Lesson 5: physics and collision detection.in gaming using unity C#

1. **Unity Physics System: Introduction**
   Well-known text representation of geometry in real world; objects get affected by collisions, gravity, friction, joints and other forces. Similarly in a game; game objects also must be affected by collisions, gravity, friction, joints and other forces.

   To implement such real world physics behaviors; Unity uses:

   2 different physics engines:
   NVIDIA PhysX physics engine to handle 3D physics
   Box2D physics engine to handle 2D physics

   2 different set of components:
   To handle 3D physics it has Physics components like:
   - Rigidbody, Box Collider, Capsule Collider, Fixed Joint, Spring Joint etc.

   To handle 2D physics it has Physics2D components like:
   - Rigidbody2D, Box Collider2D, Capsule Collider2D, Fixed Joint2D, Spring Joint2D etc.

   2 different modules for scripting:
   Physics module has classes, structures, enumerations, etc. for handling 3D Physics
   - Rigidbody, Collider, Collision, Joint, RaycastHit etc.
   Physics2D module has classes, structures, enumerations etc. for handling 2D Physics
   - Rigidbody2D, Collider2D, Collision2D, Joint2D, RaycastHit2D etc.

Rigidbody Component Overview

1. Rigidbody (3D Physics)

Purpose: The Rigidbody component in Unity's 3D physics system allows GameObjects to be affected by forces, gravity, and collisions. It provides realistic movement and interaction within a 3D environment.

Key Properties:

- **Mass**: Determines the weight of the object. Heavier objects require more force to move.
- **Drag**: Affects how the object's movement slows down over time due to resistance. Higher values make the object decelerate faster.

- **Angular Drag**: Affects how the object's rotation slows down over time. Higher values make the object stop rotating faster.
- **Use Gravity**: Toggles whether the object is affected by gravity.
- **Is Kinematic**: If checked, the Rigidbody will not be affected by forces or collisions. This is useful for objects that you control manually through code.
- **Interpolation**: Helps to smooth out the movement of the Rigidbody by interpolating between physics updates. Options include None, Interpolate, and Extrapolate.
- **Collision Detection**: Determines how collisions are detected. Options include Discrete (standard collision detection), Continuous (prevents fast-moving objects from passing through other colliders), and Continuous Dynamic (for objects moving very fast).

Example 1: Basic Rigidbody Setup

1. **Setup:**
   - Create a new Unity project.
   - Add a 3D object, such as a Cube (GameObject > 3D Object > Cube).
   - Select the Cube in the Hierarchy, go to the Inspector, and click "Add Component".
   - Search for and add the Rigidbody component.
2. **Configure Rigidbody:**
   - Set the Mass to 5.
   - Adjust Drag to 2 and Angular Drag to 1.
   - Ensure Use Gravity is checked.
   - Leave Is Kinematic unchecked.
3. **Apply Force:**
   - Create a new C# script named ApplyForce.
   - Attach the script to the Cube.

```
using UnityEngine;


public class ApplyForce : MonoBehaviour
{
    public float forceAmount = 500f;
    private Rigidbody rb;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            rb.AddForce(Vector3.up * forceAmount);
        }
    }
}
```

