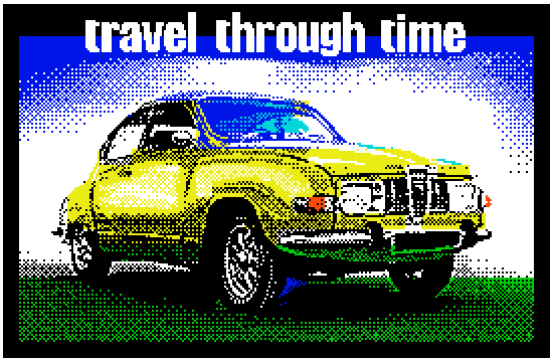
English edition (v1.01) by A.M.

Creating computer games is a wickedly fascinating process. When working on a large project, a huge number of questions and challenges always arise – ones whose solutions you won't find in books like "How to Write a Game in Assembly". Our book will not teach you how to program, but – we are sure – it will be very interesting to anyone who has played our games and wondered "How did they do that?"

The facts are presented in the book in random order. Some other games related to the *Travel Through Time* series (hereinafter sometimes referred to as *TTT*) are also mentioned, as well as details that did not make it into the final version of the game.

1.

The original game design was entirely dedicated to one well-known car brand. In the end, we removed all references to car names (we will not mention them in the book either) to avoid claims from copyright holders, and also added other cars and other vehicles.



The first version of the splash screen has been preserved, as well as a very grand description of the original project:

"The first racing game for ZX Spectrum with photorealistic road graphics. It's time to forget about striped fields and track curbs, monotonous levels and roadside objects that sprout up like mushrooms. The tracks in this game are truly alive. Forests and open spaces, hills, cliffs and rock walls, tunnels, intersections, functioning petrol stations, settlements with different types of buildings, bridges and even... a working railway crossing! What else have you not seen in racing games on ZX Spectrum? There are all kinds of competitions on offer: time trials, duels, mass starts and even orienteering, based on real maps of areas in Sweden. And yes, in this type of competition, intersections are fully functional. The points earned in the races are used to upgrade the car. All spare parts have a certain degree of influence on handling, acceleration and maximum speed. In addition, a separate upgrade procedure has been implemented for the turbo, with even more parameters. When a new car appears, the turbo is transferred to it, and its parameters are preserved".

As you can see, not all of the original ideas were implemented (some of them – for example, the upgrade – did not fit into the final concept), but many other interesting details appeared.

2.

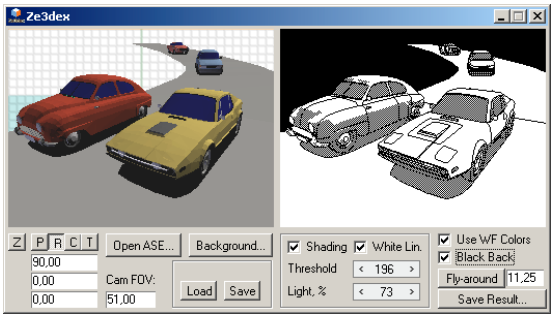
The game uses more than a dozen graphic formats and output procedures – for each element, the most efficient methods of output and data storage are used. For example, rendering the player's car and rendering the posts on the road is done by one procedure, but the opponents' cars is done by a completely different one, as it requires automasks (autocalculated transparency masks), pixel-precise positioning, clipping, and vertical scaling (to a small extent). Separate procedures are used for panoramic shots, objects "popping up" from behind the horizon, trees in the middle ground, colour sprites in the interface and cutscenes, large character images, and so on.

3.

Like in *DRIFT!*, all in-game cars, as well as some of the competitors' cars, were first prepared as 3D models, and then rendered from different angles. The models were detailed with ease of subsequent rendering in mind.

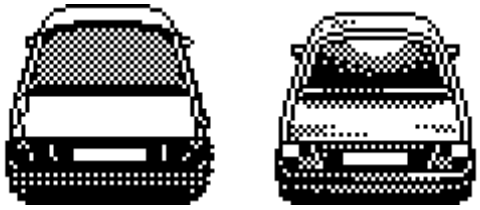
4.

When rendering the cars, we used our own utility for converting 3D models into pixel art. We had previously used this program, called *Ze3dex*, when creating the game *DRIFT!* It allows to render virtually any source model in pixels, but it works significantly better with specially prepared models having a small number of polygons and an emphasis on the necessary structural elements.



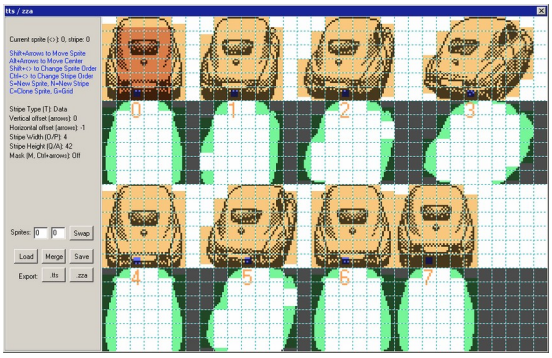
This method is fundamentally different from the typical case, where it's not the model that's converted, but its render (visualization). Below, we show how the image would look with a typical graphics conversion, and how it looks with a model conversion.

Nevertheless, while large images, the size of a ZX Spectrum screen, produced using this program are initially of acceptable quality, sprites such as those in *Travel Through Time* require substantial reworking. In this game, they have effectively been redrawn from scratch.



5.

The graphic format used for the player's cars and roadside objects is quite interesting. It was originally developed for this game, hence its name: Travel through Time Sprite (.tts). Before the game was released, we had already used it in other games – *DRIFT!* and *Maureen Miles* (known as *Just a Gal* in the ZOSYA release). A sprite editor was written for it.



The essence of this format is that sprites are composed of individual parts of varying sizes, which may or may not use a mask. This allows sprites with complex shapes to be stored in a very compact form, and the output uses a set of procedures that are optimal for each sprite fragment. There is a subroutine for outputting a 1-character-wide fragment without a mask, one for the same character but with a mask, another pair of procedures doing the same but with mirror imaging; another set of routines doing the same tasks on 2-character-wide sprites... It looks cumbersome, but it works very quickly.

There are limitations: for example, in *TTT*, we use a fragment width equal to 1, 2, or 4 character cells, and the height is always a multiple of 2 lines, as is the vertical positioning of a sprite in the game. One can also decide that a certain fragment variant in the game (say, 2 characters wide with a mask and mirroring) will never be used, and thus remove the unneeded procedures from the code.

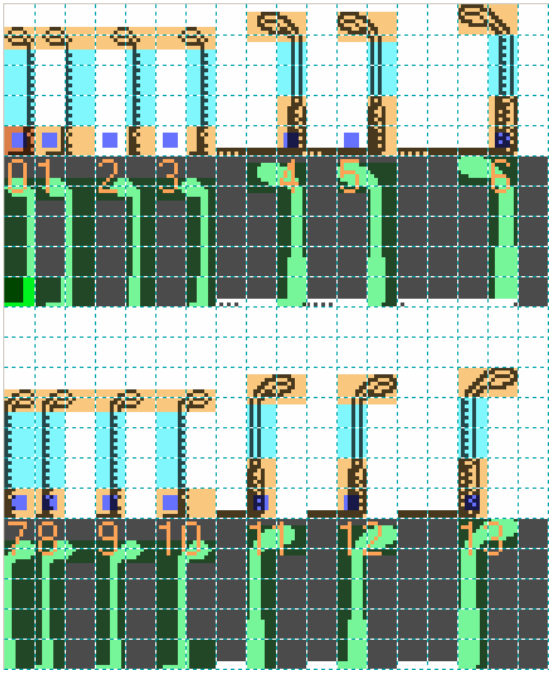
But there's more to this format. It also allows to save "columns" consisting of a pair of repeating bytes as sprite fragments – and this is not only very compact, but also allows to change the height of this column when rendering the sprite. As you might guess, this is ideal for displaying roadside posts. Moreover, for the subsequent pieces of the sprite, one can use positioning at the top of the last column, or with an additional offset.

An interesting format, in general. However, its drawbacks are significant: the horizontal positioning resolution is limited to character cells, and sprite cropping is difficult to implement.

6.

As mentioned above, the roadside objects use the same sprite format as the player's cars. It doesn't support on-the-fly horizontal offsets (procedural offsets are, in any case, a very poor option for these objects due to their slow rendering speed), so the offsets are pre-drawn.

For the furthest sprites 4 horizontal offset options are drawn, then 2, and 1 value for the closest sprite. This is because the horizontal movement of distant poles is much slower than that of nearby ones, requiring more precise positioning. This solution can be found in other racing games as well.



The middle section of the poles is made of a pair of repeating pixels (the principle is described above). When the sprite is rendered, the initial height is adjusted based on the distance to the object.

You can also notice that for some offset sprites, certain small details are simply ignored to avoid wasting space. This is also a kind of optimization – it's completely unnoticeable in the game.

Masks are used where necessary.

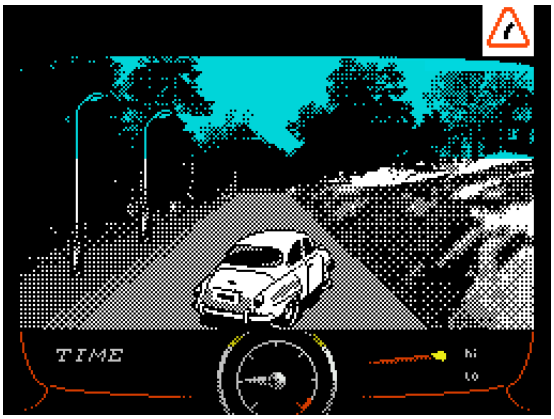
Left and right variants are used for streetlights and turn signs. Other roadside objects on the left and right sides of the road are identical. In *Rubinho Cucaracha* and *Travel Unlimited*, lampposts also use a single variant for both sides of the road.

7.

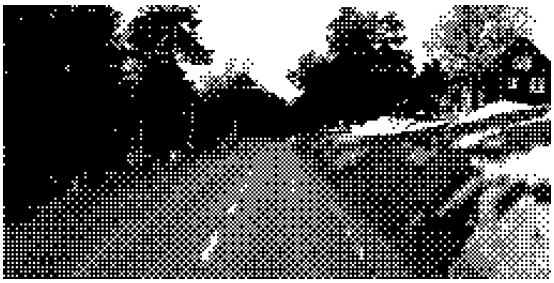
To ensure everything fits in memory, much of the data is stored in compressed form and is uncompressed as needed. Both a standard packer and custom packing methods are used for different data types. For example, for the text a token system is used.

8.

This game screen mockup was the starting point for game development.

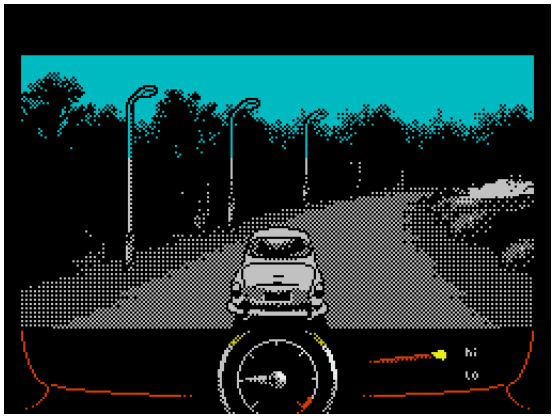


It was noted how the depiction of roads and surrounding objects in racing games for the ZX Spectrum is usually far from reality and generally very arbitrary. This hastily converted photograph served as the basis for the layout:

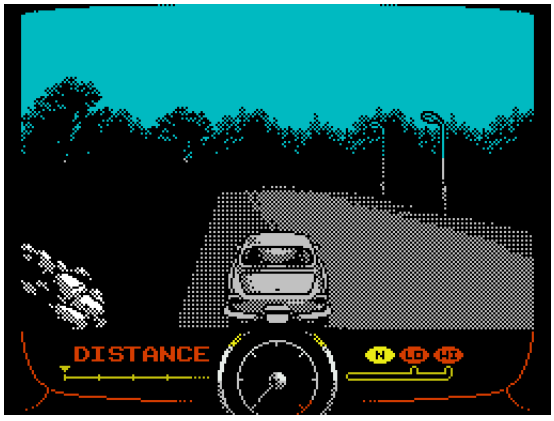


Subsequent work on the game was conducted with this model in mind, as it presented a completely different atmosphere compared to all existing games. We attempted to capture this atmosphere within the game, taking into account the existing technical limitations. For example, we depicted the dark outlines of trees slowly floating in the middle ground, the rather jagged roadside (in this case, our engine's capabilities perfectly matched the original image), and overall, the imagery in our game – with its black fills – leans toward closed spaces, something rarely seen in racing games for the ZX Spectrum. This makes the occasional panoramic views on the horizon all the more interesting.

Of course, not everything could be implemented in full accordance with the original layout. For example, it would have been very difficult to create the hilly terrain behind the roadside with houses and trees on it (all of this in motion and at a decent speed). We carried out some work to simplify the image.



...Well, can we at least put some stone boulders at the very beginning of the race? That way, we don't need to draw a large number of variants at different scales.



In the end, the game only included one dark rocky ridge along the road, which for some reason appears in a single level, the 45th.

9.

So, in addition to typical roadside objects, the TRAVEL engine uses a kind of "middle ground". This is a unique element not found in other games: objects in the distance that move slower than roadside posts, but still appear and disappear off-screen, unlike the fixed background. There are two types of such objects: trees that almost blend into the background, and buildings and structures that "pop up" from behind the horizon.

Another unique feature of this game, which has already been mentioned, is the panoramic views. These are a set of detailed images that sometimes appear on the horizon instead of the standard backgrounds. Unlike the latter, they unfortunately do not move horizontally, but they create an unparalleled atmosphere.

Some other objects, such as overpasses under which cars pass, while not entirely unique, also have interesting features. All such objects will be discussed below.

10.

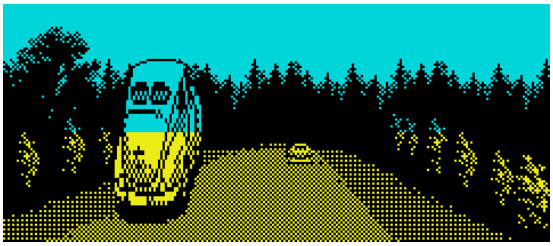
A simple, linear sprite format without masks is used to depict opponents' cars (as well as in cutscenes and for other purposes). The beauty of this approach lies in the sprite rendering process: it utilizes precise, per-pixel positioning both horizontally and vertically, an automatic mask, screen-edge cropping, and the ability to vertically scale the sprite. Given all the functionality and the ability to scroll sprites of any width on the fly, the process obviously can't be "hyperfast". Nevertheless, it handles the needs of our engine perfectly. After all, at a large scale, we typically see only one opponent, while the remaining sprites, if any, are much smaller and rendered quickly.

As for edge cropping, it can be either automatic or forced. This is for cases where the car's center is so far to the left of the screen that it rolls over to the right. But the engine knows that the car is still on the left, and it needs to be cropped on the left.

In addition, if necessary, the discreteness of the horizontal positioning can be adjusted.

11.

Sprites are vertically scaled as follows: a counter is incremented with each sprite line, and if it reaches a preset threshold value, it is reset to zero and the line is duplicated. Thus, the maximum scaling (2x) will occur at a threshold value of 1.



With a value of 2, the sprite will be 1½ times its height, as every second line of the original sprite will be duplicated. The game uses moderate threshold values, greater than 8. Scaling is only performed for the "closest" and largest sprites. A value of -1 (255), or any other value greater than the height of the sprite, disables scaling. Stretching the sprite looks ugly if the duplicated lines fall on areas of the sprite with horizontal lines.

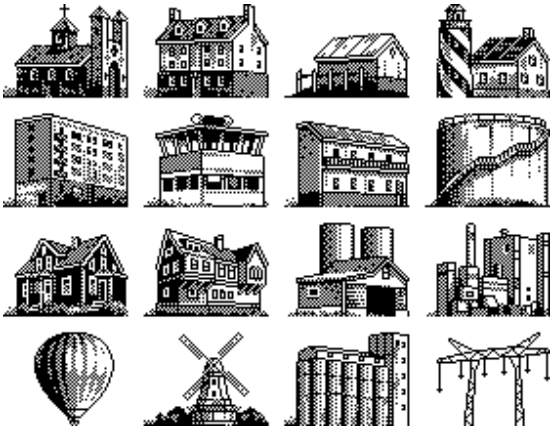
Knowing in advance the scale values to use can help to draw a sprite more effectively.

12.

The first working versions of the engine didn't yet use this format, and the coarse movement of opponents' cars on the screen (as far as the .tts format allowed) was a stumbling block for further development. Imagine cars leaving the start line in a "stair-step" pattern. Maintaining precomputed offsets for each scale of each car is prohibitively expensive.

13.

To create visual variety, the game uses a set of mid-ground objects appearing from beyond the horizon. There are 16 of them in total:



Some appear only once in the game, while others are repeated. The object index is specified in the level header if such entity is present on the track.

These objects always appear on an incline, when the road gently turns right, and they therefore always move left and up, eventually disappearing behind the left curb. They are also drawn with a perspective that suits their placement. They are not scalable.

If, at the beginning of work on these objects, we had already been using a procedure for sprites with automasking and pixel-by-pixel positioning, we would certainly have used it to render them. But instead, we set ourselves the ambitious goal of implementing very fast rendering with smooth movement. As a result, a large buffer is used for them, where copies of the sprite with offsets are prepared in advance.

Furthermore, their rendering uses separate routines that include the necessary cropping capabilities. It's safe to say that our initial approach was extremely suboptimal. Even if rendering a sprite with an automask and on-the-fly offset is slower, it wouldn't be noticeable for such a rare, short sequence. This approach would have allowed to save a significant amount of space, which we could have used increase the rendering capabilities. It would have been possible to draw objects on the right, and even to apply some scaling.

In general, the objects are good, but the procedures are not optimal.

14.

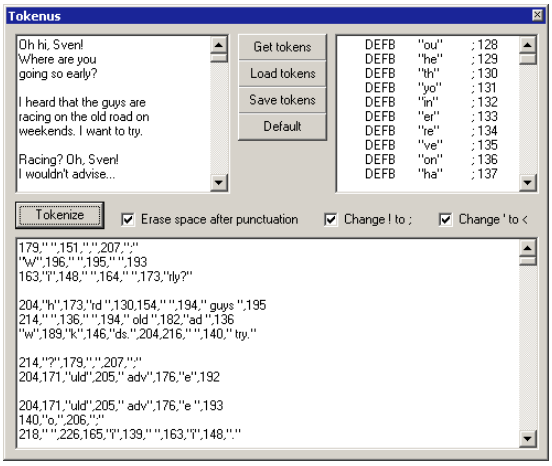
Initially, we wanted to make the speedometer needle using raster graphics. We changed our minds after estimating the required number of sprites.

In the end, the needle was drawn using vector graphics. A special utility was written to calculate the segment coordinates. The resulting table was inserted into the program code. The needle has 62 positions, which is more discrete than the speed value change. It is redrawn directly on the screen, and the previous position is deleted. To minimize engine lag, the procedure is performed every other frame, and, of course, only if the needle's position has changed.

Interestingly, the coordinates of the inner point, which changes over a small range, are written more compactly in the table – as a single byte. This doesn't offer much benefit, as such a record still needs to be interpreted by the program.

