1.1 Start your 3D Engines

Steps:

Example of project by end of lesson

Length: 70 minutes

Overview: In this lesson, you will create your very first game project in Unity Hub. You will choose and position a vehicle for the player to drive and an obstacle for them to hit or avoid. You will also set up a camera for the player to see through, giving them a perfect view of the scene. Throughout this process, you will learn to navigate the Unity Editor and grow comfortable moving around in 3D Space. Lastly, you will customize your own window layout for the Unity Editor.

Project Outcome: You will have a vehicle and obstacle positioned on the road and the camera set up perfectly behind the vehicle. You will also have a new custom Unity layout, perfectly optimized for editing.

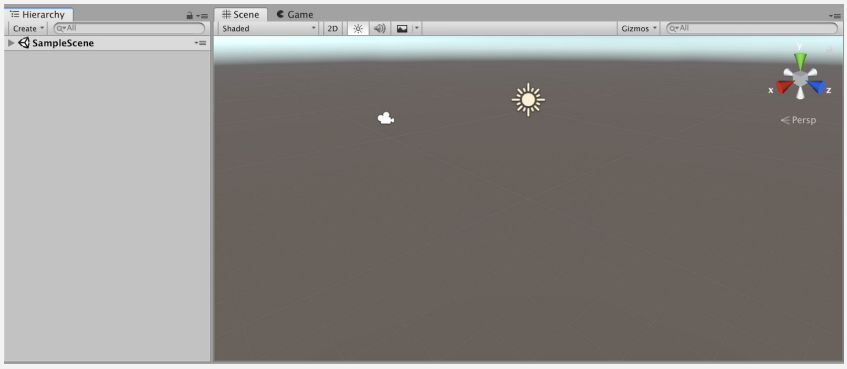
Learning Objectives: By the end of this lesson, you will be able to:
- Create a new project through Unity Hub
- Navigate 3D space and the Unity Editor comfortably
- Add and manipulate objects in the scene to position them where you want
- Position a camera in an ideal spot for your game
- Control the layout of Unity Editor to suit your needs

Step 1: Make a course folder and new project

The first thing we need to do is create a folder that will hold all of our course projects, then create a new Unity project inside it for Prototype 1.

1. On your **desktop** (or somewhere else you will remember), *Right-click > create* ***New Folder***, then name it "Create with Code"
2. Open **Unity Hub** and click **New**
3. Name the project "Prototype 1", select the correct **version of Unity**, set the location to the new "**Create with Code**" folder, and select the **3D** template
4. Click **Create Project**, then wait for Unity to open

- **Don't worry:** Unity might take a while to open, so just give it some time

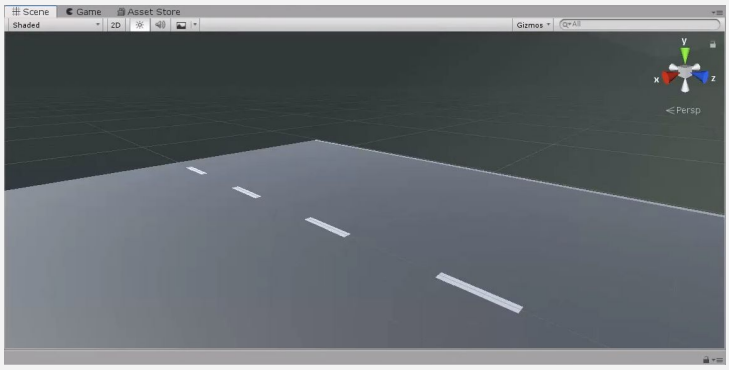


Step 2: Import assets and open Prototype 1

Now that we have an empty project open, we need to import the assets for Prototype 1 and open the scene

1. Click on one of the links to access the Prototype 1 starter files, then **download** and **import** them into Unity
2. In the **Project** window, in *Assets > Scenes >* double-click on the **Prototype 1 scene** to open it
3. Delete the **Sample Scene** without saving
4. **Right-click + drag** to look around at the start of the road

