6.2 Research and Troubleshooting

Steps:

Step 1: Make the vehicle use forces

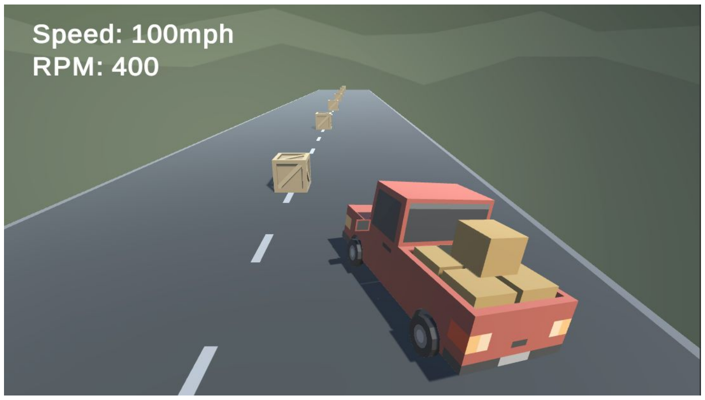
Step 2: Prevent car from flipping over

Step 3: Add a speedometer display

Step 4: Add an RPM display

Step 5: Prevent driving in mid-air

Example of project by end of lesson



Length:	75 minutes
Overview:	In this lesson, you will attempt to add a speedometer and RPM display for your vehicle in Prototype 1. In doing so, you will learn the process of doing online research when trying to implement new features and troubleshoot bugs in your projects. As you will find out, adding a new feature is very rarely as simple as it initially seems - you inevitably run into unexpected complications and errors that usually require a little online research. In this lesson, you will learn how to do that so that you can do it with your own projects.
Project Outcome:	By the end of this lesson, the vehicle will behave with more realistic physics, and there will be a speedometer and Revolution per Minute (RPM) display.,
Learning Objectives:	By the end of this lesson, you will be able to: - Use Unity Forums, Unity Answers, and the online Unity Scripting Documentation to implement new features and troubleshoot issues with your projects

Step 1: Make the vehicle use forces

If we're going to implement a speedometer, the first thing we have to do is make the vehicle accelerate and decelerate more like a real car, which uses forces - as opposed to the Translate method.

1. Open your **Prototype 1** project and make a backup
2. Replace the Translate call with an AddForce call on the vehicle's Rigidbody, renaming the "speed" variable to "horsePower"
3. Increase the ***horsePower*** to be able to actually move the vehicle
4. To make the vehicle move in the appropriate direction, change AddForce to AddRelativeForce

- **New Concept:** using Unity Documentation
- **New Concept:** using Unity Answers
- **New Concept:** AddRelativeForce
- **Don't worry:** Still a big issue where the vehicle can drive in air and that it flips over super easily!

```
private Rigidbody playerRb;

void Start() {
  playerRb = GetComponent<Rigidbody>();
}

void FixedUpdate() {
  ~~transform.Translate(Vector3.forward * speed * verticalInput);~~
  playerRb.AddRelativeForce(Vector3.forward * verticalInput * horsePower);
}
```

Step 2: Prevent car from flipping over

Now that we've implemented real physics on the vehicles, it is very easy to overturn. We need to figure out a way to make our vehicle safer to drive.

1. Add wheel colliders to the wheels of your vehicle and edit their radius and center position, then disable any other colliders on the wheels
2. Create a new ***GameObject centerOfMass*** variable, then in Start(), assign the playerRb variable to the centerOfMass position
3. Create a new **Empty Child** object for the vehicle called "Center Of Mass", reposition it, and assign it to the **Center Of Mass** variable in the inspector
4. **Test** different center of mass positions, speed, and turn speed values to get the car to steer as you like

- **New Concept:** Wheel colliders
- **New Concept:** Center of Mass
- **Don't Worry**: We can still drive the vehicle when it's sideways or upside down
- **Warning:** This is still not the *proper* way to do vehicles - should actually be rotating / turning the wheels

```
[SerializeField] GameObject centerOfMass;

void Start() {
  playerRb.centerOfMass = centerOfMass.transform.position;
}
```

Step 3: Add a speedometer display

Now that we have our vehicle in a semi-drivable state, let's display the speed on the User Interface.

1. Add a new ***TextMeshPro - Text*** object for your "Speedometer Text"
2. Import the **TMPro library**, then create and assign new create a new **TextMeshProUGUI** variable for your ***speedometerText***
3. Create a new float variables for your ***speed***
4. In Update(), **calculate** the speed in mph or kph then **display** those values on the UI

- **Warning**: Will be going fast through adding the text, since we did this in prototype 5
- **New Concept:** RoundToInt

```
using TMPro;

[SerializeField] TextMeshProUGUI speedometerText;
[SerializeField] float speed;

private void Update() {
  speed = Mathf.Round(playerRb.velocity.magnitude * 2.237f); // 3.6 for kph
  speedometerText.SetText("Speed: " + speed + "mph");
}
```

Step 4: Add an RPM display

One other cool feature that a lot of car simulators have is a display of the RPM (Revolutions per Minute) - the tricky part is figuring out how to calculate it.

1. Create a new "RPM Text" object, then **create** and **assign** a new **rpmText variable** for it
2. In Update(), calculate the the RPMs using the Modulus/Remainder operator (%), then display that value on the UI

- **New Concept:** Modulus / Remainder (%) operator

```
[SerializeField] TextMeshProUGUI rpmText;
[SerializeField] float rpm;

private void Update() {
  rpm = Mathf.Round((speed % 30)*40);
  rpmText.SetText("RPM: " + rpm);
}
```

Step 5: Prevent driving in mid-air

Now that we have a mostly functional vehicle, there's one other big bug we should try to fix: the car can still accelerate/decelerate, turn, and increase in speed/rpm in mid-air!

1. Declare a new **List** of **WheelColliders** named ***allWheels*** (or frontWheels/backWheels), then assign each of your wheels to that list in the inspector
2. Declare a new ***int wheelsOnGround***
3. Write a ***bool IsOnGround()*** method that returns true if all wheels are on the ground and false if not
4. Wrap the **acceleration**, **turning**, and **speed/rpm** functionality in if-statements that check if the car is on the ground

- **New Concept:** looping through lists
- **New Concept**: custom methods with bool returns
- **Tip**: if you use frontWheels or backWheels, make sure you only drag in two wheels and only test that wheelsOnGround == 2

```
[SerializeField] List<WheelCollider> allWheels;
[SerializeField] int wheelsOnGround;

if (IsOnGround()) {[ACCELERATION], [ROTATION], [SPEED/RPM]}

bool IsOnGround () {
  wheelsOnGround = 0;
  foreach (WheelCollider wheel in allWheels) {
    if (wheel.isGrounded) {
      wheelsOnGround++;
    }
  }
  if (wheelsOnGround == 2) {
    return true;
  } else {
    return false;
  }
}
```

Lesson Recap

New Concepts and Skills	● Searching on Unity Answers, Forum, Scripting API ● Troubleshooting to resolve bugs ● AddRelativeForce, Center of Mass, RoundToInt ● Modulus/Remainder (%) operator ● Looping through lists ● Custom methods with bool return