# [The Unity Tutorial For Complete Beginners](https://youtu.be/XtQMytORBmM)

Game Maker's Toolkit

2,019,859 views Dec 3, 2022

=== Files and Downloads ===

Assets - <https://www.dropbox.com/sh/h5vez7ltgbmfnib/AADSCiI2dEKptcR7ydv8xR2Ba?dl=0>

Scripts - <https://pastebin.com/QiLkpeJe>

Unity Project - <https://github.com/Britishgaming/GMTK-Unity-Tutorial>

Installing workspace

1. Search google unity.com
2. Download Unity Hub > Create Unity ID
3. Install Unity Editor > Add modules
4. Microsoft Visual Studio Community > Continue > Install
5. Gaming > Game development with Unity > Untick unity hub in Installation details > Install
6. In Unity Hub > Click New Project > Choose All templates > 2D
7. Project Settings > Project name > Flarpy Blorb > Create project

Creating game

Step 1: became familiar with the Unity interface

1. Splits into 4 panels (hierarchy, project, scene, inspector)
2. Project panel contains everything that is in our game
3. Sprites, Sound effect, Script, Tiles, Font and so on
4. Right click in Project panel > Import New Asset > birdbody.png > pipelonger.png > Import
5. Hierarchy panel contains stuff that’s in the current scene which in most games, will be a level
6. Right click in Hierarchy > Create empty > Game Object
7. GameObject is essentially an invisible container. It has a position in space, a rotation and a scale.

Absolutely everything in our level will be a GameObject with components – the bird, the pipes, even the user interface and the camera

1. Inspector panel is for GameObjects to put a name Bird on top of the field and change the GameObject’s position, rotation, and scale, under Transform.
2. In Add Component, pick Rendering, and pick Sprite Renderer, drag the birdbody.png to the Sprite Renderer > Sprite field, then in Scene Panel you will see graphics
3. At the right of Scene Panel/View > Click Game Panel/View > Change Free Aspect to Full HD (1920x1080)
4. In Hierarchy panel > Click Main Camera > Change Camera Size 13.23 > Change Background Hexadecimal F4ABA2 > Click the Play button at top

Step 2: physics and programming code

1. In Hierarchy Panel > Click Bird
2. In Inspector Panel > Click Add component > Physics 2D > Rigidbody 2D
3. Click Play button, the bird should fall > Click Play button again to stop
4. In Inspector Panel > Add component > Physics 2D > Circle Collider 2D > Click Scene View
5. In Inspector Panel > Circle Collider 2D > Change Offset x: -0.03 y: -0.45 Radius: 1.91
6. Add component > New Script > Name: BirdScript
7. In Inspector Panel > Double click BirdScript to open
8. In Rigidbody 2D component > Info > Greyed out field for Velocity
9. In BirdScript.cs, create reference to Rigidbody 2D
10. In Inspector Panel > Drag Rigidbody2D component into slot in Bird Script (Script) component
11. Back in Visual Studio BirdScript.cs, set velocity in void Update()

// Update is called once per frame

void Update() {

if (Input.GetKeyDown(KeyCode.Space))

{

myRigidbody.velocity = Vector2.up \* 10;

}

}

Step 3: spawning new objects

1. In Hierarchy > Right click Create empty GameObject > Name: Pipe > Right click on Pipe > Create empty GameObject > Top Pipe
2. In Hierarch > Click Top Pipe > In Inspector Panel > Click Add Component > Rendering > Sprite Renderer
3. Drag pipe.png to Sprite Renderer:Sprite field
4. Click Add component > Physics 2D > Box Collider 2D > Change Size X: 4.73 Y: 21.04 > Transform: Position Y: 20
5. In Hierarchy > Right click Top Pipe > Click Duplicate > Name: Bottom Pipe > Transform: Scale Y:-1 > Transform: Position Y:-20
6. In Hierarchy > Click Pipe > Add component > New Script > PipeMoveScript > Create and Add

public float moveSpeed = 5;

// Update is called once per frame

void Update()

{

transform.position = transform.position + (Vector3.left \* moveSpeed) \* Time.deltaTime;

