**Practical 1: Setup DirectX 11, Window Framework and Initialize Direct3D Device, Loading models into DirectX 11 and rendering.**

1. **Setup DirectX 11, Window Framework and Initialize Direct3D Device**

**Steps:**

1. Create a New Windows Forms Application:

* Open Visual Studio and create a new Windows Forms Application project.

1. Add Required References:

* Add references to Microsoft.DirectX and Microsoft.DirectX.Direct3D libraries in your project.

1. Design the Form:

* Open the Form Designer and make any necessary adjustments to your form (like size, title, etc.).

1. Declare Device Variable:

* In your form class, declare a variable for the Direct3D device.

1. Initialize the Form:

* Call the InitializeComponent() method in the form's constructor.

1. Set Up Presentation Parameters:

* Create a method InitDevice() to set up the presentation parameters for the Direct3D device.

1. Create the Direct3D Device:

* Instantiate the Direct3D device inside the InitDevice() method.

1. Render Method:

* Create a Render() method to handle rendering operations.

1. Handle Paint Event:

* Subscribe to the form’s Paint event and call the Render() method.

**C:windows:.net:DirectX Managed Code**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using Microsoft.DirectX;

using Microsoft.DirectX.Direct3D;

namespace WindowsFormsApplication5

{

public partial class Form1 : Form

{

Microsoft.DirectX.Direct3D.Device device;

public Form1()

{

InitializeComponent();

InitDevice();

}

private void InitDevice()

{

PresentParameters pp = new PresentParameters(); // CREATE OBJECT

pp.Windowed = true;

pp.SwapEffect = SwapEffect.Discard;

device = new Device(0, DeviceType.Hardware, this, CreateFlags.HardwareVertexProcessing, pp);

}

private void Render()

{

device.Clear(ClearFlags.Target, Color.Blue, 0, 1);

device.Present();

}

private void Form1\_Paint\_1(object sender, PaintEventArgs e)

{

Render();

}

}

}

**Final Steps:**

1. Run the Application:

* Build and run the application to see the rendered texture and text on the form.

1. Debugging:

* If the application does not run correctly, check for exceptions, and ensure your texture path is valid.

1. **Loading models (image and fonts) into DirectX 11 and rendering.**

**Steps:**

1. Extend the Form Class:

* Continue in the same form class or create a new one for loading models and fonts.

1. Declare Texture and Font Variables:

* Add variables to hold your texture and font.

1. Initialize Font:

* Create a method InitFont() to initialize the font.

1. Load Texture Method:

* Create a method LoadTexture() to load the image texture.

1. Initialize the Device Again:

* Call InitDevice(), InitFont(), and LoadTexture() in the constructor.

1. Render Textures and Fonts:

* Update the Render() method to draw the texture and font.

1. Paint Event Again:

* Ensure the Paint event calls the updated Render() method.

**Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using Microsoft.DirectX;

using Microsoft.DirectX.Direct3D;

namespace WindowsFormsApplication10

{

public partial class Form1 : Form

{

Microsoft.DirectX.Direct3D.Device device;

Microsoft.DirectX.Direct3D.Texture texture;

Microsoft.DirectX.Direct3D.Font font;

public Form1()

{

InitializeComponent();

InitDevice();

InitFont();

LoadTexture();

}

private void InitFont()

{

System.Drawing.Font f = new System.Drawing.Font("Arial", 16f, FontStyle.Regular);

font = new Microsoft.DirectX.Direct3D.Font(device, f);

}

private void LoadTexture()

{

texture = TextureLoader.FromFile(device, "C:\\Users\\Public\\Pictures\\Sample Pictures\\Desert.jpg", 400,

400, 1, 0, Format.A8B8G8R8, Pool.Managed, Filter.Point, Filter.Point, Color.Transparent.ToArgb());

}

private void InitDevice()

{

PresentParameters pp = new PresentParameters();

pp.Windowed = true;

pp.SwapEffect = SwapEffect.Discard;

device = new Device(0, DeviceType.Hardware, this, CreateFlags.HardwareVertexProcessing, pp);

}

private void Render()

{

device.Clear(ClearFlags.Target, Color.CornflowerBlue, 0, 1);

device.BeginScene();

using (Sprite s = new Sprite(device))

{

s.Begin(SpriteFlags.AlphaBlend);

s.Draw2D(texture, new Rectangle(0, 0, 0, 0), new Rectangle(0, 0, device.Viewport.Width,

device.Viewport.Height), new Point(0, 0), 0f, new Point(0, 0), Color.White);

font.DrawText(s, "Desert", new Point(0, 0), Color.White);

s.End();

}

device.EndScene();

device.Present();

}

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Render();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

}

}

**Final Steps:**

1. Run the Application:

* Build and run the application to see the rendered texture and text on the form.