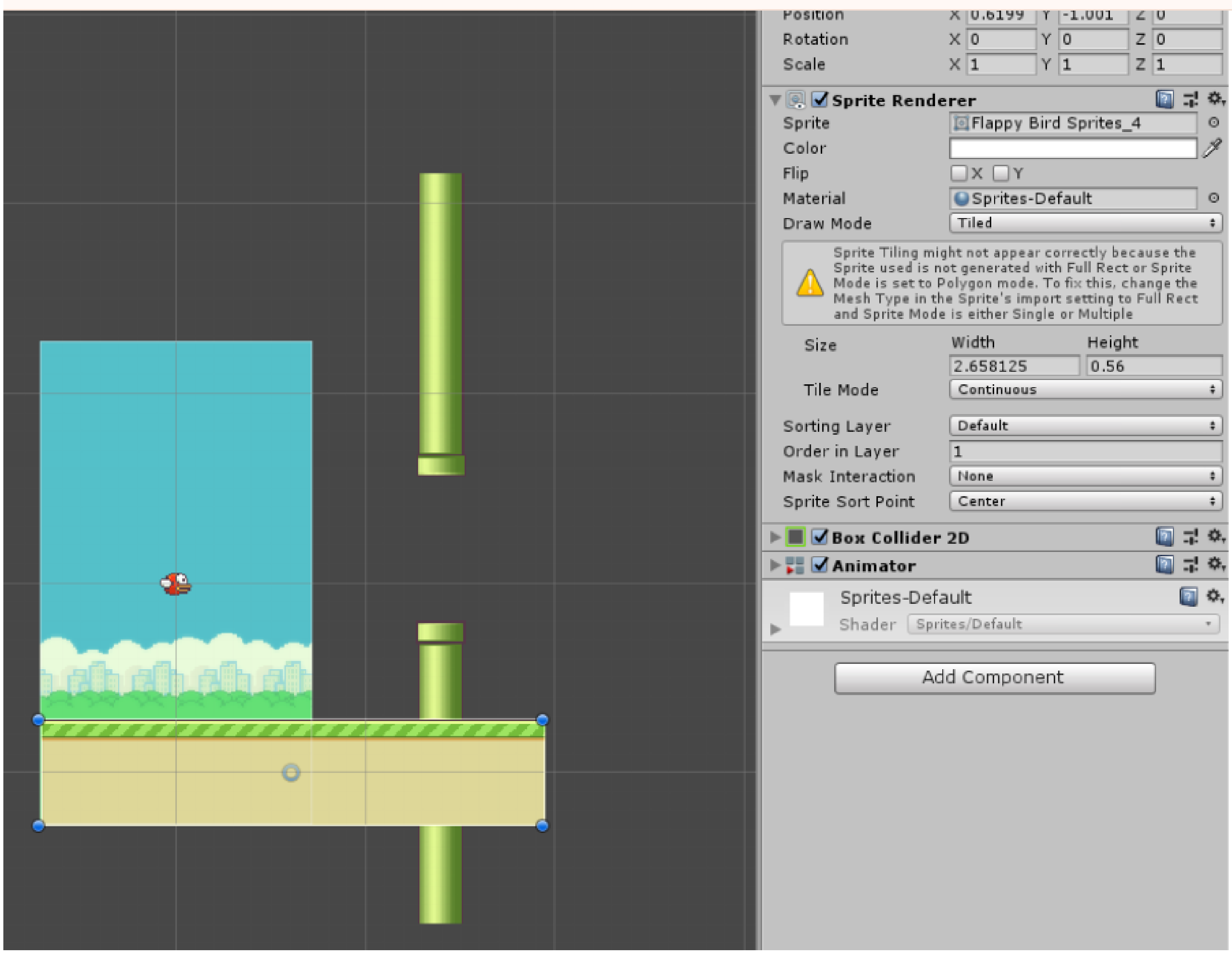
4. **Run the Scene:**
   - Press Play and press the Space bar to apply an upward force to the Cube.

Class practical Revision task

Concept to learn: **physics and collision detection gaming in unity**

Step-by-Step Guide to Create a Flappy Bird Clone

Output deliverable screen shot



```
Position        X 0.6199   Y -1.001   Z 0
Rotation        X 0        Y 0        Z 0
Scale           X 1        Y 1        Z 1
▼ ✓ Sprite Renderer
Sprite              Flappy Bird Sprites_4
Color
Flip                ☐ X ☐ Y
Material            Sprites-Default
Draw Mode           Tiled
  Sprite Tiling might not appear correctly because the
  Sprite used is not generated with Full Rect or Sprite
  Mode is set to Polygon mode. To fix this, change the
  Mesh Type in the Sprite's import setting to Full Rect
  and Sprite Mode is either Single or Multiple
  Size              Width       Height
                    2.658125    0.56
  Tile Mode         Continuous
Sorting Layer       Default
Order in Layer      1
Mask Interaction    None
Sprite Sort Point   Center
► ✓ Box Collider 2D
► ✓ Animator
Sprites-Default
Shader  Sprites/Default
```

1. Set Up the Project

1. **Install Unity Hub**: Download and install Unity Hub from the Unity website. Unity Hub helps manage your Unity projects and versions.
2. **Create a New Project**:
   - Open Unity Hub and click on the "New Project" button.
   - Choose the "2D" template and name your project "FlappyBirdClone".
   - Click "Create" to set up the new project.

2. Set Up the Game Scene

1. **Create the Game Scene**:
    - Open Unity and make sure you're in the "Scene" view.
    - Save your scene as MainScene (File > Save As).
2. **Import Assets**:
    - Import the necessary assets like images for the bird, pipes, and background. You can create these yourself or download free assets from the Unity Asset Store or other resources.

Assets sources

Sprites: https://www.spriters-resource.com/mobile/flappybird/sheet/59894/ Sounds: https://www.sounds-resource.com/mobile/flappybird/sound/5309/

-

3. **Set Up Background**:
    - Drag and drop the background image into the scene.
    - Adjust its size and position to cover the camera view.
4. **Create the Bird**:
    - Drag and drop the bird sprite into the scene.
    - Rename it to "Bird".
    - Add a Rigidbody2D component to the Bird object (Component > Physics 2D > Rigidbody2D).
    - Set the Gravity Scale to 1 (you may adjust this later for fine-tuning).
5. **Create Pipes**:
    - Import and add pipe sprites to the scene.
    - Create a prefab for the pipe. Right-click on the pipe in the hierarchy and choose "Create Prefab".
6. **Create the Ground**:
    - Import and add a ground sprite to the scene.
    - Ensure it has a BoxCollider2D and is positioned at the bottom of the screen.