- **Warning:** You're free to look around, but don't try moving yet
- **Warning:** Be careful playing with this interface, don't click on anything else yet
- **New Concept:** Project Window

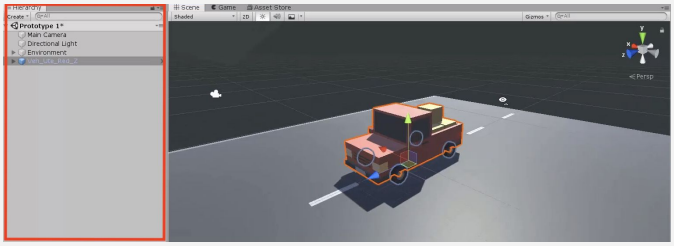


Step 3: Add your vehicle to the scene

Since we're making a driving simulator, we need to add our own vehicle to the scene.

1. In the **Project Window**, open *Assets > Course Library > Vehicles*, then drag a vehicle into the **Hierarchy**
2. **Hold right-click + WASD** to fly to the vehicle, then try to rotate around it
3. **Press F** in the Scene view to focus on it, then use the **scroll wheel** to zoom in and out and **hold the scroll wheel** to pan
4. Press F to focus on it, **hold alt+left-click** to rotate around it perfectly
5. If anything goes wrong, press **Ctrl/Cmd+Z** to Undo until it's fixed

- **New:** Hierarchy
- **New: Undo** (Cmd/Ctrl + Z) and **Redo** (Cmd+Shift+Z / Ctrl+Y)
- **Warning:** Mouse needs to be in scene view for F/focus to work
- **New Technique:** Scroll Wheel for Zoom and Pan

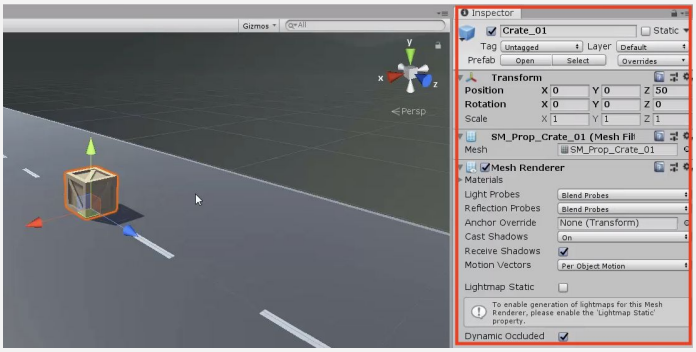


Step 4: Add an obstacle and reposition it

The next thing our game needs is an obstacle! We need to choose one and position it in front of the vehicle.

1. Go to *Course Library > Obstacles* and **drag an obstacle** directly into **scene view**
2. In the Inspector for your obstacle, in the top-right of the Transform component, click the *Gear Icon* > ***Reset Position***
3. In the **Inspector,** change the **XYZ Location** to **0,0,25**
4. In the hierarchy, *Right-click* > ***Rename*** your two objects as "Vehicle" and "Obstacle"

