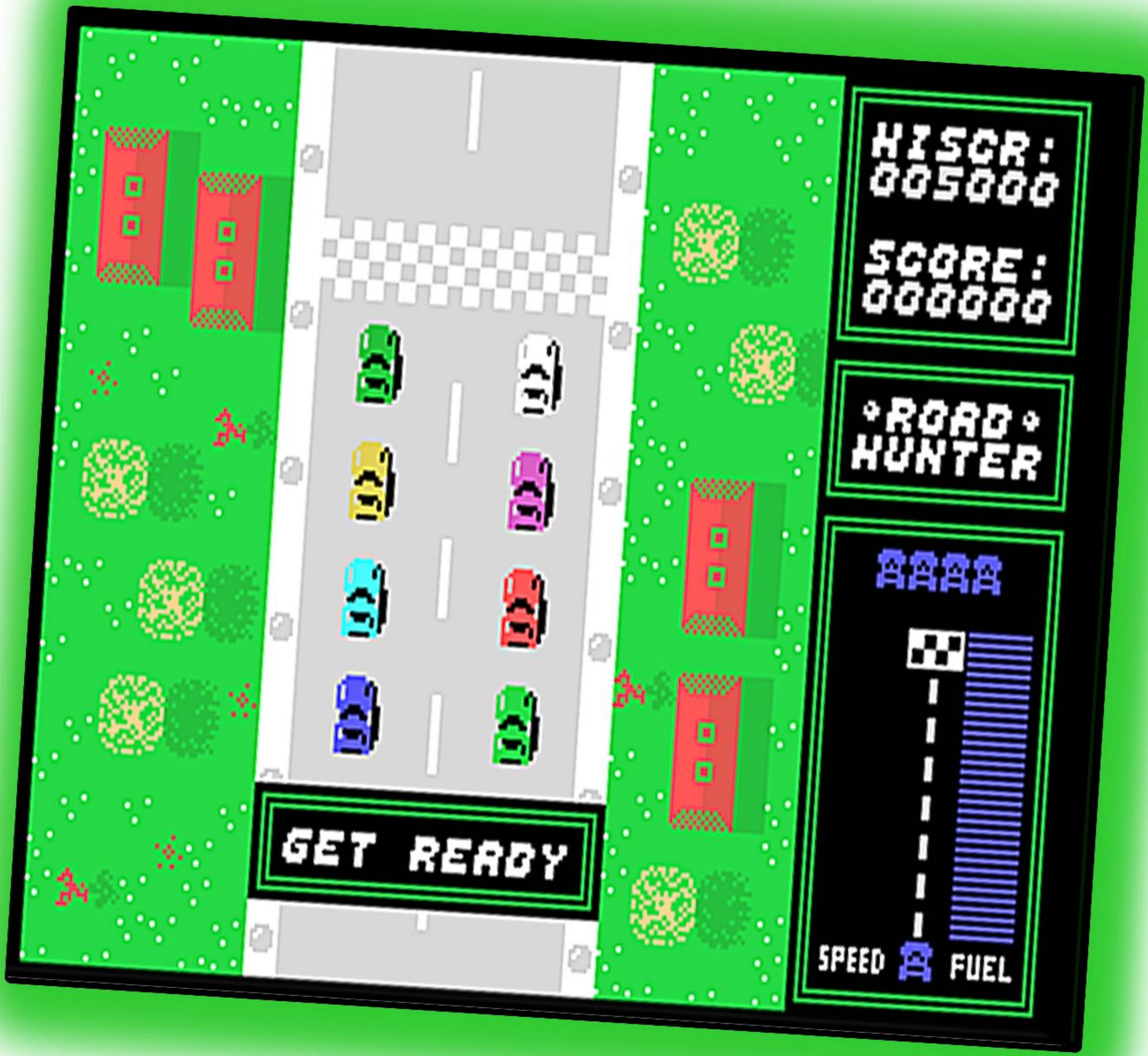

ROAD HUNTER

© 2014 By: Rasmus Moustgaard
rasmus.moustgaard@gmail.com
Version 1.0, February 21, 2014
Original manual modified by: Kevan Coleman

Introduction

Road Hunter is a racing game for the TI-99/4A home computer where you score points by bumping into other cars, knocking them off the road or even shooting them out of the way! You'll need good driving skills and fast reflexes to avoid ending up on the side of the road yourself. You'll also have to make sure that you don't run out of fuel, because if you do, it's game over! Fortunately for you, the car is equipped with an advanced system that allows you to refuel on the fly, by using fuel drums that others have dropped on the road. Occasionally, guns may even turn up on the road, and these make it easy to clear a path ahead of you. Be aware, if you bump into other vehicles while carrying a gun, you'll probably lose it.

