ERHVERVSAKADEMI AARHUS

Car game.

Programming a car game with a little help from an LLM.

Exercise.

A car game in Python, with help from an LLM.

Sila. October 18th, 2024.

In this exercise: Work with an LLM to create a car game. Focus on the car game functionality. Notice: Writing clean code is obviously always important. But here you are asked to "play along" with the code you are given by the LLM. And only refactor the code, when you think the code given to you, from the LLM, becomes too "smelly". You can even ask the LLM to help you with refactoring.

So, here we go. We are going to create a car game with the help of an LLM.

Exciting.

Exercise 1.

So, we begin chatting with ChatGPT (version ChatGPT o1-preview).

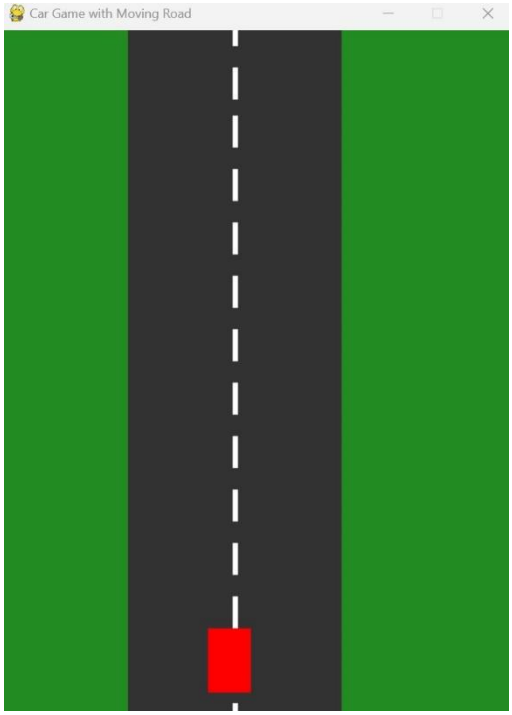
I want to make a car game. First I need Python code that can visually show a lane where a car is dring on. It should be possible to control the car with the arrow keys, and inside the code there should be easy access to parameters that indicate where the car currently is on the road, as well as a parameters that tell where the sides of the road is. So that the driver can get indication if he is no longer on the road.

Tanker om 34 sekunder

Sure! Below is a simple Python program using Pygame that displays a car moving on a lane (road). You can control the car using the left and right arrow keys. The code includes parameters that indicate the car's position and the sides of the road, allowing you to detect when the car is off the road.

Interestingly, and a little surprisingly, this worked fine out of the box.

Running the code, it looks like this:



Perhaps not the most exciting car game you ever saw, but a good start.

You can now ask ChatGPT for similar code, or you can run the code CarGameInitialVersion.py.

You can control the car with arrows, and the car does appear to be moving. So, that's all fine.

The car is yellow off road, and red on the road. Ok. Fine.

Still, you can, of course, change everything you are not pleased with, or ask ChatGPT to do it for you.

Exercise 2.

In order to make the game more exciting, I will ask for a more winding road.

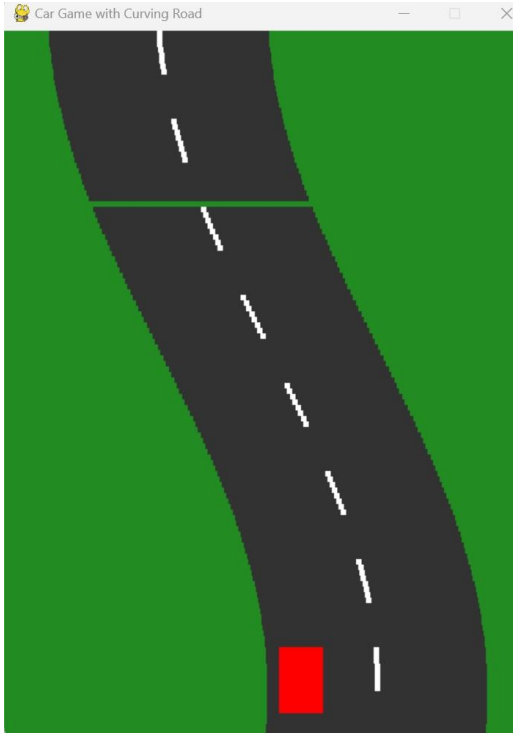
Can you make changes to the code so that you can see the road winding on the screen?