- **New Concept:** XYZ location, rotation and scale
- **New Concept:** Inspector

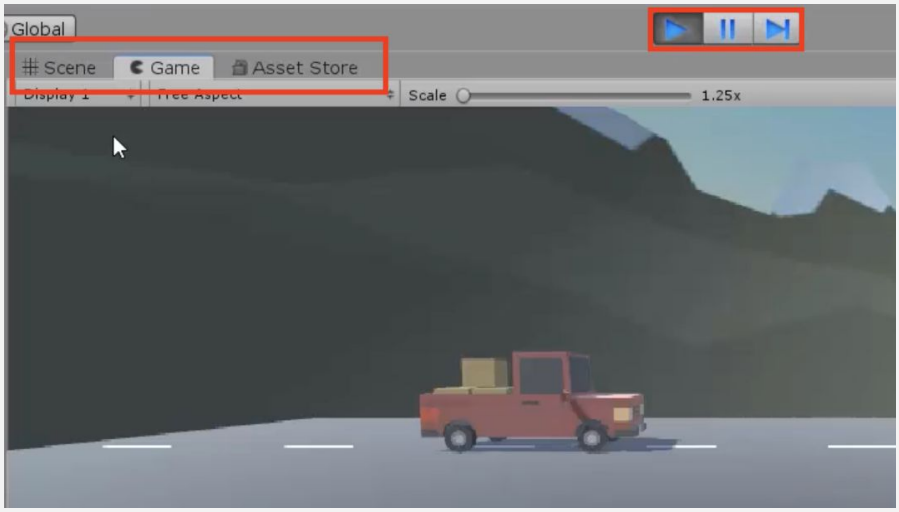


Step 5: Locate your camera and run the game

Now that we've set up our vehicle and obstacle, let's try running the game and looking through the camera.

1. Select the **Camera** in the hierarchy, then **press F** to focus on it
2. Press the **Play button** to run your Game, then press Play again to **stop** it

- **New Concept:** Game View vs Scene View
- **New Technique:** Stop/Play (Cmd/Ctrl + P)

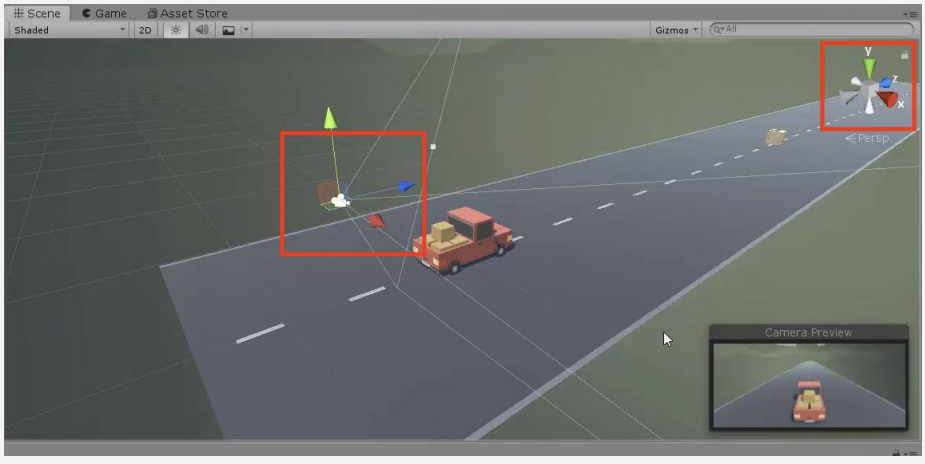


Step 6: Move the camera behind the vehicle

In order for the player to properly view our game, we should position and angle the camera in a good spot behind the vehicle

1. Use the **Move** and **Rotate tools** to move the camera behind the vehicle looking down on it
2. **Hold Ctrl/Cmd** to move the camera by whole units

- **New Technique:** Snapping (Cmd/Ctrl + Drag)
- **New Concept:** Rotation on the XYZ Axes

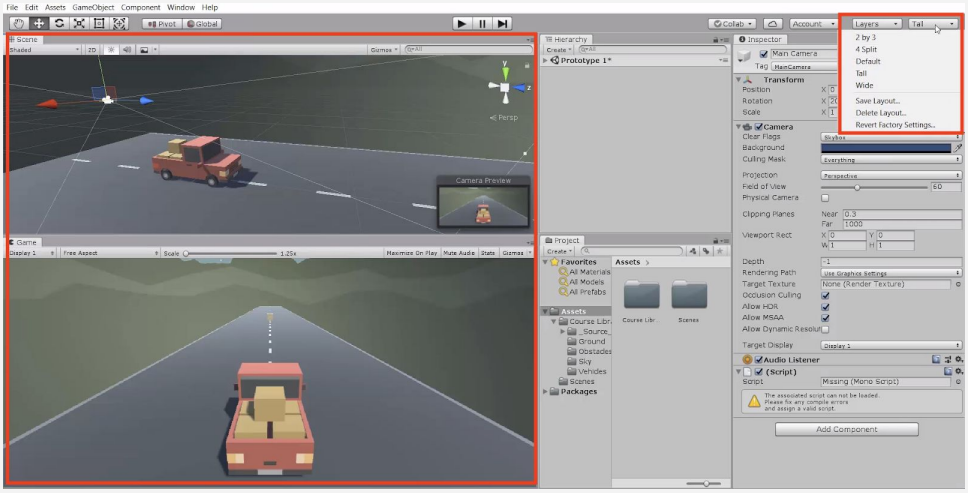


Step 7: Customize the interface layout

Last but not least, we need to customize the Unity Editor layout so that it's perfect for editing our project.