}

1. Continually spawn new pipes: In Hierarchy > Long left click Pipe and drag to Project Panel (creates a Prefabricated GameObject or Prefab)
2. In Hierarchy > Delete Pipe GameObject > In Hierarchy > Right click > Create empty GameObject > Name: Pipe Spawner > Transform: Position X: 28.5 Y: 0.11163 Z: -0.214 > Add component > PipeSpawnScript > Create and Add

public GameObject pipe;

public float spawnRate = 2;

private float timer = 0;

public float heightOffset = 10;

// Start is called before the first frame update

void Start()

{

spawnPipe();

}

// Update is called once per frame

void Update()

{

if (timer < spawnRate)

{

timer = timer + Time.deltaTime;

}

else

{

spawnPipe();

timer = 0;

}

}

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);

}

1. In Hierarchy > Click Pipe Spawner > In Project panel > Drag the Pipe (Prefab) > Inspector Panel > Pipe Spawn Script (Script) > Pipe field
2. Fix too many spawn: In Hierarchy > Click Bird > In Inspector > Transform: Position X: 0
3. In PipeMoveScript

public float deadZone = -45;

// Update is called once per frame

void Update()

{

transform.position = transform.position + (Vector3.left \* moveSpeed) \* Time.deltaTime;

if (transform.position.x < deadZone)

{

Debug.Log("Pipe Deleted");

Destroy(gameObject);

}

}

Step 4: UI, game logic, and collisions

1. In Hierarchy > Right click inside Panel > UI > Legacy > Text > Click Canvas
2. In Inspector > Canvas Scaler: UI Scale Mode > Scale With Screen Size > Reference Resolution X: 1920 Y: 1080
3. In Hierarchy > Click Text (Legacy) > In Inspector > Rect Transform Pos X: -682 Pos Y: 282 Width: 342.4 Height: 373.1 > Text Text: 0 > Text Font: Arial > Text Character Font Style: Bold > Text Character Font Size: 212 > Text Color: FFFFFF
4. In Heirarchy > Right click inside Panel > Create empty GameObject > Logic Manager > Add component > New Script > LogicScript > Create and Add

using UnityEngine.UI;

public class LogicScript : MonoBehaviour

{

public int playerScore;

public Text scoreText;

[ContextMenu("Increase Score")]

public void addScore()

{

playerScore = playerScore + 1;

scoreText.text = playerScore.ToString();

}

}

1. In Hierarchy > Drag Text (Legacy) > In Inspector > Logic Manager > Logic Script (Script) > Score Text
2. In Project Panel > Double click Pipe (Prefab) > In Hierarchy Panel > Right click Pipe > Create empty GameObject > Middle > Add component > Physics 2D > Box Collider 2D > Change Size X: 0.85 Y: 18 > Tick box Is Trigger > Add component > PipeMiddleScript > Create and Add
3. In PipeMiddleScript

public LogicScript logic;

public void OnTriggerEnter2D(Collider2D collision)

{

}

1. In Hierarchy > Click Logic Manager > In Inspector > Click Tag > Choose Add Tag… > Tags > Click Plus Sign > New Tag Name: Logic > Save
2. In Hierarchy > Click Logic Manager > Click Tag > Choose Logic
3. In Hierarch > Click Bird > > In Inspector > Click Layer > Choose Add Layer… > User Layer 3 > Bird
4. In Hierarchy > Click Bird > Click Layer > Choose 3
5. In PipeMiddleScript

// Start is called before the first frame update

void Start()

{

logic = GameObject.FindGameObjectWithTag("Logic").GetComponent<LogicScript>();

}

public void OnTriggerEnter2D(Collider2D collision)

{

if (collision.gameObject.layer == 3)

{

logic.addScore(1);

}

}

1. In LogicScript

using UnityEngine.UI;

public class LogicScript : MonoBehaviour

{

public int playerScore;

public Text scoreText;

[ContextMenu("Increase Score")]

public void addScore(int scoreToAdd)

{

playerScore = playerScore + scoreToAdd;

scoreText.text = playerScore.ToString();