Tanker om 7 sekunder

Certainly! To visually represent a winding road where you can see the road curving ahead, we'll need to simulate a 3D perspective in a 2D environment. This technique is often referred to as **pseudo-3D rendering**.

We'll model the road as a series of horizontal slices (segments), each drawn as a trapezoid (quadrilateral), where the width and position of each segment vary to simulate curves and perspective. The road will appear to narrow towards the top of the screen, giving the illusion of depth.

Ok. That's actually more than I asked for.



Looks good visually.

Inside the code I am already beginning to be nervous though, about the "update road in slices part", and how this will play out as we move on with the project.

```
# Update road slices
    for slice_info in road_slices:
        # Move the slice down the screen
        slice_info['y'] += scroll_speed
```

```
# If the slice is off the bottom, reset to the top
if slice_info['y'] > HEIGHT:
    slice_info['y'] -= HEIGHT + slice_height
```

Well. So, if a slice is down below, we change its coordinates? Easy code, probably, but not super, intuitively, easy to understand code. A little “smelly”, Imho. I.e. how will this work out for us as we move on, later, with adding more functionality to the game?

And all of this talk about a 3D perspective in a 2D environment. Hallucinating?

Anyway. Nevermind. We will take the code for now.

Try out the code by running the file CargameWindingRoad.py. Or ask ChatGPT for your version of the code.

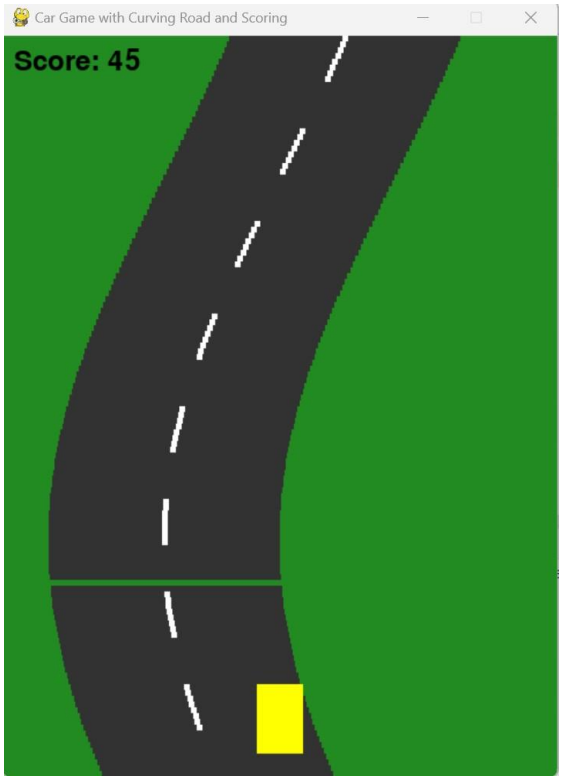
Exercise 1.3.

So far, so good. But its not a game unless the user receives points for doing well, and no points for not doing so good…

Getting that out of ChatGPT was not so easy… And took several prompts and much explaining.

Looking at the very few changes needed to make that work, you can’t help thinking that it would have been easier just to add the scoring to the code yourself.

Anyway, the end result of the discussion can be seen in Cargame.py.



Try out the code. Or make your adjustments to the code to add points to the game.

```
# Render the score
score_text = font.render(f"Score: {score}", True, BLACK)
window.blit(score_text, (10, 10))  # Draw score at top-left corner
```

Exercise 4.

Now, we need to add some traffic to the game.