1. In the top-right corner, change the layout from "Default" to "**Tall**",
2. Move **Game view** beneath Scene view
3. In the **Project** window, click on the little drop-down menu in the top-right and choose "**One-column layout**"
4. In the layout Dropdown, **save a new Layout** and call it "My Layout"

- **New Concept:** Layouts



Lesson Recap

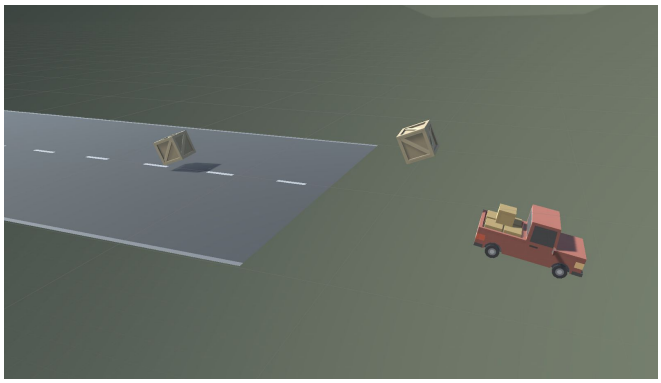
New Functionality	• Project set up with assets imported • Vehicle positioned at the start of the road • Obstacle positioned in front of the vehicle • Camera positioned behind vehicle
New Concepts and Skills	• Create a new project • Import assets • Add objects to the scene • Game vs Scene view • Project, Hierarchy, Inspector windows • Navigate 3D space • Move and Rotate tools • Customize the layout
Next Lesson	• We'll really make this interactive by writing our first line of code in C# to make the vehicle move and have it collide with other objects in the scene

unity

1.2 Pedal to the Metal

Steps:

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:

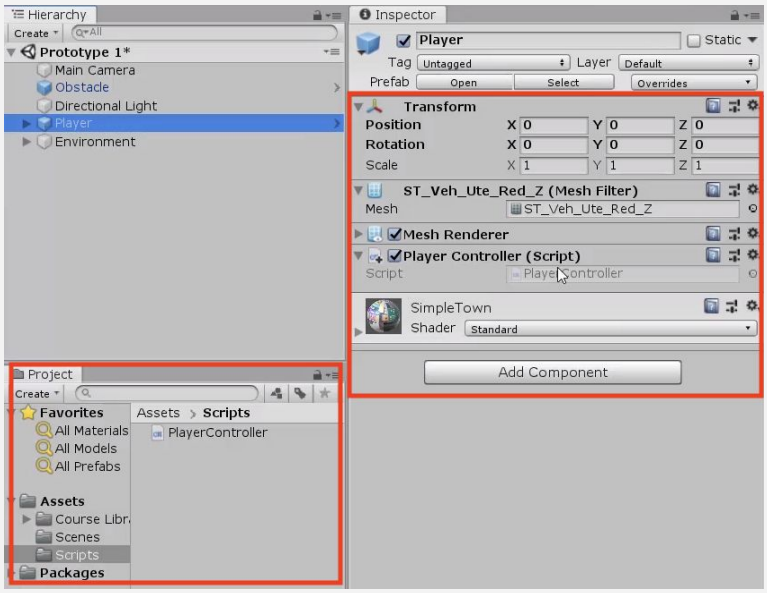
- Create C# scripts and apply them to objects
- Use Visual Studio and a few of its basic features
- Write comments to make your code more readable
- Utilize fundamental C# methods and classes like transform.Translate and Vector3
- Add Rigidbody and Collider components to allow objects to collide realistically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

