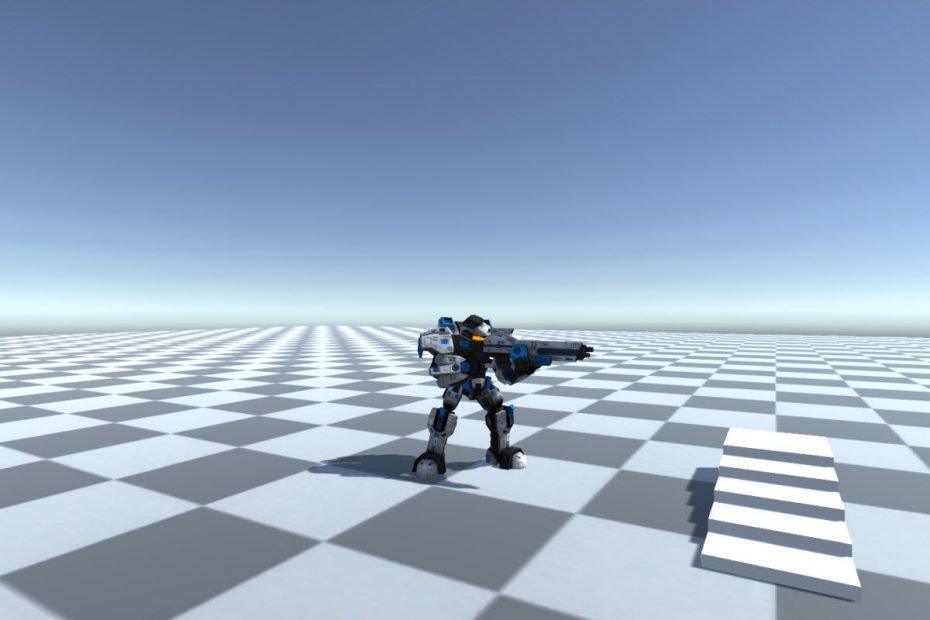
# Third-Person Camera Control in Unity using C#



**Solutions to Programming Assignments**

Shamim Akhtar

### Programming Assignment 1 - Implement Player Movement

In this programming assignment, we will implement the **PlayerMovement** script for our Player.

Double click and open the **PlayerMovement.cs** file.

Add the following variables.

[HideInInspector]

public CharacterController mCharacterController;

public Animator mAnimator;

public float mWalkSpeed = 1.0f;

public float mRotationSpeed = 50.0f;

In the Start method, we cache the CharacterController component in the mCharacterController variable.

void Start()

{

mCharacterController = GetComponent<CharacterController>();

}

Implement the Update method to handle inputs and apply the movement to the Player.

void Update()

{

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

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

float speed = mWalkSpeed;

if (Input.GetKey(KeyCode.LeftShift))

{

speed = mWalkSpeed \* 2.0f;

}

if (mAnimator == null) return;

transform.Rotate(0.0f, hInput \* mRotationSpeed \* Time.deltaTime, 0.0f);

Vector3 forward =

transform.TransformDirection(Vector3.forward).normalized;

forward.y = 0.0f;

mCharacterController.Move(forward \* vInput \* speed \* Time.deltaTime);

mAnimator.SetFloat("PosX", 0);

mAnimator.SetFloat("PosZ", vInput \* speed / 2.0f \* mWalkSpeed);

}

**Discuss in class how you can further refactor this code. Your tutor will guide you on refactoring.**

### Programming Assignment 2 – Track Third-Person With Player Height

Track third-person camera control with Player height. Add the Player height to the **Update** method.

public override void Update()

{

const float playerHeight = 2.0f;

Vector3 targetPos = mPlayerTransform.position;

targetPos.y += playerHeight;

mCameraTransform.LookAt(targetPos);

}

Click **Play** and view the behaviour.

**Discuss in class how you can further refactor this code. Your tutor will guide you on refactoring.**

### Programming Assignment 3 – Implement the Update Method for TPCFollow

In this programming task, you will implement some sections of the **Update** method for the **TPCFollow** class.

public override void Update()

{

// Now we calculate the camera transformed axes.

// We do this because our camera's rotation might have changed

// in the derived class Update implementations. Calculate the new

// forward, up and right vectors for the camera.

Vector3 forward = mCameraTransform.rotation \* Vector3.forward;

Vector3 right = mCameraTransform.rotation \* Vector3.right;

Vector3 up = mCameraTransform.rotation \* Vector3.up;

// We then calculate the offset in the camera's coordinate frame.

// For this we first calculate the targetPos

Vector3 targetPos = mPlayerTransform.position;

// Add the camera offset to the target position.

// Note that we cannot just add the offset.

// You will need to take care of the direction as well.

Vector3 desiredPosition = targetPos

+ forward \* GameConstants.CameraPositionOffset.z

+ right \* GameConstants.CameraPositionOffset.x

+ up \* GameConstants.CameraPositionOffset.y;

// Finally, we change the position of the camera,

// not directly, but by applying Lerp.

Vector3 position = Vector3.Lerp(mCameraTransform.position,

desiredPosition, Time.deltaTime \* GameConstants.Damping);

mCameraTransform.position = position;

}

### Programming Assignment 4 – Implement the TPCTopDown Camera Control

In this programming task, you will implement the **TPCTopDown** third-person camera control. **TPCTopDown** is a simple Top-Down camera mode where the camera looks down on the Player from an altitude. This mode will not use the **mRotationOffset** and the x and z values of the **mPositionOffset**. Give it a try.

Follow a similar line of thoughts as the previous few implementations.

Step 1: Derive a new **TPCTopDown** class from **TPCBase** class.

public class TPCTopDown : TPCBase

{

public TPCTopDown(Transform cameraTransform, Transform playerTransform)

: base(cameraTransform, playerTransform)

{

}

}

Step 2: Override the **Update** method

public class TPCTopDown : TPCBase

{

public TPCTopDown(Transform cameraTransform, Transform playerTransform)

: base(cameraTransform, playerTransform)

{

}

public override void Update()

{

}

}

Step 3: In the Update method, get the player position in a temporary variable **targetPos**.

public override void Update()

{

// For the topdown camera we do not use the x and z offsets.

Vector3 targetPos = mPlayerTransform.position;

}

Step 4: Add the **CameraPositionOffset**.y value to the **targetPos**.

public override void Update()

{

// For the topdown camera we do not use the x and z offsets.

Vector3 targetPos = mPlayerTransform.position;

targetPos.y += GameConstants.CameraPositionOffset.y;

}

Step 5: Use **Lerp** to move the camera to this **targetPos**.

public override void Update()

{

// For the topdown camera we do not use the x and z offsets.

Vector3 targetPos = mPlayerTransform.position;

targetPos.y += GameConstants.CameraPositionOffset.y;

Vector3 position = Vector3.Lerp(mCameraTransform.position, targetPos, Time.deltaTime \* GameConstants.Damping);

}

Step 6: Rotate the camera to look down.

public override void Update()

{

// For the topdown camera we do not use the x and z offsets.

Vector3 targetPos = mPlayerTransform.position;

targetPos.y += GameConstants.CameraPositionOffset.y;

Vector3 position = Vector3.Lerp(mCameraTransform.position, targetPos, Time.deltaTime \* GameConstants.Damping);

mCameraTransform.position = position;

mCameraTransform.rotation = Quaternion.Euler(90.0f, 0.0f, 0.0f);

}