15.

The token system used for text packaging includes the *Tokenus* utility, which was written during the development of this game.



It can automatically identify the most common two-character combinations in the source text (for best results, you need to feed it the most complete set of texts used in the game possible), and also allows you to add other combinations of letters of varying lengths. In our case, we added function words such as “the”, “are”, “you”, “here”, “what”, and so on, the names of all the characters, as well as words specific to our game, such as “racing” or “time”. A total of up to 128 combinations can be specified. (The other 128 byte values are reserved for letters, other symbols, and control codes.) After that, the program converts the source texts into a more compact form using the indices of the described tokens.

Accordingly, the parser in our game’s code must also know all these letter combinations and their indices in order to decode the text. It should be noted that this method is only effective when there is a large amount of text and direct access to it is required. (After all, the parser and token set take up a lot of space!) But this suits us just fine. A typical packer would handle the task more efficiently, but it would require unpacking a large block of text into a separate buffer, which is usually very inconvenient.

16.

All road signs in the game, except for turn-direction signs, are of the same type: warning signs. They consist of two sprites: a blank sign on a pole and an image – a pictogram.

The horizontal positioning of the sign is less precise than for posts. This is due to the need to display pictograms, all of which are presented in a single version, without offsets and at the same scale.

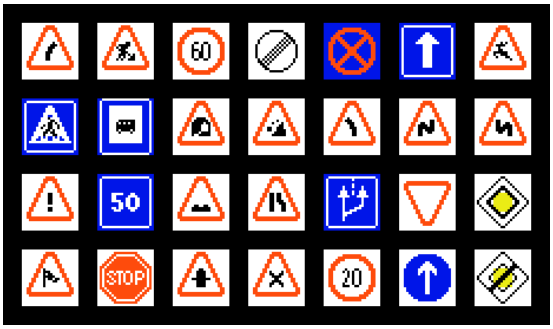
At the furthest distance, the pictogram is not displayed (there is a kind of abstract image), and the sign is positioned with an accuracy of 4 pixels. For the other two scale options, horizontal positioning is precise, and only one version of the pictogram is displayed on the sign.

Most pictograms are 1 character wide and 8 pixels high or less. Only the “Road narrowing (on both sides)” sign did not fit into this size and uses an additional character with a mask.

The last four pictograms, starting with “Other dangers”, are not used in the game; they are not included in the sprite set. Initially, there was an idea to include deer crossing the road in the distance (of course, not every time after the sign, which would be completely ridiculous) and intersections, but this had to be left for the next games in the series.

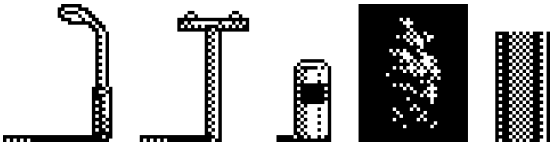
17.

Incidentally, the first mock-up showed the road sign not on the road itself, but on the interface panel. This is exactly the technique used in our game *Knights of the Night Roads*. And some of the icons migrated to *TTT* from there:



18.

The number of objects on the roadside in the game is quite small. In addition to two types of signs, these include: lamps, power line poles, small posts, bushes, and bridge supports.

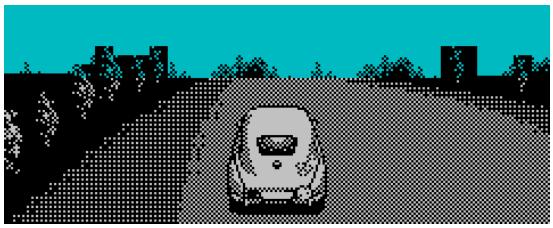


And we see them already in the first levels. Isn't that a bit sparse for such a long game? Actually, we did it deliberately. In many racing games of the 8-bit era, roadside objects change from level to level: the fir trees of the first level are replaced by cacti in the second, the background of the road changes from green to yellow, and rocks appear on the horizon instead of forests. Rubinho Cucaracha, for example, uses this classic scheme.

But in *TTT*, we decided to achieve variety in a different way: using a set of mid-ground objects and unique panoramas, changing the visible landscape from closed to open, gradually adding elements such as overpasses, tunnels, rivers, and so on. The colour attributes are, of course, also different for different levels, and there is also night and winter. And if the appearance of the streetlights along the road changed on some levels, it would have little effect on the overall diversity. We decided that instead, it would be better to add another panoramic view.

19.

The sprites depicting the bushes are rendered on a black background and without masks, to save memory and improve performance. The game is designed so that they are displayed exclusively against dark ground or water. They cannot be displayed on slopes, as then, against a light sky, they would appear as black rectangles with a pattern, like this:



20.

In *Crazy Cars II* / *F40 Pursuit Simulator*, typical power poles are depicted along the road on most platforms, but on the Atari ST, for example, we see wires strung between the poles. We tried a similar approach for TTT. We already had a line-drawing procedure, and the detection of the pole tops was also used for certain purposes. The biggest problem would have been the wires extending off-screen as the car passed a pole. Some early work was done on implementing these procedures, but we abandoned it early on because it looked very unsightly given the specific nature of our poles. The result was very flat (literally, since the pole tops on the screen are at similar heights for all distances), jerky, and generally looked like a display defect. Pity, because it seemed like an interesting idea.

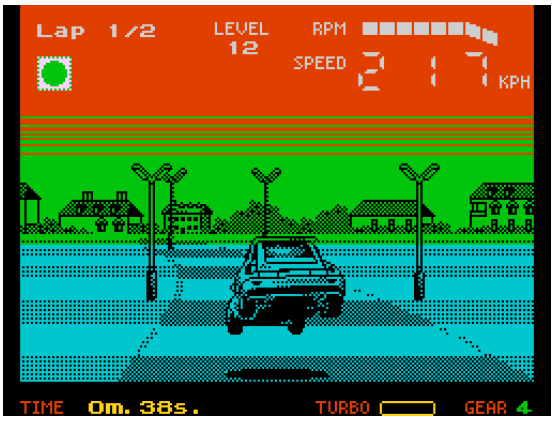
21.

We debated for a long time whether to make the game a single entity or split it into two separately downloadable parts. The obvious advantage of the second approach was that we could fit in more graphics for cutscenes and background objects, increase the number of cutscenes, dialogues and levels, and at the same time, one part would load faster than the whole game.

This immediately led to a disadvantage: both parts together would take up much more than 128 kilobytes, and a significant part of the code would simply be duplicated. We also saw several other disadvantages of dividing the game into two parts related to the plot. Finally, the title "*Travel Through Time Volume 1: Northern Lights, Part 1*" seemed a bit of a mouthful.

22.

There are no jumps in the game (except for small bounces on uneven surfaces), so the shadow is one with the car sprite. It was already made in the original 3D models. In newer versions of the engine (for example, in the spin-off *Travel Unlimited*), jumps were added, and the shadow "separates" from the car only for those sprites that depict the car in mid-air.

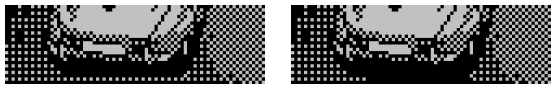


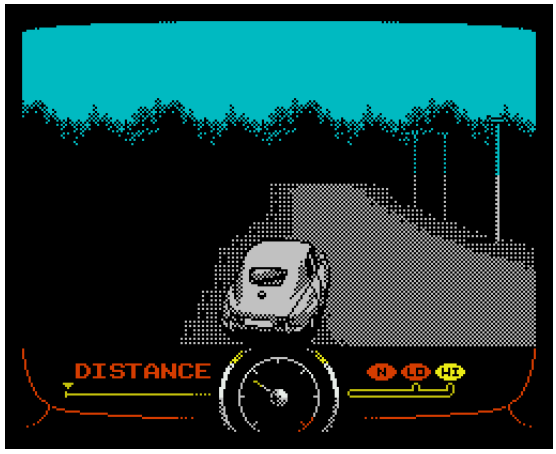
And in the second part of the main TTT series, the jumps are even more refined: your opponents' cars also jump, and the bumps in the road (which are not very noticeable in *Travel Unlimited*) are depicted with a smoothly changing texture.

23.

At low speeds, the player's car sways back and forth when it's on the margin of the road (or on uneven roads), and at medium and high speeds, it starts to bounce by two pixels. We probably would have made it just one pixel, but... remember the capabilities of the .tts format.

To avoid the shadow to appear to be "glued" to the car, we increased the shadow when jumping by simply drawing a single short black (or rather, pixel-filled) stripe in a fixed location on the game screen – on the second line from the bottom in the centre. Why the second line? Because the texture of the first line at the bottom of the roadside is already black.

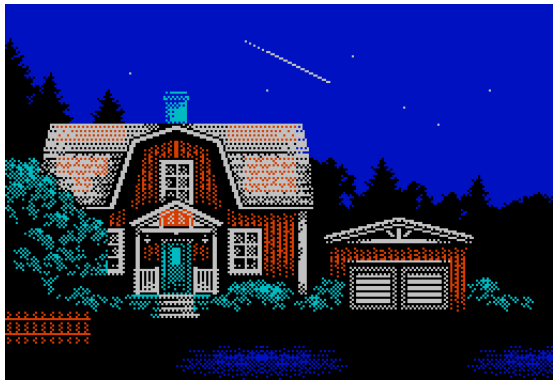




It's a small thing, but improves how it looks. However, on hills, when the car moves slightly upward, we stop using this technique.

24.

The falling star in the opening scene is not a sprite, but procedurally generated.



25.

It's worth mentioning that in the 8-bit era, there were a wide variety of car racing gameplay options, with different camera angles and gameplay representation principles. Even if we ignore top-down, side-on (there were those, too), and isometric views, and focus on the typical behind-the-car or inside-the-car view, three main types can still be distinguished:

1) The perspective is constant, the camera always moves along the road centered, and the car moves left and right across the screen when the road or steering wheel turns. For example, the first *Crazy Cars* game, *Turbo Esprit*.

2) The car is always centered on the screen, the camera is fixed to it, and the perspective changes as the car and camera move across the road (during turns); the camera does not rotate and always faces along the road. This is the most common type, used for both behind-the-car games – *WEC Le Mans*, *Chase H.Q.* – and in-car games, like *Chequered Flag*.

3) Complete freedom of movement. Both the camera and the car can rotate 360 degrees. You can drive anywhere. This is a rare type due to the difficulty of implementing it on 8-bit computers. For example, *Hard Driving* or our *DRIFT!*

Occasionally, something in between these two basic types is encountered. For example, in *Crazy Cars II* (which is generally of the second type), you can drive almost perpendicular to the road, and the camera rotates accordingly.

In any case, for *TTT*, we chose the second option, the most common. Its main difficulty lies in the changing perspective (meaning the imaginary lines along which the road's boundaries and all objects are located), which imposes limitations on the depiction of large objects along the roadside: the walls of buildings facing the road must match the camera position and the curves of the road.

You can simplify the task by placing houses only on straight sections, but the perspective of the side walls still has to change based on the position of the car. It is for this reason that such games rarely depict driving through the city, and if they do, as in *Chase H.Q.* for example, only "pencil-like" buildings following the principles of pillars and trees are shown.

And now for the disappointing spoiler: in *TTT1*, there is no city either, no buildings with a changing perspective. However, the following image shows that we did conduct some research in this area.

The idea was that if, on straight sections, the houses were depicted quite far from the roadside, then perhaps the perspective change could be avoided. Ultimately, we left this issue for future installments of the series. ("It's impossible to fit everything into one game!")

26.

And in this screenshot, we see a different type of house that would look good – on the contrary – precisely in the turns, without any change in perspective.

But unfortunately, there was no place for it in the first game of the TTT series.

27.

All rival cars, except special vehicles, use the same sprites for the two smallest scales. These sprites also have pre-drawn offsets for faster rendering and use a different sprite format – with masks.

28.

All opponents' cars have five scale options (not counting the two common distant ones). Nearby variants are also slightly scaled when displayed on-screen. The angle is chosen based on the fact that we typically see cars not directly in front of us, but slightly off to the side. The sprite's orientation changes based on the relative positions of the player and opponent, as well as the road's curvature.

Some opponent car sprites have additional phases for smoother changes in perspective:

Moreover, their number may vary (additional sprites are drawn only for certain distances):

29.

Some of the competitors' cars had to be modified based on how they looked on the track. For example, this convertible had three drawbacks: it looked too small, it was rotated too much, and it raised questions about whether it had a driver and passenger (a passenger was necessary to allow mirroring).

All these shortcomings were corrected in the final version, the last of which by "installing" the roof.

30.

We tried making one of the cars black for more variety. It did not look good in the screenshots and was removed. It might still be used in a future game, though.

31.

A small "modification package" was prepared for one of the cars to make it into a slightly more modern vehicle.

This feature was excluded from the final game: the modifications did not change the appearance too much (for sure, few people would have noticed them), but they made the car look worse. By this time, there was practically no memory left, and we cut off any questionable details from the game.

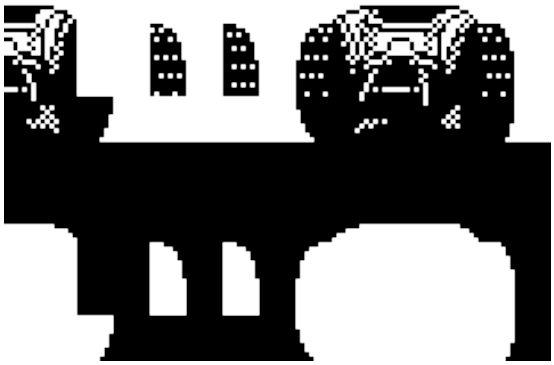
32.

The last of the competitors' cars, the most modern one, was completely redrawn, as the first version turned out to be drawn at suboptimal angles. Although the final version does not look very different at first glance, it looks better on the road.

To find the right angles, we created a three-dimensional model. Since the car is only visible from behind, the front part was not included in this model.

33.

One level uses a tractor as the game's vehicle. Like the cars, its basic sprites are in .tts format, but it has a number of interesting features.



Firstly, there are far fewer sprites than for cars: only one degree of rotation is represented. Secondly, to save memory, only half of the sprite depicting straight-ahead movement is stored; the other half is displayed mirrored. This is also the only game vehicle that shows wheel movement. Finally, there's the front of the tractor, which is represented in a different format, with an automask. This is necessary for smoother movement of this element. And lastly, Sven's torso is rendered.

In gameplay, the tractor's maximum speed has been significantly reduced, which is logical. It must be said that at high speed, it would be impossible to complete this level, where you have to drive side by side with Uncle Björn.

34.

The smoke from the exhaust pipes of the cars and tractor is realized using this set of small sprites with automasks:

A pair of these sprites move upward and gradually change appearance from the first to the last sprites in the set.

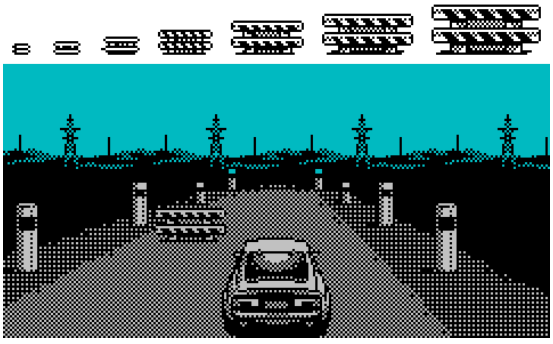
There's a car with exhaust pipes on both sides, and for it, the smoke doesn't appear twice, but constantly shifts sides. It actually looks "even better".

When the cars start moving, the smoke disappears. Only for the tractor driven by Sven does it appear continuously. This procedure was originally intended for this vehicle. The smoke sprites are even stored in the same set as Sven the tractor driver and the tractor part.

35.

Technically speaking, the barriers on the road are the same as cars. Most of their AI and engine sounds are disabled, and their speed is set to zero. The collision detection procedure is the same, and rival cars interact with them in the same way as they do with each other.

Visually, the barriers have transparent parts between the slats. In reality, there is a texture painted there that repeats the texture of the road, since this type of sprite uses an auto mask, which does not allow transparent parts inside the sprite.



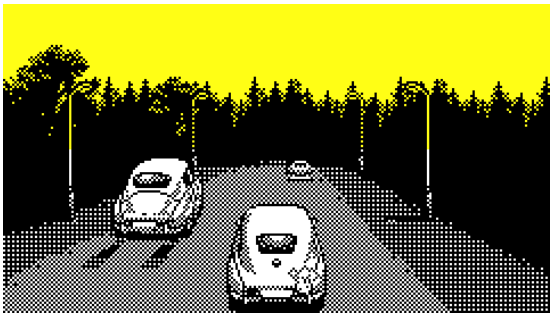
36.

Some objects (barriers and turn signs) can be knocked down. (Or they could, but the player is prohibited from doing so, like in the slalom level.) It works like this: when a collision is detected and processed, the original object is removed, and an animation is triggered showing it flying off.

This animation is completely unrelated to the original object and, in fact, isn't even tied to game coordinates: the collision always occurs in the center of the screen, next to the car, and to animate the objects flying off, it's enough to use the same sequence of screen coordinates each time. Regardless of the car's speed and its position on the road, a knocked down sign moves sideways offscreen and slightly upward, while a knocked down barrier moves up and down.

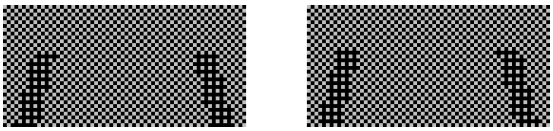
37.

This mock-up shows tyre marks when an opponent's car starts. It never made it to implementation. The game only features the braking marks of the player's car.

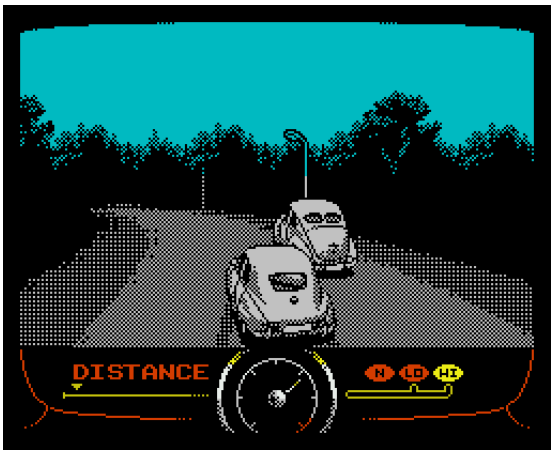


38.

Braking marks are a procedural output onto the road surface of a pattern like this one, consisting of two frames with fixed coordinates:



There's no perspective change or fade-in/out animation. But it looks perfectly acceptable in-game. And, of course, we didn't forget to disable this procedure for the motorcycle level.

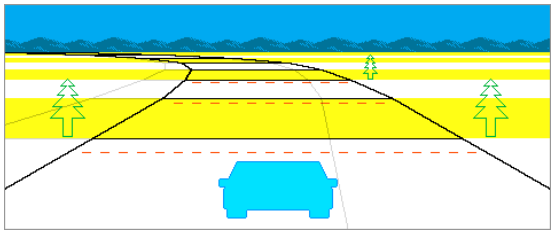


It could have been done with sprites, as in the game *DRIFT!*, but the procedural method is more compact and faster.

39.