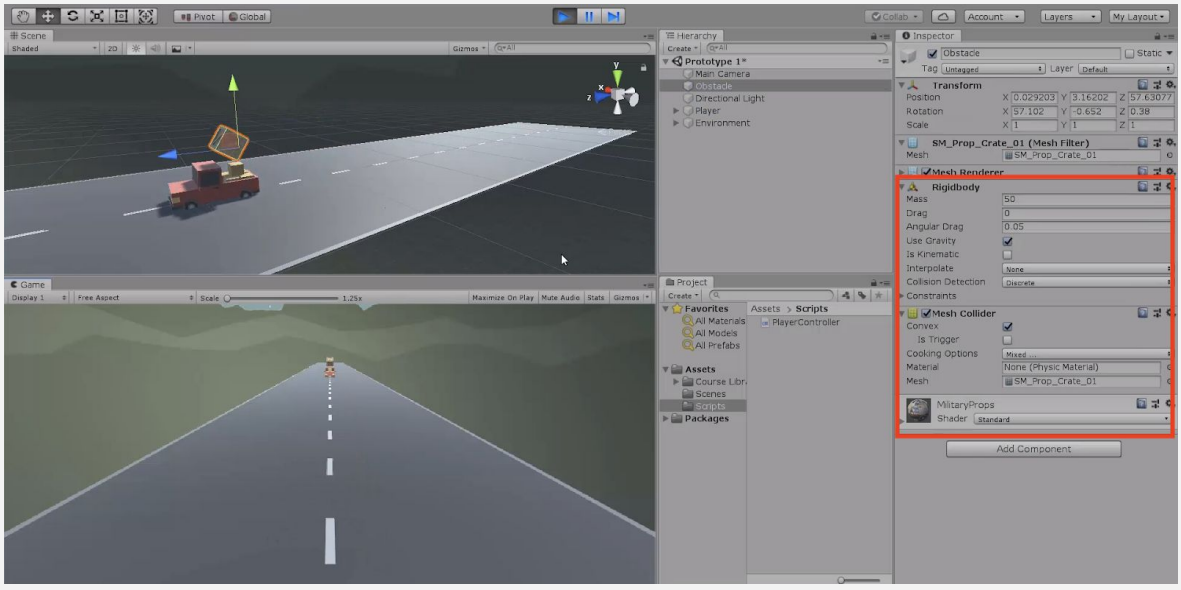
```
void Update()
{
  // Move the vehicle forward
  transform.Translate(Vector3.forward * Time.deltaTime * 20);
}
```

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

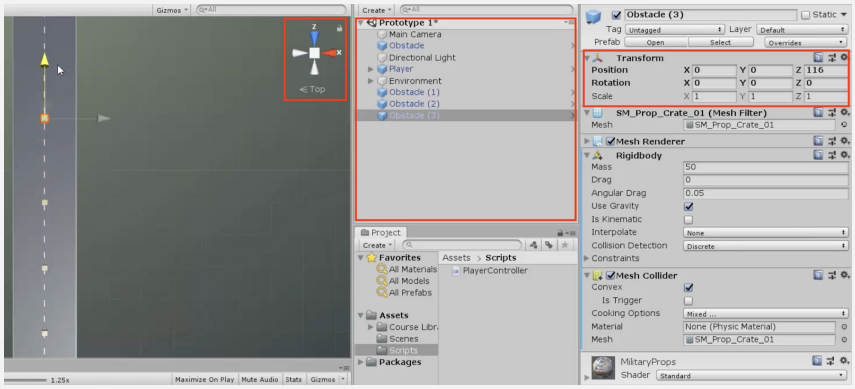


Step 7: Duplicate and position the obstacles

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.

1.3 High Speed Chase

Steps:

Example of project by end of lesson

Length: 50 minutes

Overview: Keep your eyes on the road! In this lesson you will code a new C# script for your camera, which will allow it to follow the vehicle down the road and give the player a proper view of the scene. In order to do this, you'll have to use a very important concept in programming: variables.

