unity

1.2 Pedal to the Metal

Steps:

Step 1: Create and apply your first script

Step 2: Add a comment in the Update() method

Step 3: Give the vehicle a forward motion

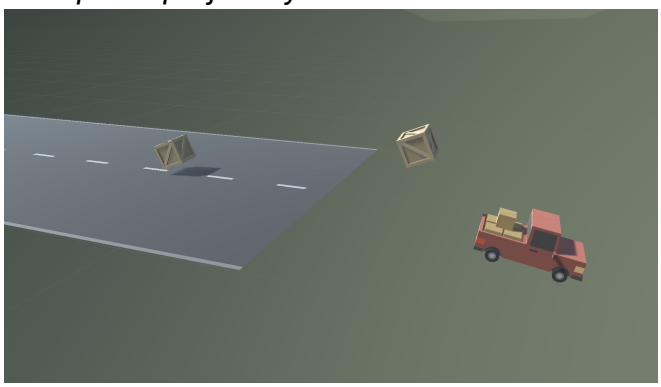
Step 4: Use a Vector3 to move forward

Step 5: Customize the vehicle's speed

Step 6: Add RigidBody components to objects

Step 7: Duplicate and position the obstacles

Example of project by end of lesson



Length: 70 minutes

Overview: In this lesson you will make your driving simulator come alive. First you will write your very first lines of code in C#, changing the vehicle's position and allowing it to move forward. Next you will add physics components to your objects, allowing them to collide with one another. Lastly, you will learn how to duplicate objects in the hierarchy and position them along the road.

Project Outcome: You will have a moving vehicle with its own C# script and a road full of objects, all of which may collide with each other using physics components.

Learning Objectives: By the end of this lesson, you will be able to:

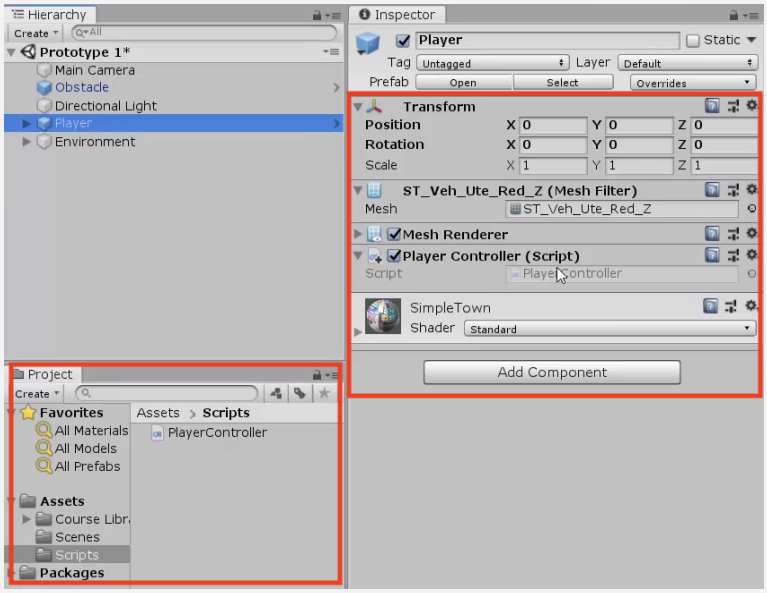
- Create C# scripts and apply them to objects
- Use Visual Studio and a few of its basic features
- Write comments to make your code more readable
- Utilize fundamental C# methods and classes like transform.Translate and Vector3
- Add Rigidbody and Collider components to allow objects to collide realistically
- Duplicate objects in the hierarchy to populate your scene

Step 1: Create and apply your first script

We will start this lesson by creating our very first C# script that will control the vehicle's movement.

1. In the Project window, *Right-click > Create* ***> Folder*** named "Scripts"
2. In the "Scripts" folder, *Right-click > Create >* ***C# Script*** named "PlayerController"
3. **Drag** the new script onto the **Vehicle object**
4. **Click** on the Vehicle object to make sure it was added as a **Component** in the Inspector

- **New Concept:** C# Scripts
- **Warning:** Type the script name as soon as the script is created, since it adds that name to the code. If you want to edit the name, just delete it and make a new script
- **New Concept:** Components



Step 2: Add a comment in the Update() method

In order to make the vehicle move forward, we have to first open our new script and get familiar with the development environment.

1. **Double-click** on the script to open it in **Visual Studio**
2. In the ***Update()*** method, add a comment that you will: ***// Move the vehicle forward***

- **New:** Start vs Update functions
- **New:** Comments

```
void Update()
{
  // Move the vehicle forward
}
```

Step 3: Give the vehicle a forward motion

Now that we have the comment saying what we WILL program - we have to write a line of code that will actually move the vehicle forward.

1. Under your new comment, type ***transform.tr***, then select **Translate** from the autocomplete menu
2. Type **(**, add **0, 0, 1** between the parentheses, and complete the line with a semicolon (**;**)
3. Press **Ctrl/Cmd + S** to save your script, then run your game to test it

- **New Function:** transform.Translate
- **New Concept:** Parameters
- **Warning:** Don't use decimals yet. Only whole numbers!

```
void Update()
{
  // Move the vehicle forward
  transform.Translate(0, 0, 1);
}
```

Step 4: Use a Vector3 to move forward

We've programmed the vehicle to move along the Z axis, but there's actually a cleaner way to code this.

1. **Delete** the 0, 0, 1 you typed and use auto-complete to **replace it** with ***Vector3.forward***

- **New Concept:** Documentation
- **New Concept:** Vector3
- **Warning:** Make sure to save time and use Autocomplete! Start typing and VS Code will display a popup menu with recommended code.

```
void Update()
{
  // Move the vehicle forward
  transform.Translate(0, 0, 1 Vector3.forward);
}
```

Step 5: Customize the vehicle's speed

Right now, the speed of the vehicle is out of control! We need to change the code in order to adjust this.

