IBM BLOG POST

When we first set out to tackle the race, we quickly realized that downloading **TORCS** and the *gym_torcs* files was just the **tip of the iceberg**. The bigger challenge was understanding what the code was doing.

Thanks to what we learned from **IBM SkillsBuild's** Project Design and Requirements Gathering, we decided to create a detailed list of requirements to get a clear picture of what needed to happen. From there, we **prioritised**: understanding the code came first, tasks like drafting a presentation can be saved for later.

To tackle the work efficiently, we split the team. Some of us dove straight into the code, analysing its structure and logic, while others explored different ways to implement our AI car, carefully considering which approach would be the most practical and effective.

At first, the code felt like a completely foreign language. But this is where **IBM Granite** made a real difference. It helped us break down the code and understand it faster than we ever could have on our own, transforming a **daunting wall** of scripts into a clear roadmap for building our AI car.

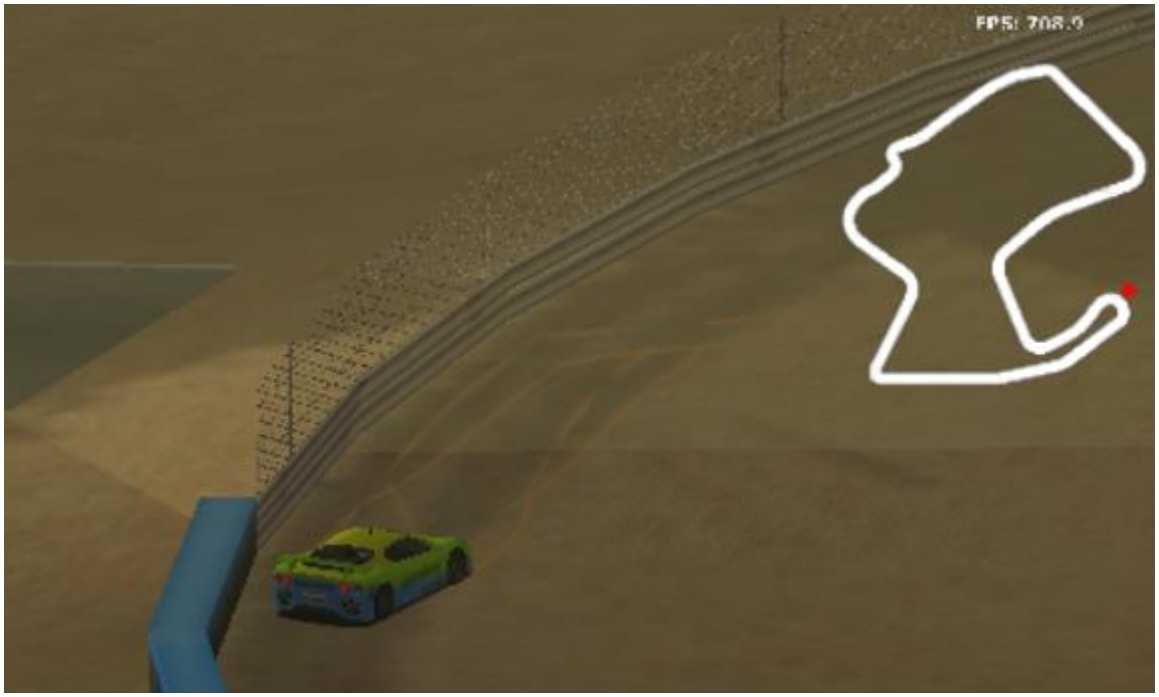
```
sensors= [ # Select the ones you want in the order you want them.
'stucktimer',
'fuel',
'distRaced',
'distFromStart',
'opponents',
'wheelSpinVel',
'z',
'speedZ',
'speedY',
'speedX',
'targetSpeed',
'rpm',
'skid',
'slip',
'track',
'trackPos',
'angle',
]
```

```
def __init__(self):
    self.actionstr= str()
    self.d= { 'accel':0.2,
              'brake':0,
             'clutch':0,
              'gear':1,
             'steer':0,
             'focus':[-90,-45,0,45,90],
              'meta':0
              }
```

IBM Granite was a game-changer for us when it came to understanding specific chunks of code, especially the sensors and their ranges. While some of the sensor labels seemed straightforward, it was crucial to clarify what each sensor actually does and its purpose. For instance, the role of the **track sensor** wasn't immediately obvious - but with **Granite**, we learned that it stores the distances from the car to the track edges at various angles. That insight made a **huge difference**!