Project Outcome: The camera will follow the vehicle down the road through the scene, allowing the player to see where it's going.

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

- Declare variables properly and understand that variables can be different data types (float, Vector3, GameObject)
- Initialize/assign variables through code or through the inspector to set them with appropriate values
- Use appropriate access modifiers (public/private) for your variables in order to make them easier to change in the inspector

Step 1: Add a speed variable for your vehicle

We need an easier way to change the vehicle's speed and allow it to be accessed from the inspector. In order to do so what we need is something called a variable.

1. In PlayerController.cs, add ***public float speed = 5.0f;*** at the top of the **class**
2. Replace the **speed value** in the Translate method with the **speed variable**, then test
3. **Save** the script, then edit the speed value in the **inspector** to get the speed you want

- **New Concept:** Floats and Integers
- **New Concept:** Assigning Variables
- **New Concept:** Access Modifiers

```
public float speed = 20;

void Update()
{
  transform.Translate(Vector3.forward * Time.deltaTime * 20 speed);
}
```

Step 2: Create a new script for the camera

The camera is currently stuck in one position. If we want it to follow the player, we have to make a new script for the camera.

1. Create a new **C# script** called FollowPlayer and attach it to the **camera**
2. Add ***public GameObject player;*** to the top of the script
3. Select the **Main Camera, then**, **drag** the player object onto the **empty player variable** in the Inspector
4. In ***Update()***, assign the camera's position to the player's position, then test

- **Warning:** Remember to capitalize your script name correctly and rename it as soon as the script is created!
- **Warning:** It's really easy to forget to assign the player variable in the inspector
- **Don't worry:** The camera will be under the car... weird! We will fix that soon

```
public GameObject player;

void Update()
{
  transform.position = player.transform.position;
}
```

Step 3: Add an offset to the camera position

We need to move the camera's position above the vehicle so that the player can have a decent view of the game.

1. In the line in the Update method add ***+ new Vector3(0, 5, -7)***, then test

- **New Concept:** Vector3 in place of coordinates
- **Tip:** You need "new Vector3()" because 3 numbers in a row could mean anything
- **New Concept:** FixedUpdate
- **Warning:** Remember to update your comments and maintain their accuracy!

```
public GameObject player;

void Update()
{
  transform.position = player.transform.position + new Vector3(0, 5, -7);
}
```

Step 4: Make the offset into a Vector3 variable

We've fixed the camera's position, but we may want to change it later! We need an easier way to access the offset.

1. At the top of **FollowPlayer.cs**, declare ***private Vector3 offset;***
2. Copy the ***new Vector3()*** code and **assign** it to that variable
3. **Replace** the original code with the ***offset*** variable
4. **Test** and **save**

- **Don't worry:** Pay no mind to the read only warning
- **Tip:** Whenever possible, make variables! You never want hard values in the middle of your code

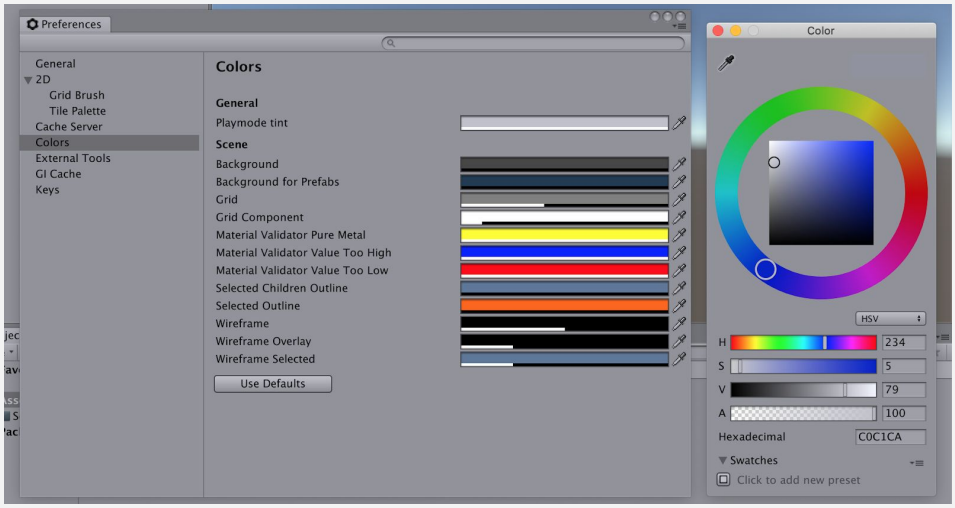
```
public GameObject player;
private Vector3 offset = new Vector3(0, 5, -7);

void Update()
{
  transform.position = player.transform.position + new Vector3(0, 5, -7)
offset;
}
```