Making a good prompt for that appears to be a tricky thing. Certainly, it was a struggle to run the code that was coming from ChatGPT (by all means, try it yourself).

So, in order to proceed, we might need to do some “chain of thought” (COT) prompting here.

First, we need a place to spawn the new cars, at which stage the idea about moving the road in “slices” becomes annoying?

Anyway, we agreed on code like this:

```
# Spawn traffic cars
current_time = pygame.time.get_ticks()
if (current_time - last_spawn_time) > traffic_spawn_interval:
    last_spawn_time = current_time

    # Spawn traffic for slice
    slice_info = road_slices[0]
    # Traffic y coordinate
    traffic_lane_y = slice_info['y']

    # Calculate the curvature offset for this slice
    curve = math.sin((slice_info['y'] + total_distance) * curve_frequency) * curve_amplitude
    # Update the x-coordinate of the slice
    traffic_lane_x = (WIDTH - road_width) // 2 + curve + slice_info['traffic_road_offset']

    # Create traffic car rect
    traffic_rect = pygame.Rect(traffic_lane_x, traffic_lane_y, traffic_width, traffic_height)
```

But that’s, of course, not enough, we also need to add code that checks if our car is running into the traffic. After some “debate”, we settled on this code:

```
# Check if  traffic collides with car
slice_info = road_slices[0]
# Traffic y coordinate
traffic_lane_y = slice_info['y']
if (abs(car_y - traffic_lane_y) < 20) and (slice_info['DoneCrash'] == False):
    # Calculate the curvature offset for this slice
    curve = math.sin((slice_info['y'] + total_distance) * curve_frequency) * curve_amplitude
    # Update the x-coordinate of the slice
```

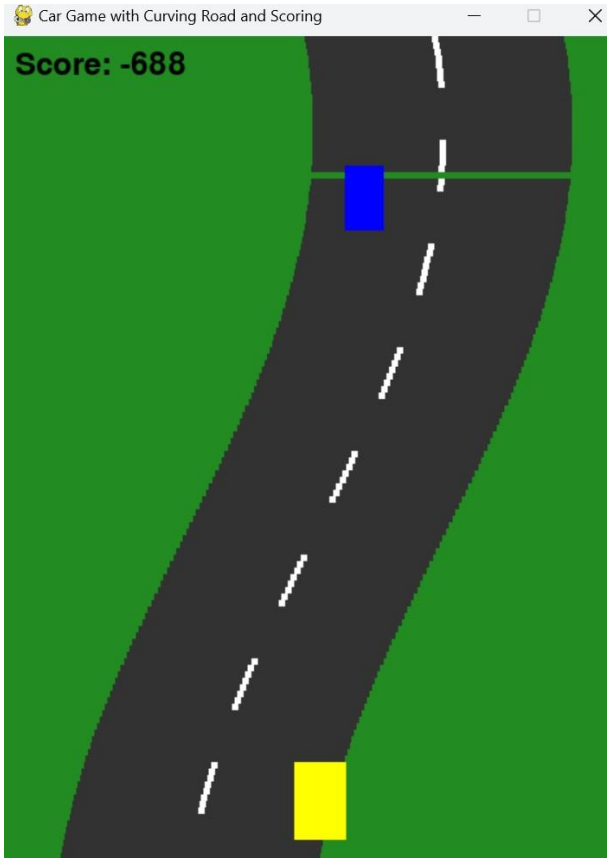
```
    traffic_lane_x = (WIDTH - road_width) // 2 + curve + slice_info['traffic_road_offset']
    if abs(car_x - traffic_lane_x) < 20:
      # Crash
      slice_info['DoneCrash'] = True
      crash_text = font.render(f"Crash: -1000 Points", True, RED)
      window.blit(crash_text, (10, 50))  # Draw score at top-left corner
      # Update the display
      pygame.draw.rect(window, WHITE, car_rect)
```

A lot of the “magic” comes down to understanding which slice is holding the traffic, and where that is compared to our car. That you need to recalculate “curvature offset”, and what have you, for doing that is…well, smelly, imho. So, this probably needs some refactoring, to make the coding of other, later features easier.

