Junior Programmer — Condensed Tutorial

Player Control (Mission 1) + Basic Gameplay (Mission 2)

This tutorial follows the **same projects, same assets, same ideas** as the Unity Learn pathway but cuts straight to what matters. Prototype 1 = driving sim. Prototype 2 = top-down food launcher. All the filler and repetition is removed — just the core concepts with code.

Phase	Covers	~Time
1. Prototype 1 Setup	Scene, 3D objects, camera positioning	10 min
2. First Script — Movement	C# basics, Translate, Vector3, speed variable, Rigidbody	10 min
3. Camera Follow	FollowPlayer script, offset, LateUpdate, public GameObject ref	5 min
4. Player Input	Input.GetAxis, Rotate vs Translate, Input Manager config	10 min
5. Prototype 2 Setup	New project, top-down camera, horizontal movement, bounds	10 min
6. Prefabs & Projectiles	Prefabs, Instantiate, GetKeyDown, Destroy out of bounds	10 min
7. Spawn Manager	Arrays, Random.Range, SpawnManager, InvokeRepeating	10 min
8. Collisions	Colliders, Triggers, OnTriggerEnter, Tags, Destroy on hit	10 min

Prerequisites: Unity 6.x installed via Unity Hub. That's it.

Phase 1 — Prototype 1: Scene Setup

Same as Lesson 1.1 from the course. You're setting up a vehicle on a road with obstacles.

1. Create Project

Unity Hub > New Project > **3D (Built-in RP)** > name it **Prototype1**. Create a folder structure: Assets/Scripts, Assets/Materials.

2. Import Starter Files

Download the **Prototype 1 Starter Files** from the Unity Learn page. Double-click the .unitypackage to import. Open the prototype scene from the imported assets.

3. Set Up the Scene

• Drag a **vehicle** from the imported assets into the Hierarchy.
• Drag an **obstacle** into the scene. Use the Move tool (W) to position it on the road ahead of the vehicle.
• Hold **Ctrl/Cmd** while moving to snap to whole units.
• **Duplicate** the obstacle (Ctrl+D) a few times and spread them along the road.

4. Position the Camera

Select **Main Camera**. Use Move (W) and Rotate (E) to position it **behind and above the vehicle**, looking down the road. Approximate position: (0, 5, -7) relative to the vehicle.

5. Customize Layout

Upper-right corner of the Editor > change layout from Default to **Tall**. Arrange Game view under Scene view so you can see both.

CONCEPT: The **Hierarchy** lists all objects in the scene. The **Inspector** shows the selected object's components. The **Scene view** is your editing workspace. The **Game view** shows what the camera sees at runtime.

Phase 2 — First Script: Vehicle Movement

Covers Lessons 1.2 (Pedal to the Metal). Your first C# script.

1. Create the Script

In the Scripts folder: Right-click > Create > MonoBehaviour Script > name it **PlayerController**. Drag it onto the vehicle in the Hierarchy (or select the vehicle > Add Component > search PlayerController).

2. Basic Forward Movement

Open the script (double-click). Replace contents with:

```
using UnityEngine;
public class PlayerController : MonoBehaviour
{
public float speed = 5.0f;
void Update()
{
// Move vehicle forward every frame
transform.Translate(Vector3.forward * Time.deltaTime * speed);
}
}
```

CONCEPT: transform.Translate(Vector3.forward) moves the object along the Z axis (forward in Unity). Without Time.deltaTime it moves per-frame (fast on good PCs, slow on bad ones). Multiplying by **Time.deltaTime** converts it to per-second — frame-rate independent.

CONCEPT: public float speed — the **public** keyword exposes this variable in the Inspector panel. You can change it at runtime without editing code. **private** hides it from the Inspector.

3. Add Rigidbody to Objects

Select the vehicle > Add Component > **Rigidbody**. Do the same for each obstacle. This gives objects physics (mass, gravity, collision detection). Without Rigidbody, objects pass through each other.

