18ITD54

Hall Ticket Number:

IV/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

November,2022 **Information Technology**

Seventh Semester

Introduction to Game Development

Time: Three Hours **Maximum:** 50 Marks

Answer Question No. 1 Compulsorily. (10X1 = 10 Marks)
*Answer **ANY ONE** question from each Unit.* (4X10=40 Marks)

1.	a)	Define a sprite?	CO1,L2	
	b)	Illustrate different types of Primitive Colliders in Unity.	CO1,L3	
	c)	How an UI Element is created.	CO2,L3	
	d)	Describe the use of Update() method in the HealthBar script?	CO2,L2	
	e)	Explain the use of Spawn Points?	CO2,L1	
	f)	Discuss the tools in the Tile Palette.	CO1,L4	
	g)	Explain Wander Variables?	CO3,L2	
	h)	Describe MEL script?	CO3,L2	
	i)	What is an UV mapping?	CO4,L1	
	j)	Discuss Pixel to Units?	CO4,L4	
		Unit - I		
2.	a)	Illustrate about Colliders and Tags and Layers.	CO1,L2	**5M**
	b)	List and explain about Animation Parameters.	CO1,L3	**5M**
		(OR)		
3.	a)	Describe about Painting with Tile Palettes.	CO1,L1	**5M**
	b)	Explain about Layer-Based Collision Detection.	CO1,L3	**5M**
		Unit - II		
4.	a)	Demonstrate about Anchor and Adjusting the Anchor Points.	CO2,L4	**5M**
	b)	Create the Prefabs and Build the Slot Script.	CO2,L2	**5M**
		(OR)		
5.	a)	What are Singletons and explain about Creating the Singleton.	CO2,L3	**5M**
	b)	Explain the steps in Building a Spawn Point Prefab.	CO2,L4	**5M**
		Unit - III		
6.	a)	Demonstrate about The Wander Algorithm.	CO3,L3	**5M**
	b)	Illustrate Enemy Walk Animation	CO3,L2	**5M**
		(OR)		
7.	a)	Explain Setting up a scene in Maya.	CO3.L1	**5M**
	b)	Write about exporting FBX file from Maya and configuring it after import in unity.	CO3,L2	**5M**
		Unit - IV		
8.	a)	Discuss about Setting materials' names in Maya.	CO4,L3	**5M**
	b)	Explain about Texture atlases 2D gamming.	CO4,L2	**5M**
		(OR)		
9.	a)	Describe about Coding the Boolean-based transitions.	CO4,L2	**5M**
	b)	Explore the operations required to set up a 2D image as a Sprite type in Unity.	CO4,L1	**5M**

1

a) A Sprite is a two dimensional image (2D) that can be used in 2D games of unity.

b) An approximation of the objects shape using a type of Collider called a “Primitive Collider” is also less processor intensive. There are two types of Primitive Colliders in Unity 2D: Box Collider 2D and Circle Collider 2D.

c) UI Elements are game objects that encapsulate specific, commonly needed user-interface functionality such as buttons, sliders, labels, a scroll bar, or input field. Unity allows developers to build out custom user-interfaces quickly by offering premade UI Elements instead of requiring that the developer create them from scratch.

d) The purpose of update method in health bar script is to monitor the lives of player, and to maintain the score values.

e) We want to be able to create or “spawn” characters—a player, or an enemy, at a specific location in the scene. If we’re spawning enemies, then we may also want to spawn them at a regular interval as well.

f) Tools in Tile Palette include Select, Move Selection, Paintbrush, Box Fill, Pick New Brush, Erase and Flood Fill.

g) Wander variables will be used to set the speed at which the Enemy pursues the Player, the general wandering speed when not in pursuit, and the current speed that will be one of the previous twopeeds

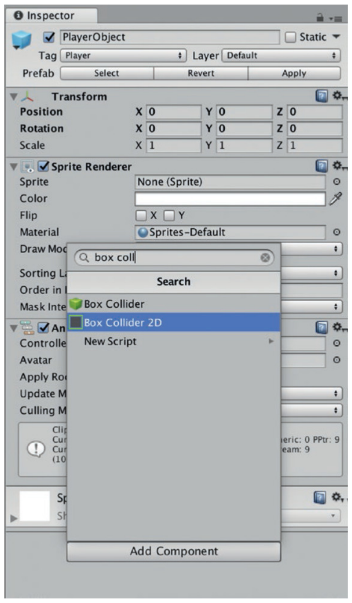
h) Maya Embeded Language used in 3D modeling of the object.

i) UV mapping is the 3D modeling process of projecting a 3D model's surface to a 2D image for texture mapping

j) The Pixel to Units property defines how many pixels in the sprite are there in one unit (1 unit is 1 meter in Unity) in the game world. We don't need to edit this property; you can leave it at its default value of 100. Remember, however, that this gets important if you plan to use physics in your game.

2. a) Illustrate about Colliders and Tags and Layers. CO1,L2 **5M**
   b) List and explain about Animation Parameters. CO1,L3 **5M**

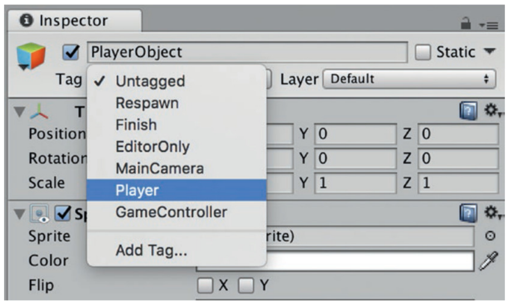
a) Colliders are added to GameObjects and used by the Unity Physics Engine to determine when a collision has taken place between two objects.