Let's examine the general principles of calculating and displaying roads on the screen for graphical engines of this type.

The road in the map consists of segments of equal length, each of which has a set of properties that may vary depending on the engine: turn value, roadside objects, road width, markings, and so on. Some games specify the height (or height increment) for building real ascents and descents.



To display the road on the screen, in addition to the current segment (and, of course, the characteristics of all visible segments), two more values are required: the position of the vehicle on the segment (i.e., the distance travelled from its beginning; as a rule, it ranges from 0 to 255) and the transverse position of the vehicle (and camera) relative to the centre of the road. These parameters are used to calculate the location of the road boundaries and of the related roadside objects on the screen, and to construct a pseudo-perspective.

The number of segments displayed is usually small (in *TTT1* there are 8 on the screen). The segments are always perpendicular to the camera; in turns, they can only be shifted left and right (unlike in real 3D perspective, where we would notice their rotation). By the way, a consequence of this are the "physical properties" of such pseudo-turns: the distance travelled by the car along the inner radius is the same as along the outer radius. In advanced engines, this can be adjusted, but that is a separate topic.

In the schematic diagram above, the main lines show an approximate representation of the road for a situation where the vehicle is somewhere at the beginning of the segment and in the centre of the road, and a turn is visible ahead. In accordance with perspective, the screen height of the segments decreases towards the horizon, and the edges of the road converge (usually not completely) towards the centre of the horizon when there are no turns. Since the perspective is not real, its accuracy depends solely on the programmer's vision.

When the car starts moving, the height of the segments on the screen begins to change (dotted lines), and with it the location of objects. And when the car and camera move across the road (either the player steers or the car "skids" in a turn), the perspective changes: as one approaches the right shoulder, the right edge of the road becomes more vertical, and the left edge becomes more horizontal (light grey lines in the image).

The road calculation procedure must take into account all of the above and output at least a set of horizontal coordinates for the left and right edges of the road for each line displayed on the screen. In the case of "real" descents and ascents, when each segment also has a height (not screen height, but "above sea level"), the procedure will be more complicated.

With the most accurate engine implementation, the road and objects on the screen are drawn segment by segment, starting with the farthest visible segment. First, the shoulder and road textures are rendered (textures may vary from segment to segment) along with curbs (if any), then roadside objects (the best option is to bind objects to the far edge of the segment), and then competitor cars. Yes, they really should be rendered here so that they are correctly occluded by nearby objects (this happens on sharp turns) and nearby road segments (if the road changes elevation sharply).

One can add a check to avoid rendering the texture of segments hidden behind nearby segments. However, roadside objects and cars must be rendered in any case – they should stick out from behind hills.

40.

Even if described briefly, calculating and rendering roads is not a simple procedure. But the funny thing is that their implementation in the *TTT* engine violates many of the "canons" described above. For example, the road is not rendered segment by segment, but completely before the objects are rendered, and from the bottom up, and in general, it is not the road itself that is rendered... But more on that later.

For now, let's just note that one third of the screen, 64 lines, is allocated for drawing the road, and the road boundaries are calculated not for each line, but for pairs of lines, i.e. we get 32 values for the left kerb and 32 for the right. First, these values are calculated for the segments, and then a very rough interpolation is performed for the lines between them.

41.

In TTT1, each segment of the road map is described by one byte. Only the direction of the turn is specified (the two most significant bits; if both are off, the road is straight), and the remaining 64 values are assigned to objects. The degree of turning, which can only assume 2 possible values, is determined by the sequence of values of two segments: 0-1-0-1... is a smooth turn, 1-1-1-1... is a sharp turn.

Changes in road width and ascents/descents are not specified for each segment, but are implemented using special procedures.

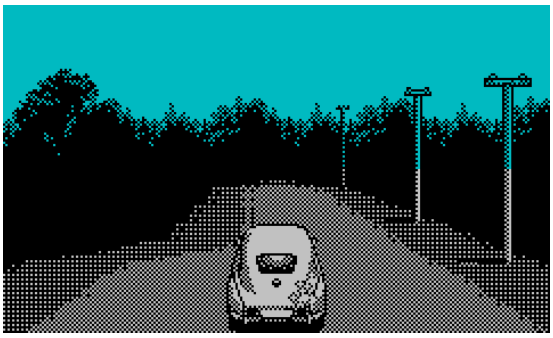
42.

When the speed increases, the turning angle is limited. This can be seen in games far beyond the ZX Spectrum. At the same time, we absolutely needed an additional sprite for the player's cars to allow them to move at very low speeds, almost perpendicular to the road (and on sharp turns).

The transition to such sprite is somewhat abrupt (especially for the vehicle shown above), but it was unavoidable. Only the player-controlled tractor was spared, but it's essentially simplified and has its own unique turn visualization features.

43.

As soon as the map encounters a "Road narrowing (right or left)" sign, a counter for the procedure that will perform this narrowing is started. The counter increases with each segment passed, and soon the road boundary calculation program will begin to apply the narrowing on the specified side.



The counter does not stop there, and once reached another threshold, the road returns to its normal width in the same way, then the procedure is turned off. One can also force the narrowing to end by selecting the "turn off special objects" marker on the map.

However, behind the "Road narrowing (on both sides)" sign, no actual narrowing occurs. Rather, the warning signals a narrowing of the roadside on "difficult sections", as is usually the case in reality.

44.

Ascents and descents are a separate, larger topic. Let's start by looking at these two images:

Which one is a descent and which one is an ascent? Or maybe it's just a narrowing of the road? In fact, there is no clear answer. If we focus only on the road, we can see the first image at the beginning of the descent (the road plunges somewhere), and then, when we are already descending, we see the second image: at the bottom, the descent ends, and the road returns to horizontal. The same applies to the ascent, only at first we see the second image (the road goes uphill in the distance), and then, as we approach the top of the ascent, we see the first: the horizon and the entire landscape are still hidden behind the top.

Accordingly, in order to understand exactly whether it is a descent or an ascent, it is important to see the change in the picture, but even more important is the location and movement of the horizon line and other objects. Nothing conveys descents and ascents as well as the movement of the background.

So, as we approach the ascent, the road in the distance, along with the horizon line, begins to rise very slowly, as if creating a foundation for the future ascent. And at its beginning, the background rapidly collapses downwards, the road bends accordingly, and now we see mainly the sky. At the top of the hill, the road straightens out, and the background returns from behind the horizon.

That would be fine, but how can we implement this in our engine if it does not initially have the functionality to elevate segments? Well, we are not the first to face this challenge...

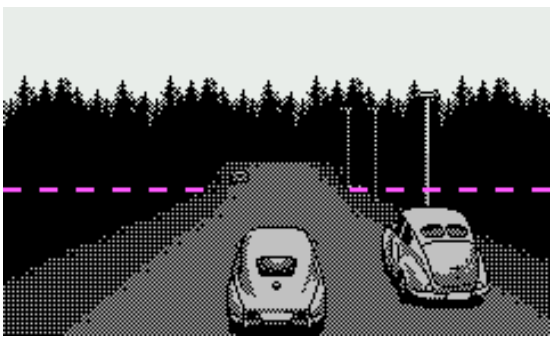
The road boundary calculation procedures use rather crude methods that, when there's a descent or ascent, additionally "bend" the road. This method is extremely "fake", but it looks good in the TRAVEL engine, especially when large mid-ground objects emerge from the horizon on an ascent. Descents, however, are a little less well done and don't look as impressive.

By the way, it's interesting to note, when playing various racing games for the ZX Spectrum, how differently descents and ascents appear. Sometimes the horizon remains fixed, while the road rises from below like a sheet of paper (*Full Throttle 2*, *Beverly Hills Cop*). But this isn't the implementation one should aim for.

45.

As we said, the road, along with the horizon line, sometimes rises upwards. But how can we display it if we only show the road in one third of the screen? If we calculate its boundaries for only 32 pairs of lines?

As trivial as it may seem, we need to calculate a few more values and display an additional section of the road (in addition to the eight displayed segments) beyond the one-third of the screen. Of course, this raises a number of questions: after all, we display road objects only for eight segments. We took an interesting approach: for distant cars, we added a correction that slightly raises the car's screen coordinates when there's a road receding into the distance, as shown in the following screenshot (the dotted line marks the end of the main section of the road). Yes, this is actually a car from the eight displayed segments, but visually, it's already in the additional segments. Another "fake" for the sake of visual beauty. And what about the poles in this additional section of the road? It's simple: they aren't displayed. Now you know.



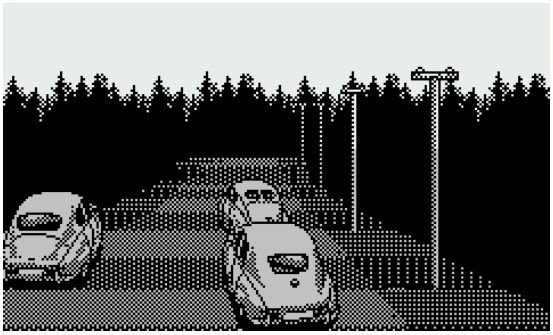
The procedures for rendering the additional part of the road are also closely related to rendering the background, cleaning the necessary areas and drawing the road texture.

46.

Another nuance: when the background moves up and down, the colour attributes at the horizon also change. In the game, colour is not formed in the screen buffer, but is always displayed directly on the screen. The two parts of the game window at the beginning of the level are coloured in specific colours representing the sky and the ground. Accordingly, only the lines on the horizon are recoloured during the race. Sometimes this is noticeable. Many other racing games do the same.

47.

Sections of "uneven road", preceded by the appropriate sign, are visually represented by a striped texture, and the car rocks back and forth on them.

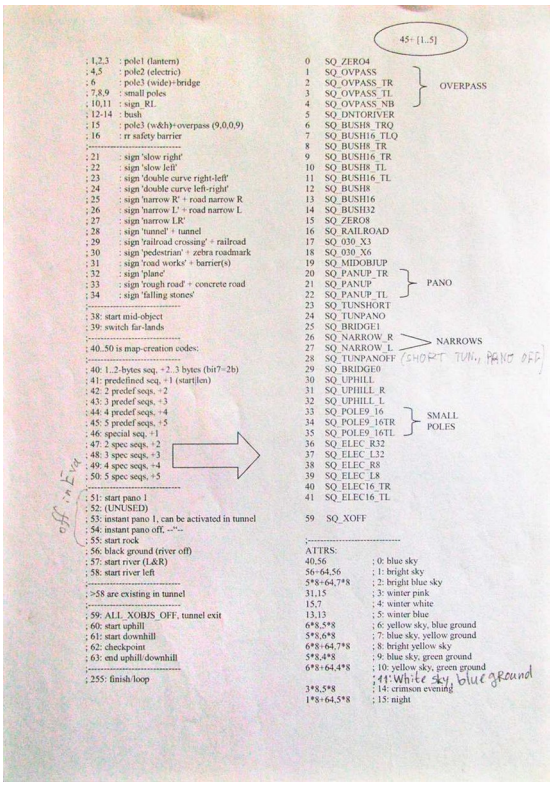


Please note: stripes are also displayed on the additional section of the road, but they are (what can you do?) fake again. Basically, it's all a game of smoke and mirrors. But in the second part of *TTT*, the procedure has been refined, and the striped texture of the additional section of the road is displayed more correctly.

The same texture is used in tunnels, but there it plays the role of a smooth road.

48.

This sheet contains a complete list of objects and predefined sequences found in the level map.



At the bottom of the sheet, you can also see the sky and road attributes used in the game.

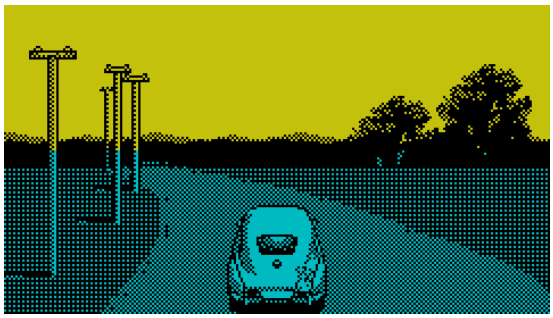
You can see that there are not many notes. In fact, this is just the final printout of this sheet, made towards the end of development.

49.

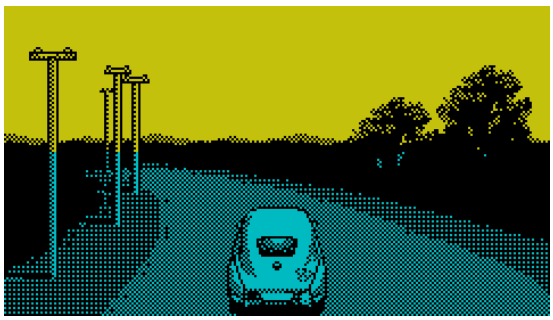
The principle of drawing roads is quite unusual: in the game, it is not the road itself that is drawn, but the roadside and the "ground" behind it. The road itself is nothing more than the initial clearing of the entire third of the screen, filling it with a texture, either a uniform "checkerboard" or a more complex version with a striped road, where the texture changes from segment to segment. There is also the possibility of a smooth transition between these textures.

Without rendering the roadsides, the road looks like this (for clarity, we show the transition from a uniform texture to a striped one):

Then the roadsides are drawn, in two passes. In the first pass, the first line from the bottom is drawn (left and right), then the third, and so on through the other lines. To do this, 32 pre-calculated values are used for the left and right edges of the road.



Then the second pass: the second line from the bottom and so on, one by one, using the same values, but reduced by the width of the roadside, which decreases towards the horizon. That is, these lines are shorter than in the first pass, and as a result, we get an image of both the roadside and the ground behind it.



In the screenshot above, you can see that the lines in the second pass, in addition to being shorter, often start higher, not from the very bottom of the screen. Therefore, the second pass runs significantly faster.

It is now difficult to say why such an unusual method of displaying the road was chosen initially, but it is used by all games running on the TRAVEL engine.

As mentioned earlier, in the most correct implementation of such engines, segments are drawn from the horizon downwards, simultaneously with all objects and vehicles. In our case, the direction of road rendering is irrelevant (unlike objects that are rendered later).

50.

In the image with missing road shoulders, a certain graphic element of the road is clearly visible on the horizon. The issue is that our image of the road, which covers one third of the screen and consists of eight visible segments, has a drawback: it ends too abruptly at the horizon.

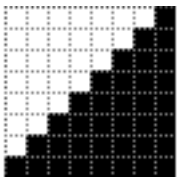
To correct this flaw, a couple of additional lines are drawn above the road on the horizon, visually extending the road and following its turns. This graphic, which we have tentatively called the "distant road", is displayed only under "favourable" conditions.

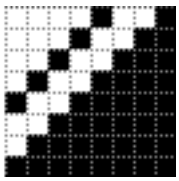
51.

Another element of this purpose can be seen in the same screenshot: these "distant poles". Eight segments are not enough. The poles appear too abruptly. So we decided to track the contents of one more segment (this is fast, as we are not calculating the road boundaries) in order to display poles that have not yet entered the main field of view. This is done for two types of poles – streetlights and power lines. Moreover, it is not the main graphics of the poles that are used, but a separate, very fast procedure that draws points (inverting pixels) across a line. Consequently, at the farthest distance, the two types of poles are identical.

52.

You have probably noticed the "ragged" display of the roadside. This is due to the fact that the horizontal lines for the last byte are drawn using a noise-injected table rather than a typical one:





The "ragged" roadside actually solves three problems at once:

• It brings our image closer to the first initial screen layout, which was based on a photograph.
• It hides the rather crudely calculated road boundary.
• It adds dynamism to the monotonous texture of the road.

Without noise, the road would look like this:

53.

The rain in one of the cutscenes is a cut in half variant of a procedure from our game *Rikki-Tikki-Tavi*.

54.

The principles behind the implementation of winter levels are quite interesting.

As we already know, our engine does not render the road itself, but rather the roadside and the ground behind it, and everything is initially designed exclusively for a “black” ground fill (unlike subsequent games on this engine, where the ground is either “white” or both options are available, as in *TTT2*). So, to depict winter, we simply used inverted colour attributes. Now we see light-coloured trees against a dark background, and so on. Look: even the shadows from the posts have turned white against the dark background!

But what about the cars? After all, when the attributes are inverted, they should look like this:

To restore the normal appearance of cars when attributes are inverted, sprite inversion is used. For the player’s car (the .tts format described above), a procedure is performed during level initialisation that inverts the sprite data, taking masks into account. These changes are reversible, and when “spring arrives”, the sprite returns to its original state (the same procedure is performed again).

We can’t use this method for rival vehicles (whose sprites use automatic masks). Therefore, we’ve written a rendering routine for them that adds on-the-fly inversion. It’s slow, but we don’t have many winter levels.

The exhaust smoke also uses the automask format, and it displays correctly in winter.

The car’s behavior has also been slightly modified in winter levels, making it drift more.

55.

There is a very unpleasant issue when testing games: the case when individual levels and cutscenes may run correctly, but when you try to play through the entire game, at a certain point a critical error suddenly occurs, the game freezes and crashes. Moreover, it is difficult to catch, because the conditions are unclear: for example, it may happen on level 70 or after level 75.

Of course, no one would have the patience to play through the game multiple times after making changes, even with cheats. So in *TTT*, we used an auto-play mode for this. The levels are completed in a very rough manner – collision detection is disabled for the cars. This build can be run in an emulator at twenty times the speed to quickly see if the game will reach the end.

However, each individual level was, of course, also completed many times in a honest way to find the right balance.

56.

To test *Travel Unlimited* (in which, incidentally, we encountered a similar error at higher levels), an even more advanced automatic gameplay system was developed, capable of navigating turns and overtaking opponents. It does not perform perfectly, which is actually beneficial: it roughly corresponds to the skill of an average player, not always completing the level on the first attempt. In addition, the game also automatically “purchases” upgrades, without which the higher levels cannot be completed.

So, we turned on the emulator and watched the game play itself.

Another very useful mode used during debugging and testing was “show cutscenes only”.

57.

The game only runs on computers with 128 kilobytes of memory. However, this is mainly due to the large volume of various additional objects and cutscenes, while the TRAVEL engine itself can handle 48 kilobytes.

Unlike *DRIFT!*, it uses a screen buffer rather than the dual screens typical of 128K. Some other games using this engine run on 48K.

58.

“There are no road markings in the game” sounded like a reproach to us, but that’s not entirely fair. We just didn’t want to repeat what we had seen many times in other games, so instead of longitudinal markings, we used transverse ones: pedestrian crossings, the finish line, and a couple of simpler elements. And we can definitely be proud of how it looks.

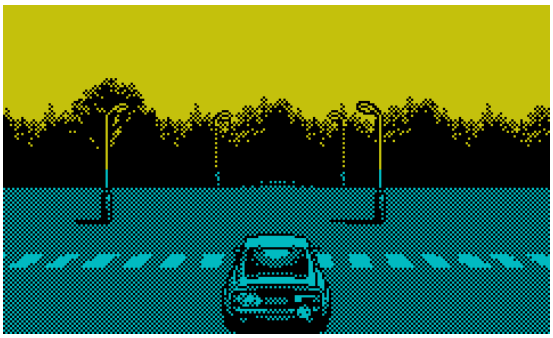
As for the longitudinal markings, there are no problems with them, and we have already inserted them in the second part of the *TTT* series.

59.