TEST: Press Play. The vehicle should drive forward down the road automatically. Try changing the **speed** value in the Inspector while playing to find a good speed. (Values reset when you stop — that's normal.)

NOTE: Play Mode Tint: Go to Edit > Preferences > Colors > set Playmode Tint to a visible color. This reminds you when you're in play mode — changes made in play mode are lost!

Phase 3 — Camera Follow Script

Covers Lesson 1.3 (High Speed Chase). The camera tracks the vehicle.

1. Create FollowPlayer Script

Create a new script **FollowPlayer**. Attach it to the **Main Camera**.

```
using UnityEngine;
public class FollowPlayer : MonoBehaviour
{
// Drag the vehicle here in Inspector
public GameObject player;
private Vector3 offset;
void Start()
{
// Save initial distance between camera and player
offset = transform.position - player.transform.position;
}
void LateUpdate()
{
// Follow player, keeping the same offset
transform.position = player.transform.position + offset;
}
}
```

2. Assign the Reference

Select **Main Camera** in the Hierarchy. In the Inspector, find the FollowPlayer component. The **Player** field is empty — drag the vehicle from the Hierarchy into that slot.

CONCEPT: public GameObject player — creates an empty slot in the Inspector. You fill it by dragging objects from the Hierarchy. This is how scripts reference other objects. Without this, scripts can't talk to each other.

CONCEPT: LateUpdate() runs after ALL Update() methods have finished. The player moves in Update(), then the camera catches up in LateUpdate(). If both used Update(), the camera might move before the player, causing visible **jitter**.

CONCEPT: offset = camera.position - player.position — calculated once in Start(). This means you can position the camera wherever you like in the editor and it'll maintain that exact distance. No hardcoded magic numbers.

TEST: Play the game. The camera should smoothly follow the vehicle down the road. No jitter.

Phase 4 — Player Input & Steering

Covers Lesson 1.4 (Step into the Driver's Seat). The player controls the vehicle.

1. Configure Input Manager (Unity 6+ only)

Edit > Project Settings > Player > Configuration > Active Input Handling > Both. Unity restarts. This lets Input.GetAxis() work alongside the newer Input System.

NOTE: Skip this step on Unity 2021/2022. Only needed for Unity 6+. If you skip it on Unity 6, input won't work at all.

2. Update PlayerController with Input

Replace your PlayerController script with:

```
using UnityEngine;
public class PlayerController : MonoBehaviour
{
public float speed = 5.0f;
public float turnSpeed = 25.0f;
private float horizontalInput;
private float verticalInput;
void Update()
{
// Read input (-1 to 1 range)
horizontalInput = Input.GetAxis("Horizontal");
verticalInput = Input.GetAxis("Vertical");
// Move forward/backward based on up/down input
transform.Translate(Vector3.forward * Time.deltaTime
* speed * verticalInput);
// Rotate (not slide) based on left/right input
transform.Rotate(Vector3.up,
turnSpeed * horizontalInput * Time.deltaTime);
}
}
```

CONCEPT: Input.GetAxis("Horizontal") returns a float from -1 (left) to +1 (right), smoothly ramping. "Vertical" does the same for up/down. These map to arrow keys AND WASD by default. Check the mappings: Edit > Project Settings > Input Manager > Axes.

CONCEPT: Rotate vs Translate for steering: Using Translate(Vector3.right) makes the car **slide** sideways. Using **Rotate(Vector3.up)** makes it **turn** like a real car — the forward direction changes. That's why we replaced the Translate Right call with Rotate.

3. Clean Up

• Make variables **private** (except speed and turnSpeed you want to tweak).
• In the Hierarchy: right-click > Create Empty > name it **Obstacles** > drag all obstacles into it. This organizes the hierarchy with parent objects as folders.

TEST: Play. Arrow keys / WASD should steer and accelerate the vehicle. It should turn like a car, not slide.