1. Debugging:

* If the application does not run correctly, check for exceptions, and ensure your texture path is valid.

**Practical 2: Learn Basic Game Designing Techniques with pygame.**

**Cmd installation:**

**Pip install pygame**

**Code:**import pygame

import sys

# Initialize Pygame

pygame.init()

# Set up display

WIDTH, HEIGHT = 600, 400

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Shapes Display")

# Define colors

WHITE = (255, 255, 255)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

BLUE = (0, 0, 255)

YELLOW = (255, 255, 0)

# Function to draw shapes

def draw\_shapes():

# Clear the screen

screen.fill(WHITE)

# Draw a rectangle

pygame.draw.rect(screen, RED, (50, 50, 100, 50)) # x, y, width, height

# Draw a square

pygame.draw.rect(screen, GREEN, (200, 50, 50, 50)) # x, y, width, height

# Draw a circle

pygame.draw.circle(screen, BLUE, (400, 75), 30) # center\_x, center\_y, radius

# Draw a triangle

points = [(300, 150), (250, 200), (350, 200)] # List of points

pygame.draw.polygon(screen, YELLOW, points) # Color, points

# Update the display

pygame.display.flip()

# Main loop

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

draw\_shapes()

**Practical 3: Develop Snake Game using pygame.**

**Cmd installation:**

**pip install pygame**

**Code:**

import pygame

import random

import sys

# Initialize pygame

pygame.init()

# Constants

WIDTH, HEIGHT = 500, 500

CELL\_SIZE = 20

FOOD\_SIZE = CELL\_SIZE

DELAY = 100

SNAKE\_COLOR = (0, 255, 0)

FOOD\_COLOR = (255, 0, 0)

BG\_COLOR = (0, 0, 0)

# Directions

offsets = {

"up": (0, -CELL\_SIZE),

"down": (0, CELL\_SIZE),

"left": (-CELL\_SIZE, 0),

"right": (CELL\_SIZE, 0)

}

# Initialize screen

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Snake Game")

# Clock for controlling frame rate

clock = pygame.time.Clock()

# Initialize score

score = 0

font = pygame.font.SysFont(None, 35)

def reset():

global snake, snake\_direction, food\_pos, score

snake = [[100, 100], [80, 100], [60, 100], [40, 100]]

snake\_direction = "right"

food\_pos = get\_random\_food\_pos()

score = 0 # Reset score

move\_snake()

def move\_snake():

global snake\_direction, score

new\_head = snake[0].copy()

new\_head[0] += offsets[snake\_direction][0]

new\_head[1] += offsets[snake\_direction][1]

if new\_head in snake or new\_head[0] < 0 or new\_head[0] >= WIDTH or new\_head[1] < 0 or new\_head[1] >= HEIGHT:

reset()

else:

snake.insert(0, new\_head)

if not food\_collision():

snake.pop()

screen.fill(BG\_COLOR)

for segment in snake:

pygame.draw.rect(screen, SNAKE\_COLOR, pygame.Rect(segment[0], segment[1], CELL\_SIZE, CELL\_SIZE))

pygame.draw.rect(screen, FOOD\_COLOR, pygame.Rect(food\_pos[0], food\_pos[1], FOOD\_SIZE, FOOD\_SIZE))

# Display the score

display\_score()

pygame.display.flip()

clock.tick(1000 // DELAY)

def food\_collision():

global food\_pos, score

if get\_distance(snake[0], food\_pos) < CELL\_SIZE:

food\_pos = get\_random\_food\_pos()

score += 1 # Increase score when food is eaten

return True

return False

def get\_random\_food\_pos():

x = random.randint(0, (WIDTH - CELL\_SIZE) // CELL\_SIZE) \* CELL\_SIZE

y = random.randint(0, (HEIGHT - CELL\_SIZE) // CELL\_SIZE) \* CELL\_SIZE

return (x, y)

def get\_distance(pos1, pos2):