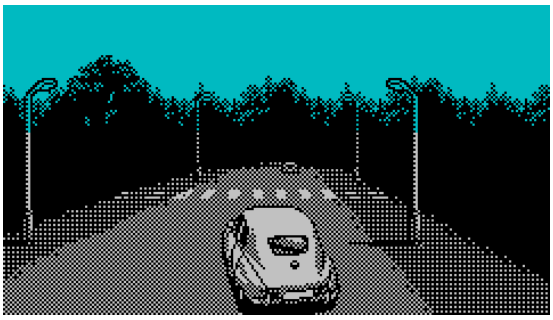
The following set of textures is used to display pedestrian crossings and the finish line:

The zebra crossing is depicted three times in it, for different distances, but for the finish line we decided to use only one variant, simply drawing a piece of different heights at different distances.

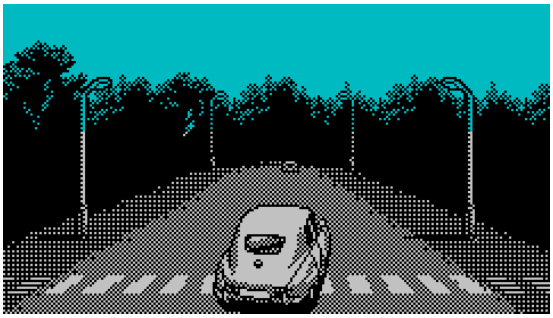
The method of drawing horizontal markings is based on the same principle of drawing a road, or more precisely, road shoulders: the markings are drawn from edge to edge of the screen, and then cropped by the shoulders, so that we only see the part on the road. If we do not draw the shoulders, we will see the following image:



So, the issue of trimming the road markings at the edges has been resolved. But what about the correctness of the perspective? To be honest, there’s nothing we can do about it. The markings are drawn exactly as they appear in the texture map. For example, just count the number of stripes here:



...and here:

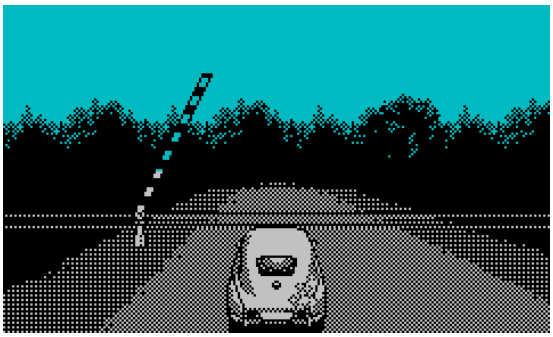


But in motion, everything looks just fine!

60.

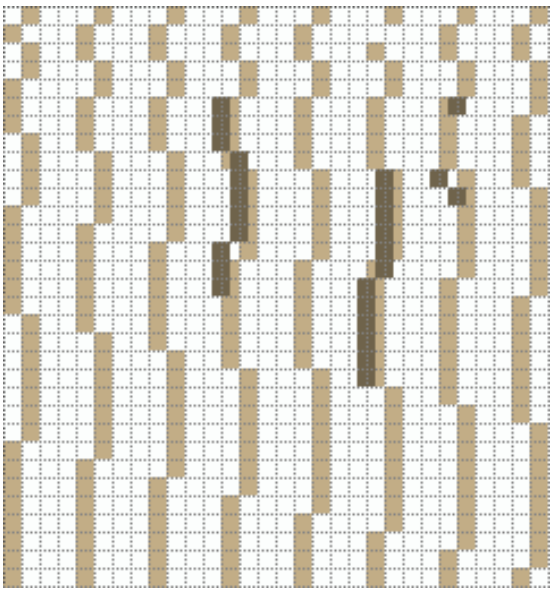
Checkpoints and rails are drawn in a similar way, but instead of using a texture, a single byte is repeated across the entire width of the screen.

Rails must extend beyond the road, so unlike markings, they are drawn after the road edges.



61.

Based on this image with cells, a table of visual dimensions of road segments (their screen heights, i.e. the number of lines) was developed:



The screen always displays 8 segments. The cell in this image is equal to the minimum possible height of a segment on the screen: two lines. 32 cells mean 64 lines of height, or one third of the screen, which is where the road is displayed. The figure shows the screen heights of the segments for 8 frames. This is the degree of precision with which we see movement on the screen. The position of the car on a segment, which has a range of 0..255, is reduced to a value of 0..7. At the start, you may notice that the road and roadside objects move somewhat jerkily at low speeds, and movement begins slightly later than the start of acceleration.

The height of the nearest segment ranges from 8 to 1 cell (16 to 2 lines), while the farthest segment shown always has the minimum height of 2 lines. The segments in the image are represented by a broken line simply for the convenience of visual perception during development, nothing more. The dark cells are corrections for some frames after testing the first version.

The final table (in pairs of lines) is as follows:

```
RDSEGLN     DEFB    8,7,5,4,3,2,2,1
            DEFB    7,7,6,4,3,2,2,1
            DEFB    6,7,6,5,3,2,2,1
            DEFB    5,7,7,5,3,2,2,1
            DEFB    4,8,6,5,4,2,2,1
            DEFB    3,8,6,6,4,2,2,1
            DEFB    2,8,7,5,4,3,2,1
            DEFB    1,8,7,6,4,3,2,1
```

62.

Before being hit by a train, the car passes freely through a closed railway bar gate. It turns out that in the face of such a disaster, this detail is completely unnoticeable, so we didn't waste any extra bytes or time on it.

Incidentally, the first level was deliberately designed in such a way that it was practically impossible to avoid contact with the train when playing for the first time and not knowing about it in advance.

63.

There are many types of races in the game: CHALLENGE, CHECKPOINTS, TIME TRIAL, DUEL, CHASE, JUST DRIVE, SPEED SCORE, and SPECIAL EVENT. Here are the icons for each of them:

In fact, there are even more types of races, because the last "special" type includes a number of completely different competitions, and even with the participation of unique vehicles, both in-game and traffic. One of these levels, "log trucks", even has its own icon on the status bar. Why not a separate type?

Some types of races occur once (CHASE), twice (JUST DRIVE) or three times (SPEED SCORE) throughout the game, while CHALLENGE, CHECKPOINTS and TIME TRIAL are the main types of competitions.

64.

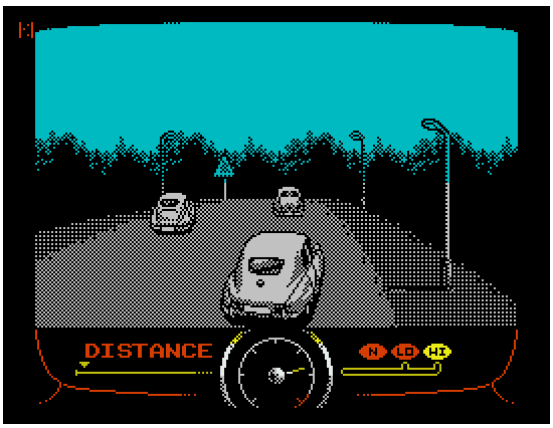
Each type of competition has its own AI for vehicles on the track (opponents, traffic, special vehicles). The behaviour of an opponent in a duel is certainly different from that in a race, and a duel on public roads is a unique competition with its own AI characteristics. However, there is also a large set of procedures that are common to all cars in most cases.

In races the number of cars on the track remains constant throughout the competition, but when driving on public roads the traffic is regularly updated: those that have long been overtaken are eliminated, and new ones are created ahead.

There's also an interesting behavioral model with the telling name "can't be overtaken". It was later used in *Rubinho Cucaracha*.

65.

To help the AI of rival cars quickly and effectively assess the road situation, we used a technology we called the "situational table." There's even a debug mode that displays the table visually. Here it is, in the upper left corner of the screen:



It contains the approximate location of the cars – both the player's and the opponents'. The table is small in size, allowing the AI to analyse the situation on the track quickly and efficiently. Without it, the behavioural procedures for each car would have to go through the parameters of all the other opponents' cars, as well as the player's car separately, and all this in an inconvenient form. It is difficult to even imagine how slow this would be.

It is thanks to the situational table that opponents can slow down behind slow cars, overtake (but without changing lanes if the player's car is next to them), and try to pass a player who has slowed down when they appear behind them. It doesn't always work, but they try.

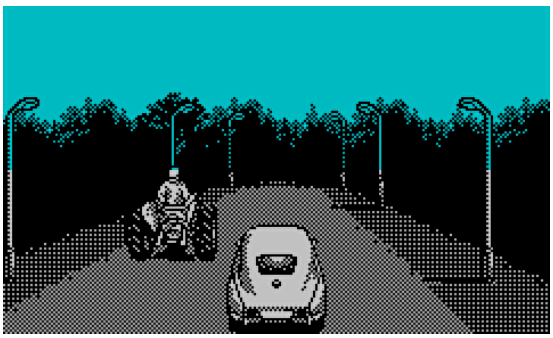
This technology played a crucial role in the gameplay of games using the TRAVEL engine. Many other highly advanced racing games (we won't point fingers) lack this feature, and the cars simply "don't see" each other or the player.

66.

The speed of the vehicle does not decrease when driving uphill; it does not depend on the terrain.

67.

Is it possible to hit Uncle Björn's tractor on the first level?



When Sven's vehicle is near the tractor, it automatically moves to the opposite side of the road, and attempts to move back are blocked. However, there is no strict enforcement, and indeed, you can achieve this, by moving in advance to the left roadside and accelerating. But this will not make any difference. On this level, collisions are allowed, and as soon as Sven catches up with the tractor, he and Björn will start a conversation.

68.

But what happens if you miss the tractor, or follow it to the end of the level through all these crossings without talking to Uncle Björn?

It won't work. From the moment the tractor appears, a looped section of the track begins. We will only be "released" after the dialogue. The tractor won't go anywhere either, but will wait at the horizon.

69.

Finally, one more question regarding the episode with the tractor: it is necessary to remove any extra vehicles from the track, if there are any. This is easily accomplished: all vehicles in front of us accelerate sharply (this is very noticeable) and disappear over the horizon. The vehicles behind can simply be removed.

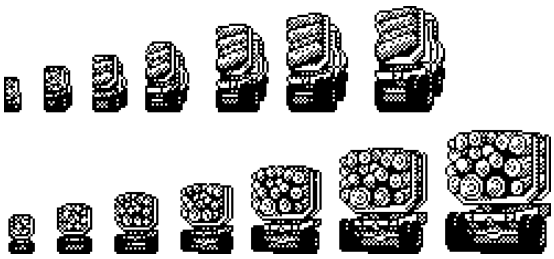
70.

All races in the game are point-to-point. There are no races consisting of multiple laps. This feature appeared in later versions of the engine.

However, in *TTT1*, some levels still have a looped section, which is implemented in a simplified way: at the "joint", the map has identical fragments, and at the right moment, the car, along with its opponents, is transferred to the specified fragment. Aside from the first meeting with Uncle Björn, the "loop" is used in the level with the timber trucks and during the duel with Johannes, where he gets into an accident. We don't really even notice it, and this method is only intended to close a certain logical hole during the race, which is most obvious in the episode with the tractor.

71.

The timber trucks consist of two vehicles that drive at a fixed distance from each other without changing speed or lane. This means that these are not just two sprites, but two data structures for vehicles. We also had to resolve several issues to ensure the visually correct positioning of their parts on the screen and their cropping when the timber truck moves off frame. The rear part takes into account the position of the front part, and a restriction is used on the maximum horizontal distance between them on the screen.



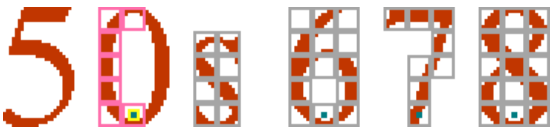
The front part, separate from the rear, looks amusingly narrow.

72.

The game is divided into four decades, from the 1950s to the 1980s, with most of the plot taking place in the 1950s and 1960s.

73.

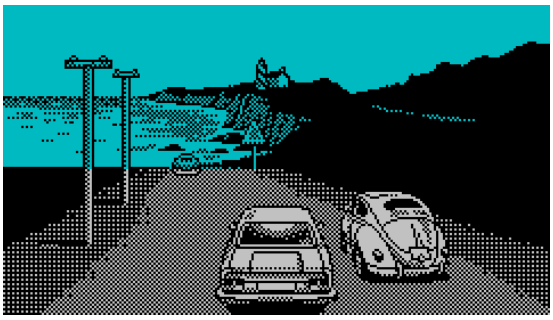
Before each decade, a splash screen appears with cars of that era and a large caption: “50s / 60s / 70s / 80s.” Naturally, these symbols are rendered using regular sprites. To save space, only a part of the number “0” is stored (2/3, to be precise), and the number “5” is stored separately, as it is displayed only once at the beginning of the game, and this memory area is subsequently used for the video buffer.



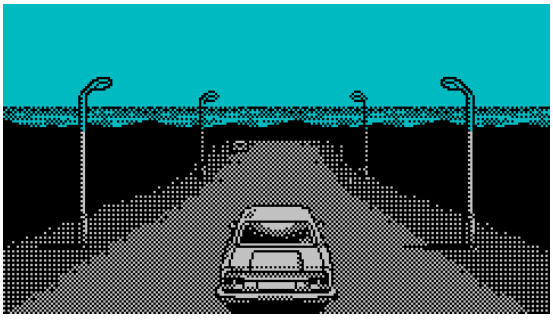
The font is also additionally packaged. The packaging saves just over 100 bytes, but in large projects, all optimisations are important – the main thing is that they do not cause harm.

74.

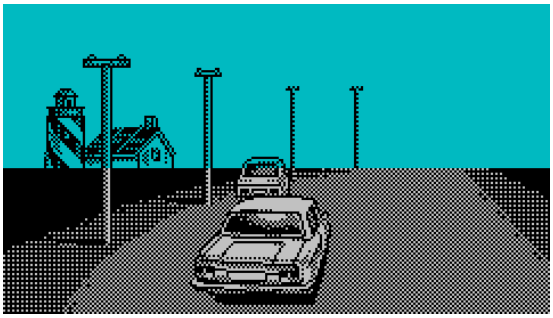
Panoramic shots often logically connect to other objects. For example, at level 45, we are treated to a beautiful view of the rocky coastline and lighthouse.



Then, after a short tunnel, the panorama changes to the open sea...

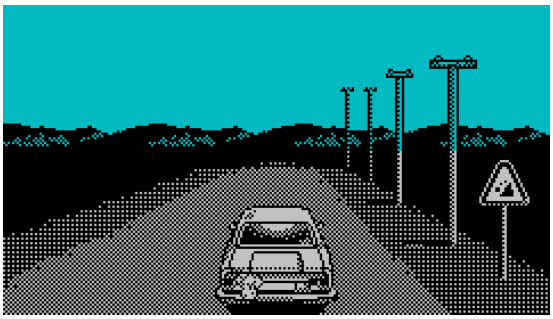


… and then, rising in front of us from behind the horizon, that same lighthouse appears, already close by.

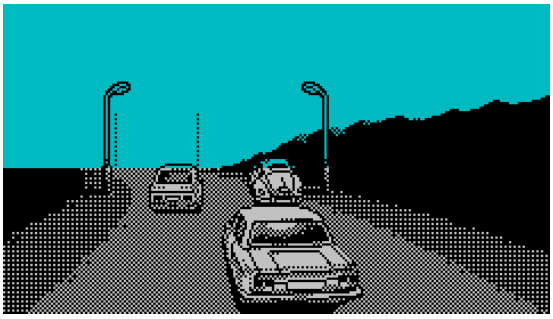


75.

This level (which, incidentally, appears after about an hour and a half of speedrunning) is notable for its detailed map design, with multiple changes of scenery and a wealth of other details. For example, the race begins at a “Falling rocks” sign.



Shortly afterwards, the road passes by a rocky ridge.



Its rendering utilizes separate rendering procedures, allowing the rocks to follow the curves of the road. The graphics format is the same as for some larger interface elements – with simple column-wise packing, which lends itself well to the uniform black fill of the rocks.

Interestingly, just two sprites were used to depict the dynamic elements, which, combined with the posts, create a good sense of movement.

76.

The game font includes the letter ö. This is specifically to write Uncle Björn's name.

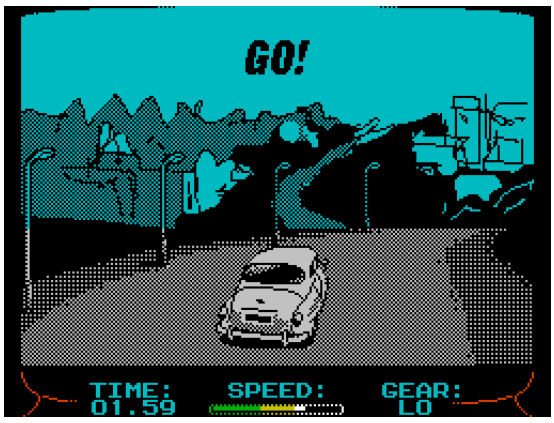
As one can see, some elements of the graphical interface are also stored as font characters and use the corresponding output procedures.

77.

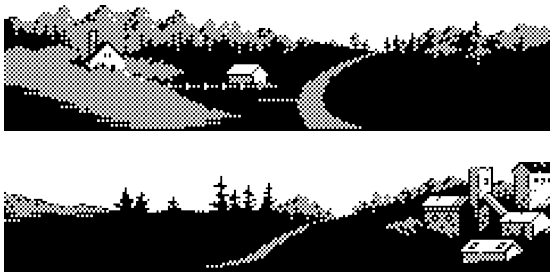
The arrow showing the distance travelled is one of the font symbols. However, it is only printed at the start of the race, and then a scrolling procedure is used to move it smoothly, moving the pixels directly on the screen. Due to the specifics of the procedures, it is impossible to precisely synchronise the calculated distance and the movement of the arrow, which is why the distance line on the screen does not end with a cut-off, but turns into dots: "the finish is somewhere here".

78.

Some panoramas were drawn based on photographs, others were drawn entirely by hand. For example, this screenshot shows the first sketch for two future panoramas:



As you might guess, at that moment it occurred to us that it was better not to overload the panorama with a large number of objects, but rather to prepare as many different panoramas as possible.



In the same screenshot, you can see one of the status panel options with a ribbon speedometer and a slightly enlarged playing area, which was not used in the final game.

79.

Regarding the size of the playing area: its height on the screen is 19 lines, and another 5 are taken up by the status bar. Actually, the top lines are "fake" – they always display a single-colour sky fill. The actual screen buffer consists of the typical 2/3 of the screen (16 lines), as in most other racing games. In addition, the top line is only used when necessary, and usually only 15 lines are displayed, which again slightly increases the speed.

The only detail: when driving under overpasses and in tunnels, the upper "fake" part of the game area has to be temporarily filled with black, but this is quite simple.

The same method with visual screen expansion is used in many subsequent games based on the TRAVEL engine: *Rubinho Cucaracha* (which features stationary clouds), *Travel Unlimited* (where the striped gradient has some dynamics on descents and ascents), and all parts of the main *TTT* series.

80.

In some levels, the main character has no chance to reach the podium, or must get on the podium but it's unable to win the race. This is implemented in a hilarious way: the cars that must finish ahead are already at the finish line! After all, at the start, we can't see the entire starting grid anyway.

81.