MISSION 1 COMPLETE. You've covered: scripts, Translate, Rotate, variables, Time.deltaTime, Vector3, Input.GetAxis, camera follow with LateUpdate, Rigidbody, Inspector references, and hierarchy organization.

Phase 5 — Prototype 2: New Project & Player Movement

Covers Lesson 2.1 (Player Positioning). New project — top-down food-throwing game. Player moves left/right, can't leave the screen.

1. Create Project & Import

Unity Hub > New 3D Project > **Prototype2**. Download and import the Prototype 2 starter files. Open the scene. If on Unity 6+, set Active Input Handling to **Both** again.

2. Set Up Objects

• Drag 1 **human character** into the scene. Rename it **Player**. Position at bottom-center.
• Drag 3 different **animals** into the scene and position them at the top.
• Drag 1 **food object** into the scene. Scale it up so it's visible from above.

3. PlayerController — Horizontal Only + Bounds

Create Scripts folder, create PlayerController, attach to Player:

```
using UnityEngine;
public class PlayerController : MonoBehaviour
{
public float speed = 15.0f;
public float xRange = 10.0f;
private float horizontalInput;
void Update()
{
horizontalInput = Input.GetAxis("Horizontal");
// Move left/right only
transform.Translate(Vector3.right * horizontalInput
* speed * Time.deltaTime);
// Clamp: keep player in bounds
if (transform.position.x < -xRange)
{
transform.position = new Vector3(-xRange,
transform.position.y, transform.position.z);
}
if (transform.position.x > xRange)
{
transform.position = new Vector3(xRange,
transform.position.y, transform.position.z);
}
}
}
```

CONCEPT: Bounding with if-statements: Check if position exceeds a limit. If so, snap it back. The pattern is: **if (pos < -limit) { pos = -limit; }**. You're manually clamping. The xRange variable lets you tweak the boundary width in the Inspector.

TEST: Play. Left/right arrows move the player. They should stop at the edges, not go off-screen.

Phase 6 — Prefabs & Projectiles

Covers Lesson 2.2 (Food Flight). Launching food with spacebar.

1. Make the Food Move Forward

Create a new script **MoveForward**. Attach to the food object.

```
using UnityEngine;
public class MoveForward : MonoBehaviour
{
public float speed = 40.0f;
void Update()
{
transform.Translate(Vector3.forward * Time.deltaTime * speed);
}
}
```

2. Make the Food a Prefab

• Create a **Prefabs** folder in Assets.
• Drag the food object from the Hierarchy into the Prefabs folder. Choose **Original Prefab**.
• **Delete** the food from the Hierarchy. The template lives in the folder now.

CONCEPT: Prefabs = reusable templates. A prefab stores the GameObject + all its components and scripts. You can spawn (Instantiate) copies at runtime. Edit the prefab once → all instances update. Essential for anything you need multiples of: bullets, enemies, pickups, etc.

3. Launch on Spacebar

Add to your **PlayerController** script:

```
// New variable at top of class:
public GameObject projectilePrefab;
// Add inside Update(), after movement code:
if (Input.GetKeyDown(KeyCode.Space))
{
Instantiate(projectilePrefab,
transform.position,
projectilePrefab.transform.rotation);
}
```

In the Inspector: select Player > drag the food **prefab from the Prefabs folder** (not from Hierarchy) into the **Projectile Prefab** slot.

CONCEPT: Instantiate(prefab, position, rotation) — spawns a clone of the prefab at the given position/rotation. We use **transform.position** so the food appears at the player's location.

CONCEPT: GetKeyDown fires once on the frame the key is pressed. **GetKey** fires every frame while held. **GetKeyUp** fires once on release. GetKeyDown prevents rapid-fire spam.

