Scene file

The file is to be read line by line, one line in file corresponds a line of the 80X25 screen.

A minimal scene file is a text file containing a sole “x” or “X” character. This minimal scene file is being considered as empty 80X25 scene, with car position at the upper-left corner.

The empty grid points can be denoted by spaces, or for better readability by dots “.” or plus signs “+”. Any character other than “x”, “X”, “<”, “^”, “>”, "v", “#” or “~” being evaluated as empty space. Empty line being considered as line full of spaces. Lines shorter than 80, being padded with spaces. Missing lines of the full set of 25 being considered as empty lines.

Lines longer than 80 being truncated.

In the 80X25 range the file must contain one and only one “x” or “X” character denoting the car position.

If car position is missing, the program quits with error message: “The car position is missing from scene file”.

In the 80X25 range of the scene file only one “x” or “X” allowed, if more than one found, the program quits with error message: “Only one car position is allowed”.

Control

To make it really physical, the human control should be implemented in a quasi real-time manner as e.g. a joystick unit. In order to gain a WASD control in “joystick” mode, I had to utilize the X11 library features. The other alternative was to get access to the low-level keyboard files of Linux, but it requires “sudo” access that nobody wants to grant to a third-party executable.
Normal program termination: “esc” key.

Rotation implementation

However the linear velocity can be adjusted in a continuous manner, the rotation speed is not similarly simple case. Why? Because even adjusting the direction angle at any arbitrary value (case B), the visualization takes place through a low resolution grid, so the controller would have no visual feedback about the accurate direction of the vehicle, so accidental collisions would occur, or the guardian agent’s warnings would cause confusion having conflict with the visual sense.

In the other case the rotation manifests only in predefined grades namely 0, 45, 90, 135 etc… (case A) so the visual feedback is fitted to the visualization of the underlying physical phenomenon (in this case not quite physical).
So I decided to consider both version with their advantages and drawbacks.
The continuous rotation happens in both cases, but takes effect in different ways.

Case A: As the car can proceed only in the predefined directions, the rotation snaps at them by the rules of floating point rounding of the angle divided by pi/4. At the last snapping, if the rotation speed is not enough to bring it to the next named direction, the difference get lost from the last snapped direction as soon as the decelerating rotation eventually stops, so the car can proceed at “clear” directions. But if the rotating keys (a or d) having pressed during the not yet stopped previous rotation, the rotation gets accelerated/decelerated from its value at that point in time.

Case B: The car can proceed in any arbitrary directions, so the grid get hit in an “aliased” manner, that is proceeding in some low angle with a main axis, there would be several unit long way at the actual main axis until transfers one unit sideways.

Note: “Case B” is deliberately omitted, because of 1. lack of time, 2. from the description “It can rotate in place by 45 degrees (= 1 turn).” it is not exactly as prescribed.

Prerequisites

sudo apt-get install cmake libx11-dev libncurses5-dev libncursesw5-dev doxygen

Build and run

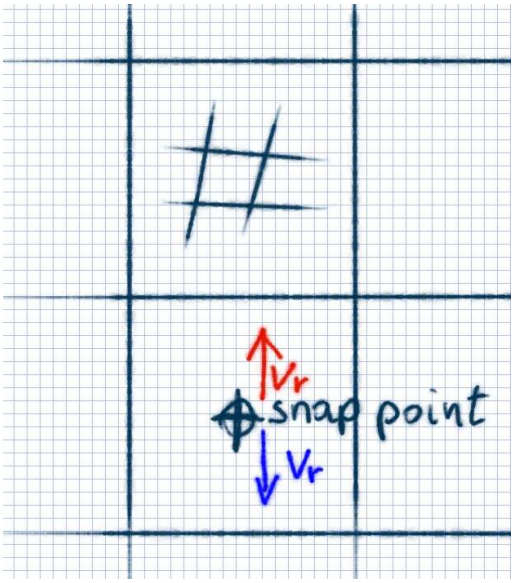
```
$mkdir build&&cd $_
$cmake ..
$make
$./flsrn /path/to/scene/file
```