Grasping the code in this way allowed us to **develop more quickly** and **efficiently**, particularly when it came to using these sensors - which, spoiler alert, we ended up making use of in our AI car. 🚗 🛞

Running the provided code files was another key moment. We realised that *`torcs_jm_par.py`* already gave us a **solid framework** to build on, letting us deploy a working AI car rapidly. But to really push for **fast** lap times, we knew we had to tweak the driving code. The default behaviour was alright, but without modifications, the car would **overshoot corners** or **crawl** through the track - definitely not what we wanted for the race!



When the rest of the team returned with their research, we came to an important decision given our **time constraints** and **limited experience with reinforcement learning**, a **rule-based AI approach** was the most practical path forward. This decision was strongly supported by **IBM SkillsBuild**, particularly courses such as **Academic Innovation with IBM**, which helped guide us on **how to structure and approach the project effectively**.

Our **coding strategy** was deliberately simple and iterative. We began by defining a set of **general racing rules**, implementing them one by one, and then using **IBM Granite** to suggest **optimisations aimed at improving lap times**. For example, a basic rule like

slowing down when the distance ahead decreases would be implemented, tested extensively, and refined by adjusting parameters to see whether it was truly effective.

As a result, a significant portion of our time was spent on **testing rather than pure development**. While this may seem counterintuitive, it was essential during the early stages to ensure that each rule behaved reliably on track. This careful, test-driven approach paid off, and by the end of this development phase, the fastest lap we achieved was - **02:33:24**.

Rank	Driver	Total	Best	Laps	Top Spd	Damages	Pit Stops
1	scr_server 1	02:33:24	02:33:24	1	150	0	0

However, this code was catered towards the **old model** of the car, once the **new F1 car** was provided, our code no longer worked.

This was initially **demoralising**, but it quickly became clear that understanding *why* this was happening was far more important than the setback itself. With the help of **IBM Granite**, we uncovered that the **F1 car operates under a completely different physics model** compared to the previous car we had been working with.

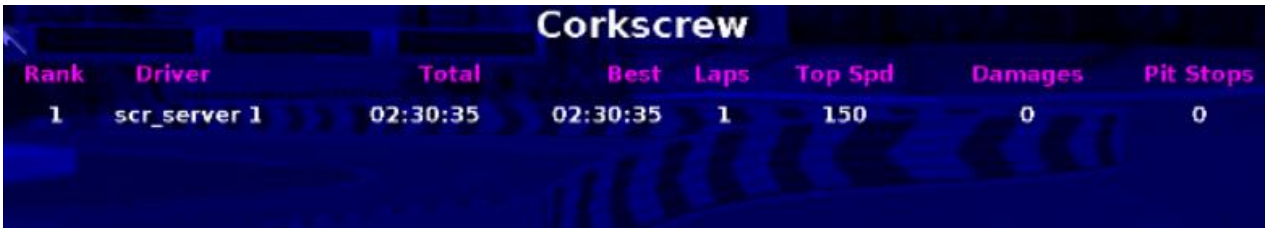
As a result, our AI car began to **understeer heavily**, largely due to excessive braking combined with continued acceleration while navigating corners. This mismatch in behaviour explained why our earlier approach was no longer producing the results we expected.

It also became apparent that the **legacy code was difficult to interpret**. Some sections were disorganised, and in certain cases, it was unclear why they worked at all. Identifying these issues gave us the clarity we needed to rethink our approach and adapt the code more effectively to the new physics model.

