Flappy Bird Simple Game Tutorial

Step 2: Physics & Programming (Unity 6)

Goal: Make the bird affected by gravity and flap upward when the player presses the **Spacebar**.

1. Add Physics Components

- **Rigidbody 2D:**
  Select the bird → *Add Component* → **Rigidbody 2D**.
  - This makes the bird fall under gravity.
- **Circle Collider 2D:**
  Add **Circle Collider 2D** for collision detection.
  - Adjust the collider's offset and slightly shrink it to make gameplay feel fairer when passing through pipes.

2. Create a Bird Script

- Add a new **C# Script** called `BirdScript` and attach it to the bird.
- Double-click it to open in **Visual Studio** (or Rider).

3. Understand Script Structure

- `Start()` → Runs **once** when the script begins.
- `Update()` → Runs **every frame** while the game is active. These are the main areas where gameplay logic will go.

4. Reference the Rigidbody

In your script, create a reference so the code can control physics:

```
public Rigidbody2D myRigidbody;
```

- Save the script and return to Unity.
- Drag the bird's **Rigidbody 2D** into the script's new slot in the **Inspector**.

5. Make the Bird Flap

Inside `Update()`, detect when the **Spacebar** is pressed and apply upward velocity:

```
if (Input.GetKeyDown(KeyCode.Space))
{
    myRigidbody.velocity = Vector2.up * flapStrength;
}
```

6. Add Adjustable Flap Strength

At the top of the script, define a variable for tuning:

```
public float flapStrength = 10f;
```

Now you can adjust **flapStrength** directly in the **Inspector** to make the bird flap higher or lower without changing code.

7. Test and Fine-Tune

- Press **Play** to test.
- Adjust **flapStrength** and **Gravity Scale** on the Rigidbody until the bird movement feels right.
- Note: Changes made *while playing* won't save after you stop the game.

Key Takeaways

- **Rigidbody 2D** = enables gravity and movement physics.
- **Collider 2D** = enables collisions.
- **Script** = controls behavior using C# code.
- Use **public variables** to tweak gameplay values easily.
- Use **if statements** to perform actions only when certain conditions (like a key press) are met.

Step 3: Spawning Objects (Unity 6)

Goal: Make pipes automatically appear, move across the screen, and delete themselves — giving the illusion that the bird is flying forward.

1. Core Idea

In Flappy Bird, the **bird doesn't move** — the **pipes move** instead. We'll:

1. Create a pipe object.
2. Make it move left.
3. Continuously spawn new pipes.
4. Remove old pipes off-screen.

2. Build the Pipe Prefab

1. **Create Parent Object:**
    - Create an empty GameObject named **Pipe**.
    - Position it near the bird to match its size and spacing.
2. **Add Top and Bottom Pipes:**
    - Inside the parent, create **Top Pipe** and **Bottom Pipe** objects.
    - Add a **Sprite Renderer** (pipe image) and a **Box Collider 2D** to each.
    - Flip the bottom pipe by setting **Scale Y = –1**.
    - Move the top and bottom pipes up/down to form a gap.
3. **Parent Control:**
    - Moving, rotating, or scaling the **Pipe (parent)** will affect both pipes together.

3. Pipe Movement Script

Attach a new C# script (e.g. `PipeMove.cs`) to the **Pipe** parent.

```
public float moveSpeed = 5f;
public float deadZone = -45f;

void Update()
{
    transform.position += Vector3.left * moveSpeed * Time.deltaTime;

    if (transform.position.x < deadZone)
    {
        Debug.Log("Pipe Deleted");
```

```
        Destroy(gameObject);
    }
}
```

✅ `Time.deltaTime` ensures consistent speed across all frame rates.
✅ The pipe deletes itself once it moves past the left side of the screen.

4. Create the Pipe Prefab

1. Drag the **Pipe** object from the **Hierarchy** into the **Project** window. → This makes it a **Prefab** (a reusable blueprint).
2. Delete the original pipe from the scene to keep it clean.

