31263 / 32004 Game Development Lab Week 5

Getting Started

1. Download the corresponding week's zip file from the Lab section of UTSOnline.
2. Unzip the project folder and open it in Unity. If there are any warnings about difference in versions, just continue. If this causes any red errors in the console once the project opens, notify the tutor.
3. Within the Weekly folders there are image and executable files starting with "Status…". These files give you a preview of what is expected for each point percentage below.
    a. If you are on Mac and the Status-100Percent App file won't run, hold control and click (right click) then select Open, if there is a security warning, acknowledge it an press open again.
    b. If you are running the Windows executable on the lab computers, you need to copy the entire executable folder into Windows(C:)/Users/<student number>/AppLockerExceptions. From there you can double run the .exe file.

Tasks

Points	Requirements

40%	Scene Set-up
	• Drag the Floor prefab from the Project Window into the Hierarchy Window and make sure its position and rotation are zeroed.
	• Drag the Player prefab into the Hierarchy Window and make sure its position and rotation is zeroed and scale is set to 1.
	  ○ Change the Y position to 0.12 to have the character standing on top of the floor.
	• Create an Empty Gameobject in the Hierarchy View called BackgroundMusic
	  ○ Attach an AudioSource component.
	  ○ In the Inspector Window with the BackgroundMusic gameobject selected, next to AudioClip to see valid audio clips in the Project Window. Select the BackgroundMusic audio clip.
	  ○ Make sure "Play on Awake" and "Loop" are ticked in the Inspector Window.

50% (P)	CharacterMovement Script Variables and Methods
	• Create a new script called CharacterMovement
	• In CharacterMovement:
	  ○ Create the following member variables
	  ▪ private Vector3 movement;

	▪ private float movementSqrMagnitude; ○ Create the following methods ▪ void GetMovementInput() {} ▪ void CharacterPostion() {} ▪ void CharacterRotation() {} ▪ void WalkingAnimation() {} ▪ void FootstepAudio() {} ○ In Update(), call all of the above methods in the above order. ○ Delete the Start() method • Attach the CharacterMovement component to the Player gameobject.
60% (P)	Movement Vector • Go to the Input Manager window and make sure that: ○ The Horizontal Axis has Positive Button is set to "d" and Negative Button set to "a", Gravity 3 and Sensitivity 3. ○ The Vertical Axis has Positive Button is set to "w" and Negative Button set to "s", Gravity 3 and Sensitivity 3. • In the GetMovementInput() method of the CharacterMovement script: ○ Set the x value of the movement vector variable to the value of the Horizontal input axis (tip: see Input.GetAxis(….) ○ Set the z value of the movement vector variable to the value of the Vertical input axis. ○ Store the squared magnitude of the movement vector in the movementSqrMagnitude variable (tip: see Vector3.sqrMagnitude) ○ Print the movement vector out to the console.
70% (C)	Position and Rotation • In CharcterMovement, create a public variable called walkSpeed with a default value of 1.5f. • In CharacterPosition() in the CharacterMovement script: ○ Move the current gameobject in the direction specified by the movement vector and Transform.Translate(). ○ Press play and use the a,s,w,d to move. ○ Combine the movement vector with the walkSpeed variable and make it frame rate independent when moving the game object. ○ Press play and move around • In CharacterRotation() ○ Set the Player gameobject's rotation using the direction specified by the movement vector (without the walkSpeed variable or frame rate independence). (tip: see Quaternion.LookRotation(…)) ○ Press play and move around • Notice that the position update is all wacky now with the rotation ○ This is because the Transform.Translate() defaults to local space translation, which is affected by an object's rotation. ○ Set Transform.Translate() to use Space.World.