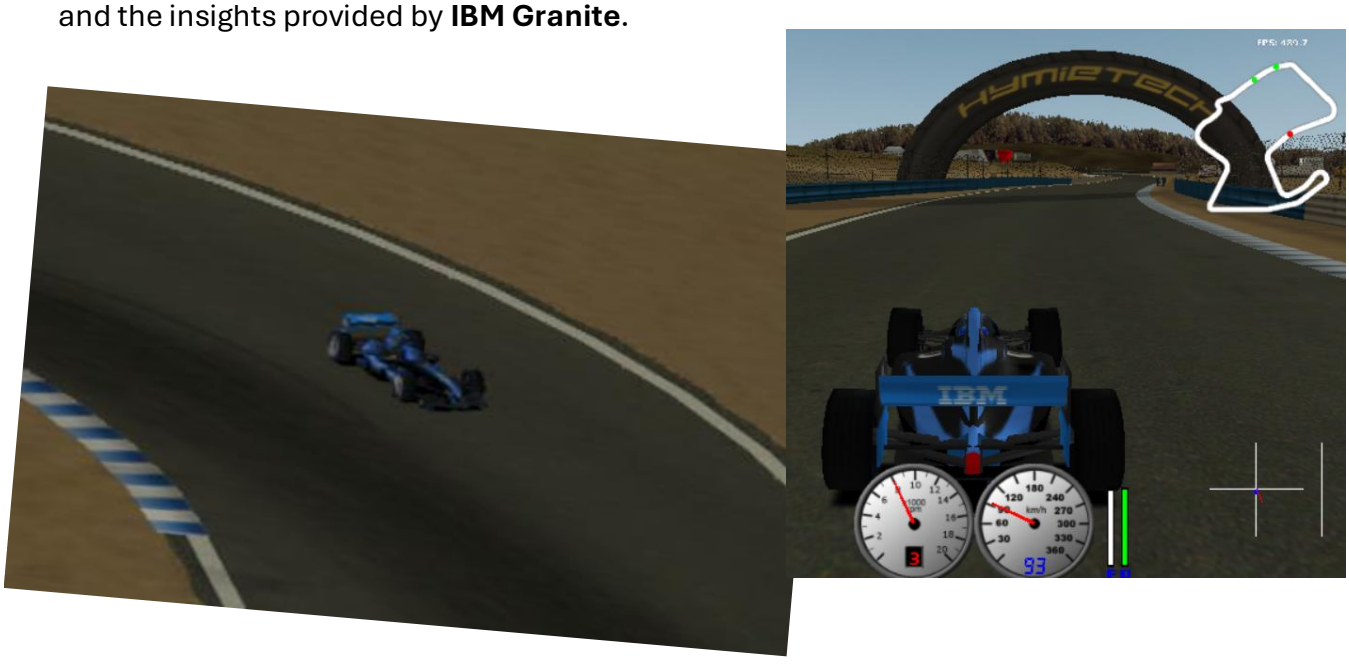
```
if (S['track'][9] < (S['speedX'] * 0.6) or isCorner(min_reading)):
    safe_speed = min(TARGET_SPEED, max(65, 30 + ((TARGET_SPEED / math.sqrt(calculateSeverity(S) + 0.01))) *  1/S['track'][9]))
    if (S['speedX'] > safe_speed):
        brake = ((S['speedX'] / STEER_GAIN) * BRAKE_THRESHOLD) + ((1 /getCornerSensor(S)) / STEER_GAIN) + 1/S['track'][9]
```

While it was disappointing to see our **previous driving code no longer perform as expected**, it also gave us the opportunity to **start again with a clearer understanding** and learn directly from our earlier mistakes. Rather than viewing it as a setback, we treated it as a reset point.

With just a **simple but deliberate change - reducing acceleration to zero as the car approached a corner -** we immediately saw measurable improvement. This small adjustment had a significant impact, allowing us to achieve a **faster lap time of 02:30:35**, even at this early stage of refinement.