Step 5: Edit the playmode tint color

If we're going to be creating and editing variables, we need to make sure we don't accidentally try to make changes when in "Play mode"

1. From the top menu, *go to Edit > Preferences* (**Windows**) or *Unity > Preferences* (**Mac**)
2. In the left menu, choose **Colors**, then edit the "**Playmode tint**" color to have a *slight* color
3. **Play** your project to test it, then close your preferences

- **Tip:** Try editing a variable in play mode, then stopping - it will revert
- **Warning:** Don't go crazy with the colors or it will be distracting



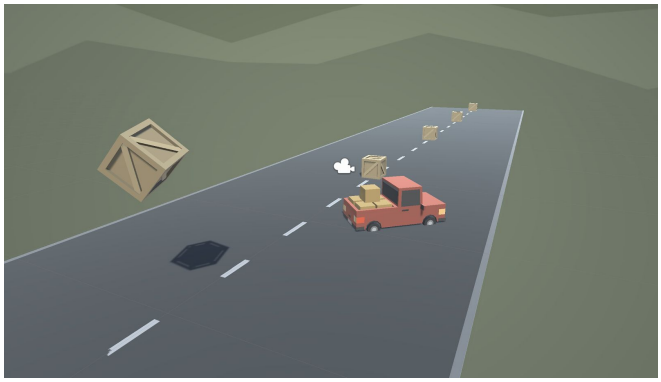
Lesson Recap

New Functionality	• Camera follows the vehicle down the road at a set offset distance
New Concepts and Skills	• Variables • Data types • Access Modifiers • Declare and initialize variables
Next Lesson	• In the next lesson, we'll add our last lines of code to take control of our car and be able to drive it around the scene.

1.4 Step into the Driver's Seat

Steps:

Example of project by end of lesson



Length: 50 minutes

Overview: In this lesson, we need to hit the road and gain control of the vehicle. In order to do so, we need to detect when the player is pressing the arrow keys, then accelerate and turn the vehicle based on that input. Using new methods, Vectors, and variables, you will allow the vehicle to move forwards or backwards and turn left to right.

Project Outcome: When the player presses the up/down arrows, the vehicle will move forward and backward. When the player presses the left/right arrows, the vehicle will turn.

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

- Gain user input with Input.GetAxis, allowing the player to move in different ways
- Use the Rotate function to rotate an object around an axis
- Clean and organize your hierarchy with Empty objects

Step 1: Allow the vehicle to move left/right

Until now, the vehicle has only been able to move straight forward along the road. We need it to be able to move left and right to avoid the obstacles.

1. At the top of PlayerController.cs, add a ***public float turnSpeed;*** variable
2. In ***FixedUpdate()***, add ***transform.Translate(Vector3.right * Time.deltaTime * turnSpeed);***
3. Run your game and use the ***turnSpeed*** **variable slider** to move the vehicle left and right

- **New Function:** Vector3.right

```
public float turnSpeed;

void Update()
{
  transform.Translate(Vector3.forward * Time.deltaTime * speed);
  transform.Translate(Vector3.right * Time.deltaTime * turnSpeed);
}
```

Step 2: Base left/right movement on input

Currently, we can only control the vehicle's left and right movement in the inspector. We need to grant some power to the player and allow them to control that movement for themselves.