	○ Press play and move around. • Notice the warnings and that the character pops back to facing forward after stopping its movement. ○ Only do the rotation if the movement vector is not a zero vector. ○ Press play and move around. • Notice that the Player moves faster when moving diagonally than moving just vertically or just horizontally. ○ This is because a vector of (1,0,0) (e.g. moving positively in the x-axis) has a magnitude of 1. However the vector (1,0,1) (e.g. moving diagonally in positive x and z) has a magnitude of about 1.4. ○ In GetMovementInput(), after setting the x and z variables of the movement vector but before the sqrMagnitude of the movement vector is stored: ▪ Use Vector3.ClampMagnitude(…) to clamp the magnitude of the movement vector to 1.0f.
80% (D)	<u>Animation</u> • Comment out the console printing of the movement vector. • Add an Animator component to the Player gameobject. ○ Press the circle button next to Controller to see valid AnimatorControllers in the ProjectWindow. Assign the MovementAnimator controller. ○ Press the circle button next o Avatar and select PlayerAvatar. ○ Make sure "Apply Root Motion" is unticked, "Update Mode" is normal, and "Culling Mode" is set to "Cull Update Transforms". • Go to the menu option "Window->Animator" and dock the Animator Window somewhere such that you can see the Game View and the Animator Window at the same time. • Select the Player gameobject to see the state machine of the MovementAnimator • Select the Parameters tab in the Animator Window and create a new parameter called MoveSpeed. ○ Select the transition line that is pointing between the states PlayerIdle to Walking. ▪ In the Inspector, untick "Has Exit Time" ▪ In the Inspector, under Conditions, add a new condition that MoveSpeed is Greater than 0.1. ○ Select the transition line that is pointing between the states Walking to PlayerIdel. ▪ In the Inspector, untick "Has Exit Time" ▪ In the Inspector, under Conditions, add a new condition that MoveSpeed is Less than 0.1. • In CharacterMovement, create a public variable to store the animator of the Player gameobject. • In the WalkingAnimation() method ○ Set the MoveSpeed parameter of the Player's MovementAnimator to have the value of the movementSqrMagnitude. • Press play.

90% (HD)	Footstep Audio
	• Add an AudioSource component to the Player gameobject in the Hierarchy Window.
	  ○ Set the AudioClip of the AudioSource to be Footstep01 (found under Audio Clips -> FX in the Project Window).
	  ○ Set the Pitch of the AudioSource to be 1.1 and make sure Play On Awake and Loop are unticked.
	• In CharacterMovement, create a public AudioSource variable called footstepSource and assign the newly added audio source to it.
	• In the CharacterMovement script, create a public array of AudioClips call footstepClips.
	  ○ In the Hierarchy Window, set this array to a size of 2 and assign Footstep01 and Footstep02 from the Project Window.
	• In CharacterMovement, also create a public AudioSource reference to the background music audio source.
	• In the FootstepAudio() method:
	  ○ If the movementSqrMagnitude is greater than 0.3f and the audio source isn't already playing a clip then:
	  ▪ Associate the other clip in the footstepClips array with the audio source.
	  ▪ Set the volume of the audio source to the movementSqrMagnitude variable.
	  ▪ Play the audio source
	  ▪ "Duck" the background music volume by reducing it to 0.5f.
	  ○ Otherwise, if movementSqrMagnitude is less than or equal to 0.3f and the footstepAudioSource is playing then:
	  ▪ Stop the footstepAudioSource from playing.
	  ▪ Return the background music to 1.0f;
	• Press play (with headphones or speakers)

100% (HD)	Collisions
	• Player's Collider:
	  ○ Select the Player gameobject. Notice that there is currently no collider on the player.
	  ▪ It is not actually interacting with the floor of the game, it is just that our animation and positioning makes it look that way.
	  ○ Add a CapsuleCollider component to the Player.
	  ▪ Give this a Center of (0,1,0) and a Height of 2
	  ▪ In the Scene View, with the Player selected, you should see a green capsule that roughly encompasses the character model.
	  ○ Add a Rigidbody component
	  ▪ Set the Constraints of this Rigidbody to freeze the x, y and z rotation. This will stop our character from falling / rolling over due to gravity and the nature of our rounded bottom capsule collider
	• Trigger:
	  ○ Create a new Cube primitive GameObject and place it at position (-3,1.5,0)
	  ▪ Rename this to "Trigger Box"
	  ○ Notice that this Cube already has a BoxCollider component on it.