}

}

Final Step 5: game over and scene management

1. In Hierarchy > Right click Canvas > UI > Legacy > Text > Right Click Canvas > Create empty GameObject > Name: Game Over Screen > Right click Game Over Screen
2. In Inspector > Rect Transform Width: 942.72 Height: 237 > Text Text: Game Over > Text Font Size: 155 > Text Font Style: Bold > Text Color: FFFFFF > Text Paragraph Alignment: Center Middle
3. In Hierarchy > Right click Game Over Screen > UI > Legacy > Button
4. In Inspector > Rect Transform Pos Y: -166.2 Width: 569.51 Height: 113.1
5. In Hierarchy > Expand Button (Legacy) > Click Text (Legacy)
6. In Inspector > Text Text: Play Again > Text Character Font Size: 68
7. In Hierarchy > Click Logic Manager > Double click Script: Logic Script
8. In LogicScript

using UnityEngine.SceneManagement;

public class LogicScript : MonoBehaviour

{

public void restartGame()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().name);

}

}

1. In Hierarchy > Click Button (Legacy) > In Inspector > On Click () > Click Plus Sign
2. In Hierarch > Drag Logic Manager to Inspector > On Click () > None (Object)
3. In Inspector > On Click ()> Runtime function dropdown > LogicScript > restartGame()
4. In Hierarchy > Click Game Over Screen
5. In Inspector > Untick Game Over Screen
6. In LogicScript

public GameObject gameOverScreen;

public void gameOver()

{

gameOverScreen.SetActive(true);

}

1. In Hierarchy > Drag Game Over Screen to Inspector > Logic Script (Script) > Game Over Screen
2. In BirdScript

public float flapStrength;

public LogicScript logic;

public bool birdIsAlive = true;

// Start is called before the first frame update

void Start()

{

logic = GameObject.FindGameObjectWithTag("Logic").GetComponent<LogicScript>();

}

// Update is called once per frame

void Update()

{

if(Input.GetKeyDown(KeyCode.Space) && birdIsAlive)

{

myRigidbody.velocity = Vector2.up \* flapStrength;

}

}

private void OnCollisionEnter2D(Collision2D collision)

{

logic.gameOver();

birdIsAlive = false;

}

1. Click File > Build Settings > Build And Run

# COMP4300 - C++ Game Programming - Lecture 01 - Course Introduction

**Dave Churchill**

**6.43K subscribers**

**Intro to Game Programming**

* Emphasis on the ***PROGRAMMING***
* This course is:
* Programming that makes video games possible
* Game Engine / Gameplay Programming
* C++ / SFML
* Actually kind of Fun!
* This course isn’t:
* A game design course (may mention design concepts)

**Introduction to C++ / SFML**

* Introduction to C++
* C++ Syntax / Semantics
* Standard Template Library (SFML)
* Simple and Fast Multimedia Library (SFML)
* Basics of SFML
* Windows, Rendering, Input Handling, etc

**2D Game Engine Programming**

* Game Engine Layout / Architecture
* Main Loop Structure / Tick Rate
* Game States & State Machines
* Asset Loading / Memory Management
* Sprites & Animations, Rendering
* Basic Shaders
* User Input Handling & Event Systems
* Data Oriented Design & Config Files
* Window / Menu / Drag & Drop Systems
* World View: Camera Systems / Viewports

**ECS Game Engine Architecture**

* Entites, Components, Systems (ECS)
* Architecture, Design, Implementation
* ECS Classes, Structure, Memory Mgmt
* Systems for Gameplay Mechanics

**Physics / Math for Games**

* Vector Math / Class Implementation
* Game Object Kinematics
* Position, Velocity, Rotation
* Acceleration, Gravity, Projectiles
* Collision Detection / Resolution
* Circles, Rectangles
* Bounding Boxes
* Mass, Momentum, Inertia

**Gameplay Programming**

* Basic Artificial Intelligence
* NPC Behavior, Steering, Path-Finding
* Entity Interactions / Dialogues
* Difficulty Settings / Game Config
* Game Events / Triggers
* Items / Inventory / Weapons
* Saving / Loading Games

**Plan of Attack**

1. Introduction to C++ / SFML
2. Assignment 1: Simple C++ / SFML

* Simple game main loop
* Drawing a Sprite to the Screen
* Playing a Sound when something happens
* (Get used to C++ / SFML syntax and compiling)

