2D Game Design Workshop - Part 1

Unity Fundamentals: Movement, Physics, and Collisions

Lucas P. Cordova, Ph.D.

In this workshop, you'll practice the fundamentals of 2D game development in Unity. By the end of Part 1, you'll have created a simple car game with movement controls.

This workshop is divided into three parts: - Part 1: Unity Fundamentals (this part) - Part 2: Enhancing Your Game with Physics and Collisions - Part 3: Polishing and Expanding Your Game

Each part builds upon the previous one, so be sure to complete them in order.

Table of contents

1 Workshop Overview

In this three-part series, you'll learn the fundamentals of 2D game development in Unity. By the end of Part 1, you'll have created a simple car game with movement controls, physics interactions, and a dynamic camera system.

1.1 Learning Objectives

By completing Part 1, you will:

- Set up a Unity 2D project with proper layout
- Understand sprites, transforms, and their properties
- Create and attach MonoBehaviour scripts
- Implement frame-rate independent movement using `Time.deltaTime`

2 Step 1: Create Your Unity 2D Project

2.1 Creating Your Unity 2D Project

1. **Open Unity Hub** and click the "New Project" button
2. **Select the 2D (Core) template** from the available options
3. **Name your project** "2DCarGame_Part1"
4. **Choose a save location** that you can easily find later
5. **Set Unity Version** to 6.0 or your latest LTS version
6. **Click "Create Project"** and wait for Unity to initialize

3 Step 2: Configure Your Workspace

3.1 Optimizing for 2D Development

1. **Select the Tall Layout**:
   - Go to `Window → Layouts → Tall`
   - This gives you more vertical space for the Scene view
2. **Rearrange the Game View**:
   - Click and drag the "Game" tab
   - Drop it at the bottom of the Unity interface
   - This allows you to see both Scene and Game views simultaneously
3. **Adjust the Scene View for 2D**:
   - Click the "2D" button in the Scene view toolbar
   - Set the camera to orthographic view if not already

> Pro Tip
>
> Save your custom layout by going to `Window → Layouts → Save Layout`. Name it "2D Development" for easy access in future projects.

4 Step 3: Create Your Car Sprite

4.1 Adding the Capsule Sprite

1. **In the Hierarchy panel**, right-click in empty space
2. Navigate to `2D Object → Sprites → Capsule`
3. A capsule sprite will appear in your scene
4. **Rename the GameObject**:
   - Select the Capsule in the Hierarchy
   - Press F2 (or right-click → Rename)
   - Type "Car" and press Enter

5 Step 4: Position and Customize Your Car

5.1 Positioning Your Car

1. Select the Car GameObject in the Hierarchy
2. In the Inspector, reset the Transform:
    - Click the three dots menu on the Transform component
    - Select "Reset"
3. Your car should now be at position (0, 0, 0)

> **Visual Customization**
>
> You can change the car's color by:
>
> 1. Selecting the Car in the Hierarchy
> 2. In the Inspector, find the Sprite Renderer component
> 3. Click the Color property and choose your preferred color

6 Step 5: Understanding Transform Properties

6.1 Transform Component Basics

The Transform component is fundamental to all GameObjects in Unity. Let's explore its three main properties:

6.2 Position

- Controls the GameObject's location in 3D space
- Uses X, Y, and Z coordinates
- In 2D games, we primarily use X (horizontal) and Y (vertical)
- Z is typically used for layering (what appears in front/behind)

6.3 Rotation

- Defines the GameObject's orientation
- In 2D, we mainly use Z rotation (rotating around the Z-axis)
- Measured in degrees (0-360)
- Positive values rotate counter-clockwise

6.4 Scale

- Determines the size of the GameObject
- Default is (1, 1, 1) for original size
- (2, 2, 1) would double the width and height
- Keep Z scale at 1 for 2D objects

7 Step 6: Practice with Transform

7.1 Hands-On Transform Practice

Try these experiments with your Car selected:

1. **Position**: Change X to 3, notice the car moves right
2. **Rotation**: Change Z to 45, see the car rotate
3. **Scale**: Change X and Y to 2, observe the car doubles in size
4. **Reset** the Transform when done experimenting

8 Step 7: Enable the Input System

8.1 Switching to Modern Input

> **i** Why the New Input System?
>
> The new Input System provides better performance, flexibility, and support for multiple input devices compared to the legacy system. It is recommended for all new projects. While both are supported, we will use the new Input System in this workshop and in the class. Setting the project to support both systems ensures compatibility with older assets.