So, well, as in many software teams, the team members were kind of yelling at each other here. Arguing about who was making good clean code, and who was making smelly, confused code.

It was time for the project manager, that was also me, to step in and say “enough”, we have a deadline to meet, so we will deliver this code, and make room for refactoring in upcoming sprints (Shipping error prone code, with the excuse that we are doing refactoring later on… Well, don’t do that IRL…).

Anyway, the game looked cool:



The code is available in the file CargameTraffic.py. But you can also try (and see if it is possible for you) to make some smart prompts, that will make it possible for you to get the code directly from ChatGPT.

Exercise 5.

Collecting data.

In this day and age, we obviously also want our car to be self-driving. In this simple setting, we could, of course, just code this with a few if-statements. But that wouldn't be much fun…

We also know that computers can learn to play Atari games like Pong from raw pixel input. So, raw pixel input could probably also work here. (But) Training that is likely to take some time, and a good graphics card.

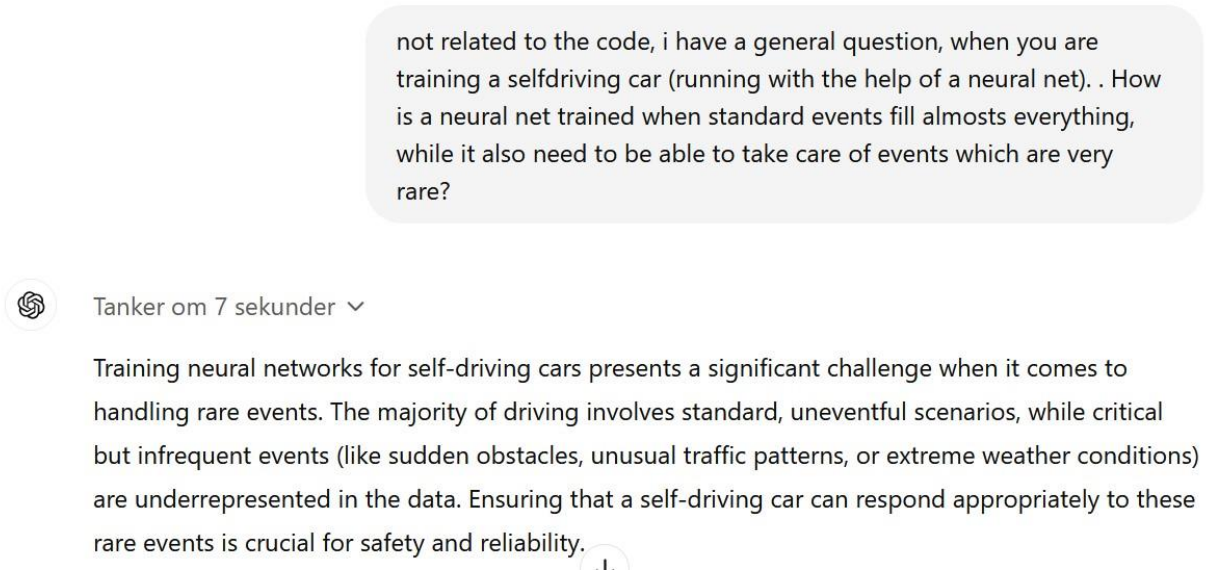
So here, it is probably better to proceed with the simplest possible input: Where are we (offset from the middle of the road) and where is the traffic (lane 1 or 2). Collect data for that along with the user action. And see if some machine learning algorithm can learn to reproduce (good) user actions.

Things are still not entirely rosy, though.

Playing the game in its current form, it quickly becomes obvious that a player will do reasonably well by just driving straight ahead. So, an algorithm trained on such data will probably also do reasonably well, by just going straight ahead. Not paying too much attention to every twist and turn of the road, and not being too concerned about traffic passing by very closely.