1. Introduce ECS Game Engine Architecture
2. While (!courseOver)

* Introduce new topic(s)
* Assignment: make game using the new topics

**ECS Game Programming**

* Why use ECS for games?
* ECS = Entities, Components, Systems
* Entity: Anything object in the game
* Player, Platform, Tile, Bullet, Enemy
* ECS is actually a software design paradigm
* Can be used for all software, not just games

**Object Oriented Programming**

* Object-Oriented Programming uses the concept of Objects and Inheritance to implement specific functionality
* Object-Oriented Design (in games) is the process of strictly using OOP in order to design the architecture of our game engine and the entities within it

**Object-Oriented**

* OOP / OO Design helps us manage data /code
* Entity: any object in game
* Each of them have (x,y) pos
* Entity used as base-class
* Tile, Player, inherit from it

**Entity-Component**

* Entity: object in game
* Component: properties that can attach to Entities
* Component examples:
* Position, Speed
* Bounding Box
* Health, Weapon, Damage

**ECS Game Programming**

* ECS uses Composition-based design
* Entity: Anything object in the game
* Player, Platform, Tile, Bullet, Enemy
* Component: Properties attached to entities
* Position, Texture, Animation, Health, Gravity
* Components are PURE DATA
* Systems: Code / logic that drives behavior
* Movement, Rendering, Sound, Physics

**ECS Example Entity/Components**

* Player
* Pos, Speed, BBox, Sprite, Health, Gravity, Input
* Enemy
* Pos, Speed, BBox, Sprite, Health, AI
* Bullet
* Pos, Speed, Angle, BBox, Sprite, Damage, Lifespan
* Tile
* Pos, BBox, Sprite

**ECS Example System**

* Movement System

for (e: entities) { e.pos += e.velocity; }

* Collision System

for (b : bullets)

for (e : enemies)

if (Physics::IsCollision(b,e))

e.health -= b.damage

b.destroy();

* Rendering System

for (e : entities) { window.draw(e.sprite, e.pos); }

**Engine Architecture**

**GameEngine**

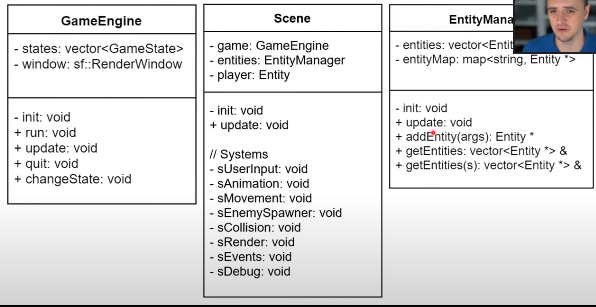
**> Scene**

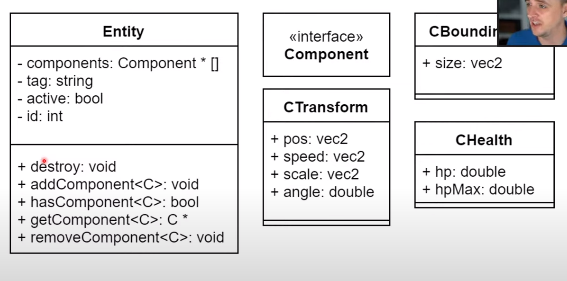
**>Systems**

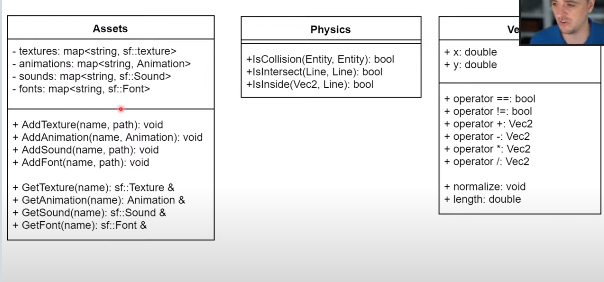
**>EntityManager**

**>Entity**

**>Component**







# COMP4300 - C++ Game Programming - Lecture 02 - Intro to C++ (1/2)

C++ References

<https://www.learncpp.com/>

Excellent tutorial, step by step

<https://en.cppreference.com/w/>

Go-to reference for libraries / functions

<http://www.cplusplus.com/doc/tutorial/>

What is C++?

C++ is a programming language that focuses on run speed & functionality

C++ is a compiled language

C++ is a mid-level programming language

High Level = Python

Low Level = Assembly