3. Scripting the Bird

1. **Create a Bird Script**:

    - Right-click in the Project window and create a new C# script named BirdController.
    - Attach the script to the Bird object.

2. **Implement Bird Movement**:
    - Open BirdController.cs and implement the following code:

```
using UnityEngine;
public class BirdController : MonoBehaviour
{
    public float flapStrength = 10f;
    private Rigidbody2D rb;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            Flap();
        }
    }

    void Flap()
    {
        rb.velocity = Vector2.up * flapStrength;
    }
}
```

3. **Adjust Bird Physics**:
    - Play around with the flapStrength value in the Inspector to make the bird's movement feel right.

4. Creating the Pipe Movement

1. **Create a Pipe Movement Script**:
    - Create a new C# script named PipeController.
    - Attach it to the pipe prefab.
2. **Implement Pipe Movement**:
    - Open PipeController.cs and implement the following code:

```
using UnityEngine;

public class PipeController : MonoBehaviour
{
    public float moveSpeed = 5f;

    void Update()
    {
        transform.Translate(Vector2.left * moveSpeed * Time.deltaTime);
    }

    void OnBecameInvisible()
    {
        Destroy(gameObject);
    }
}
```

3. **Create a Pipe Spawner**:
    - Create a new empty GameObject in the scene and name it PipeSpawner.
    - Create a new C# script named PipeSpawner.
    - Attach the script to the PipeSpawner object.

```
using UnityEngine;

public class PipeSpawner : MonoBehaviour
{
    public GameObject pipePrefab;
    public float spawnRate = 2f;
    public float heightOffset = 10f;

    void Start()
    {
        InvokeRepeating("SpawnPipe", 0f, spawnRate);
    }

    void SpawnPipe()
    {
        float height = Random.Range(-heightOffset, heightOffset);
        Vector3 spawnPosition = new Vector3(transform.position.x, height, 0);
        Instantiate(pipePrefab, spawnPosition, Quaternion.identity);
    }
}
```

4. **Assign Pipe Prefab**:
   - Drag and drop the pipe prefab into the Pipe Prefab field of the PipeSpawner component in the Inspector.

5. Add Collisions and Scoring

1. **Add Colliders**:
   - Ensure that the Bird, Pipes, and Ground have appropriate colliders (BoxCollider2D for pipes and ground, CircleCollider2D for the bird).
2. **Create a Collision Script**:
   - Create a new C# script named GameController.
   - Attach it to an empty GameObject named GameController.

- Implement the following code:

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameController : MonoBehaviour
{
    public GameObject gameOverText;

    void Start()
    {
        gameOverText.SetActive(false);
    }

    public void GameOver()
    {
        gameOverText.SetActive(true);
        Time.timeScale = 0; // Pause the game
    }
}
```

3. **Handle Collisions**:
    - Modify the BirdController script to detect collisions with pipes and ground:

```
using UnityEngine;

public class BirdController : MonoBehaviour
{
    public float flapStrength = 10f;
    private Rigidbody2D rb;
    private GameController gameController;
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        gameController = FindObjectOfType<GameController>();
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            Flap();
        }
    }