1. **Open Project Settings**:
   - Edit → Project Settings
2. **Navigate to Player**:
   - In the left panel, select "Player"
3. **Under Configuration**:
   - Find "Active Input Handling"
   - Set it to "Input System Package (New)" or "Both"

4. **Unity will prompt to restart** - click "Yes"

9 Step 8: Create the Movement Script

9.1 Creating a MonoBehaviour Script

1. **In the Project panel**, navigate to the Assets folder
2. **Right-click** in empty space within Assets
3. Select `Create → MonoBehaviour Script`
4. **Name the script** "Cruise" (press Enter to confirm)
5. **Wait** for Unity to compile the new script

9.2 Attaching the Script to Your Car

1. **Select the Car** GameObject in the Hierarchy
2. **Drag the Cruise script** from the Project panel onto the Car in the Inspector
   - Alternatively: Click "Add Component" and search for "Cruise"
3. You should now see the Cruise component in the Inspector

10 Step 9: Basic Script Structure

10.1 Setting Up the Script

Double-click the Cruise script to open it in your code editor. Replace the default code with:

```
using UnityEngine;
using UnityEngine.InputSystem;

public class Cruise : MonoBehaviour
{
    // Movement variables will go here

    void Update()
    {
        // Movement code will go here
    }
}
```

New Input System

We're using Unity's new Input System (`UnityEngine.InputSystem`) which provides better performance and more flexibility than the legacy input system. It also allows us to easily support multiple input devices.

11 Step 10: Add Movement Variables

11.1 Adding SerializeField Variables

Update your Cruise script with these movement variables:

```
using UnityEngine;
using UnityEngine.InputSystem;

public class Cruise : MonoBehaviour
{
    [SerializeField] float moveSpeed = 1f;
    [SerializeField] float steerSpeed = 5f;

    void Update()
    {
        // Movement code coming next
    }
}
```

12 Step 11: Understanding SerializeField

Understanding [SerializeField]

`[SerializeField]` makes private variables visible in the Unity Inspector, allowing you to:

- Adjust values without recompiling code
- Test different settings quickly
- Keep variables private while still being editable
- See real-time value changes during play mode

Our Variables:

- `moveSpeed`: Controls how fast the car moves forward/backward (units per frame, will be adjusted with deltaTime)
- `steerSpeed`: Controls how fast the car rotates (degrees per frame, will be adjusted with deltaTime)

13 Step 12: Implement Basic Movement

13.1 Adding Keyboard Controls

Update your Cruise script with the movement code:

```
using UnityEngine;
using UnityEngine.InputSystem;

public class Cruise : MonoBehaviour
{
    [SerializeField] float moveSpeed = 1f;
    [SerializeField] float steerSpeed = 5f;

    void Update()
    {
        float move = 0f;
        float steer = 0f;

        // Forward movement (W or Up Arrow)
        if (Keyboard.current.wKey.isPressed || Keyboard.current.upArrowKey.isPressed)
        {
            move = 1f;
        }
        // Backward movement (S or Down Arrow)
        else if (Keyboard.current.sKey.isPressed || Keyboard.current.downArrowKey.isPressed)
        {
            move = -1f;
        }

        // Left steering (A or Left Arrow)
        if (Keyboard.current.aKey.isPressed || Keyboard.current.leftArrowKey.isPressed)
        {
            steer = 1f;
        }
```

```
        // Right steering (D or Right Arrow)
        else if (Keyboard.current.dKey.isPressed || Keyboard.current.rightArrowKey.isPressed)
        {
            steer = -1f;
        }

        // Apply movement and rotation (without deltaTime for now)
        transform.Translate(0, move * moveSpeed, 0);
        transform.Rotate(0, 0, steer * steerSpeed);
    }
}
```