To display Driver Assistant warnings and logs in a second terminal:
```
$tail -f /tmp/car_log.txt
```

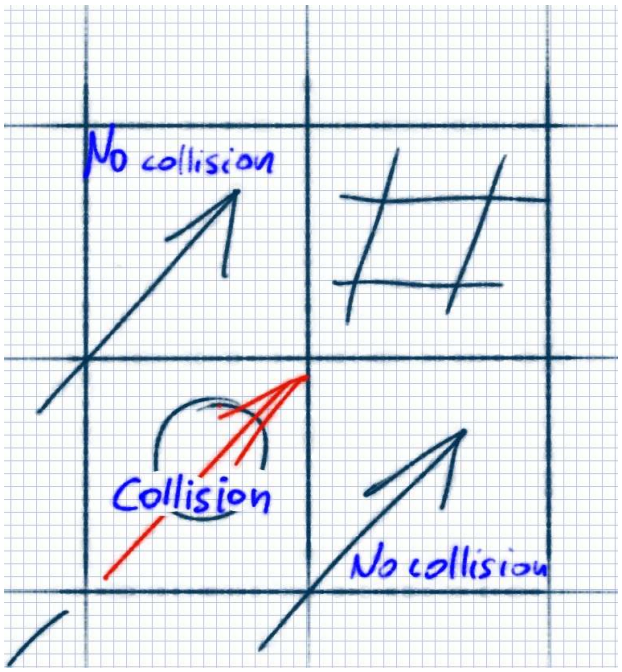
Notes

This program is written by a software engineer for being used by software engineers to test his comprehension and coding ability. That is why I dare write these words:
As I do not intend to deepen my knowledge in curses (unless somebody pays for it) turned just as many attention to it as necessary to implement the scene display driver.
That is why there is no menu, nor warning panels etc, the program simply stops with all the information displayed on the fatal event.
All the controlling parameters are stored as constants at the beginning of the “main” function, so if necessary changing them and re-building the program is a workaround for the missing menu or command line parameters.
The joystick feel of the X11 mouse control can be tested by setting the first those constant, the boolean kbTestMode true.
Why templated to screen size? Because of the use of light-speed std::array as distributed screen data storage. Not for commercial use, but as I have previously chosen the way to give parameters to the program by its constants and re-building it, there was nothing to loose.
Also worth mention that after the first clarification question (thank you for the quick answer) I decided not to ask more. Why? Not because I consider asking for details a bad practice generally. For a commercial development the best way is to clarify as many dubious issues as possible. But after the opening question, aiming to set the project in the scale of “how much physical” I considered the assignment as is, so I modeled a real-life scenario when there is room for own decisions. For example:
- The initial direction of the car. I have decided between the following two cases: 1. appoint direction 0 to the system’s zero angle – user right, or 2. appoint direction 0 to the “driver’s point of view” that is upwards. I deliberately chose 2. so the starting direction is upwards, this way the tail gets under the “O”.
- The behavior of the movable elements ‘~’. There are two extremes, perfect elastic vs. perfect inelastic collision. From the assignment “On collision '~' absorbs the vehicle's kinetic energy and moves until decelerated”, sounds like perfect elastic collision, otherwise it has been like “absorbs part of…” or “absorbs the half of...”. The interesting scenario is when the car collides not one, but a series of these elements, or at lower velocity than to move the block. I consider these events as normal stopping (not perfect elastic this way). And what happens if the accelerator key is still being

pressed? The whole array moves with the acceleration of maxAcceleration/movingElements? No, I have implemented it a simple way, two or more of the movable blocks are immovable, but not fatal. Is it physical? Yes, consider friction.



- Diagonal collision. In the lack of time I deliberately chose the simple way, so collision occurs only if both the car and the object is in the exact same diagonal.