The game is not all about destruction; you should also attempt to finish the track to collect the bonus (depending on the amount of fuel you have left). Watch out for that next track, it might be even harder than the last to complete.

Program Loading

To load and run the game you have three options:

1) Editor/Assembler option # 3 : DSK1.RH
2) Editor/Assembler option # 5: DSK1.ROADHUNT
3) Extended BASIC loader

Note: The game requires the 32K RAM memory expansion to operate.

Game Controls

	KEYBOARD	JOYSTICK
Accelerate	E	UP
Braking	X	DOWN
Steer Left	S	LEFT
Steer Right	D	RIGHT
Fire	SPACE	FIRE
Pause Play	P	

Press accelerate or fire to start the game. After 10 seconds with no activity, the game will enter into the demonstration mode.

Playing Tips

° Hitting other cars from the side is the easiest way to push them off the road, but be careful if you're driving too near the road side, you could bounce off the other vehicle and crash.

° Avoid colliding into other cars from the rear since this might send your own car off the road.

° If you hit a car in the front end it will explode immediately.

° Shoot if you can, but avoid shooting the fuel drums or you may run out of fuel.

Rasmus Moustgaard
Copenhagen, February 2014

Technical Notes

° The game was written in TMS9900 assembly language, and was assembled/compiled using WinAsm99. Testing and debugging was primarily done using Classic99, but the game has also been tested in MESS 0.150 and on a real TI-99/4A.

° The game runs at 60 frames per second (FPS), or 50 FPS on EU consoles without the F18A. Timing is done by polling the VDP status register. Interrupts are disabled at all times. All 256 bytes of scratch pad memory are used, but a copy of the original content is saved and restored again before file operations.

° The maps were made in the Magellan map editor and exported using the "Assembler Character Transition Data" option. Because of the smooth scrolling, each unique pair of neighboring characters in the vertical direction uses up one of the 256 characters, and the characters beyond 192 are reserved for the left side panel. The game is running in the normal graphics mode so to avoid color spills, two characters next to each other must have compatible color sets.

° Each map consists of 24 24x24 screens. Each track consists of 104 screens in a fixed order specified using a list of screen memory addresses. The maps are 13.5 K each so they have to be loaded from disk for each level to fit into RAM.

° The map is not read in order to detect if a car hits the side of the road. For each track, only the starting position of the left side of the road and the (fixed) width of the road is used. For each screen, the direction of the road (left, straight, right) as one number. From this information the program can calculate the position of the road sides at any point.

° Reading from VDP is kept at a minimum. Except for file operations, only the VDP status register is read.

° The first 8K of the VDP RAM is used for storing character definitions, in 4 character sets, each corresponding to a different scroll offset (0, 2, 4, 6 pixels). A routine scrolls the original patterns and uploads the data to VDP RAM once before each level. Scrolling is done simply by changing the address of the pattern table. After 4 times or 8 pixels it moves back to the first character set and switches to another name table scrolled one character or 8 pixels.
° There are two name tables in VDP RAM (one at >2000 and one at >2400). I alternate between them each time I have scrolled the screen 8 pixels. I show one table while updating the other. To even out the work between frames I update 1/4 or 1/2 of a table each frame, depending on whether the scroll speed is 2 or 4 pixels. Copying from CPU RAM to VDP RAM is done using a routine in scratch pad RAM that has 8 consecutive MOVB instructions in the loop body.

° There are also two sprite attribute tables that are alternated between. Each frame I switch to one of the tables while uploading data to the other from CPU RAM. The CPU RAM copy of the table is actually stored in scratch pad memory. The game uses a flicker (reducing) routine where I cycle the quarter of the table that gets the lowest sprite numbers, i.e. the quarter that is uploaded to the beginning of the VDP RAM table.

° Sound and music is played using my own sound list player. This is based on the format of standard sound lists but also supports loops and calls. Allowing the 'drums' to play continuously without being disrupted by the explosions required a little special coding.
Keyboard and joystick are read directly using CRU. A single call to KSCAN would reduce the frame rate by a factor two because of the delay.

* During game development I have continuously kept an eye on the number of CPU cycles used by the main loop using the debugger in Classic99. I find this essential if you want to make a fast moving game on the TI. It's easier to fix any speed issues immediately, rather than later when the code has become more entangled.