    void Flap()
    {
        rb.velocity = Vector2.up * flapStrength;
    }

    void OnCollisionEnter2D(Collision2D collision)
    {
        gameController.GameOver();
    }
}
```

4. **Test and Fine-Tune**:
    - Play the game and adjust the parameters like flap strength, pipe speed, and spawn rate to get the desired gameplay experience.

6. Adding UI Elements

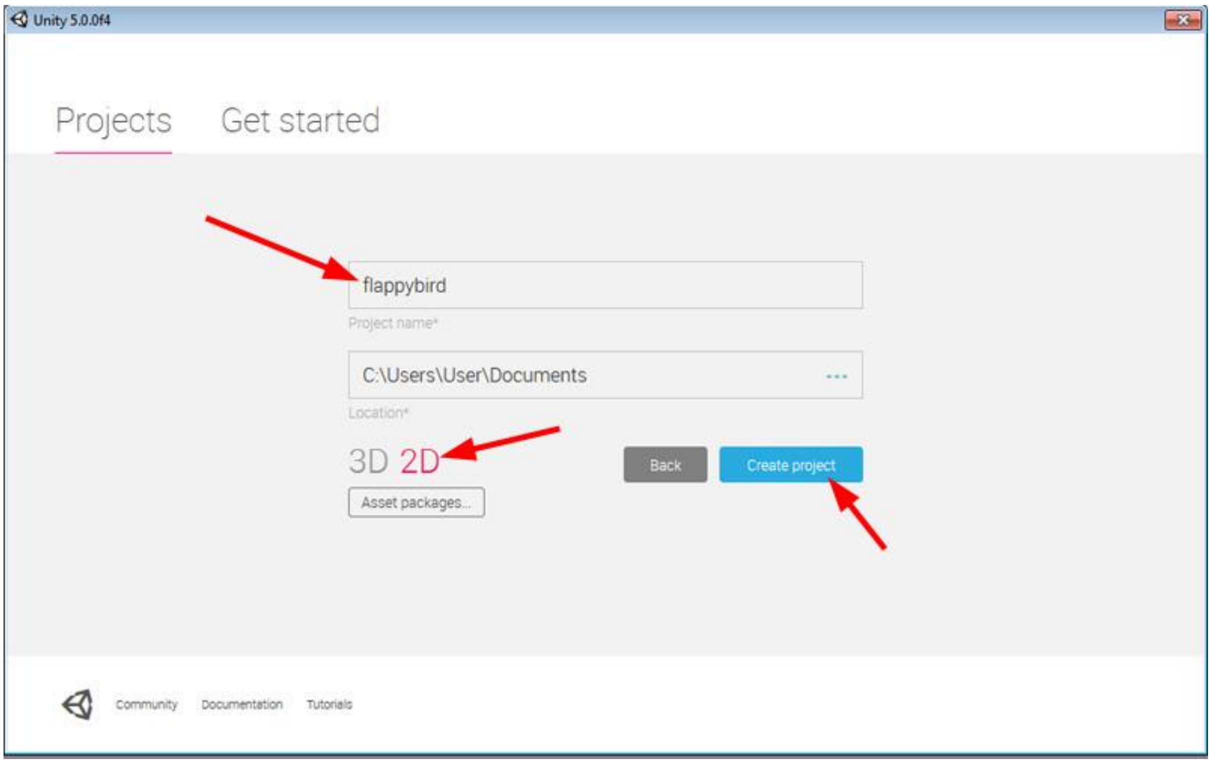
1. **Create UI for Score and Game Over**:
    - Add Text elements to display the score and game over message (GameObject > UI > Text).
    - Set up a script to update the score based on successful pipe passes and display the game over message.
2. **Implement Score Tracking**:
    - Create a script named ScoreManager and attach it to an empty GameObject named ScoreManager.
    - Implement basic scoring logic based on pipe passing.

Conclusion