This moment reinforced an important lesson for us: **small, well-informed changes can lead to meaningful performance gains**, especially when guided by careful analysis and the insights provided by **IBM Granite**.



Another key improvement focused on braking behaviour. Instead of applying heavy braking all at once, we **gently incremented braking** as a corner approached. This proved crucial, as braking too aggressively caused the car to **understeer**, negatively impacting lap times and stability.

By carefully balancing speed and control in this way, we were able to make the car both **faster and more reliable**, reinforcing the value of incremental changes backed by consistent testing.

This result was our **best ever lap-time**, until it was beaten the **next** day…

Implementing these ideas led to **consistently faster lap times**, reinforcing that our approach was working. Each adjustment, no matter how small, contributed to noticeable performance gains on track.

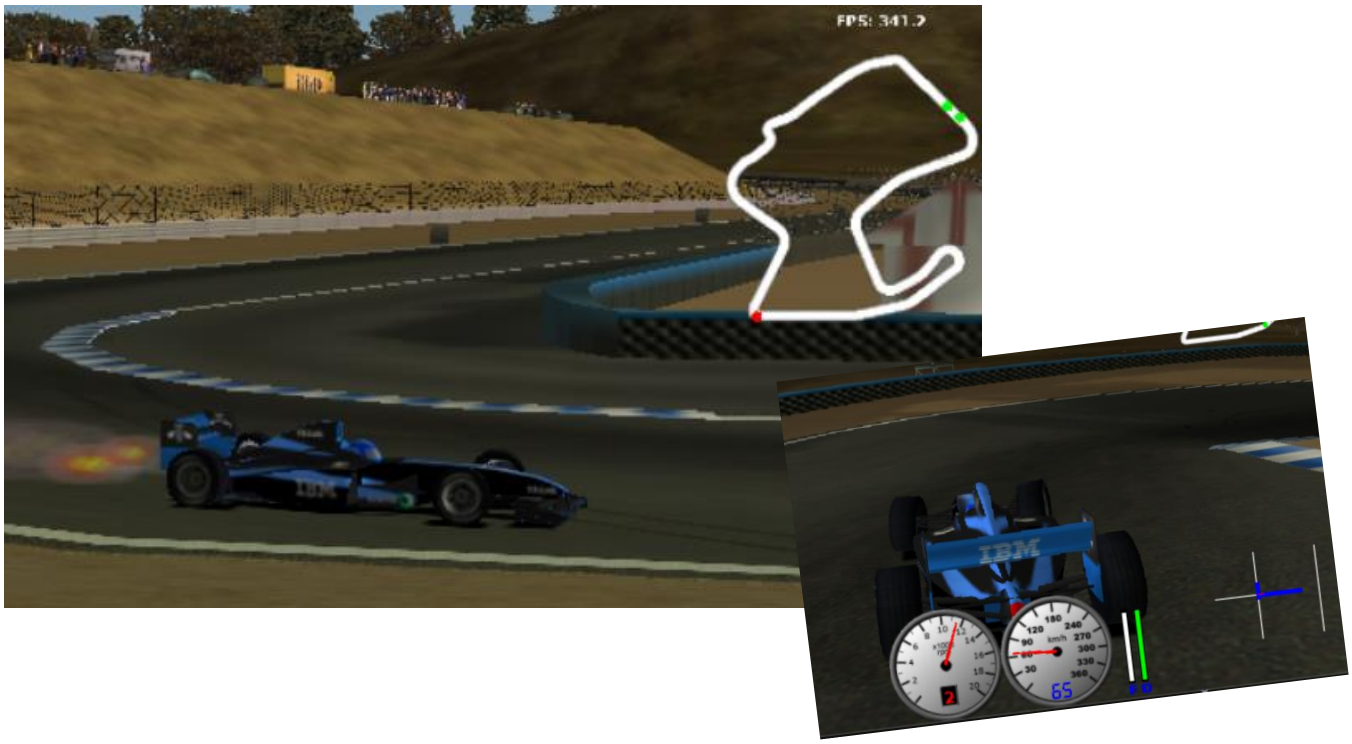
Programming the AI car was however far from easy. Our challenge was to **minimise lap times** while still ensuring the car could **reliably complete the circuit**. Striking this balance required careful tuning and constant testing.