In total, the game uses nine types of panoramic shots that "pop up" from the horizon. Of course, we rushed to show the most beautiful and detailed ones in the early levels. Later on, they became much simpler, like this one:

But this was not because we were lazy, but to save memory. After all, panoramas are stored in a compressed format, and while the first one takes up almost a kilobyte, the simplest one requires less than 150 bytes. We only saved one of the most detailed panoramas for the aforementioned level 45.

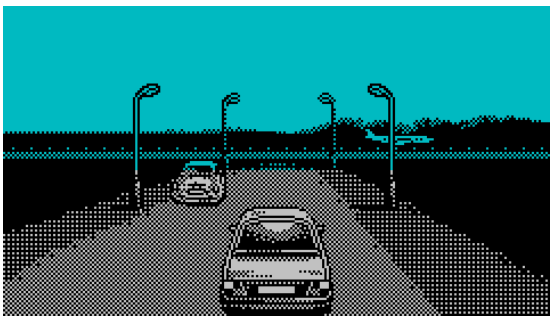
82.

Technically, the output of panoramas is arranged in a very interesting way: as already mentioned, they are stored in a packed form (a specially created format), but they are unpacked not as graphic data, but as a ready-made program for displaying these graphics on the screen as quickly as possible – "hardwired" into the code.

Of course, a separate buffer is required for unpacking. For this purpose, we used the memory area which stored the data for the train, which we will no longer encounter in the game. This area is almost identical in size to the largest of the unpacked panoramas.

83.

At level 55, we see the airfield, the simplest of the panoramic shots. But it's "decorated" with a plane taking off.



Keeping in mind the tradition of logically connecting objects, we soon see the airport control tower rising above the horizon.

Well, at the very beginning of the level, we already saw the "Low-flying aircraft" sign, and now we are convinced that it was there for a reason.

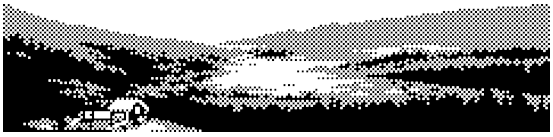
84.

Incidentally, the control tower was modelled after the old (now defunct) airport tower in the Finnish city of Oulu. After all, the action of the game at these levels has moved to Finland, following Eva.

85.

As already mentioned, some panoramic plans were deliberately simplified to save memory. For example, this panorama...

...originally looked like this:

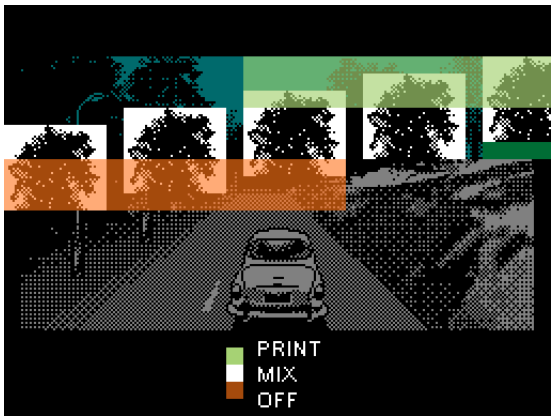


86.

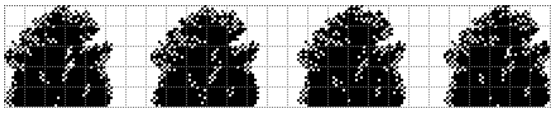
In addition to the taking-off aeroplane, we planned to depict a moving ship in one of the panoramas. We didn't really like how it looked, so it didn't make it into the final version of the game.

87.

The following mock-up was used for research work on how to display trees in the middle ground: which parts should be displayed directly, which should blend into the background, and which could be omitted given the tree's location. All this was done in order to achieve sufficient fidelity with the original prototype while maintaining high speed.



The trees in the middle ground are not only the most unique object in the game, not found in any other racing game on the ZX Spectrum: they were also very difficult to implement. Many aspects had to be taken into account when moving the trees and displaying them on the screen. They had to not only move smoothly, but also follow the curves of the road to a certain extent (without giving the impression to be moving backwards), as well as "get along" with all the changes in the landscape and background (so that, for example, they would not detach from the ground on an incline). Since the landscape in the game quickly changes from closed to open, the trees in the middle ground should leave the screen in time. Their movement actually follows its own rules, not directly related to the current game situation, but rather adapting to it.



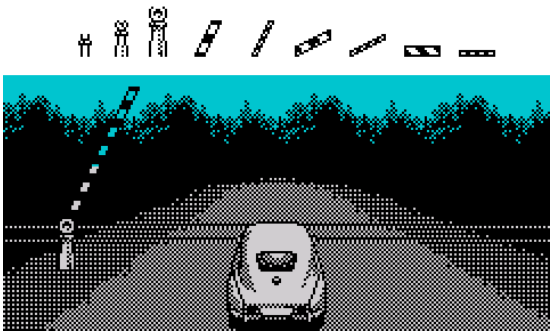
And here's another interesting detail: although it seems that the trees appear to grow larger as they move, this illusion is achieved solely by their gradual emergence (upward movement) from the background. In reality, only one tree of a single size is used.

But the horizontal offset, as you can see, is pre-determined. Moreover, copies with different offsets look slightly different from each other. In motion, this looks very interesting – it adds a kind of dynamic noise.

Naturally, trees use their own rendering procedures, taking into account all the features described above and allowing them to be smoothly moved off-screen. Exactly two trees can exist in the midground at a time – either on the same side of the road or on opposite sides.

88.

Drawing the railway bar wasn't as simple as it might seem: at different distances from the camera, it can also be in different states: closed, open, and in transition (opening or closing). Moreover, since it's positioned horizontally or diagonally, rather than vertically, our sprite "stretching" procedures aren't suitable. Furthermore, as the bar gate is a rare object, it should not take up too much memory. As a result, we composed it from separate pieces, the number of repetitions of which depends on the distance to the object. The crossbar has only two thickness options. It doesn't look very nice when driving at low speed, but overall, the problem is solved.



89.

The game logic dictates that every second railway bar on the level is closed in anticipation of an approaching train. However, only the first level has the two railway crossings. Starting from the second level, the memory area occupied by the train is used for other purposes (unpacking panoramic shots). Consequently, the train sequence is no longer possible, but the game never returns to the first level.

90.

The game uses manual gear shifting and a "two-speed" gearbox, which is so characteristic of arcade racing games. There is also a neutral position. First gear engages automatically when you start moving.

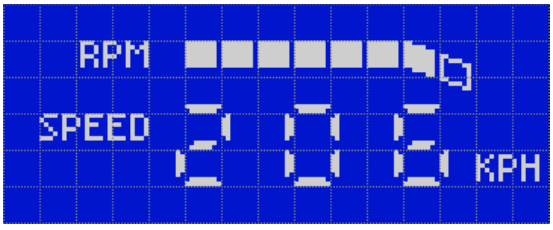
In some other games based on this engine (*Rubinho Cucaracha* and *Travel Unlimited*), gear shifting is automatic, and the joystick button activates the turbo.

91.

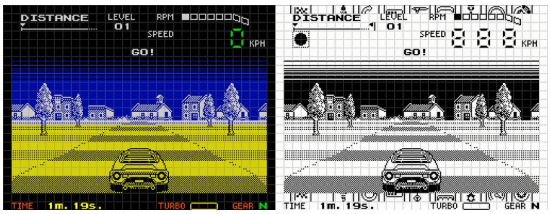
On the status bar, the current gear is shown by changing attributes – "highlighting" the corresponding icon.

This is perhaps the fastest and most economical way to display user information in the interface, and it is often used in ZX Spectrum games due to the characteristics of this computer's screen memory area.

In the same way, part of the interface graphics can be completely hidden: to do this, colour attributes with the same paper and ink values are used. This is how the digital speedometer in the game *Travel Unlimited* works: the pre-drawn value "888" on the screen is hidden via attributes.



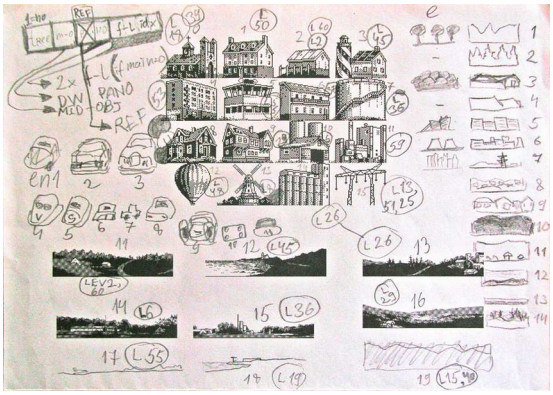
Moreover, in order to save memory, in this game all the graphic elements of the “garage” (upgrade icons) are hidden in two visually empty lines of the screen.



This approach is doable if the game does not require the screen to be completely cleared at any point. Sometimes, attributes even conceal part of the code. We did this, for example, in the game *Don Quixote*.

92.

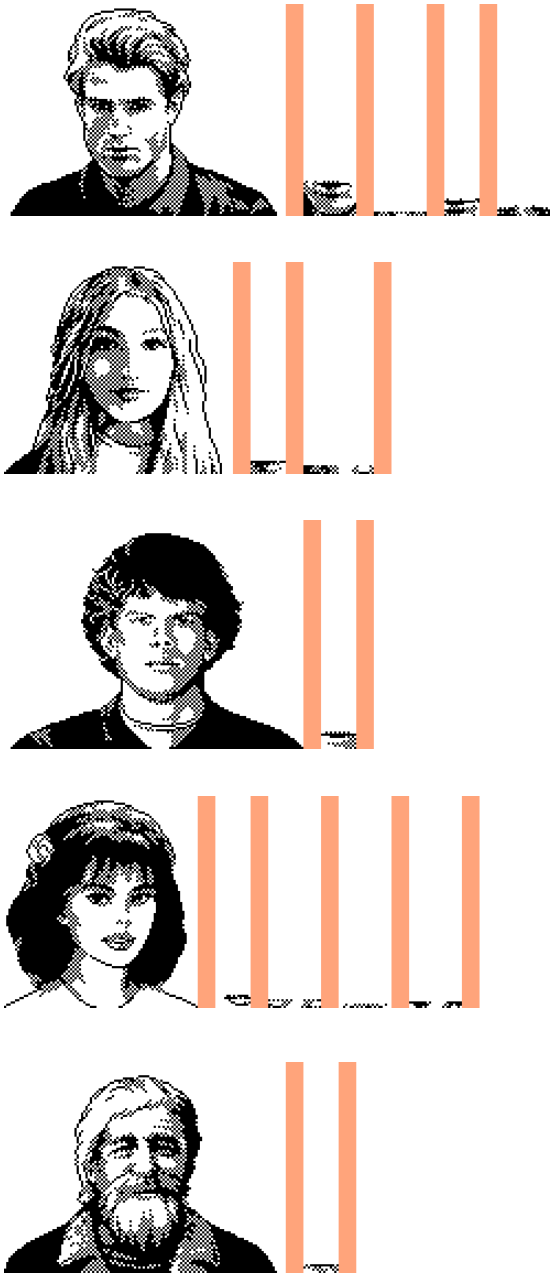
This is a very interesting sheet, exploring the use of panoramas, cars, backgrounds and “pop-up” objects on different levels.



Also, in the upper left corner, you can see the design of a one-byte format for the level header, where individual bits describe certain track parameters or the presence of such parameters (in order to read certain values later, if they are present). This compact recording method is important to us because we have so many levels.

93.

All character portraits were drawn based on random photos found online. The sprites use a simple column-packed format, which is further packed using a standard packer.



Sven and Irma have the greatest number of facial expressions (but the set is different). Johannes and Uncle Björn can only open and close their mouths.

Irma is the only one who is able to look aside: at Marie sitting next to her, and at Sven during the trip.

94.

The characters’ names were chosen based on statistics on the names of newborns in Sweden in specific years. For example, if Sven was around 20 years old in the mid-50s, then he was born in the mid-30s.

Indeed, this name was the second most popular at that time. And for the other characters, at the very least, the rule applies: “Yes, that’s what he could have been called”.

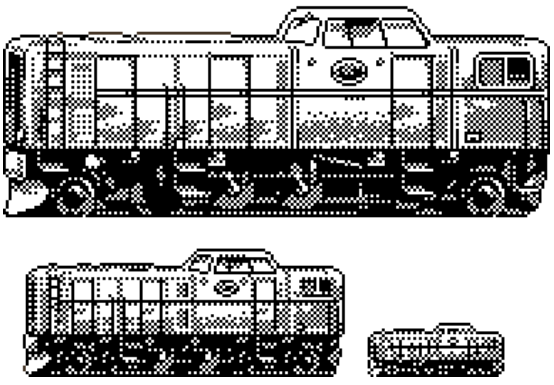
95.

The locomotive in the first level is not fictional; it is a representation of the SJ Tp model that was actually produced and used on Swedish railways. It was built in 25 exemplars between 1953 and 1954, so we can assume that the game takes place no earlier than 1953.

This same locomotive plays a rather dramatic cameo role in the film “A Man Called Ove”, which, incidentally, also devotes considerable attention to automobiles.

96.

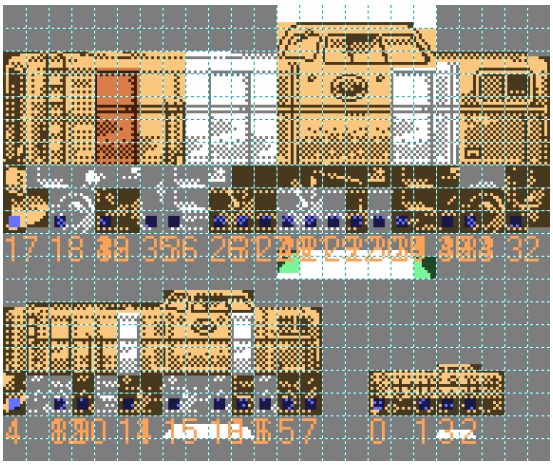
How many different scale variants are needed for the train sprite? If it were the same as for the cars, then given the locomotive’s size, it would take up a lot of memory. Fortunately, it can be displayed in a more discretized way: after all, if we manage to stop before the crossing, the train will no longer approach and grow larger. If we don’t, then the car is moving fast, and the train will also approach very quickly; the abrupt change in scale (only three variants) won’t be noticeable.



By the way, we mentioned the logic behind the appearance of the train: the program determines that we are travelling too fast and unleashes the train at the right moment. If we did not start braking in advance, it is too late to do so when the train appears. And even if the car has already passed the crossing visually, the train will still continue to move along the bottom line of the screen, and it will be impossible to avoid a collision. All this happens so quickly that one doesn’t pay attention to the details and compromises. Only if we are driving slowly enough and manage to stop before the closed railway bar does the train appear, not instantly, but after a short delay.

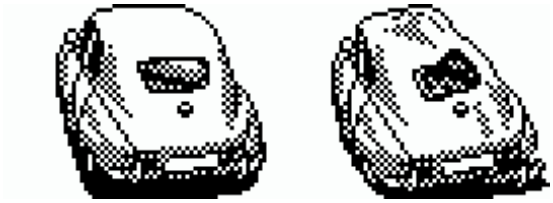
97.

Although the locomotive has only three scale options, it would still take up a lot of memory space given its size. Logically, in the game it is composed of a set of small sprites, many of which are repeated several times. This also solves the problem of image cropping at the edges of the screen, as cropping is not implemented for this type of sprite.



98.

We prepared an image of a car being destroyed in a collision with a train. It’s not used in the game, as it was barely noticeable in such a short sequence. Instead, the train quickly carries a standard car sprite off-screen.



99.

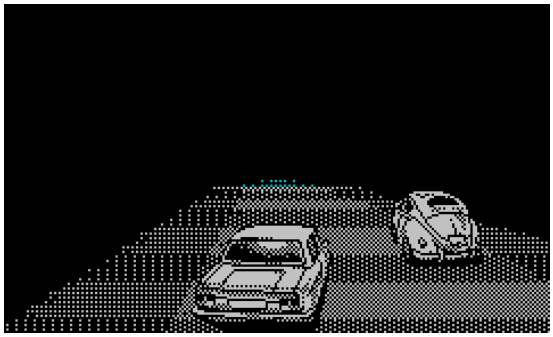
Another train-related detail that few people noticed: after the dialogue with Uncle Björn, there are no other cars on the road until the very finish. This is, of course, because they aren't designed to interact with the train. There would also be issues with the sprite rendering order, since the train is rendered outside of the object sorting procedures (which is why there are no roadside poles immediately after the crossing).

100.

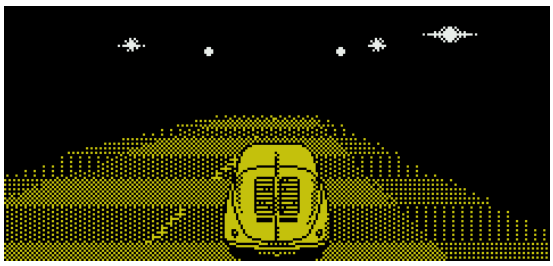
The tunnel is one of the most interesting objects in the game. Its implementation uses a large set of procedures. The most impressive parts are the tunnel entrance...

...and its exit.

This is achieved using fill procedures that draw walls and ceilings positioned along the road edges. The most important detail is the mandatory change of the road texture to "striped". After all, there are no objects in the tunnel itself, and movement is conveyed exclusively by texture. The same texture is sometimes used on some other sections of the road.



Why didn't we add lights like in *Chase H.Q.*? For example, to avoid repeating that game entirely, and also to save some memory and development time... Although, in the later second game, we improved the tunnels, primarily by adding lights. Also, in *TTT2*, cars finally started hitting walls.



In tunnels, of course, the background is not displayed, but instead a solid fill is applied. The display of all objects is reactivated just before exiting the tunnel.

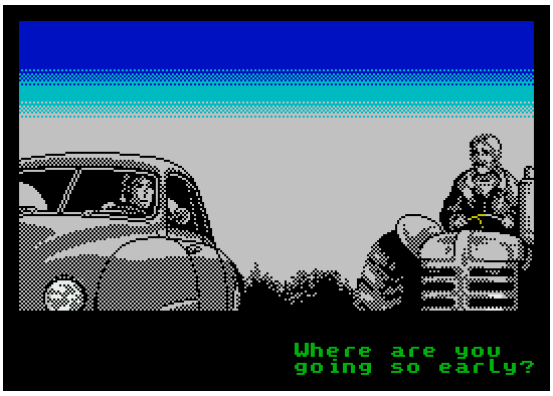
101.

Tunnels can also be used to change the background, including panoramas that can not only "pop up" from behind the horizon, but also turn on and off instantly, being masked by a short section of the tunnel.

It is also worth mentioning that each tunnel has a corresponding road sign at its entrance. It is placed automatically when the map encounters the identifier “tunnel start”.

102.

The first level of the game is the only one where a cutscene appears during the level, dividing it into two parts. This scene is one of the most detailed.