1. In **PlayerController.cs**, add a new ***public float horizontalInput*** variable
2. In ***FixedUpdate***, assign ***horizontalInput = Input.GetAxis("Horizontal");***, then test to see it in inspector
3. Add the ***horizontalInput*** variable to your left/right **Translate method** to gain control of the vehicle
4. In the inspector, edit the ***turnSpeed*** and ***speed*** variables to tweak the feel

- **New:** Input.GetAxis
- **Tip:** Edit > Project Settings > Input and expand the Horizontal Axis to show everything about it
- **Warning:** Spelling is important in string parameters. Make sure you spell and capitalize "Horizontal" correctly!

```
public float horizontalInput;

void Update()
{
  horizontalInput = Input.GetAxis("Horizontal");

  transform.Translate(Vector3.forward * Time.deltaTime * speed);
  transform.Translate(Vector3.right * Time.deltaTime * turnSpeed * horizontalInput);
}
```

Step 3: Take control of the vehicle speed

We've allowed the player to control the steering wheel, but we also want them to control the gas pedal and brake.

1. Declare a new public ***forwardInput*** variable
2. In ***FixedUpdate***, assign ***forwardInput = Input.GetAxis("Vertical");***
3. Add the ***forwardInput*** variable to the **forward Translate method**, then test

- **Tip:** It can go backwards, too!
- **Warning:** This is slightly confusing with forwardInput and vertical axis

```
public float horizontalInput;
public float forwardInput;

void Update()
{
  horizontalInput = Input.GetAxis("Horizontal");
  forwardInput = Input.GetAxis("Vertical");

  transform.Translate(Vector3.forward * Time.deltaTime * speed * forwardInput);
  transform.Translate(Vector3.right * Time.deltaTime * turnSpeed * horizontalInput);
}
```

Step 4: Make vehicle rotate instead of slide

There's something weird about the vehicle's movement… it's slides left to right instead of turning. Let's allow the vehicle to turn like a real car!

1. In ***FixedUpdate***, call ***transform.Rotate(Vector3.up, horizontalInput)***, then test
2. **Delete** the line of code that **translates Right**, then test
3. Add **** turnSpeed * Time.deltaTime***, then test

- **New:** transform.Rotate
- **Tip:** You can always trust the official Unity scripting API documentation

```
void Update()
{
  horizontalInput = Input.GetAxis("Horizontal");
  forwardInput = Input.GetAxis("Vertical");

  transform.Translate(Vector3.forward * Time.deltaTime * speed * forwardInput);
  transform.Rotate(Vector3.up, turnSpeed * horizontalInput * Time.deltaTime);
  transform.Translate(Vector3.right * Time.deltaTime * turnSpeed * horizontalInput);
}
```

Step 5: Clean your code and hierarchy

We added lots of new stuff in this lesson. Before moving on and to be more professional, we need to clean our scripts and hierarchy to make them more organized.

1. In the hierarchy, *Right-click > Create Empty* and rename it "Obstacles", then **drag** all the obstacles into it
2. **Initialize** variables with values in **PlayerController**, then make all variables **private** (except for the **player** variables)
3. Use **//** to add **comments** to each section of code

- **New:** Empty Object
- **Tip:** You don't actually need to type "private", it defaults to that
- **Tip:** Comments are important, especially for your future self

```
public private float speed = 20.0f;
public private float turnSpeed = 45.0f;
public private float horizontalInput;
public private float forwardInput;

void Update() {
  horizontalInput = Input.GetAxis("Horizontal");
  forwardInput = Input.GetAxis("Vertical");
  // Moves the car forward based on vertical input
  transform.Translate(Vector3.forward * Time.deltaTime * speed * forwardInput);
  // Rotates the car based on horizontal input
  transform.Rotate(Vector3.up, turnSpeed * horizontalInput * Time.deltaTime);
}
```

Lesson Recap

New Functionality	• When the player presses the up/down arrows, the vehicle will move forward and backward • When the player presses the left/right arrows, the vehicle turns
New Concepts and Skills	• Empty objects • Get user input • Translate vs Rotate
Next Lesson	• We made our first project! We learned alot about how unity works, we wrote our first lines of code, and we made a driving game where our player has full control over this vehicle!