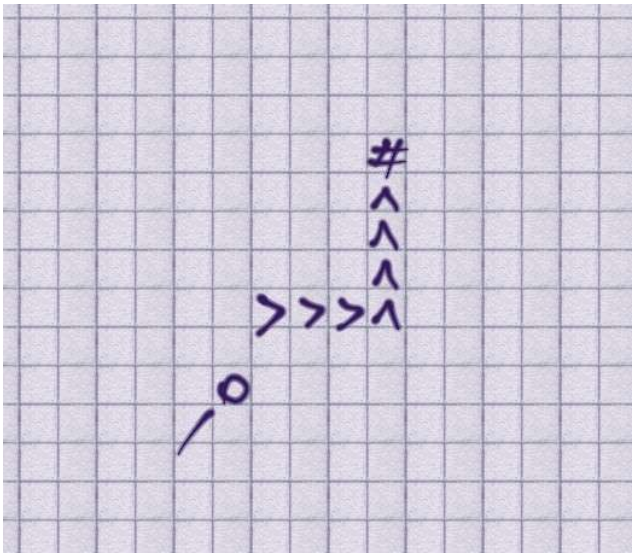


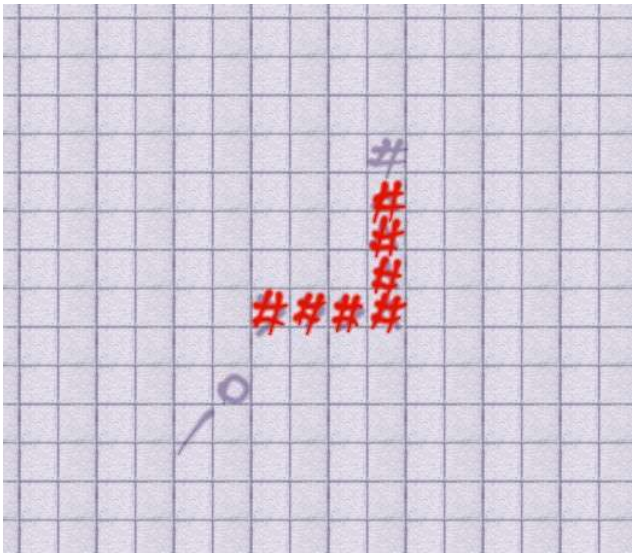
- The behavior of the force fields. If true “If the head part of the vehicle collides with an arrow '>' symbol (from any direction - note by me) the vehicle gains its maximum speed in the direction the symbol is pointing.” then “Forces acting sideways or diagonal to course apply as angular velocities.” can not be true. So I chose the first one, that is the force field sets the direction according to its own, and maximizes the velocity (also zeros the angular velocity and acceleration out). How this is physical? Best imagine as a hybrid of a mudslide and for opposite arrival, a rubber wall. (according to this, the last section of the assignment can be interpreted in three different nature phenomenons. 1: mudslide, 2: side wind, 3: edge of tornado, or being the world “perpendicular” is a misinterpretation, a rubber wall.)
- Collision. Arriving to a grid point happens according to floating point rounding rules, graded by 1.0 for perpendicular pacing, and graded by sqrt(2) for diagonal pacing, the residual velocity values to be stored in the “inter-cellar” member of the car data. If the car arrives to a grid point with a wall element opposite side neighbor, there are two scenarios: 1. the remaining velocity is negative, 2. the same is positive. In the first case the car safely parks at the grid point next to the wall, in the second case collision occurs, and program ends with the message on the event.

Driver assistant (guardian agent thread)

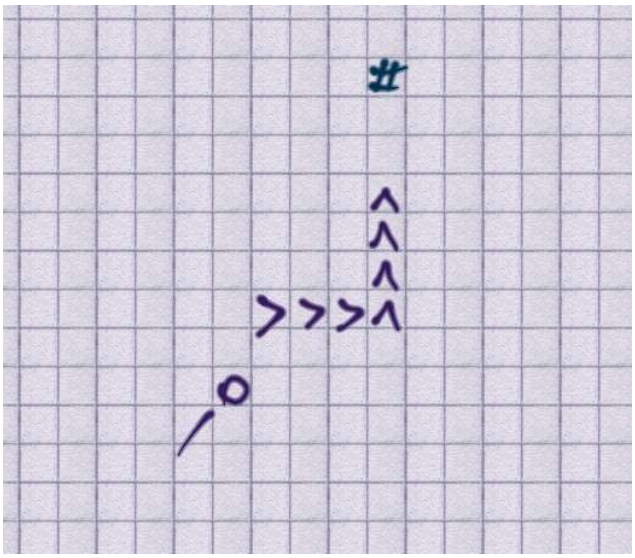
The easy way would be to map the whole universe and provide the driver assistant with this map. The other, more viable way is to let the driver assist “sense” the environment, and make real-time calculations based on the actual car data, that is direction, acceleration and velocity.