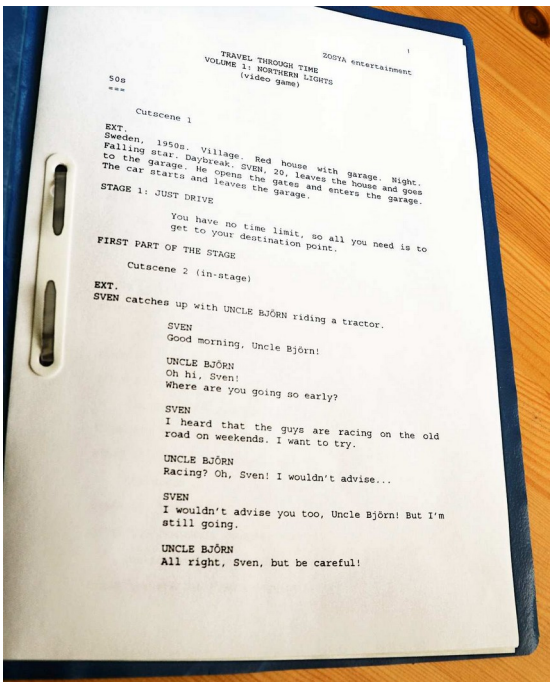


In addition to sprites, animation here is conveyed by scrolling the car up and down and, even more interestingly, scrolling the trees – not only to the right, but also slightly downwards. The same procedure is used in the scene of the trip with Irma. The graphics of the trees themselves are the background graphics of the level.

Only Sven doesn’t look like himself.

103.

The game script, written in screenplay format (American style), contains 21 pages and 36 scenes. The duration of cut scenes in the game is about 17 minutes. This is really close to the classic formula of “1 page of script = 1 minute of screen time”.



TRAVEL THROUGH TIME
VOLUME 1: NORTHERN LIGHTS
(video game)

ZOSYA entertainment

50s

Cutscene 1

EXT.
Sweden, 1950s. Village. Red house with garage. Night.
Falling star. Daybreak. SVEN, 20, leaves the house and goes to the garage. He opens the gates and enters the garage.
The car starts and leaves the garage.

STAGE 1: JUST DRIVE

You have no time limit, so all you need is to get to your destination point.

FIRST PART OF THE STAGE

Cutscene 2 (in-stage)

EXT.
SVEN catches up with UNCLE BJÖRN riding a tractor.

SVEN
Good morning, Uncle Björn!

UNCLE BJÖRN
Oh hi, Sven!
Where are you going so early?

SVEN
I heard that the guys are racing on the old road on weekends. I want to try.

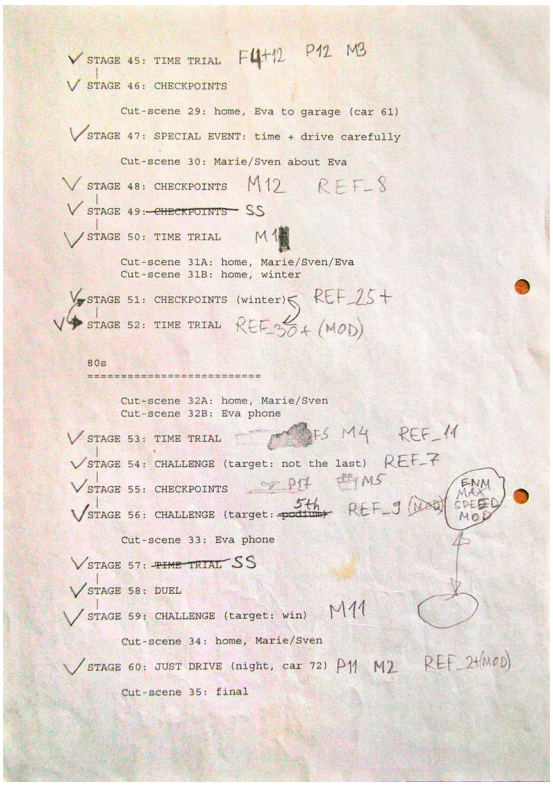
UNCLE BJÖRN
Racing? Oh, Sven! I wouldn’t advise...

SVEN
I wouldn’t advise you too, Uncle Björn! But I’m still going.

UNCLE BJÖRN
All right, Sven, but be careful!

104.

Before writing the full script, a list of the game’s levels and cutscenes was compiled, along with a brief description of some of the main events.



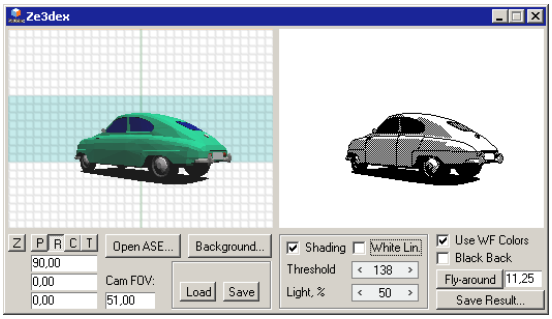
STAGE 45: TIME TRIAL F4+12 P12 M3
STAGE 46: CHECKPOINTS
Cut-scene 29: home, Eva to garage (car 61)
STAGE 47: SPECIAL EVENT: time + drive carefully
Cut-scene 30: Marie/Sven about Eva
STAGE 48: CHECKPOINTS M12 REF_8
STAGE 49: ~~CHECKPOINTS~~ SS
STAGE 50: TIME TRIAL M1
Cut-scene 31A: home, Marie/Sven/Eva
Cut-scene 31B: home, winter
STAGE 51: CHECKPOINTS (winter) REF_25+
STAGE 52: TIME TRIAL REF_30+ (MOD)

80s

Cut-scene 32A: home, Marie/Sven
Cut-scene 32B: Eva phone
STAGE 53: TIME TRIAL F5 M14 REF_11
STAGE 54: CHALLENGE (target: not the last) REF_7
STAGE 55: CHECKPOINTS P11 M5
STAGE 56: CHALLENGE (target: 5th) REF_9
Cut-scene 33: Eva phone
STAGE 57: ~~TIME TRIAL~~ SS
STAGE 58: DUEL
STAGE 59: CHALLENGE (target: win) M11
Cut-scene 34: home, Marie/Sven
STAGE 60: JUST DRIVE (night, car 72) P11 M2 REF_24(MOD)
Cut-scene 35: final

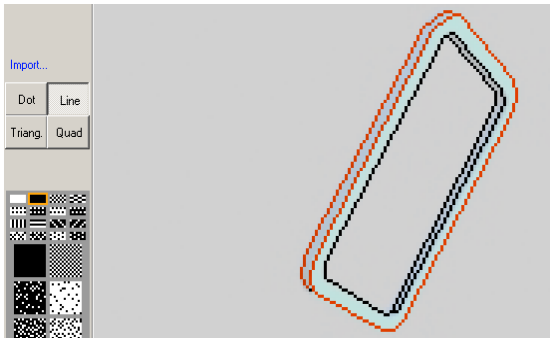
105.

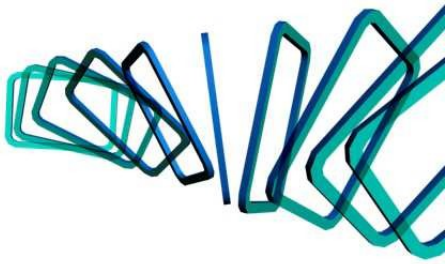
The images of cars on the screens introducing each decade were, of course, also created using the *Ze3dex* utility, but unlike the gameplay graphics, they were only minimally refined, as the result is acceptable for large images.



106.

The animation of the frame flying out from behind the screen was done in 3D, and then, using a hastily written utility, it was manually redrawn into vector format – the coordinates of the points for displaying the segments.





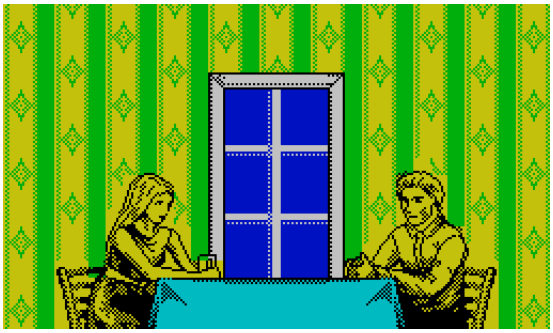
In the game, the same line drawing procedure is used for the speedometer needle. This resulted in 10 frames of vector graphics, while the final frame uses raster graphics.

107.

For the game *Rubinho Cucaracha*, the engine was mercilessly simplified to its minimum functionality. Subsequently, when creating *TTT2*, this simplified engine was taken as a basis, onto which tunnels, flyovers and other previously discarded objects were "bolted on" again. Some may find this illogical, but coders know how difficult it is to work with an "overloaded" engine, in which procedures are scattered "in the gaps in the memory pages".

108.

The artist copied the wallpaper pattern of Sven's house from the one in the house where he was when he created this image.

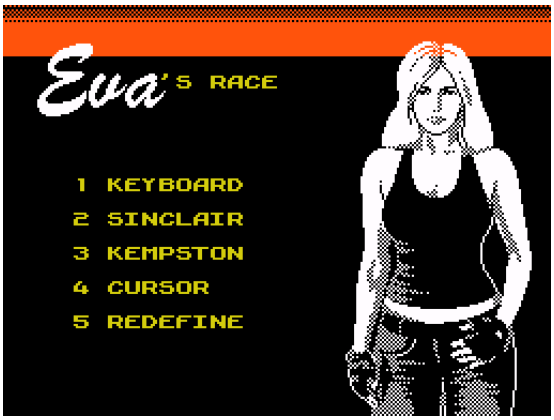


109.

The game consists of 60 levels, and there are also 20 bonus levels that become available after completing the game and reloading it (at the end of the main game, we are told how to lauch them).

110.

The bonus levels are designed as a hidden game with its own title, *Eva's Race*. In them, we play as Eva, just like in the section of main game set in the 1980s.



In fact, the bonus level tracks are basically the same as those in the main game, with a few minor differences: they lack panoramic shots and objects appearing over the horizon, and they use a different set of backgrounds. Here are the matchings between bonus and main level tracks:

Bonus	1	2	3	4	5	6	7	8	9	10
Main	6	53	3	54	46	2	7	4	51	58

Bonus	11	12	13	14	15	16	17	18	19	20
Main	50	30	12	8	34	19	24	10	13	57

The bonus levels are in general slightly easier to complete than their corresponding main levels, as they exclusively feature newer cars, which are faster.

Overall, this game within the game takes up minimal additional memory. Even its large menu image is initially stored in the memory area that will be used as a screen buffer. Then, for the following visualisations, it is copied to an area not used in the bonus game.

In other words, the most we could gain by not making this bonus game would be a little more space for the main game's intro (for example, for a picture in the menu). No additional cutscenes would fit anyway.

111.

The font in the bonus game has been slightly changed. Why? Because we can, we'd say. To this end we used a new set of upper parts of numbers and capital letters.

112.

What's more, for *Eva's Race* we wanted to change the entire interface panel as well. But we decided that was completely pointless

113.

According to logic, the bonus levels should drive on the right: the action clearly takes place in Finland, and Sweden had already switched to right-hand traffic by this time. However, we see road signs on the left. It's simple: this is a factual error. The first 38 levels of the main game use left-hand traffic, but in the bonus levels, the countdown restarts – and traffic is on the left again. If there were more than 38 bonus levels, not 20, we would have seen the traffic switch, as in the main game. The error wasn't noticed in time.

114.

The background in the game is represented by images that repeat horizontally and shift left and right at different speeds depending on the turn. The width of the images is only 64 pixels (8 characters), and the maximum height is 24 pixels. Yes, the scrolling is fast, and a lot of images can fit in memory, but one must agree that the size is very small. Many other racing games, both ours and from others, use a background that is as wide as the screen, not to mention the killer four-screen (1024 pixels) backgrounds of the game *DRIFT!* The only thing that helps to accept such a small size is that the mid-ground objects, blending in with the background, add variety and dynamics. Well, that and the knowledge that in *Chase H.Q.* the background pieces are only slightly wider, of course.

A total of 19 backgrounds were drawn for *TTT1*. Some are only used in the bonus game.

The height and filling of the background decisively influences the sense of enclosure or, on the contrary, of openness.

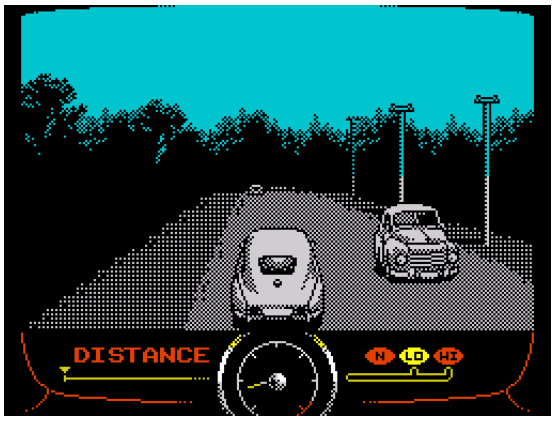
115.

The work on the game engine began in 2013. Although the project was put on hold for a long time (and then reimagined), the game is still based on that same road rendering code. When we created *DRIFT!* and *Maureen Miles* with completely different engines, we already knew that the TRAVEL engine would become the basis for our racing games. And although there are simpler projects in development on this engine, we wanted to release the main game of the future *TTT* series first. We plan to release several more games in this series, as well as other racing games, including for the 48K.

One of such games is *Rubinho Cucaracha*.

116.

The game was planned to feature oncoming cars in some levels, but they looked excessively fast even at near-zero speeds, so so we decided to abandon this idea. A set of sprites for one such car was prepared. Another idea was to have an oncoming car passing by at the very beginning of the game, when the player's car is parked on the left side of the road. This detail would have further emphasized the fact that traffic in Sweden drove on the left side of the road at the time.



However, we considered it irrational to store a full set of car graphics for such a small vignette.

One can imagine how interesting it would be to see oncoming traffic in the level with the timber trucks. Alas, in this case, overtaking would become a lottery.

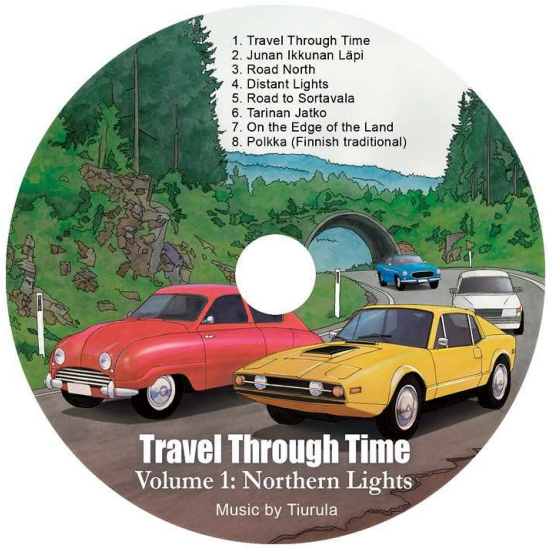
117.

The music in the game consists of AY arrangements of compositions by the band Tiurula from two albums (the eponymous *Tiurula* and *Tie Pohjoiseen*): the game features the tracks “Junan Ikkunan Läpi”, “Road to Sortavala”, “Night in Hiitola”, and a short excerpt from “Ensilumi” to accompany failures. The main theme was written specifically for the game.

The CD included with the game features instrumental versions performed by Tiurula. During gameplay, you can choose to listen to either AY or CD.

The list of songs on the CD differs from those used in the game:

1. Travel Through Time
2. Junan Ikkunan Läpi
3. Road North
4. Distant Lights
5. Road to Sortavala
6. Tarinan Jatko
7. On the Edge of the Land
8. Polkka (Finnish traditional)



118.

Why doesn’t the game use music during races? First of all, it would significantly slow down the gameplay. It’s not so much about the music player itself, but rather about adapting certain procedures to handle interrupts. So instead, we worked on the sound of the engines – both for the player’s car and their opponents’. Yes, we are aware of racing games that feature music during the race, but speed is still much more important. Besides, there is already plenty of music in the game.

119.

Some technical details: to increase speed, the game actively uses the method of copying and outputting via the stack (for those familiar with assembler, these are POP–PUSH or LD–PUSH constructs) with interrupts disabled. These methods in particular do not combine well with playing music. Where the use of the stack is unpractical, LDI chains are used, which, although significantly slower than copying via the stack, are faster than the LDIR instruction (which in “self-respecting” games is used only outside the main loop).

For rendering the shoulder texture and other solid fills, the fastest option was a chain of LD (HL),r: INC L instructions with entry point calculation.

In cutscenes where music is playing, instead of employing the fast copy procedure on the screen buffer, we had to use slower methods, but that's not critical there.

120.

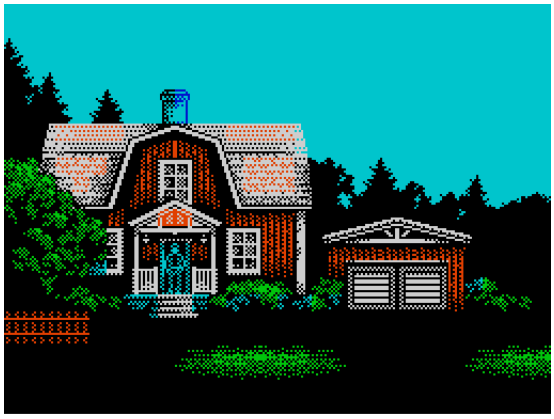
One episode of the game describes Sweden's transition to right-hand traffic – the so-called *Dagen H*, which took place on 3 September 1967. When we were still considering dividing the game into two parts and were less constrained by data volume, we thought about creating a short game scene on the streets of Stockholm, where all you had to do was to slowly re-park your car on the other side of the road. Since there were no plans to further roam around the city, we could have cheated a little: we could have drawn the nearby buildings in great detail and on a large scale, while minimising the need to scale objects. However, it would not have been possible to completely abandon this need within the framework of our engine – the buildings would have needed dynamic procedures related not only to scaling but also to perspective changes.

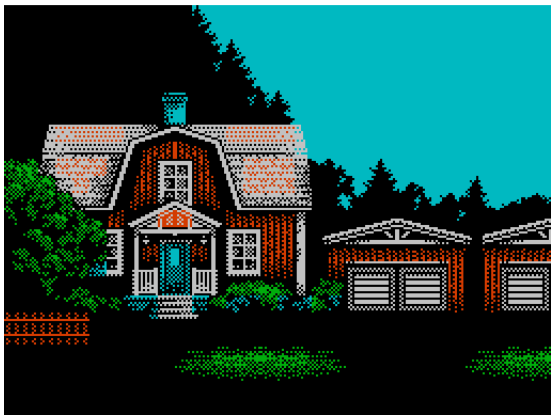
This would take up a lot of memory in any case, but one could do the same as with the train in the first level: put this episode at the beginning of the planned second block of the game, and then not return to it and use the freed memory for other purposes.

In the end, we decided against that scene, and we wove this important event into the game's plot in a completely different way, resulting in a more dramatic and touching episode.

121.

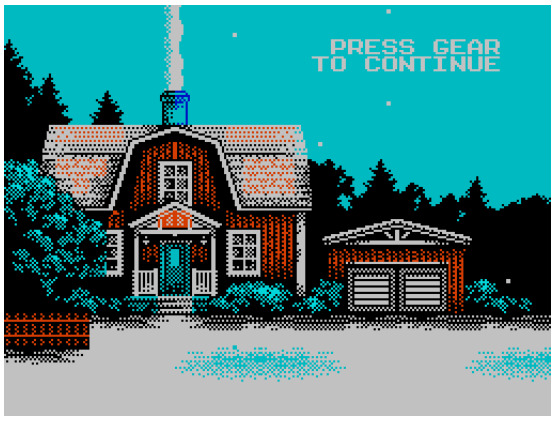
Probably not many people have noticed, but the trees behind Sven's house have grown over the years. This speaks volumes about our attention to detail. A second garage has also appeared (which is, of course, more noticeable).