This challenge was amplified by the track itself - **Laguna Seca**. With its tight corners and demanding elevation changes, the circuit left little room for error, forcing our AI to be precise rather than simply be fast!

This track presented two particularly **problematic corners - the Corkscrew** and **the final corner**. Both were **highly unforgiving**, leaving little margin for error.

To overcome this, we relied heavily on **extensive testing**, gradually identifying the **highest safe speed** at which the car could consistently navigate each corner without losing control. Rather than chasing raw speed, we focused on finding a balance

between **aggression and stability**, ensuring the AI could handle these sections lap after lap.



Once the **prototyping phase** was complete, we shifted our focus to **cleaning up the code**. Drawing on both our own learning, **IBM SkillsBuild** and guidance from **IBM Granite**, we aimed to make the codebase **clearer and more readable.**

This process involved breaking down long sections of logic and **refactoring them into well-defined functions**, each with a clear purpose. For instance, we created a dedicated function to **determine whether a corner was approaching**, using data from the **track sensors** to make that decision.

By organizing the code in this way, we not only improved readability but also made it far easier to **test and modify.**

```python
# Checks if there is a corner coming up
#
def is_corner(S, min_reading):
    if min_reading < CORNER_READING or S['track'][9] < S['speedX'] * 0.65:
        return True

    return False
```

```python
# Checks if the car should start slowing down - and therefore increase the braking to an extent
#
def slow_down(S):
    max_forwards_sensors = max(S['track'][7:12])

    if max_forwards_sensors < S['speedX'] * 0.65:
        return True

    return False
```

With the codebase cleaned up, we moved on to **fine-tuning parameters** and observing how each adjustment influenced lap times. This stage was crucial, but it quickly became clear that manual testing would be both **time-consuming and inefficient**.

To streamline the process, we designed a **simple automation system**. Parameters and resulting lap times were **logged to a CSV file**, and we wrote a script that could **automatically launch TORCS, start a new race, execute the AI driver, and record the relevant performance data** once the run was complete.

Here, **IBM Granite** played a vital role. It helped us understand which **Python libraries** were best suited for the task, particularly *pandas* for handling structured data and *subprocess* for starting up **TORCs**. This approach allowed us to test faster, gather consistent data, and focus our efforts on **analysing results rather than repeating manual steps**.

We also created a dedicated file - *fastest.py*-which contains the **cleaned-up driving logic** and the **optimal parameters** responsible for our best-performing laps. This gave us a reliable reference point and made it easy to distinguish experimentation from proven performance.

At this stage, the focus shifted entirely to **finding the fastest possible laps**. With a solid foundation in place, each new run became an opportunity to push the limits just a little further.

So far, our **fastest recorded lap time** stands at - **02:02:01.**

This is a milestone that reflects the cumulative impact of careful testing, iteration, and the support provided by **IBM SkillsBuild** and **IBM Granite** throughout the project.

This project was an incredible **learning experience** and, honestly, a lot of **fun**. From experimenting with AI driving rules to pushing our car through the twists of **Laguna Seca.**

Thanks to **IBM SkillsBuild** and **IBM Granite**, we could approach the challenge with confidence - breaking down complex code, testing ideas efficiently, and optimising our AI car like true engineers.

We're **grateful for the opportunity** to take on this race, work as a team, and see our ideas come to life on the track. The lessons we learned and the enjoyment we had made this experience truly unforgettable, and the fast lap times are just the cherry on top!

The full code for this project can be found through the following link - *https://github.com/Simple-wood/IBM-TORCs*