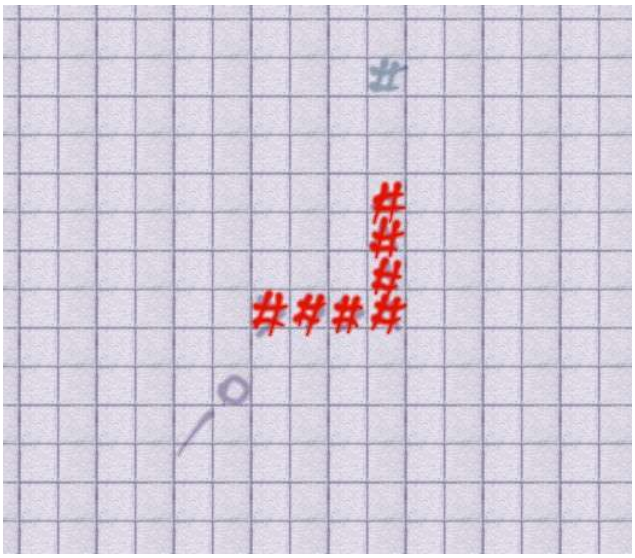
I chose the second scenario, so the guardian agent looks just ahead the pacing direction of the car, and stops the acceleration (apply emergency brake) as dangerous situation being formed.
There is only one exception of this, a pre-registered deathtrap if you like:
If force field chain brings the car unavoidably to a wall, in this case the agent simply re-casts the elements of the force field chain to wall elements.





As force field maximizes the speed, this is also a fatal situation:





An other kind of trap is “><” (see ./misc/scene2.txt) in which case there is nothing to do to the guardian agent, but this situation is of game over also, the only way out is “esc” key. (Physical analogy: if the mudslide sinks into a well, and the car is tumbling on the top of it)
The moving ‘~’ elements can get into force fields. The only way I can handle this situation in the lack of time, if these elements disappear (physical analogy: the truck tires get into the mudslide, and sink into them). Otherwise I should handle the case, the force field array being gradually filled up with mobile blocks if there is a wall element at the end. Possible solution → these mobile elements turn into wall elements. But not in this narrow time frame I have now.
The algorithm to issue warnings and to apply emergency brake:
- Safe range is the distance the car proceeds until guardian agent’s response has been issued. Safety range is calculated, based on the maximum distance in worst case scenario time slippage. Time slippage is continuously updated based on measurement.
- Calculates the brake range in which collision possibilities to be checked
- 1.5 times brake range for issuing warnings, simple brake range for disabling acceleration.
- Checks if there is significant angular velocity (larger than epsilon). If so, calculates the position the rotation snaps, and whether from that position there is an object on the new track. Warnings or emergency break is applied on that calculation.

Doxygen

Doxygen’s main aim to document interfaces, e.g. classes and enumerations, but has little to nothing to do with algorithms and code. Why? Because each and every piece of code you want to include in your document have to be duplicated, and the most hated things by coders are… yes, redundancies. At this point the code almost unreadable, and hard to maintain, as you have to enter each such documented snippets’ modifications twice.
But… Having an old bug (or unpublished feature?) makes Doxygen somehow applicable to function body also. This bug is that local code sample can be inserted in the comment (so in the documentation) as “@snippet this”, if the bookmarks are NOT in doxygen-style comment, like “///[bookmark]”, but traditional c-style as “//[bookmark”, because in the first case the bookmarks appear in the doc, that makes it kind of unusable but in the second case do not. This phenomenon was widely utilized throughout my code comments.