This guide provides a basic framework for creating a Flappy Bird clone in Unity. You can extend and refine the game by adding features like different levels, sound effects, and animations. Testing and tweaking will be essential to get the game feeling just right. Enjoy your game development journey!

Hint to complete the Exercise Above

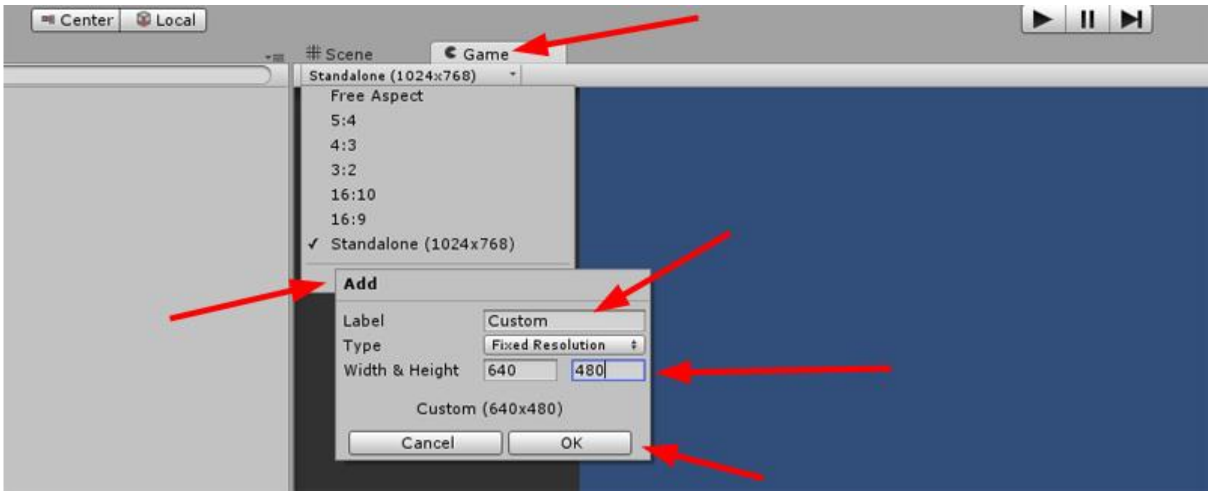
First, let's create a new project, give it a name and above all set it to a 2D project.



flappybird

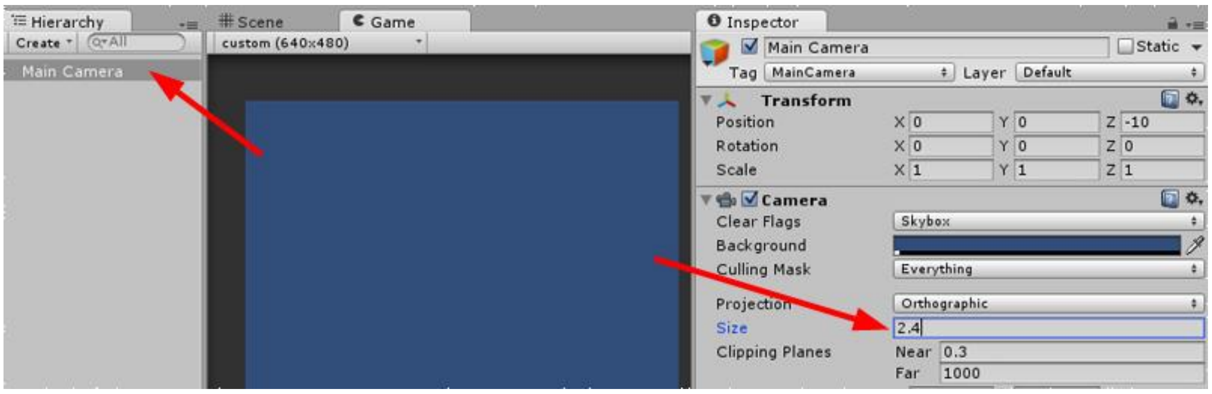
C:\Users\User\Documents

Our first prpject will be a game to be played on the web, so let's change the resolution to 640×480



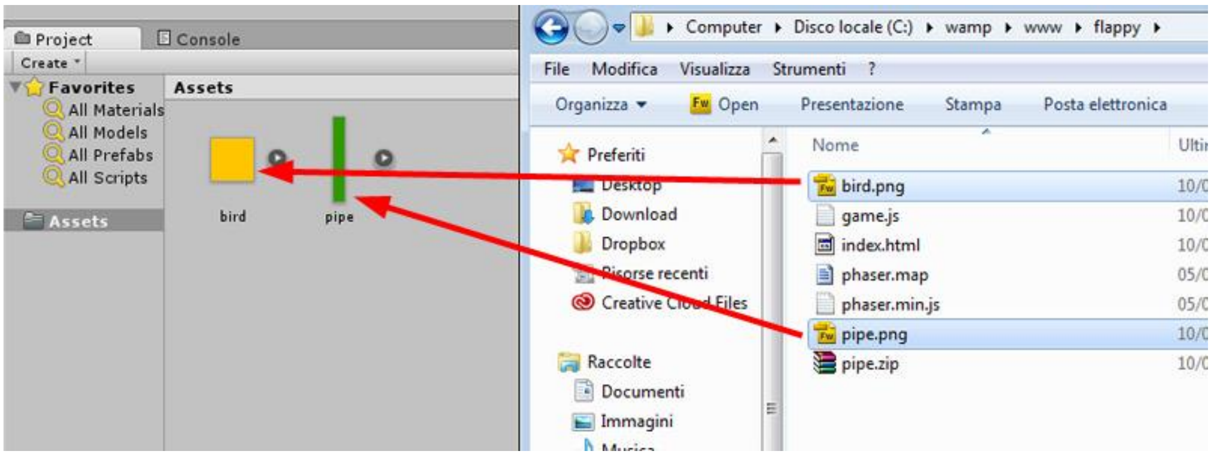
Label: Custom
Type: Fixed Resolution
Width & Height: 640 480
Custom (640x480)

In the Main Camera inspector, set the size to game height/200, that is 480/200 = 2.4

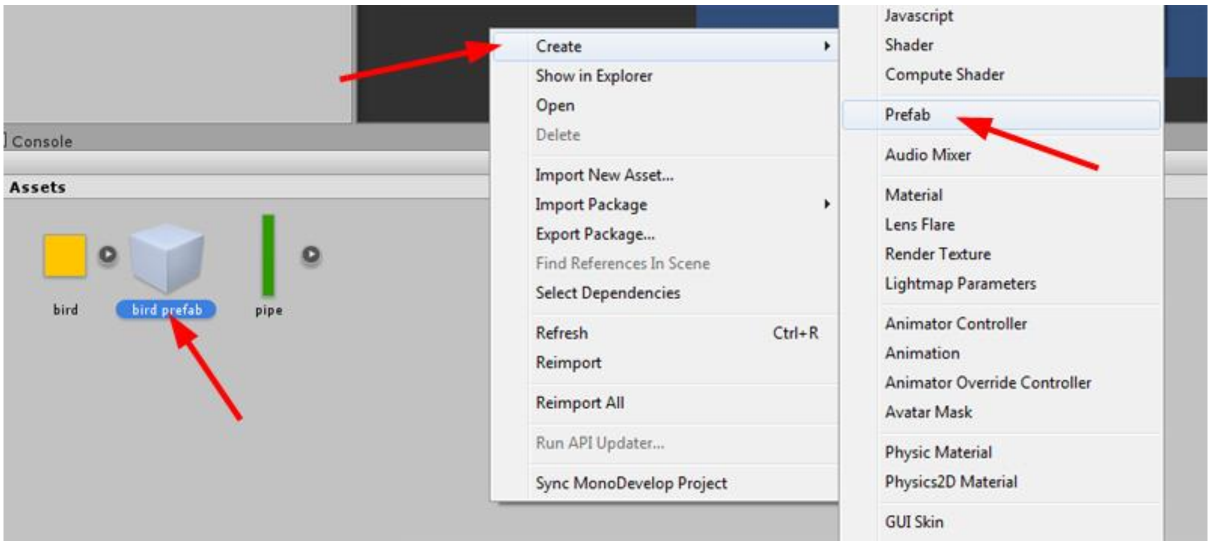


Main Camera
Tag: MainCamera Layer: Default
Position X 0 Y 0 Z -10
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Clear Flags: Skybox
Background
Culling Mask: Everything
Projection: Orthographic