5. Pipe Spawner Setup

1. Create an empty GameObject called **PipeSpawner** and position it **just off the right side** of the camera view.
2. Add a new script (e.g. `PipeSpawner.cs`):

```
public GameObject pipe;
public float spawnRate = 2f;
public float heightOffset = 10f;

private float timer = 0f;

void Start()
{
    SpawnPipe();
}

void Update()
{
    if (timer < spawnRate)
    {
        timer += Time.deltaTime;
    }
    else
    {
        SpawnPipe();
```

```
        timer = 0f;
    }
}

void SpawnPipe()
{
    float lowestPoint = transform.position.y - heightOffset;
    float highestPoint = transform.position.y + heightOffset;

    Instantiate(pipe,
        new Vector3(transform.position.x, Random.Range(lowestPoint,
highestPoint), 0),
        transform.rotation);
}
```

✅ Spawns pipes at random heights between two limits.
✅ Uses a **timer** and **`Time.deltaTime`** for frame-rate-independent intervals.
✅ Calls the same spawn function in `Start()` so the first pipe appears instantly.

6. Testing and Fine-Tuning

- In Unity, drag the **Pipe Prefab** into the `pipe` slot in the spawner's Inspector.
- Adjust:
    - **spawnRate** → how often new pipes appear.
    - **heightOffset** → how far pipes vary vertically.
    - **moveSpeed** → how fast pipes move.

7. Debugging

Use `Debug.Log("Pipe Deleted")` in the **Console** to confirm when pipes are destroyed. This helps verify that old pipes are being properly cleaned up.

Recap

- **Prefabs** are reusable GameObject templates.
- **Spawners** use `Instantiate()` to create new instances.
- **Timers** + `Time.deltaTime` make consistent intervals.
- **If / Else** statements control logic flow.
- Always **destroy** unused GameObjects to free memory.

🎮 Step 4: Logic and UI

Goal: Create a scoring system that updates when the bird successfully passes through pipes, and display that score using Unity's UI system.

1. Create the Score UI

1. **Add a Text Object**
   - In the **Hierarchy**, go to **UI → Text (Legacy)**.
   - Unity will automatically create a **Canvas** and **EventSystem**.
2. **Adjust the Canvas Settings**
   - Select the **Canvas** object.
   - In the **Canvas Scaler** component:
     - Set **UI Scale Mode → Scale With Screen Size**.
     - Set **Reference Resolution →** `1920 x 1080`.
3. **Customize the Text**
   - Rename the text object to **ScoreText**.
   - Set default text to **"0"**.
   - Increase **Font Size** (e.g. 100).
   - Adjust **Rect Transform** position to the top center of the screen.
   - Use **width/height**, not scale, to resize UI elements.

2. Create the Logic Manager

1. **Add a New Empty GameObject**
   - Name it **LogicManager** (or simply **Logic**).
2. **Add Script:** **`LogicScript.cs`**

```
using UnityEngine;
using UnityEngine.UI;  // Needed for UI components

public class LogicScript : MonoBehaviour
{
    public int playerScore;
    public Text scoreText;

    [ContextMenu("Add Score")] // allows manual testing
    public void AddScore(int scoreToAdd)
    {
        playerScore += scoreToAdd;
        scoreText.text = playerScore.ToString();
```

```
    }
}
```

3. **Setup References**
    - Drag the **ScoreText** object from the Hierarchy into the `Score Text` field of the **LogicManager** in the Inspector.

3. Add the Trigger Between Pipes

1. **Open the Pipe Prefab**
    - In the **Project** window, double-click the **Pipe** prefab.
2. **Add Middle Collider**
    - Create a new empty child object → name it **Middle**.
    - Add a **Box Collider 2D** component.
    - Resize the collider to fit the gap between pipes.
    - Tick **"Is Trigger"** (so it detects passing but doesn't collide).

4. Add Pipe Trigger Script

Create a new script named `PipeMiddleScript.cs` and attach it to the **Middle** object.

```
using UnityEngine;

public class PipeMiddleScript : MonoBehaviour
{
    public LogicScript logic;
    public int birdLayer;

    void Start()
    {
        logic =
GameObject.FindGameObjectWithTag("Logic").GetComponent<LogicScript>()
;
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.layer == birdLayer)
        {
```

```
            logic.AddScore(1);
        }
    }
}
```

5. Link Everything with Tags and Layers

1. **Tag the Logic Manager**
    - Select **LogicManager** → **Tag** → **Add Tag** → **+** → **Logic**.
    - Assign the **Logic** tag to the LogicManager GameObject.
2. **Set Bird Layer**
    - Select the **Bird** GameObject → **Layer** → **Add Layer** → **Bird**.
    - Assign the Bird GameObject to this layer.
    - Note the layer number (displayed beside the name).
3. **Configure PipeMiddleScript**
    - In the Inspector, set **Bird Layer** = the correct layer number.