122.

The winter image of Sven's house is composed almost entirely of recolored summer sprites. Falling snow, smoke from the chimney, and the musical composition "Night in Hiitola" by Tiurula add to the atmosphere.



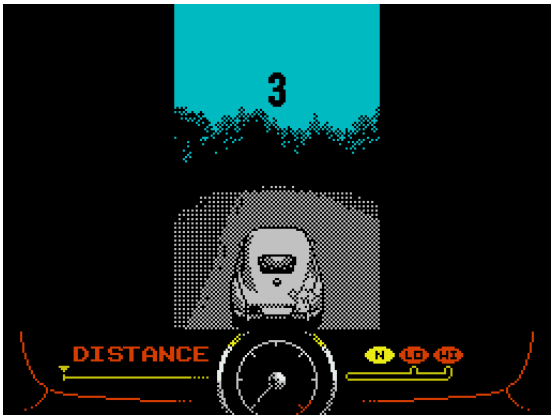
There's also the same scene at night (repainted and with a few sprite changes), in autumn with yellow trees, and in early autumn – just a few "yellowing" attributes.

123.

The frame rate in the game is around 25 FPS when there are no rival cars nearby. This is higher than in many existing racing games with similar content on the screen.

124.

At the beginning of the level, we see the effect of vertical "curtains" – the game area gradually appears from the centre of the screen. This is implemented using attributes. At the beginning of the level, the screen is coloured with zero attributes ("black on black"), and the curtains gradually colour it in the colours set for the sky and ground.



There are also closing curtains that paint the screen black. When they are active, other procedures concerning attributes, such as changing them on the horizon, are not possible.

125.

Collision checking with opponent vehicles (or, more precisely, each traffic vehicle with the player's vehicle) is fairly simple: if an opponent's vehicle is below a certain threshold (defined by a constant) on the screen, the horizontal distance to the center of the player's vehicle is checked. If this also is below another threshold, a collision occurs. Most vehicles are similar in size, so they share the same hitbox. Only the player-controlled motorcycle undergoes some modifications to the collision checking procedure.

126.

When computing the road, positioning roadside objects and opponent vehicles, the engine uses data smoothing:

VALUE = (CALCULATED VALUE + PREVIOUS VALUE) / 2

For even smoother animation, this is done twice. The main disadvantage of smoothing is the delay in the movement of objects, which is very noticeable in the game when observing the movement of the posts relative to the roadside. However, we could not do without smoothing: the initial calculations are too rough, and the movements would be irregular. Roadside objects, instead of smoothly moving sideways, would jerk left and right.

The lag in movement is more noticeable at high speeds. To compensate this negative effect, in these circumstances only one iteration of motion smoothing is applied to roadside objects, while at low speeds two passes are used.

Before the race begins, for roadside objects that appear on the screen the position calculations are made multiple times, otherwise we would observe the posts slowly moving to their assigned place due to the position smoothing effect. However, another implementation suggests itself – temporarily disabling smoothing.

127.

Interestingly, thanks to motion smoothing, the effect of rival cars changing lanes was implemented for free. It happened that, to simplify the data, we initially decided to store not the exact horizontal position of traffic cars on the road, but only a flag indicating whether

they were on the left or right side of the road. But when we needed to add the ability to change lanes (for which we simply flip the lane flag), it turned out that the cars on the screen were already able to do that smoothly. They do it a little too fast, but this generally reflects their representation in the data structure, where, as mentioned, there is only “right” or “left”.

128.

To calculate the horizontal coordinate of the opponent’s cars on the screen, we take the values of the left and right edges of the road for the line of the screen on which the car is located. We compute the middle of the road, and then, depending on which of the two lanes the car is in, we calculate the middle of such lane is found. This coordinate is smoothed based on the previous value.

Interestingly, when the road narrows, the cars seem to steer on their own, even though no procedure for this has been written.

But here’s the problem: we can’t always determine the middle of the road and each of the lanes precisely: if the road extends beyond the screen, the edge values are equal to the screen border: when calculating, we don’t go beyond the byte values. As a result, the road border seems to fall vertically downwards on the screen; considering the perspective, this is equivalent to a sudden narrowing. Instead of approaching the screen boundary on the perspective line, the car will suddenly start moving vertically downwards.

No elegant solutions were found. To remedy the situation, we locate the point where the road is “clipped”, and then somehow adjust the cars’ positions. The cars closest to the edge receive an additional shift to the side depending on their vertical position on the screen (this is done for the 32 bottom lines of the game screen).

129.

A very nice detail: in the episode where we first see Marie and Sven meeting, two stars fall one after the other above the house, rather than just one, as in the opening scene. Here is that moment in the script:

```
Cutscene 14

EXT.
Sven’s home. There’s a SECOND
GARAGE next to the first. Night.
TWO falling stars.

INT.
Sven’s home. SVEN and MARIE.
```

It’s funny that, before this moment in the story, they had never spoken to each other.

130.

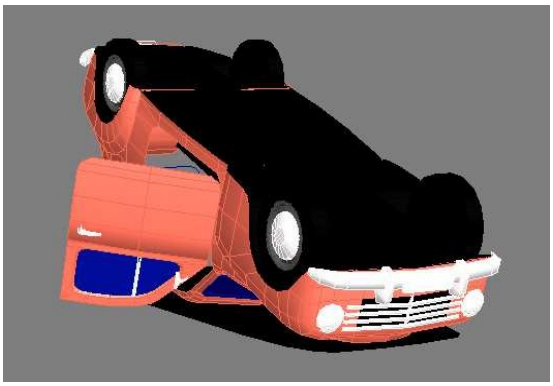
The chase level is an obvious tribute to *Chase H.Q.* It’s a pity that it only fits into the plot once (and even then, it’s somewhat of a stretch).

Similar to the *Chase H.Q.* series, we also included a scene depicting the criminal’s arrest. (The first *Chase H.Q.* on the ZX Spectrum didn’t have this; it only featured portraits of the criminals.)

We got by with using gameplay graphics and a small set of sprites.

131.

The overturned car was drawn based on a visualisation of a three-dimensional model.



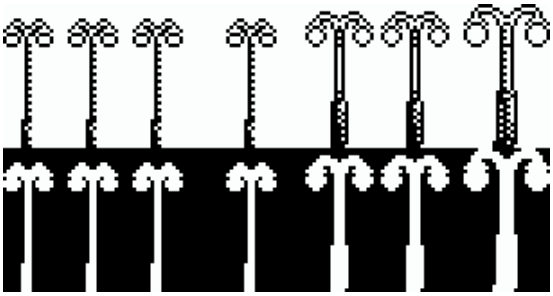
As with the train, we didn't need many sprites of different sizes, especially since the game logic dictates that we don't drive close to the overturned car.

132.

The game uses almost all of the computer's memory. There are no more than 20 bytes left in each of the available memory areas.

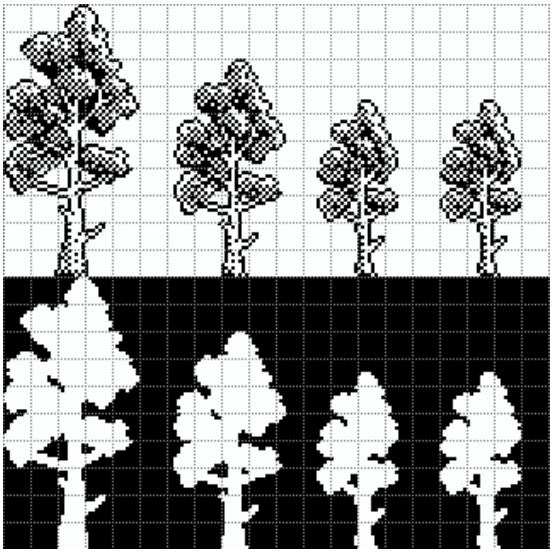
133.

The lamppost that we later used in *Rubinho Cucaracha* was drawn while working on *TTT1*. That's the name of the file: pole_other_game.



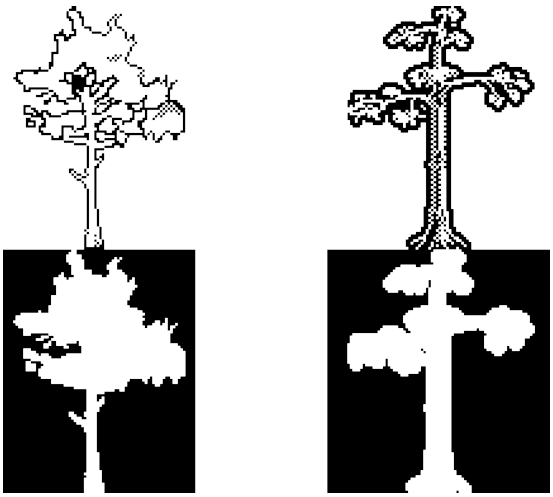
134.

The trees in the game are drawn in mid-ground and background, while only small bushes are found along the roadsides. However, during the development process several tree variants were drawn and tested.



For this tree, an offset has been pre-rendered only for the distant state, and overall the number of scale variants is clearly insufficient.

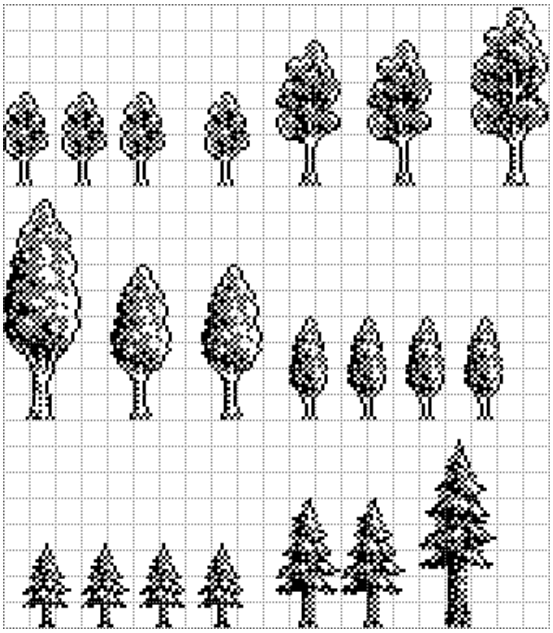
The main problem with using trees in this engine is that a pleasant-looking, smooth increase in size requires multiple scale options, which takes up a lot of memory; in addition, the large sprite size significantly slows down the game, and there is the issue of sprite clipping at the edges of the screen, which, for example, is not so significant for thin pillars.



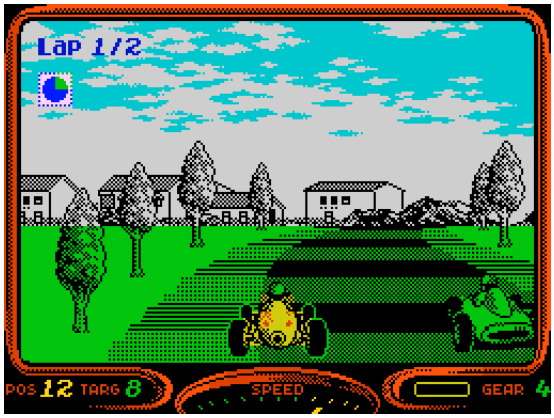
The last tree, unfinished and rather ugly, was used primarily to test the stretching of individual parts of the sprite. We also tried using a sprite format with smooth stretching, automasking, and clipping – similar to what we use for opponent cars – but ultimately decided to abandon roadside trees.

135.

In the simpler and more casual game *Rubinho Cucaracha*, where we didn't place any particular demands on realism, we used three types of trees. Tree designs from the games *TTT1* and *DRIFT!* were used with some modifications.

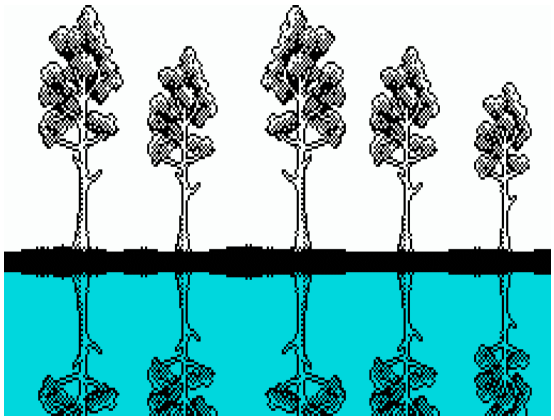


Note that for the farthest sprites, 4 variants of horizontal offset were drawn; for the middle level, 2; and for the nearest sprite, only one single variant. This is the same approach that was used for the poles in *TTT*.



136.

The next image, which practically begs to be included in some adventure game, also emerged during the course of our "work on the trees". It's clearly missing a knight on horseback.



137.

The main drawback of sprites with automatic masking is that they can only be applied to objects with fairly simple, closed, convex outlines. (Based on how the automask is constructed, the contour must remain convex for horizontal segments.) For example, with cars with open wheels, part of the background would be erased between the wheel and the body.

In *TTT1*, all opponent car sprites have convex outlines, except for very small, insignificant details. However, when creating *Rubinho Cucaracha*, we had to figure out how to reconcile automatic masking with exposed

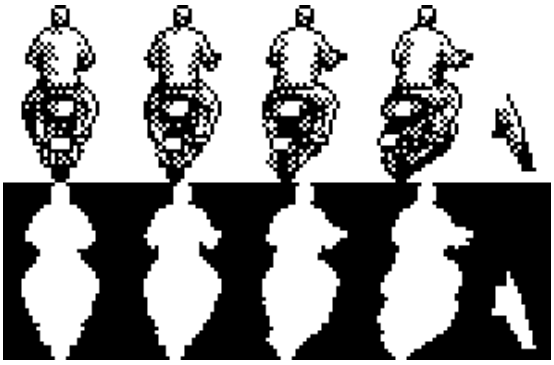
wheels. Partly for this reason, opponent cars have a wider body than the player's car (which, as in other games based on the TRAVEL engine, uses a different format – .tts with masks).

As a result, there are still some "holes" left in the sprites of Rubinho's opponents, as we can see in the following picture, but they are very minor.

138.

When the speed of his motorbike is near zero, Sven puts out his leg.

This is a sprite that is simply rendered on top of the main one, and it looks good regardless the motorcycle's angle.



For the motorcycle level, we generally made do with a very compact set of sprites.

139.

The ability to change the ground texture during a race is used to depict water: sometimes the road crosses a river or passes next to a body of water.

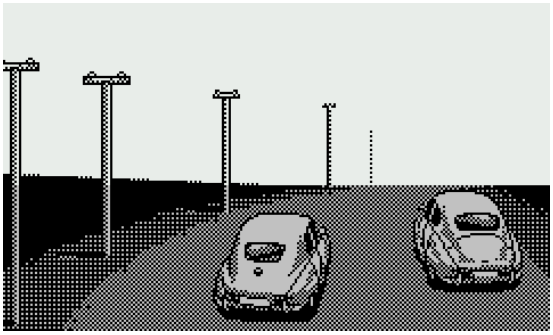
Due to engine limitations, the texture must be dark in any case, but small wave crests on it look fine. Additionally, an animation has been added to convey water movement (between rendering lines, a procedure shifts the texture).

Additionally, to make the water look a bit more realistic, graphic segments were added near the horizon, similar to what was done with the "distant road".

Unlike a full implementation of road texture swapping, the ground (water) texture changes in a somewhat simplified way – by whole character rows (8 pixel lines each), so it doesn't exactly align with the movement of road segments. In motion, though, it still looks fine. Like many other shortcomings, this was also improved in the second part of *TTT*.

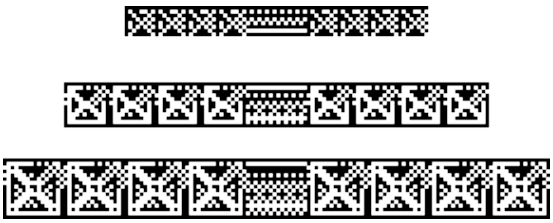
140.

On the upslopes, you'll notice that the horizon line isn't completely flat; it rises toward the edges of the screen. This is another additional graphic element (similar to the additions to water and the "distant road") that enhances the visuals. It's activated only when needed and blends well with both the background and the panoramas.

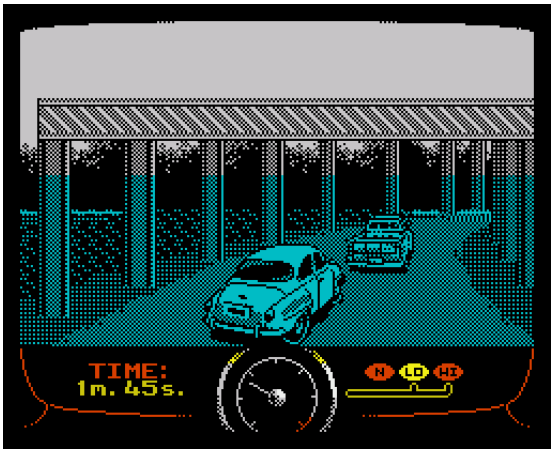


141.

The first version of the bridge designs was more detailed:



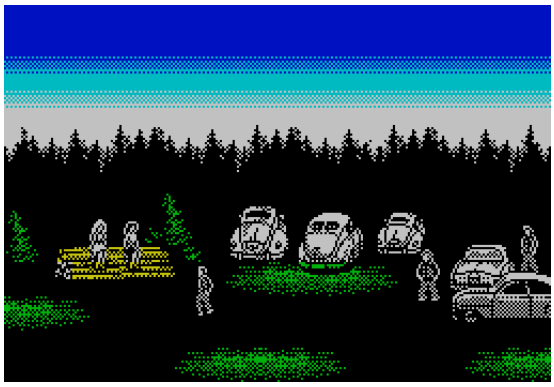
The game uses a simplified version, in which horizontal structures are procedural:



The coordinates of the horizontal beams of the bridges are tied to the tops of the posts. Since the nearest beams on the screen cover all subsequent ones, only they are rendered. It was also necessary to handle the case when posts on one side of the road don't appear on screen, because screen coordinates are used to position the beams. In this case, one can reference the nearest visible post and draw the beam in the opposite direction to the edge of the screen.

142.

Although the game is replete with cutscenes, the locations where the action unfolds are very few. Some appear only once, but most are used for multiple scenes.



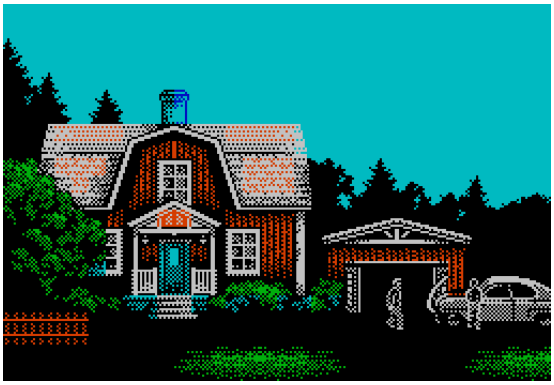
This forest location is one of the most frequently recurring. It is where the characters' relationships develop over a significant part of the story. Two other frequently encountered locations are the exterior of Sven's house (starting with the opening scene) and later its interior.

At the beginning of development, we didn't know what the cutscenes would look like. This fortunate insight – combining large character portraits with very small graphics depicting the action – allowed us to fit so many scenes into memory.