return ((pos2[0] - pos1[0]) \*\* 2 + (pos2[1] - pos1[1]) \*\* 2) \*\* 0.5

def change\_direction(new\_direction):

global snake\_direction

if new\_direction == "up" and snake\_direction != "down":

snake\_direction = "up"

elif new\_direction == "down" and snake\_direction != "up":

snake\_direction = "down"

elif new\_direction == "left" and snake\_direction != "right":

snake\_direction = "left"

elif new\_direction == "right" and snake\_direction != "left":

snake\_direction = "right"

def display\_score():

score\_text = font.render(f"Score: {score}", True, (255, 255, 255)) # White color for score

screen.blit(score\_text, (10, 10)) # Position the score at the top left

# Main loop

reset()

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.KEYDOWN:

if event.key == pygame.K\_UP:

change\_direction("up")

elif event.key == pygame.K\_DOWN:

change\_direction("down")

elif event.key == pygame.K\_LEFT:

change\_direction("left")

elif event.key == pygame.K\_RIGHT:

change\_direction("right")

move\_snake()

**Practical 4: Creating 2D Infinite Scrolling Background**

**Code:**

Steps:

1. Create Background Sprites:

* Design seamless backgrounds (e.g., a landscape or sky) in your favorite graphic design software.
* Import these sprites into Unity by dragging them into the Assets folder.

1. Set Up Scene:

* Create two GameObjects (e.g., Background1 and Background2) in your scene.
* Assign the background sprites to these GameObjects.
* Position Background1 at (0, 0) and Background2 at (width, 0), where width is the width of your background sprite.

1. Create Scrolling Script:

* Create a new C# script (e.g., BgScroller.cs) in your Assets folder.
* Use the following corrected code:

using UnityEngine;

public class BgScroller : MonoBehaviour {

public float speed = 0.5f;

private Material material;

void Start() {

// Get the material of the renderer

material = GetComponent<Renderer>().material;

}

void Update() {

// Calculate the offset

Vector2 offset = new Vector2(Time.time \* speed, 0);

material.mainTextureOffset = offset;

}

}

1. Attach Script:

* Drag the BgScroller script onto both Background1 and Background2 GameObjects.

1. Adjust Speed:

* In the Inspector, set the speed variable for both backgrounds to control how fast they scroll.

1. Test:

* Press play to see the infinite scrolling effect. You should see the backgrounds moving horizontally.

1. Enhancements:

* To create a parallax effect, you can add more background layers with different speeds. For instance, a foreground layer could scroll faster than a background layer to give depth to your scene.

**Practical 5: Create 2D Target Shooting Game**

**Cmd installation:**

**Pip install pygame**

**Code:**

import pygame

import random

# Screen

pygame.init()

width, height = 800, 600

screen = pygame.display.set\_mode((width, height))

pygame.display.set\_caption("Shooting Game")

# Colors

white = (255, 255, 255)

red = (255, 0, 0)

blue = (0, 0, 255)

# Character

character\_size = 50

character\_speed = 5

character = pygame.Rect(width // 2 - character\_size // 2, height - character\_size,

character\_size, character\_size)

# Bullets (triangles) properties

bullet\_size = 10

bullets = []

# Enemy (circle) properties

enemy\_radius = 20

enemies = []

# Initialize score

score = 0

# Clock for controlling frame rate

clock = pygame.time.Clock()

# Main game loop

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_SPACE:

bullet = pygame.Rect(character.centerx - bullet\_size // 2, character.top,

bullet\_size, bullet\_size)

bullets.append(bullet)

# Move bullets

for bullet in bullets:

bullet.y -= 10

if bullet.top < 0:

bullets.remove(bullet)

# Spawn enemies

if random.randint(1, 100) <= 2:

enemy\_x = random.randint(enemy\_radius, width - enemy\_radius)

enemy = pygame.Rect(enemy\_x - enemy\_radius, 0, enemy\_radius \* 2,

enemy\_radius \* 2)

enemies.append(enemy)

# Move enemies

for enemy in enemies:

enemy.y += 5

if enemy.top > height:

enemies.remove(enemy)

# Move character

keys = pygame.key.get\_pressed()

if keys[pygame.K\_LEFT]:

character.x -= character\_speed

if keys[pygame.K\_RIGHT]:

character.x += character\_speed

# Check for collisions

for bullet in bullets:

for enemy in enemies:

if bullet.colliderect(enemy):

score += 1

bullets.remove(bullet)

enemies.remove(enemy)