- Not what we want. We want the car to be able to self-drive, in such a way that ***it follows the road closely, and not being in the same lane as traffic.***

I.e how can we make sure that the self-driving car also deals with "rare" events. ChatGPT says:



Yes.

And ChatGPT gets even more to the point later on in its answer:

4. Handling Class Imbalance

- Resampling Techniques:
  - **Oversampling Rare Events:** Duplicate or slightly modify rare event data to increase its representation in the dataset.
  - **Undersampling Common Events:** Reduce the number of standard event examples to balance the dataset, though this risks losing valuable information.
- **Synthetic Minority Over-sampling Technique (SMOTE):** Generate synthetic examples of the minority class by interpolating between existing examples.
- **Adaptive Resampling:** Adjust the sampling strategy dynamically during training based on model performance.

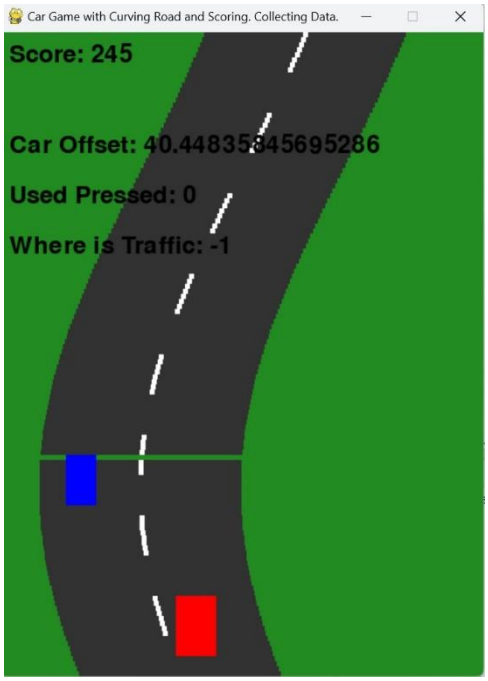
So, maybe **here** it would be possible to train an algorithm on collected data, if we just oversample the rare events (when the user is actually doing something, rather than just going straight ahead).

And, eventually, after a lot of back and forth with ChatGPT, there was even some code for doing that. But, sadly, again, smelly code.

So, on with the thinking cap again: What about making it easier by forcing the driver to make small moves all the time? I.e. not allowing the driver to just go straight? And then "cheating" by collecting those datapoint from another computer program, running the car…?

Indeed, there are many possibilities when it comes to data collecting.

In the ***first version*** the user is driving the car, and we are collecting road offsets (where is the car compared to the car), and traffic position (which lane).



Getting that code out of ChatGPT was not easy, though… And making it work took some final manual adjustments (many).

The code is available in CarGameCollectData.py

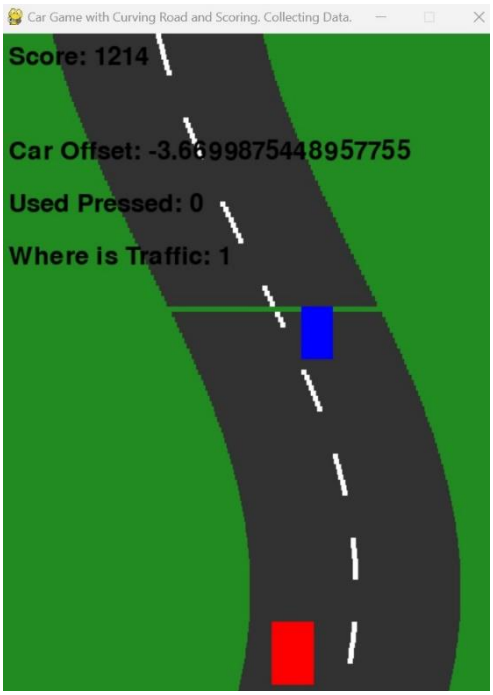
It collects data in this format (offset, lane, useraction):