Size: 2.4
Clipping Planes: Near 0.3 Far 1000

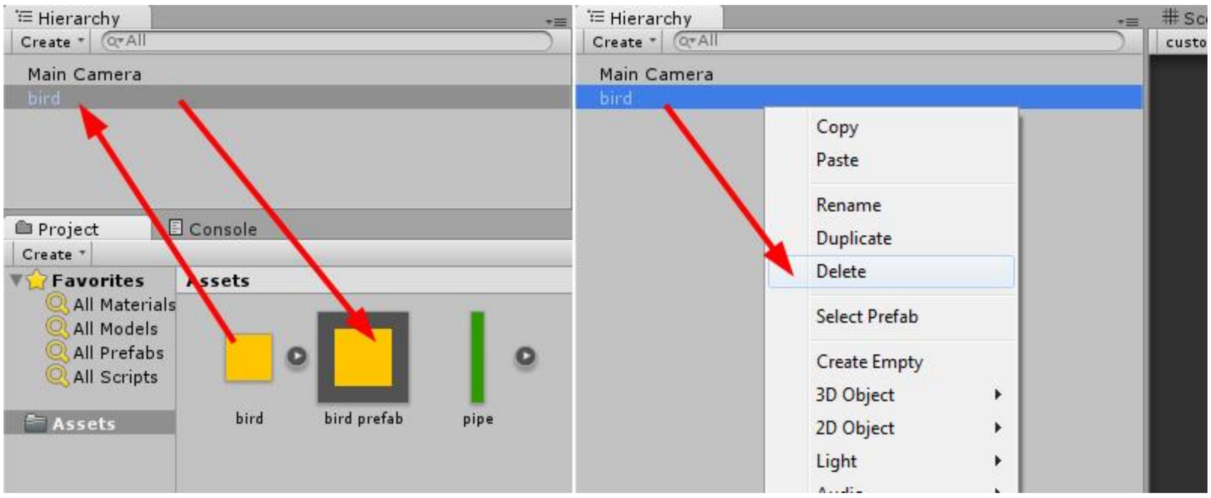
It's now time to import some graphic assets: I am using the same png files used in the post HTML5 Flappy Bird prototype using Phaser states, extended classes and Arcade physics. Just drag and drop them in the project window, or create your own bird and pipe png images, then drag them in the project window.



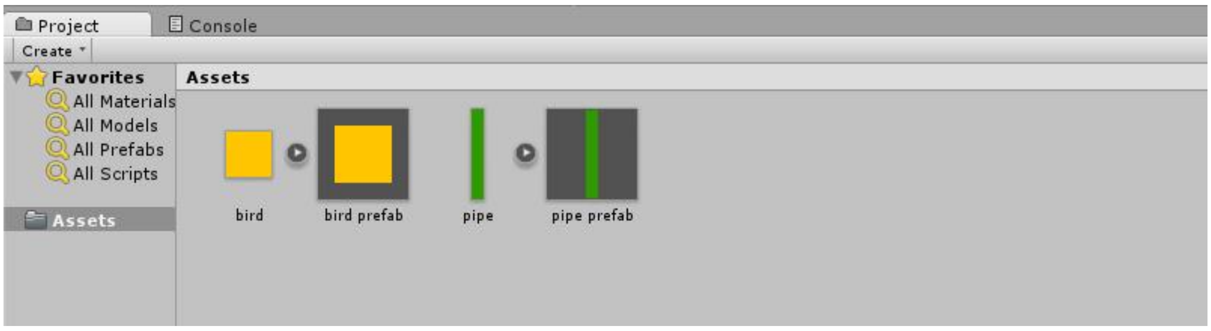
The assets you just uploaded are just plain images, so we need to create something more complex for Unity to use in the game. Unity features **prefabs**, which are complex objects which can have images, scripts and component. Right click with the mouse in the project window, then create -> prefab. Call your newly created prefab **bird prefab**.



To add an image to a prefab, first we drag the image in the hierarchy window, then we drag the image instance in the prefab, then we can delete the image from the hierarchy window. Now our prefab has an image assigned to it. Look how I placed the bird image in bird prefab:

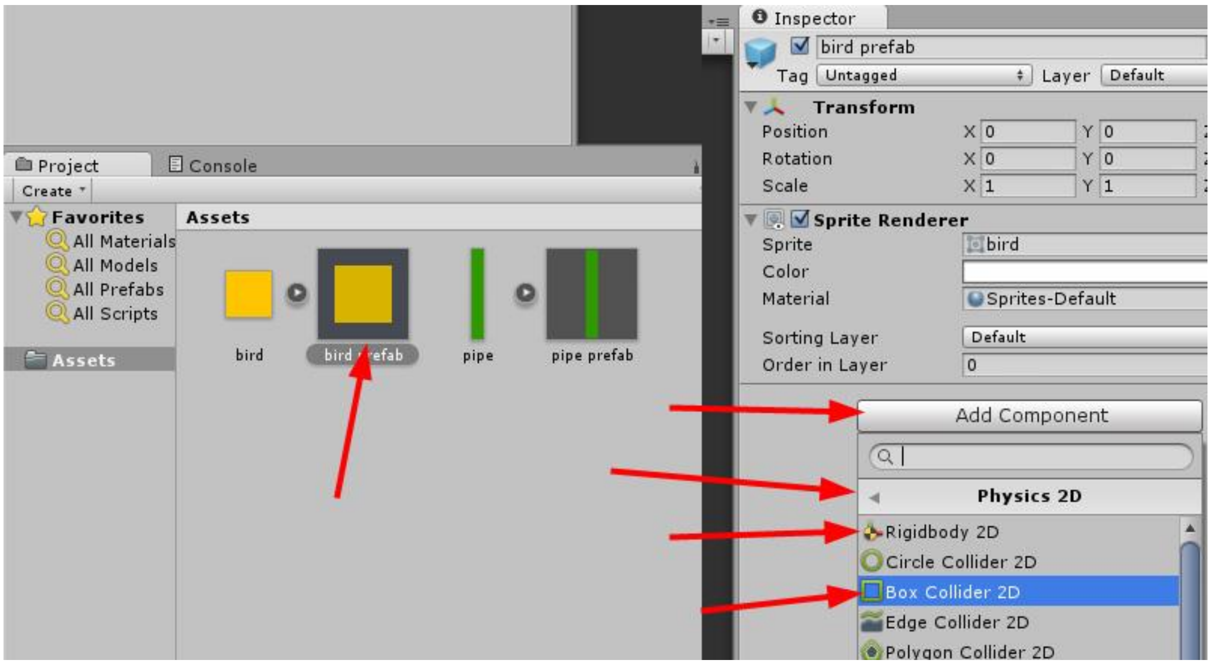


Now you should be able to apply the same concept to create a pipe prefab and place pipe image in it.



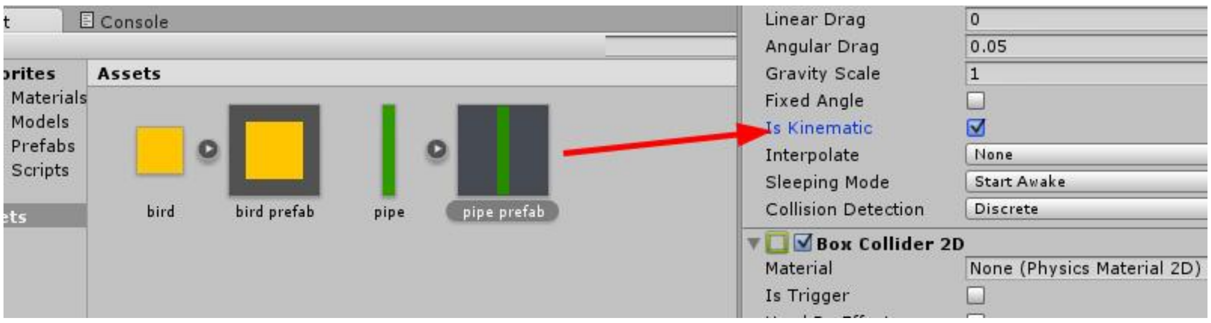
We just said a prefab an can also have components, and our prefabs are going to have two components: select the bird prefab, from inspector window select “Add Component” then under “Physics 2D” add both “Rigidbody 2D” and “Box Collider

2D”. Leave all options at their default values. This will assign the bird a physics rigid body and a physics collider.



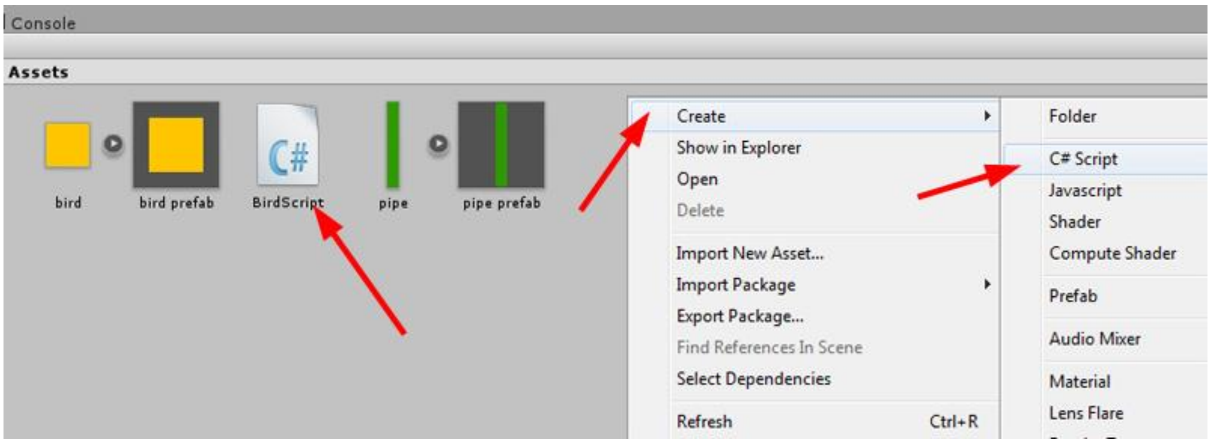
bird prefab
Tag Untagged Layer Default
Transform
Position X 0 Y 0
Rotation X 0 Y 0
Scale X 1 Y 1
Sprite Renderer
Sprite bird
Color
Material Sprites-Default
Sorting Layer Default
Order in Layer 0

Do the same for pipe prefab, just check “Is Kinematic” checkbox so pipes won’t be affected by gravity.