6. Test the System

- Press **Play**.
- Each time the bird passes through the pipes, the score should **increase by 1**.
- Confirm that the **UI text updates live**.
- You can test manually in Play Mode using the gear icon → **"Add Score"** (from Context Menu).

7. Future-Proofing

✅ Using `AddScore(int scoreToAdd)` makes the function flexible — you can later add different point values for other goals or collectibles.

✅ Using tags + `FindGameObjectWithTag()` ensures that the spawner-created pipes can dynamically find the LogicManager during runtime.

💡 Recap

Concept	Description
UI Elements	GameObjects for on-screen text, buttons, etc.
Canvas Scaler	Keeps UI size consistent across resolutions.

Logic Manager	Central invisible GameObject that tracks score and rules.
Trigger Collider	Detects when objects overlap, without physical collision.
FindGameObjectWithTag()	Finds GameObjects dynamically by tag.
GetComponent<>	Accesses specific scripts or components attached to an object.
Public Functions	Can be accessed and run from other scripts.

💀 Step 5: Game Over System

Goal: Show a “Game Over” screen when the bird crashes into a pipe, and allow the player to restart the game.

1. Build the Game Over Screen

1. **Create the UI:**
    - Select your existing **Canvas** (from Step 4).
    - Right-click → **Create Empty** → name it **GameOverScreen**.
    - Inside **GameOverScreen**, add:
        - **Text (Legacy)** → rename to **GameOverText** → set text to `"Game Over"`.
        - **Button (Legacy)** → rename to **RestartButton**.
2. **Customize the UI:**
    - Adjust layout and size so the “Game Over” text appears centered at the top.
    - Resize and reposition the button below the text.
    - Change the button’s label (child text object) to `"Restart"` or `"Play Again"`.
    - Ensure **Canvas Scaler** is still set to:

```
UI Scale Mode → Scale With Screen Size
Reference Resolution → 1920 x 1080
```

3. **Disable by Default:**
    - In the Inspector, **uncheck** the GameOverScreen’s checkbox to hide it at the start.

2. Add Game Restart Logic

Open your existing `LogicScript.cs` (from Step 4) and add the following code **below your AddScore function**:

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement; // <-- Add this

public class LogicScript : MonoBehaviour
{
    public int playerScore;
    public Text scoreText;
    public GameObject gameOverScreen;
```

```
    public void AddScore(int scoreToAdd)
    {
        playerScore += scoreToAdd;
        scoreText.text = playerScore.ToString();
    }

    public void RestartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }

    public void GameOver()
    {
        gameOverScreen.SetActive(true);
    }
}
```

3. Connect the Button

1. Select the **RestartButton** in the Hierarchy.
2. In the **Button** component → find **On Click ()** event list.
3. Click **+** to add a new event.
4. Drag your **LogicManager** GameObject into the field.
5. From the dropdown, choose: **LogicScript** → **RestartGame()**

✅ Now, clicking the button will reload the current scene.

4. Trigger Game Over on Collision

Open your **Bird script** (from Step 2). We'll modify it to detect collisions and stop the game when the bird crashes into pipes or the ground.

```
using UnityEngine;

public class BirdScript : MonoBehaviour
{
    public Rigidbody2D myRigidbody;
    public float flapStrength;
```