14 Step 13: Understanding the Input Code

14.1 Key Components

Understanding the Input System Code:

- `Keyboard.current`: References the current keyboard device
- `.isPressed`: Returns true while the key is held down
- We support both WASD and arrow keys for accessibility

15 Step 14: Test Your Movement

15.1 Testing Your Controls

1. **Save your script** (Ctrl/Cmd + S)
2. **Return to Unity** and wait for compilation
3. **Press Play** button to enter Play Mode
4. **Test both control schemes**:
    - W or ↑: Move forward
    - S or ↓: Move backward
    - A or ←: Turn left
    - D or →: Turn right
5. **Note**: Both WASD and arrow keys work, so players can use their preferred control scheme

Movement Speed Issue

You'll notice the car moves VERY fast! This is because movement is happening every frame (30-120 times per second). We'll fix this in the next step with Time.deltaTime.

16 Step 15: Understanding Frame Rate Dependence

16.1 The Problem with Frame-Dependent Movement

Without `Time.deltaTime`:

- A computer running at 30 FPS moves the car slower
- A computer running at 120 FPS moves the car 4x faster
- Game becomes unplayable across different devices

17 Step 16: What is Time.deltaTime?

17.1 Making Movement Frame-Independent

`Time.deltaTime` represents the time in seconds since the last frame:

- At 60 FPS: ~0.0167 seconds per frame
- At 30 FPS: ~0.0333 seconds per frame
- Multiplying movement by deltaTime ensures consistent speed regardless of frame rate

18 Step 17: Implement Time.deltaTime

18.1 Update Your Movement Code

Update your Cruise script to include Time.deltaTime. This will need to be adjusted later when we change speed values:

```
using UnityEngine;
using UnityEngine.InputSystem;

public class Cruise : MonoBehaviour
{
    [SerializeField] float moveSpeed = 5f;
    [SerializeField] float steerSpeed = 100f;
```

```
8
9       void Update()
10      {
11          float move = 0f;
12          float steer = 0f;
13
14          if (Keyboard.current.wKey.isPressed || Keyboard.current.upArrowKey.isPressed)
15          {
16              move = 1f;
17          }
18          else if (Keyboard.current.sKey.isPressed || Keyboard.current.downArrowKey.isPressed)
19          {
20              move = -1f;
21          }
22
23          if (Keyboard.current.aKey.isPressed || Keyboard.current.leftArrowKey.isPressed)
24          {
25              steer = 1f;
26          }
27          else if (Keyboard.current.dKey.isPressed || Keyboard.current.rightArrowKey.isPressed
28          {
29              steer = -1f;
30          }
31
32          // Calculate frame-independent movement
33          float moveAmount = move * moveSpeed * Time.deltaTime;
34          float steerAmount = steer * steerSpeed * Time.deltaTime;
35
36          // Apply movement and rotation
37          transform.Translate(0, moveAmount, 0);
38          transform.Rotate(0, 0, steerAmount);
39      }
40  }
```

19 Step 18: Adjust Speed Values

19.1 Setting Appropriate Speeds

With `Time.deltaTime`, you'll need to increase your speed values:

1. Select your Car in the Hierarchy

2. In the Inspector, find the Cruise component
3. Set new values:
    - **Move Speed**: 5 (units per second)
    - **Steer Speed**: 100 (degrees per second)

> **Testing Frame Independence**
>
> To verify frame independence is working:
>
> 1. In Play Mode, open the Stats window (click "Stats" in Game view)
> 2. Note your FPS
> 3. Go to Edit → Project Settings → Quality
> 4. Change VSync Count between "Don't Sync" and "Every V Blank"
> 5. Your car should move at the same speed regardless of FPS changes