# Check for character collision with enemies

for enemy in enemies:

if character.colliderect(enemy):

running = False

# Clear the screen

screen.fill(white)

# Draw bullets

for bullet in bullets:

pygame.draw.polygon(screen, blue, [(bullet.left, bullet.bottom), (bullet.centerx,

bullet.top), (bullet.right, bullet.bottom)])

# Draw character

pygame.draw.rect(screen, red, character)

# Draw enemies

for enemy in enemies:

pygame.draw.circle(screen, red, enemy.center, enemy\_radius)

# Display score

font = pygame.font.Font(None, 36)

score\_text = font.render(f"Score: {score}", True, red)

screen.blit(score\_text, (10, 10))

# Update display

pygame.display.flip()

# Limit frame rate to 60 FPS

clock.tick(60)

# Game over display

font = pygame.font.Font(None, 72)

game\_over\_text = font.render("Game Over", True, red)

screen.blit(game\_over\_text, (width // 2 - game\_over\_text.get\_width() // 2, height // 2 -

game\_over\_text.get\_height() // 2))

pygame.display.flip()

# Wait for a few seconds before closing the game

pygame.time.wait(3000)

# Clean up

pygame.quit()

**Practical 6: Create Camera Shake Effect in Unity**

Following are the steps to create a Camera Shake Effect in Unity:

STEP 1 : Start Unity

STEP 2 : Create new project 2D. Add project name

STEP 3 : Add folders such as Sprites, Scenes, Scripts, etc

STEP 4 : Add Assets by right-click > Import Asset

STEP 5 : Import the Following assets : Background and UFO

STEP 6 : Drag the assets to the Hierarchy > Click and hold asset and drag under MainCamera

SImilarly, Drag and add the UFO under the background

STEP 7 : Adjust the MainCamera by changing the size

STEP 8 : Select the UFO from Hierarchy. To make the UFO layer appear above the Background, Add a sorting layer, and select the layer. Update the following settings for UFO. Add a circle collider and adjust the radius. Adjust gravity Scale

Rigidbodies are components that allow a GameObject to react to real-time physics. This includes reactions to forces and gravity, mass, drag and momentum. You can attach a Rigidbody to your GameObject by simply clicking on Add Component and typing in Rigidbody2D in the search field.

STEP 9 : Select the Background from Hierarchy. Add a box collider from Add component > Box Collider 2D. Do this step 4 times for each border of the box collider. Also, set the Shake Frequency which set the intensity of the Shake Effect

STEP 10 : To implement the camera shake effect, add a new script as follows:

On clicking on the new script, VS code Editor will open. Add the following code:

public Transform cameraTransform = default;

private Vector3 \_orignalPosOfCam = default;

public float shakeFrequency = default;

void Start()

{

\_orignalPosOfCam = cameraTransform.position;

}

void Update()

{

if (Input.GetKey(KeyCode.S))

{

CameraShake();

}

else if (Input.GetKeyUp(KeyCode.S))

{

StopShake();

}

}

private void CameraShake()

{

cameraTransform.position = \_orignalPosOfCam + Random.insideUnitSphere \* shakeFrequency;

}

private void StopShake()

{

cameraTransform.position = \_orignalPosOfCam;

}

The final output will result in the Camera Shake Effect on pressing the

key ‘S’:

**Practical 7: Create Snowfall Particle effect in Unity**

We can achieve this effect using Unity's Particle System component. Following is a step- by-step guide on how to create a basic snowfall particle effect:

1. Create a New Particle System:

• In the Unity Editor, select the GameObject where you want to add the snowfall effect.

• Go to the menu bar and select GameObject > Effects > Particle System.

This will create a new Particle System component attached to the selected GameObject.

2. Configure the Particle System:

• In the Inspector window, you'll see the Particle System component's settings.

Adjust these settings to create a snowfall effect:

• Duration: Set this to a value that suits the length of your snowfall scene. For an ongoing snowfall, you can set it to a high value or loop the system.

• Start Lifetime: This determines how long each snowflake particle will stay on screen. Set it to a value that makes the snowflakes fall for an appropriate amount of time (e.g., 5 seconds).

• Start Speed: Adjust the initial speed of the snowflakes. Typically, a small value like 1-5 units per second will work.

• Start Size: Set the size of the snowflakes. They should be small, like 0.05 to

0.2.