```
    public LogicScript logic;
    public bool birdIsAlive = true;

    void Start()
    {
        logic =
GameObject.FindGameObjectWithTag("Logic").GetComponent<LogicScript>()
;
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space) && birdIsAlive)
        {
            myRigidbody.velocity = Vector2.up * flapStrength;
        }
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        logic.GameOver();
        birdIsAlive = false;
    }
}
```

✅ **`OnCollisionEnter2D()`** triggers when the bird hits a pipe or ground.
✅ **`birdIsAlive`** prevents further flapping after death.

5. Testing and Fine-Tuning

1. **Play the game.**
    - When the bird hits a pipe → the **Game Over screen** should appear.
    - The bird should stop responding to input.
    - Clicking **Restart** should reload the scene.
2. **Optional Tweaks:**
    - Add a fade-in effect or animation to the Game Over screen.
    - Use **TextMeshPro** instead of legacy UI for higher-quality fonts.
    - Add sound effects for collision and restart.

6. Export the Build

To turn your project into a playable game:

1. Go to **File** → **Build Settings**.
2. Click **Add Open Scenes**.
3. Select your target platform (e.g., Windows, macOS).
4. Click **Build**, choose a folder, and let Unity export.
5. Run the built `.exe` or `.app` — your Flappy Bird clone is complete!

💡 Recap

Concept	Description
SceneManager	Used to load or restart scenes.
GameOver Screen	A UI panel that activates only on failure.
Public Function	Can be called by UI buttons or other scripts.
Boolean (bool)	True/False variable for checking state (e.g., bird alive or not).
OnCollisionEnter2D	Runs automatically when two colliders hit each other.
Active State Toggle	Enables/disables GameObjects via `SetActive(true/false)`.

✅ Project Complete!

You now have:

- Player movement and physics
- Pipe spawning and destruction
- Score tracking with UI
- Game Over and restart logic

🎉 You've built a fully functional 2D game in Unity 6 — and learned foundational programming concepts like:

- Prefabs and spawning
- Collisions and triggers
- Scene management
- UI events
- Boolean logic and state control

🕹️ Next Steps to Finish & Improve Flappy Bird (Unity 6)

1. Add Game Over Conditions

- **Goal:** End the game when the bird goes off-screen or hits something.
- **How:**
  - In your **Bird script**, use a trigger like `OnCollisionEnter2D()` or check the bird's Y position.
  - When game over happens, stop movement and show a "Game Over" UI.

2. Fix the Score Bug

- **Problem:** The score keeps increasing even after the game is over.
- **Fix:**
  - In your scoring script, add a check — only increase score **if the game is not over**.

3. Add Sound Effects 🎵

- **Goal:** Make sounds for flapping, scoring, or game over.
- **How:**
  - Add an **Audio Source** to your **Game Manager** (or Logic Manager).
  - Import `.wav` or `.mp3` files (e.g., jump, point, hit sounds).
  - In script:

```
public AudioSource audioSource;
public AudioClip scoreSound;

void AddScore() {
    audioSource.PlayOneShot(scoreSound);
}
```

4. Add Cloud Particles ☁️

- **Goal:** Add atmosphere with background clouds.
- **How:**
  - Go to **GameObject → Effects → Particle System**.
  - Adjust shape to "Box" or "Cone," reduce emission rate, and use a cloud sprite.
  - Set it behind the pipes and bird in the hierarchy.

5. Animate Bird Wings 🐦

- **Goal:** Make the bird flap its wings.
- **How:**
  - Open the **Animation window**.
  - Record and rotate the wing sprites or swap between two images.
  - Save as “Flap” animation, then use an **Animator Controller** to loop it.

6. Add a Title Screen 🎬

- **Goal:** Start the game from a menu instead of directly.
- **How:**
  - Create a new scene called **MainMenu**.
  - Add “Play” and “Quit” buttons (using **UI** → **Button**).
  - Use `SceneManager.LoadScene("GameScene");` on button click.
  - Go to **File** → **Build Settings** → click “Add Open Scenes” to include both.

7. Save and Show High Score 💾

- **Goal:** Keep the best score even after quitting.
- **How (PlayerPrefs):**

```
int best = PlayerPrefs.GetInt("HighScore", 0);
if (score > best) {
    PlayerPrefs.SetInt("HighScore", score);
}
highScoreText.text =
PlayerPrefs.GetInt("HighScore").ToString();
```

8. Expand and Get Creative 🎨

- Add new mechanics like:
  - Shooting missiles at targets.
  - Different bird skins or environments.
  - Random obstacles or day/night cycles.

9. Keep Learning 🚀

Try remaking other simple games in Unity 6:

- Pong
- Space Invaders

- Breakout
- Chrome Dino
- Angry Birds mini clone

This helps you focus on **coding and game logic** instead of art.