4. Destroy Out of Bounds

Create **DestroyOutOfBounds**. Attach to the food **prefab** (double-click prefab to edit):

```
using UnityEngine;
public class DestroyOutOfBounds : MonoBehaviour
{
private float topBound = 30f;
private float bottomBound = -10f;
void Update()
{
if (transform.position.z > topBound)
{
Destroy(gameObject);
}
if (transform.position.z < bottomBound)
{
Destroy(gameObject);
}
}
}
```

NOTE: Every Instantiate() allocates memory. If you never Destroy(), the scene fills with invisible objects and performance tanks. Always clean up off-screen objects.

5. Make Animals into Prefabs Too

• Rotate all 3 animals by 180° on the Y axis (so they face downward toward the player).
• Select all 3 animals > Add Component > **MoveForward**. Set their speeds (5-10).
• Add **DestroyOutOfBounds** to each as well.
• Drag all 3 into the **Prefabs** folder. Delete them from the Hierarchy.

TEST: Play. Press Space — food should launch upward. Check the Hierarchy: food objects should appear then vanish when they go past the top. Try dragging animal prefabs into the scene during play mode to see them walk down.

Phase 7 — Spawn Manager & Randomization

Covers Lessons 2.3 (Random Animal Stampede) and 2.4 (Collision Decisions — spawn part).

1. Create the Spawn Manager

Create an **Empty GameObject** in the Hierarchy. Name it **SpawnManager**. Create a new script **SpawnManager** and attach it.

```
using UnityEngine;
public class SpawnManager : MonoBehaviour
{
// Drag all 3 animal prefabs here (set Size = 3)
public GameObject[] animalPrefabs;
private float spawnRangeX = 10f;
private float spawnPosZ = 20f;
private float startDelay = 2f;
private float spawnInterval = 1.5f;
void Start()
{
InvokeRepeating("SpawnRandomAnimal",
startDelay, spawnInterval);
}
void SpawnRandomAnimal()
{
// Pick random animal from array
int animalIndex = Random.Range(0, animalPrefabs.Length);
// Pick random X position
Vector3 spawnPos = new Vector3(
Random.Range(-spawnRangeX, spawnRangeX),
0,
spawnPosZ);
Instantiate(animalPrefabs[animalIndex],
spawnPos,
animalPrefabs[animalIndex].transform.rotation);
}
}
```

2. Wire the Array

Select SpawnManager in the Hierarchy. In Inspector, find **Animal Prefabs**. Set **Size = 3**. Drag each animal prefab into Element 0, 1, 2.

CONCEPT: GameObject[] animalPrefabs — an **array**. Holds multiple items of the same type. Access elements by index: animalPrefabs[0], [1], [2]. **Random.Range(0, array.Length)** picks a random valid index. Note: the max is **exclusive** for ints.

CONCEPT: InvokeRepeating("MethodName", delay, interval) — calls the named method repeatedly. First call after **delay** seconds, then every **interval** seconds. Much simpler than writing your own timer in Update(). The method name is passed as a **string**.

TEST: Play. After 2 seconds, random animals should start spawning at random X positions at the top and walking downward. A mix of all 3 types should appear.

Phase 8 — Collisions & Game Over

Covers Lesson 2.4 (Collision Decisions). Making things actually interact.

1. Add Colliders & Rigidbodies

Open each prefab (double-click in the Prefabs folder) and configure:

Prefab	Collider	Is Trigger?	Rigidbody	Use Gravity?
Food (projectile)	Box Collider (Add)	✗ No	Add	✗ Off
Each Animal	Box Collider (Add)	✓ Yes	Add	✗ Off

Also add a **Box Collider** to the **Player** in the scene (not a prefab).

CONCEPT: Trigger Collider = objects pass through each other but fire OnTriggerEnter events. **Regular Collider** = solid, physics-based blocking. For OnTriggerEnter to fire: at least one object needs a **Rigidbody**, and at least one collider must be a **Trigger**.

2. Tag Your Objects

• Player is already tagged **Player** (default Unity tag).
• For each animal prefab: open it > Tag dropdown > Add Tag > create **Animal** > select it.
• This lets collision code identify *what* hit *what*.