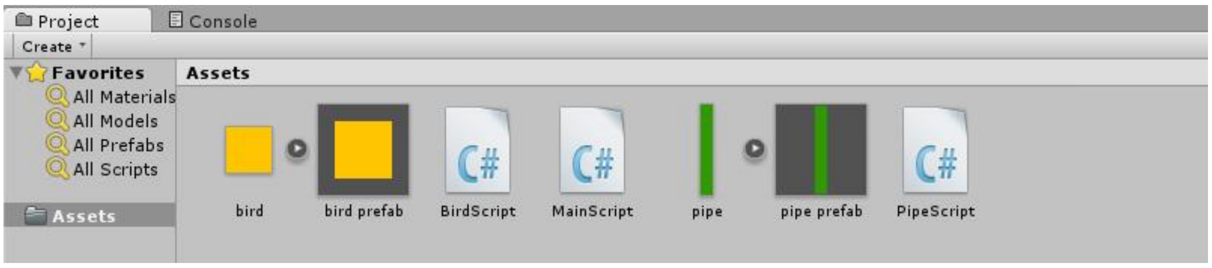


Linear Drag 0
Angular Drag 0.05
Gravity Scale 1
Fixed Angle ☐
Is Kinematic ✓
Interpolate None
Sleeping Mode Start Awake
Collision Detection Discrete
Box Collider 2D
Material None (Physics Material 2D)
Is Trigger ☐

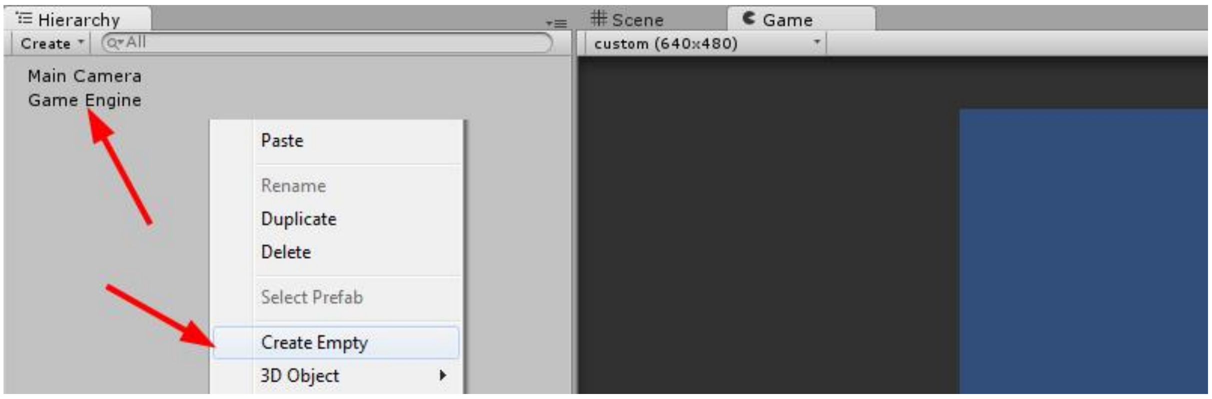
Prefabs can also include scripts, so we are going to create a script right clicking in project window and selecting “Create” -> “C# Script”. Call the script BirdScript.



In the same way create PipeScript and MainScript.



While it's obvious PipeScript and BirdScript will be assigned respectively to pipe and bird prefabs, MainScript will be the engine of the game and will be assigned to an empty object we will create by right clicking on hierarchy window and selecting "Create Empty". Call this object "Game Engine".



Complete step by step guide with pictures and code guides blog link

https://generalistprogrammer.com/unity/unity-2d-flappy-bird-how-to-make-flappy-bird/