        - Make this Collider a Trigger.
    - In the CharacterMovement script, use the OnTriggerExit() method so that when the player STOPS colliding with the trigger, the following is printed to the console "Trigger Exit: <GameObject name> : <GameObject Position>" where <GameObject name> is the name of the gameobject that the trigger is attached to and <GameObject Position> is the current position of that gameobject. E.g. the output may be "Trigger Enter: Box : (-2, 1.5, 0)"
    - Press play and walk through the trigger.
- Rigidbody Collider:
    - Create a new Cube primitive GameObject and place it at position (3,1.5,0)
        - Rename this to "Physics Box" and add a Rigidbody component to it.
        - Add a Rigidbody component to this Physics Box
    - In your CharacterMovement script, use the OnCollisionEnter() method so that when the player BEGINS colliding with the box, the following is printed in the console "Collision Enter: <GameObject name> : <Last Contact Point>" where <Last Contact Point> is the position of one of the touching points of the collision (see Collision.contacts, just use the first element of the array)
        - Note the difference between the method parameters for OnTrigger…() methods and OnCollision…() methods.
- Static Collider:
    - Create another Cube primitive, place it (0,1.7,4) and with scale (5,3,1)
    - Rename this to "Wall".
        - Create new tag called "Impassable" and give the Wall gameobject this tag.
        - Notice that the attached BoxCollider is affected by the scale of the gameobject so it is automatically stretch to fit the scale of the wall.
        - Also notice that without a Rigidbody component, this Wall is now a Static Collider
    - In CharacterMovement, use the method OnCollisionStay() to detect when a gameobject with the tag "Impassable" has been collided with and print out "Collision Stay: <GameObject name>" in the console.
    - Press play and walk into the wall
- Raycasting
    - Create a new Cube primitive and place it at (0, 1, -4)
        - Rename this "Enemy" and give it a new tag called "Freeze"
    - Create a new method called RaycastCheck() that returns a bool
        - In this method, perform a Raycast from the Player's position plus (0,0.5,0) (because otherwise the raycast would shoot from the feet and not be high enough for the enemy), facing forward from the Player, and with a distance of 5 units.
        - If the raycast hits something
            - Print to the console "Raycast Hit: <GameObject name>" with the name of the first gameobject that was hit by the ray
            - Check if that gameobject has has the tag "Freeze" and return true, otherwise return false
    - In Update()
        - Call RaycastCheck()

	▪ If it returns false, perform the actions as normal ▪ If it returns true, prevent the position, animation, and audio from being updated, but allow the rotation to occur as normal. ▪ The resulting behaviour is that the player cannot move towards the Enemy (see Status-100Percent) • CLEAN, EFFICIENT, ELEGANT CODE!
Extra	Show your tutor the above and submit that. This section is just for you to test and learn from. Important for Assessment 4: • Remember the restriction that you must only use kinematic Rigidbodies? • In this example, your Player and Physics Box are not kinematic, so these would break the rules of Assessment 4. • Press Play. Notice which objects the player and raycasts do and do not collide with and which objects the Raycast collides with. • For Assessment 4, you should plan to make use of Triggers and Raycasts to simulate collisions instead of using non-kinematic Rigidbodies or look into how you force kinematic rigidbodies to register collisions (but beware of the cost of doing this)

Submission

When you complete the activity to the grade threshold that you want, you then need to:

1. Complete the "Status-StudentSubmission.txt" file in the highest level of the project folder.
2. Remove all other "Status-…" files and folders to reduce the size of your project.
3. Zip the entire project folder.
4. Re-name the zip file to **"[student ID]-LabWeek[week number].zip".**
5. Submit the zip file to UTSOnline for the associated link for this week in the Lab **before Monday 9am of the following week**.
6. Failure to follow any of these could result in a 0% mark for that week.
7. You will also **demo your submission to your tutor** at the **start of the following week's lab.**

If you finish the activity early, show it to your tutor before you submit it on UTSOnline so they can help you make some final corrections and mark it at that time.