1. Add **** Time.deltaTime*** and run your game
2. Add **** 20*** and run your game

- **New Concept:** Math symbols in C#
- **New Function:** Time.deltaTime

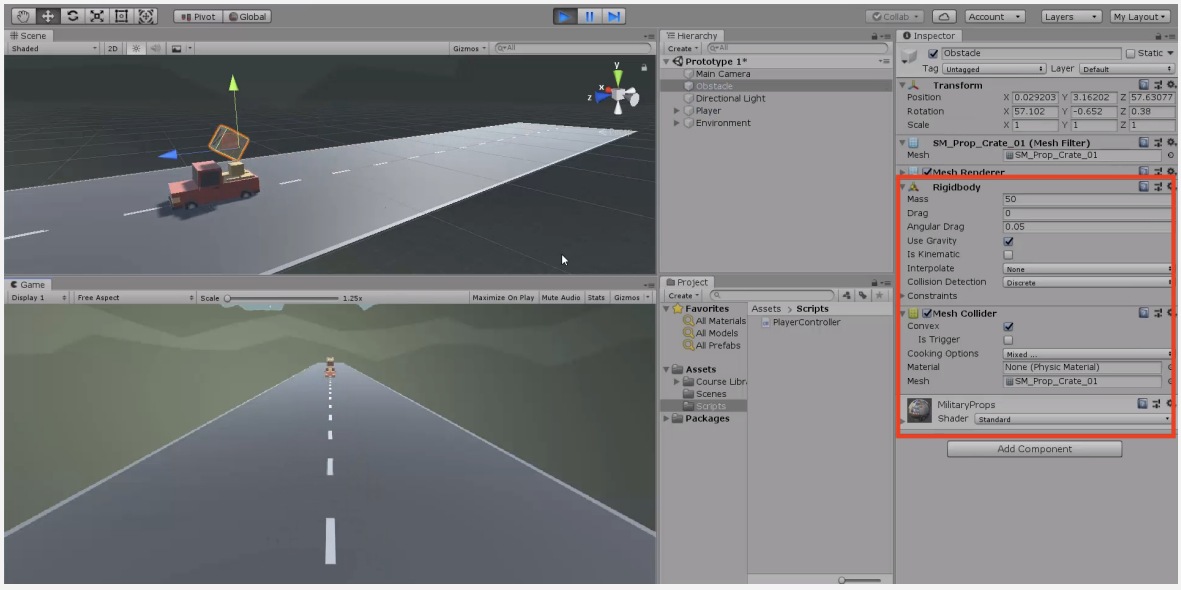
```
void Update()
{
  // Move the vehicle forward
  transform.Translate(Vector3.forward * Time.deltaTime * 20);
}
```

Step 6: Add RigidBody components to objects

Right now, the vehicle goes right through the box! If we want it to be more realistic, we need to add physics.

1. Select the **Vehicle**, then in the hierarchy click **Add Component** and select **RigidBody**
2. Select the **Obstacle**, then in the hierarchy click **Add Component** and select **RigidBody**
3. In the RigidBody component properties, increase the **mass** of vehicle and obstacle to be about what they would be in **kilograms** and test again

- **New Concept:** Rigidbody Component
- **New Concept:** Collider Component
- **Tip:** Adjust the mass of the vehicle and the obstacle, and test the collision results

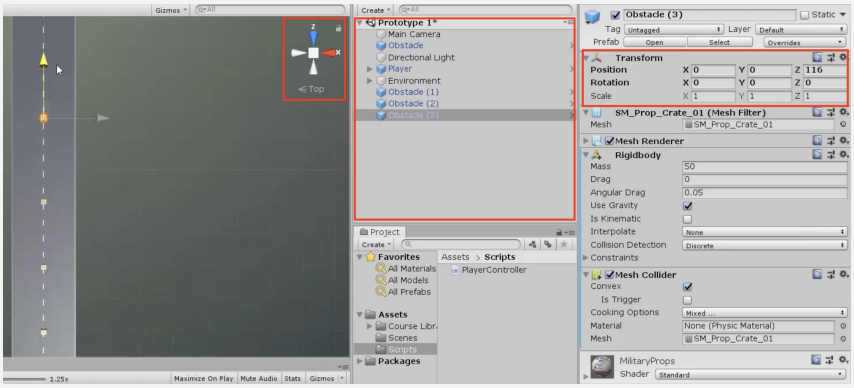


Step 7: Duplicate and position the obstacles

Last but not least, we should duplicate the obstacle and make the road more treacherous for the vehicle.

1. Click and drag your obstacle to the **bottom of the list** in the hierarchy
2. Press **Ctrl/Cmd+D** to duplicate the obstacle and move it down the **Z axis**
3. Repeat this a few more times to create more obstacles
4. After making a few duplicates, select one in the hierarchy and **hold ctrl + click** to select multiple obstacles, then **duplicate** those

- **New Technique:** Duplicate (Ctrl/Cmd+D)
- **Tip:** Try using top-down view to make this easier
- **Tip:** Try using the inspector to space your obstacles exactly 25 apart



Lesson Recap

New Functionality	• Vehicle moves down the road at a constant speed • When the vehicle collides with obstacles, they fly into the air
New Concepts and Skills	• C# Scripts • Start vs Update • Comments • Methods • Pass parameters • Time.deltaTime • Multiply (*) operator • Components • Collider and RigidBody
Next Lesson	• We'll add some code to our camera, so that it follows the player as they drive along the road.