143.

You've probably also noticed that the cutscenes also use gameplay graphics. The forest location uses the same background as the races, the same opponents' cars, and the grass sprite is also used in the location with Sven's house. His car, however, is drawn from a side view specifically for the cutscenes (the sprite shows only the visible part, and there's an additional piece):

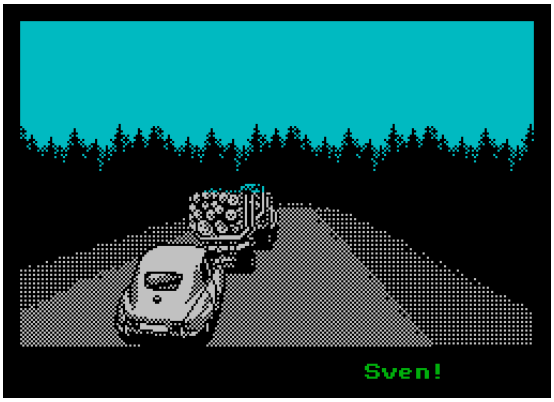
In the car repair scene, the open hood is drawn on top of the closed one, part of which is partially painted over. We had to come up with all sorts of tricks for optimization! The additional piece of the car is also needed here, just to position it correctly by the garage.



144.

The scene on the ocean shore is almost entirely made up of gameplay graphics. Only the figures of Marie and Sven and a small piece of the rocks were added.

And this cutscene with the logging truck uses not only fully gameplay graphics, but also some of the engine's procedural rendering routines.



145.

In cutscenes in Sven's house, with multiple characters present, it was difficult to tell who was speaking. Adding a colored stripe under the character solved this problem.

146.

By the way, have you noticed the color coding of the dialogue? Sven always "speaks" in yellow, Uncle Björn in green, Marie in blue, Irma and Eva in purple (there's a silly mistake for Irma in the scene with the logging trucks you just saw), and Johannes in white.

147.

We prepared sprites for a bad crash with the wrecked car spinning, but this idea was ultimately abandoned.

148.

In one of the final scenes, Sven and Marie are sitting at a table reading a newspaper. Initially, they were depicted, as before, on opposite sides of the table, with an attempt to place the newspaper in Sven's hand. It didn't look right, and the idea suddenly struck me not to draw any newspaper at all (implying it's just lying flat on the table), but instead to move Marie to sit next to Sven.

And it worked! They seem to be looking at the same point. We got away with a minimum of additional sprite pieces.

149.

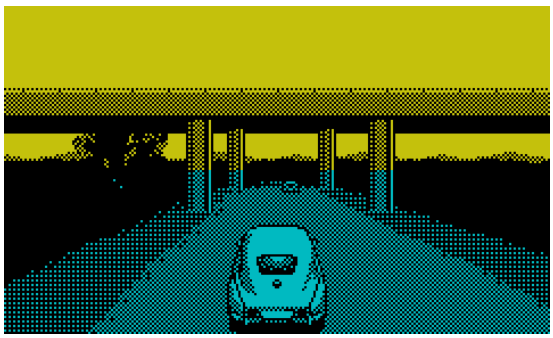
The picture of Eva sitting on a window sill was redrawn from a photograph taken specifically for this purpose.

For optimization, the vertical parts of the windows are made with a repeating sprite:

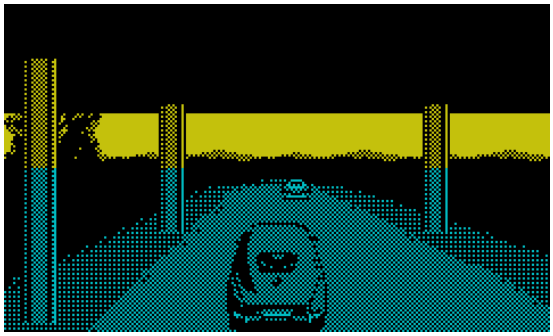
150.

The overpasses that cars drive under are another very interesting object. In their implementation, they are partly like bridges, and partly like tunnels.

Over the two pairs of supports, the ceilings are drawn procedurally, with scale and perspective corresponding to their placement.

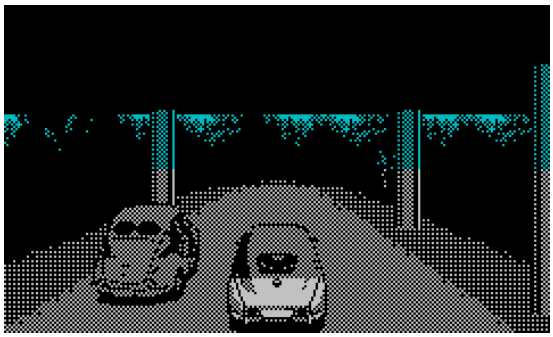


When driving under them, they block the sky, but the most interesting detail is that they cast a shadow on the cars, which appears and disappears dynamically.



This is implemented by overlaying a procedural texture onto an already fully rendered image, matching the road texture. It is applied to the area where the car is drawn, and of course it affects not only the car but any objects under that texture. The car “darkens”, while the road remains unchanged.

Moreover, if there are other cars passing under the overpass at the same time as the player’s car, additional pieces of this texture are drawn to cover these cars.



151.

At the beginning of the game you can try to complete a level an infinite number of times, but at later levels you're given a limited number of attempts before the game is over. To visualize these attempts we drew two icon variations. The game uses the second one. The procedure for displaying remaining attempts was copied from the game *Bonnie and Clyde* along with the file name. The image with the car and arrow is called "bc_hat" because in the cat game, lives were represented by hats.

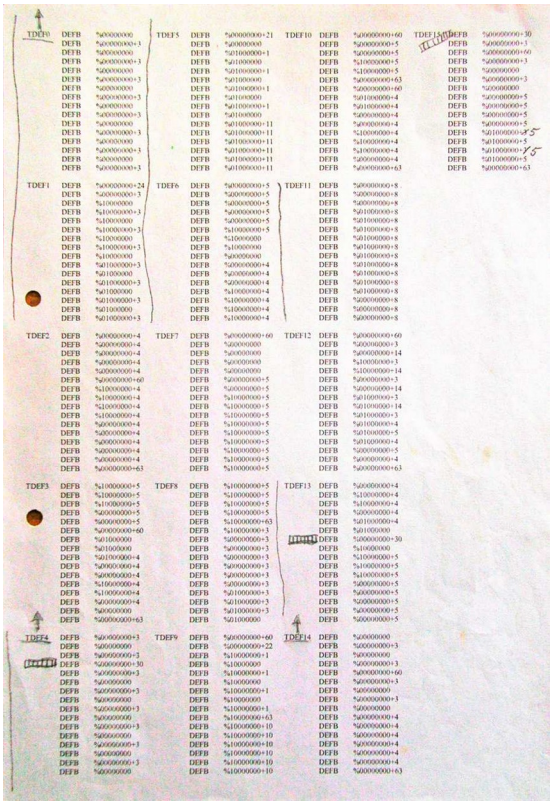
152.

In a time trial, if the finish line or sector checkpoint is crossed, but only after the time has run out, the game counts it as losing. Which makes sense, even though in many other games this this happens differently.

In a duel, if the player and opponent visually cross the finish line at the same time (even if the opponent appears slightly ahead), the victory is usually awarded to the player.

153.

Level maps in all racing games for 8-bit computers are generally quite uniform: they consist of similar sections of straights and turns, to which all the objects implemented in the game (if any) are added. And uniformity is actually a good starting point for optimization. To fit a large number of tracks into memory, we use methods to store maps as compactly as possible. In all games on the TRAVEL engine, a reserved memory area is used for the current track, where the map is built before the stage begins using procedures and fairly compact source data. In *TTT1*, this relies on a large number of predefined segment sequences. In the image below, a "typical" chain of 256 segments is shown, divided into 16 sections. These segments include turns and straightaways, indicate roadside objects, some signs, pedestrian crossings, and inclines. A game map can simply specify: "use the standard sequence, starting from section #0 and with a length of 3 sections".



Moreover, for some track alterations (say, entering a tunnel or starting a narrow road), the same data can be used, but it will be rendered slightly differently on the screen: for example, in tunnels, only the turns will remain.

We didn't hesitate to use this same array almost unchanged in *Rubinho Cucaracha* and *Travel Unlimited*.

But this isn't the only set of predefined segment sequences in the map. There's also a list of short but more unique sequences. You saw it on the same sheet as the list of all objects. For example, this is how we defined overpasses with appropriate approach routes, short tunnels, different types of bridges, long rows of road posts, and so on.

There's also the option of accessing a map from another track. Let's say memory is running low, and the player is unlikely to remember what happened on level 5 by the time they're on level 55. Especially since the competition type is different, as are the cars, the background, the color scheme, or maybe it's night or winter... One can simply write a command to jump to the data at a specified marker and repeat part of the other track.

154.

The character animations for cutscenes are collected in one file.

By the way, to walk up and down, the characters use only one sprite, which is mirrored.

155.

To show a car driving out of a garage, a set of sprites with changing lighting is used.

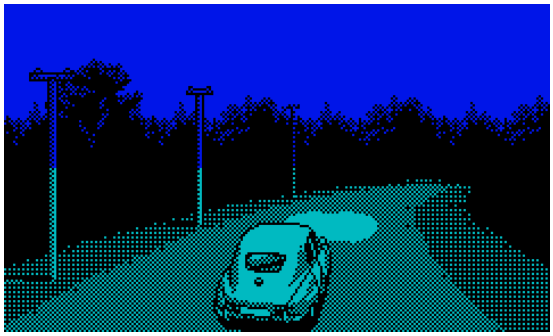
In the same way, Sven, entering the garage, disappears into the shadows.

156.

The visualization of the collisions with flying debris is clearly inspired by the game *Chase H.Q.* The debris uses a four-frame animation and flies off along various trajectories.

157.

Night races are no different from daytime races in terms of gameplay or technology. A darker color is used for the sky, but the main visual difference is, of course, the headlights.



This is implemented using just one single specially drawn sprite: the image itself is "white", and the mask is chequered.

When this sprite is rendered, the "white" pixels of the headlights always land on the "black" pixels of the road, creating a solid bright spot. At the same time, other objects seem to actually be "illuminated" without being completely overwritten, since the grid of "white" pixels also falls on them.

It was only some time after the game's release, while working on the second part, that we realized that we'd actually overdone it with the "light" texture, which should have been much simpler:

In this case, the cars in the headlights' light will no longer show the erroneous checkered rectangle that was noticeable in *TTT1*. In the second part of the game, the corrected texture is used.

158.

Uncle Björn's tractor, like the logging trucks, uses its own long-range sprites, so the number of scales drawn is greater than for the cars.

For the two closest distances, the wheels' rotation is drawn: every second frame, additional pieces are rendered on top of the already drawn sprite.

At the nearest distance, part of the tractor driver's torso is drawn as a separate, much narrower sprite for optimization.

159.

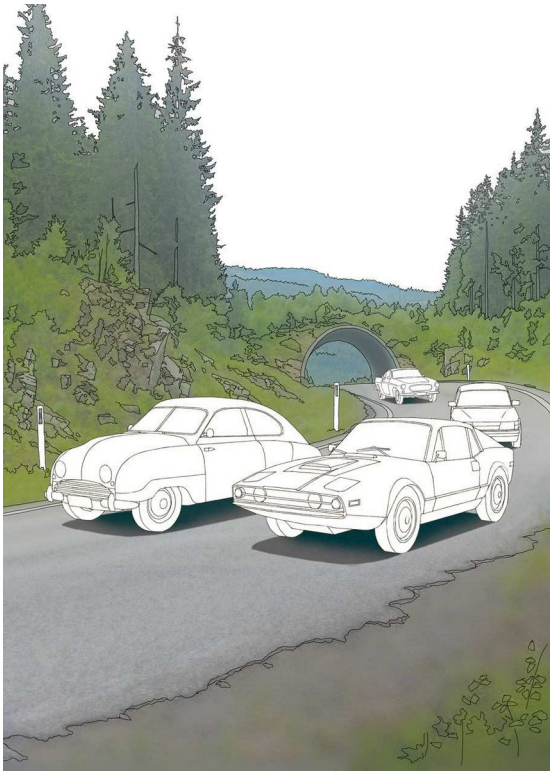
For the countdown, we used a dynamic font, a relatively rare feature in ZX Spectrum games. The numbers appearing on the screen grow in size. To achieve this, we used two sets of characters and the capabilities of one of our sprite formats: first, the numbers are displayed in a smaller size for a very short time, then in a larger size, with gradual vertical stretching applied (using the same sprite format as for the opponent cars). For the "GO!" text, we limited ourselves to a single font size.

160.

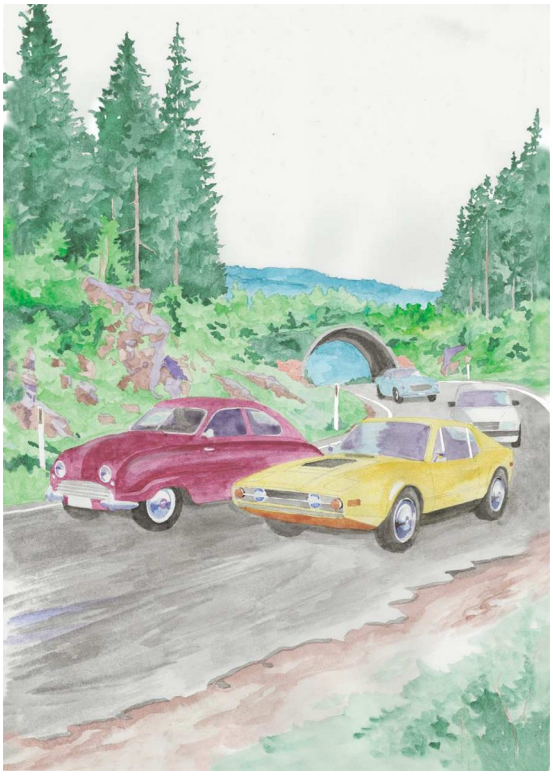
The work on the title screen and the poster was carried out simultaneously. Of course, the existing 3D car models were used. Then, based on photographs (technically it's Norway, not Sweden, though part of the game's story takes place there)…

…prepared the base image for the poster's background…

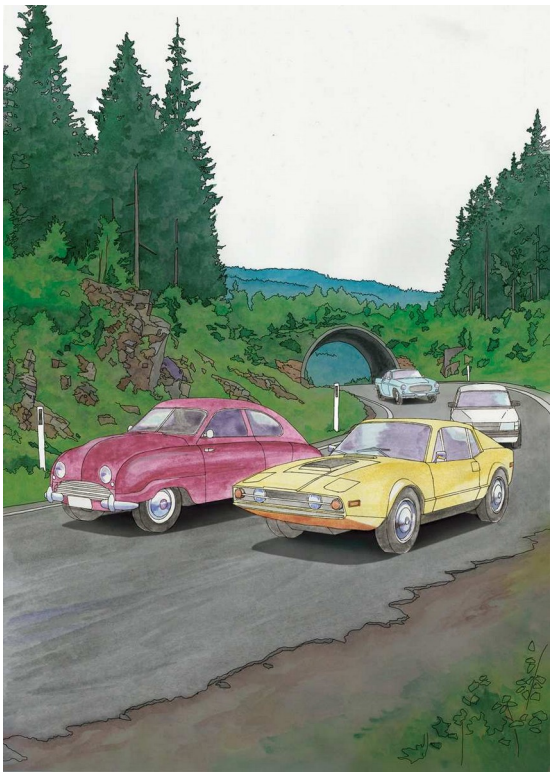
...we added contour strokes, blur, and embedded the renders of the 3D models...



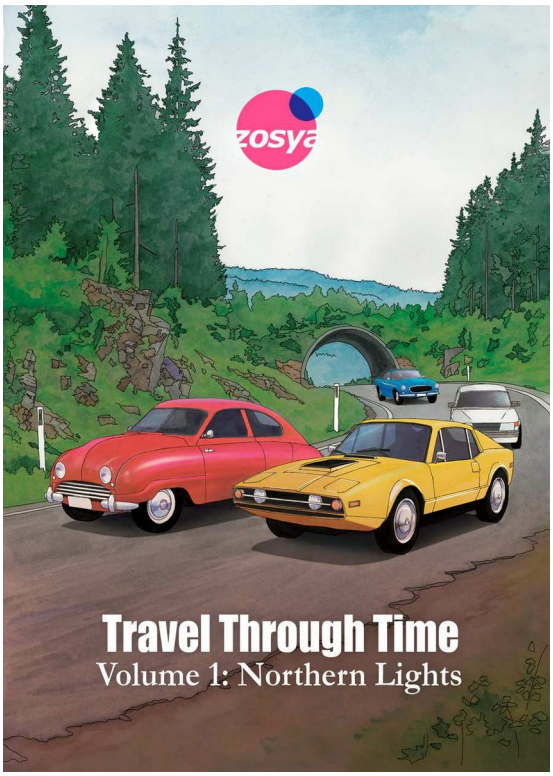
...drew a watercolor sketch on paper...



...merged it with the existing image...

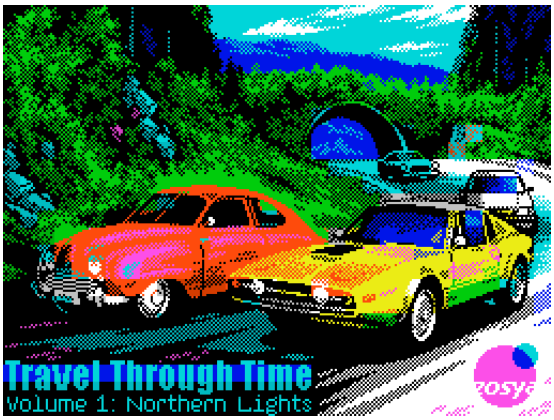


…made numerous digital touch-ups and came up with the final poster:



161.

And similar methods were used to create the loading screen on the ZX Spectrum:



162.

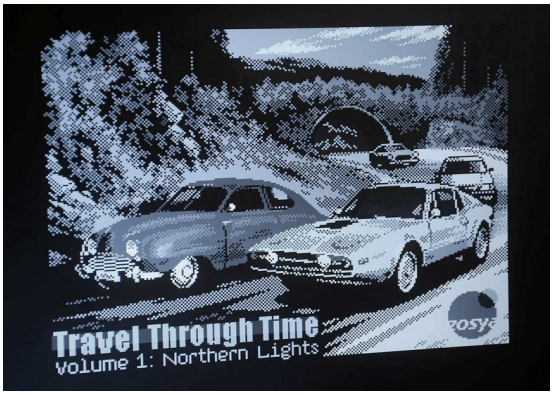
The design of the final scene included images of trees. They didn't fit in memory, and besides, they didn't really enhance the picture. You can also see a more detailed reflection animation. In the game, a single mirrored sprite is used instead.

163.

The game trailer is the most popular of all our videos.

164.

One of our favorite "divertissements" was checking how the game's graphics (especially the intro) looked in monochrome. After all, that's exactly how many of us saw computer games in the late '80s and early '90s.



165.

Are we happy with the result? It's not a thing we can say about every game, but in this case – definitely yes. *Travel Through Time Volume 1: Northern Lights* can certainly be considered one of our best games and one of the best racing games for the ZX Spectrum. Sales showed that the community agrees with us.