• Start Color: Choose a light blue or white color to resemble snow.

• Emission: Configure the rate at which particles are emitted. Set the rate to create a dense snowfall. For example, you might try starting with 100 particles per second.

• Shape: Choose "Cone" as the shape and adjust the angle and radius to control the area where snowflakes will spawn.

• Gravity Modifier: Apply a downward force (negative value) to simulate gravity. Use a small negative value, like -0.1, to make snowflakes fall gently.

3. Texture for Snowflakes:

• You can use a custom texture for your snowflakes. In the Particle System settings, under the Renderer module, set the Material to a particle material that uses a snowflake texture. Ensure the snowflake texture is set to have a transparent background.

4. Tweak Additional Settings:

• You can further enhance the effect by adjusting settings like Color Over

Lifetime, Size Over Lifetime, and Rotation Over Lifetime to add variation to the snowflakes.

5. Loop the Snowfall (Optional):

• If you want the snowfall to continue indefinitely, check the "Looping" option in the Particle System settings.

6. Play the Scene:

• Press the Play button in Unity to see your snowfall effect in action.

7. Optimization:

• Be mindful of performance. If you have a lot of particles, it can impact your game's performance. Adjust the particle count and other settings as needed for your project.

You can further refine and customize the effect by experimenting with different settings and textures to achieve the desired look for your game or scene.

**Practical 8: Develop Android Game with Unity**

**STEPS:**

Sure! Here’s a more concise version of your steps for developing an Endless Runner game in Unity:

### Practical 8: Develop Android Game with Unity

\*\*1. Set Up Unity Project\*\*

- Open Unity Hub, create a new 2D project named "EndlessRunner", and click "Create".

\*\*2. Import Required Assets\*\*

- Import sprites for the player character and obstacles from sources like OpenGameArt.

\*\*3. Create Game Objects\*\*

- In the Hierarchy, create:

- Player: Right-click > 2D Object > Sprite, name it "Player".

- Obstacle: Right-click > 2D Object > Sprite, name it "Obstacle".

- Ground: Right-click > 2D Object > Sprite, name it "Ground".

\*\*4. Set Up Player\*\*

- Select the Player object, add a Rigidbody2D, set Gravity Scale to 1, and add a BoxCollider2D.

\*\*5. Create Player Movement Script\*\*

- Right-click in the Project window, create a C# Script named "PlayerController" and add:

```csharp

using UnityEngine;

public class PlayerController : MonoBehaviour

{

public float jumpForce = 10f;

private Rigidbody2D rb;

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

void Update()

{

if (Input.GetButtonDown("Jump"))

{

rb.velocity = new Vector2(rb.velocity.x, jumpForce);

}

}

}

```

- Attach this script to the Player object.

\*\*6. Set Up Ground\*\*

- Scale the Ground object and add a BoxCollider2D.

\*\*7. Create Obstacle Script\*\*

- Create a script named "ObstacleSpawner" and add:

```csharp

using UnityEngine;

public class ObstacleSpawner : MonoBehaviour

{

public GameObject obstaclePrefab;

public float spawnTime = 2f;

public float spawnY = 0f;

void Start() { InvokeRepeating("SpawnObstacle", spawnTime, spawnTime); }

void SpawnObstacle()

{

float randomX = Random.Range(1f, 10f);

Vector2 spawnPosition = new Vector2(randomX, spawnY);

Instantiate(obstaclePrefab, spawnPosition, Quaternion.identity);

}

}

```

\*\*8. Set Up Obstacle Prefab\*\*

- Create a prefab from the Obstacle object, then delete it from the Hierarchy.

\*\*9. Add Obstacle Spawner\*\*

- Create an empty GameObject named "Spawner", attach the ObstacleSpawner script, and assign the Obstacle prefab.

\*\*10. Add Collision Detection\*\*

- Update PlayerController:

```csharp

void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.CompareTag("Obstacle"))

{

Debug.Log("Game Over!");

}

}

```

- Tag the Obstacle prefab as "Obstacle".

\*\*11. Build Settings for Android\*\*

- Go to File > Build Settings, select Android, and click "Switch Platform". Configure Player settings.

\*\*12. Test and Build\*\*

- Click Play to test the game and make adjustments. Then, click "Build" to generate the APK.

\*\*Final Touches\*\*

- Add sound effects, implement a scoring system, and consider menus and game restart functionality.