94	91,60769	-1	0
95	90,57666	-1	0
96	89,48903	-1	0
97	88,34547	-1	0
98	87,14669	-1	0
99	85,89345	-1	0
.00	84,58653	-1	0
.01	83,22674	-1	0
.02	76,81495	-1	-1
.03	70,35202	-1	-1
.04	63,83887	-1	-1
.05	57,27645	-1	-1
.06	50,66573	-1	-1
.07	44,00773	-1	-1
.08	37,30348	-1	-1
.09	30,55403	-1	-1

Again: Making a relatively simple algorithm that can drive based on this training data is probably a little bit tricky though? And will take some additional adjustments. Certainly, here, the code for making this work quickly became (very) smelly, and will not be included here.

Still, feel free to experiment with this though.

To make it much, much simpler, another data collection was also tried. Here, a computer runs the car, and is forced to go either a little left or right on all data collection points.



The collected data looks like this (offset, lane, user/computer action):

266	90,0222	1	-1
267	87,52194	1	-1
268	85,02009	1	-1
269	82,51511	1	-1
270	80,00543	1	-1
271	77,4895	1	-1
272	74,96577	1	-1
273	72,43268	1	-1
274	69,88869	1	-1
275	67,33228	1	-1
276	64,76191	1	-1
277	62,17606	1	-1
278	59,57323	1	1
279	56,95192	1	1
280	54,31064	1	1
281	51,64791	1	1
282	48,96228	1	1
283	46,25231	1	1

The code is available in CarGameCollectData_SelfDrive.py

Feel free to adjust the code, to work on your own way of collecting data to train on (in later steps).

Exercise 6.

So, is it possible for an algorithm to learn how to drive just as the user, or the computer?

Yes. For the simple (computer generated) data, this is a standard scenario, so ChatGPT can easily provide standard (clean) code for learning that task.

Running the code (with only a few manual corrections) on the collected data (game_data.xlsx) gives a perfect result (confusion matrix) for the test data:

[[311 0]

[0 280]]

And therefore an accuracy of 100 %. I.e. this was easy to learn with a simple algorithm.

Feel free to experiment with this by looking at the code in CarActionPrediction.py

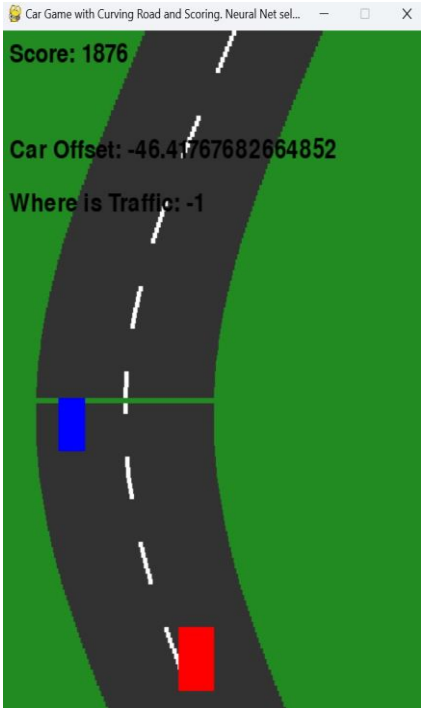
Exercise 7.

The final step. We are now ready to let the neural net drive the car.

Asking ChatGPT to copy paste the new neural net code into the existing game was not so easy though (You might have more luck with this?). It was like asking a co-worker to make a small correction to a simple error, and then afterwards realize that the co-worker had made major (random?) changes all over the program.

Not, what we wanted.

After some back and forth the copy paste of the neural net code into the CarGameTraffic.py ended up being a manual job. Still, a pretty easy job though.



The code is available in CarGameTraffic_NeuralNetSelfDrive.py.

Conclusion.

And that is it. A car game in Python, even capable of self-drive.

With many possibilities for add-ons: More traffic, increased speed, better self-drive, different roads etc.

Feel free to explore and how you can improve the game.