3. Create DetectCollisions Script

Create this script and attach it to **each animal prefab**:

```
using UnityEngine;
public class DetectCollisions : MonoBehaviour
{
void OnTriggerEnter(Collider other)
{
// Food hits animal: destroy both
Destroy(gameObject);
Destroy(other.gameObject);
}
}
```

CONCEPT: OnTriggerEnter(Collider other) is called automatically by Unity when another collider enters this trigger. The **other** parameter is the collider that entered. **Destroy(gameObject)** destroys the object this script is on. **Destroy(other.gameObject)** destroys the other object.

4. Detect Game Over

Update **DestroyOutOfBounds** to detect when an animal gets past the player:

```
// Updated DestroyOutOfBounds.cs
using UnityEngine;
public class DestroyOutOfBounds : MonoBehaviour
{
private float topBound = 30f;
private float bottomBound = -10f;
void Update()
{
if (transform.position.z > topBound)
{
Destroy(gameObject);
}
else if (transform.position.z < bottomBound)
{
// Is this an animal that got past?
if (gameObject.CompareTag("Animal"))
{
Debug.Log("Game Over! Animal escaped.");
}
Destroy(gameObject);
}
}
}
```

CONCEPT: CompareTag("TagName") checks an object's tag. More efficient than gameObject.tag == "TagName". Tags must be created first (Tags & Layers panel) before you can assign them to objects.

TEST: Play. Animals spawn and walk down. Press Space to throw food. When food hits an animal, **both disappear**. If an animal reaches the bottom, you see "Game Over" in the Console (Window > General > Console). Verify the Hierarchy: objects should be created and destroyed continuously.

MISSION 2 COMPLETE. You've now covered all core concepts from both missions: Prefabs, Instantiate, Arrays, Random.Range, InvokeRepeating, Colliders, Triggers, OnTriggerEnter, CompareTag, Destroy, and Tags.

Quick Reference — All Key Concepts

Concept	What / Why	Mission
transform.Translate()	Move object along a direction. Multiply by Time.deltaTime for frame-rate independence.	1
transform.Rotate()	Rotate object around axis. Use instead of Translate for turning (car/ship steering).	1
Vector3.forward/right/up	Direction shorthands: (0,0,1), (1,0,0), (0,1,0). Local to object's orientation with Translate.	1
Time.deltaTime	Seconds since last frame. Converts per-frame to per-second movement.	1
public / private	public = visible in Inspector & editable. private = hidden, internal only.	1
Input.GetAxis()	Returns -1 to 1 float. Smooth, analog-style. "Horizontal" / "Vertical".	1
Input.GetKeyDown()	Returns true on the single frame a key is first pressed. One-shot actions.	2
LateUpdate()	Runs after all Update() calls. Use for camera follow to prevent jitter.	1
Rigidbody	Gives object physics (mass, gravity, collision). Required for OnTriggerEnter.	1+2
Prefabs	Reusable object templates in project folder. Spawn copies with Instantiate().	2
Instantiate()	Spawns a clone of a prefab at a position/rotation. Core spawning mechanism.	2
Destroy()	Removes a GameObject. Always clean up off-screen objects.	2
GameObject[]	Array of GameObjects. Access by index [0], [1]... Random.Range for random picks.	2
Random.Range()	Random int (exclusive max) or float (inclusive max). Randomize spawns/positions.	2
InvokeRepeating()	Call a method on a timer. Params: method name (string), start delay, interval.	2
Collider + Is Trigger	Trigger = pass through but detect. Regular = solid physics blocking.	2
OnTriggerEnter()	Callback when another collider enters a trigger. Needs Rigidbody on at least one.	2
CompareTag()	Check object's tag efficiently. Tags created via Tags & Layers panel.	2

5 scripts total | **~75 minutes** | All Junior Programmer Mission 1 & 2 core concepts covered.