**Practical 9: Design and Animate Game Character in Unity.**

**STEPS:**

### Step 1: Import Character Model

1. **Download a Model**: Get a 3D character from the Unity Asset Store or Mixamo.
2. **Import**: In Unity, right-click in the Assets folder and select Import New Asset to add the model.

### Step 2: Set Up Character

1. **Create GameObject**: Drag the model into the Hierarchy.
2. **Add Components**: Add a Rigidbody and a Capsule Collider to the character.

### Step 3: Create Animations

1. **Use Mixamo**: Upload your character to Mixamo, select animations (Idle, Walk, etc.), and download.
2. **Import Animations**: Import these animations into Unity and set the Animation Type to Humanoid.

### Step 4: Animator Controller

1. **Create Controller**: Right-click in Assets, create Animator Controller, and name it (e.g., CharacterAnimator).
2. **Add States**: Drag your animation clips into the Animator window and create transitions.

### Step 5: Control with Script

1. **Create Script**: Right-click, create C# Script, name it CharacterController.
2. **Add Code:**

using UnityEngine;

public class CharacterController : MonoBehaviour

{

public float moveSpeed = 5f;

public Animator animator;

private Rigidbody rb;

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

void Update()

{

float moveHorizontal = Input.GetAxis("Horizontal");

float moveVertical = Input.GetAxis("Vertical");

Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);

rb.MovePosition(transform.position + movement \* moveSpeed \* Time.deltaTime);

animator.SetBool("isWalking", movement.magnitude > 0);

}

}

1. **Link Animator**: Attach the script to your character and link the Animator component.

### Step 6: Set Animation Parameters

1. **Add Parameter**: In the Animator, add a Bool parameter called isWalking.
2. **Configure Transitions**: Set transitions between Idle and Walk states based on isWalking.

### Step 7: Test

* **Play the Game**: Press Play and use WASD or arrow keys to move the character and see animations.

### Final Touches

* Add more actions like jumping and sound effects as needed.

**Practical 10: Create Intelligent enemies in Unity**

**STEPS:**

### Step 1: Set Up Enemy Character

1. **Import Model**: Import a 3D enemy model into Unity.
2. **Create GameObject**: Drag the model into the Hierarchy.
3. **Add Components**: Add a Rigidbody and a Capsule Collider.

### Step 2: Create Enemy AI Script

1. **Create Script**: Right-click in Assets, create C# Script, name it EnemyAI.
2. **Add Code**:

using UnityEngine;

public class EnemyAI : MonoBehaviour

{

public Transform player;

public float moveSpeed = 3f;

public float chaseRange = 5f, attackRange = 1.5f;

void Update()

{

float distance = Vector3.Distance(transform.position, player.position);

if (distance < attackRange) Attack();

else if (distance < chaseRange) Chase();

else Idle();

}

void Chase() => transform.position = Vector3.MoveTowards(transform.position, player.position, moveSpeed \* Time.deltaTime);

void Attack() => Debug.Log("Attacking the player!");

void Idle() => Debug.Log("Idle...");

}

Absolutely! Here’s a concise version for creating intelligent enemies in Unity:

### Step 1: Set Up Enemy Character

1. **Import Model**: Import a 3D enemy model into Unity.
2. **Create GameObject**: Drag the model into the Hierarchy.
3. **Add Components**: Add a Rigidbody and a Capsule Collider.

### Step 2: Create Enemy AI Script

1. **Create Script**: Right-click in Assets, create C# Script, name it EnemyAI.
2. **Add Code**:

csharp

Copy code

using UnityEngine;

public class EnemyAI : MonoBehaviour

{

public Transform player;

public float moveSpeed = 3f;

public float chaseRange = 5f, attackRange = 1.5f;

void Update()

{

float distance = Vector3.Distance(transform.position, player.position);

if (distance < attackRange) Attack();

else if (distance < chaseRange) Chase();

else Idle();

}

void Chase() => transform.position = Vector3.MoveTowards(transform.position, player.position, moveSpeed \* Time.deltaTime);

void Attack() => Debug.Log("Attacking the player!");

void Idle() => Debug.Log("Idle...");

}

### Step 3: Assign Player Target

1. **Create Player Object**: Add a player GameObject.
2. **Link Player**: Drag the player object to the Player field in the EnemyAI component.

### Step 4: Test Your Enemy AI

1. **Press Play**: Move the player close to the enemy to see it chase and attack.