An approximation of the objects shape using a type of Collider called a "Primitive Collider" is also less processor intensive. There are two types of Primitive Colliders in Unity 2D: Box Collider 2D and Circle Collider 2D.

Select the PlayerObject and then select the Add Component button in the Inspector. Search for and select "Box Collider 2D" to add a Box Collider 2D to the PlayerObject as seen in Figure.

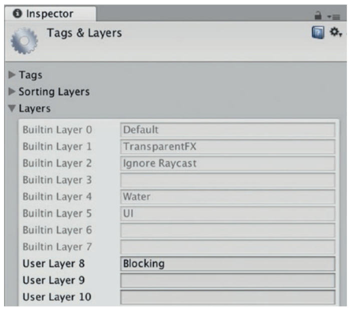
Tags:



Tags allow us to label GameObjects for easy reference and comparison while our game is running.

Select the PlayerObject. Under the Tag drop-down menu on the very top left of the Inspector, select the Player tag to add a tag to our PlayerObject, as seen in Figure.

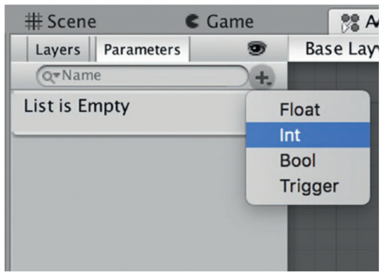
Layers :



Layers are used to define collections of GameObjects. These collections are used in collision detection to determine which layers are aware of each other and thus can interact. We can then create logic in a Script to determine what to do when two GameObjects collide.

As we can see in Figure 3-22, we want to create a new "User Layer" called "Blocking". Type "Blocking" into the User Layer 8 field. Now select the PlayerObject again to view its properties in the Inspector. Select the Blocking Layer we just created from the drop-down menu (see Figure) to add our PlayerObject to that Layer.

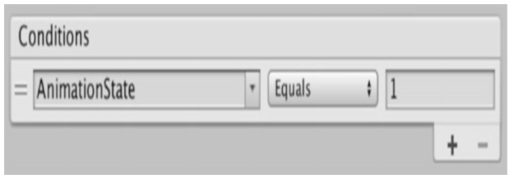
b) Animation Parameters are variables defined in the Animation Controller and are used by scripts to control the Animation State Machine. Animation Parameter that we create in our Transitions and in our MovementController script to control the PlayerObject and make her walk around the screen.



If during gameplay this condition is ever true, then the Animator will transition to that Animation State and the corresponding Animation Clip will play. Because this Animator component is attached to the PlayerObject, the Animation Clips will be displayed at the Transform component's location in the Scene. We use a script to set this Animation Parameter condition to be true and trigger the state transition.

On the bottom of the inspector, you'll see an area titled, "Conditions." Click the plus symbol in the lower-right and select AnimationState, Equals, and enter 1 (Figure 3-41). We've just created a condition that says: if the Animation Parameter called "AnimationState" equals 1, then enter this Animation State and play the Animation. This is how we will trigger state changes from the script we're about to write.

```
if (movement.x > 0)
{
animator.SetInteger(animationState, (int)
CharStates.walkEast);
}
```



If movement along the x axis is greater than 0, then the player is pressing the key to go right. SetInteger() takes two parameters: a string, and an int value. The first value is the Animation Parameter (Figure) we created earlier in the Unity Editor called, "AnimationState."

We've conveniently stored the name of this Animation Parameter in a string called "animationState" in our script and we'll pass that as the first parameter to SetInteger().

The second parameter to SetInteger() is the actual value to set for AnimationState. Because each value in our CharStates enum corresponds with an int value.

3.	a)	Describe about Painting with Tile Palettes.	CO1,L1	**5M**
	b)	Explain about Layer-Based Collision Detection.	CO1,L3	**5M**

a) Select the paintbrush tool from the Tile Palette, and then select a tile from the Tile Palette. Use the paintbrush to paint on the Tilemap in the Scene view. If you make a mistake, you can hold down the Shift key to use the tile paintbrush as an eraser. When the paintbrush is selected, you can hold down Option (Mac)/Alt (PC) + the left mouse button to pan around the Tilemap.
Use Option (Mac)/Alt (PC) + left mouse button to pan around the Tile Palette, left-click to select a tile, and left-click and drag to select a group of tiles.
Tools in the Tile Palette.
Select—Select areas of the grid or specific tiles
Move Selection—Move around selected areas
Paintbrush—Select a tile from the Tile Palette then use the Paintbrush to paint on the Tilemap
Box Fill—Paint a filled area using the actively selected tile
Pick New Brush—Use an existing tile from the Tilemap as a new brush
Erase—Remove a painted tile from the Tilemap (Shortcut: hold down Shift)
Flood Fill—Fill an area with the actively selected tile
The Import Settings in the Inspector should be set to the following:
Texture Type: Sprite (2D and UI)
Sprite Mode: Multiple
Pixels Per Unit: 32
Filter Mode: Point (no filter)
Ensure the Default button is selected at the bottom and set Compression to: None
Now go into the Sprite Editor by clicking its respective button in the Inspector. Press the Slice button in the upper-left and then Grid by Cell Size from the Type menu. Use 32 × 32 for the X and Y pixel size. We are reusing the Sprite slicing techniques.

b) Layers are used to define collections of GameObjects. Collider components that are attached to GameObjects on the same Layer will be aware of each other and can interact. We can create logic based off of these interactions to do things such as pick up objects. There's also a technique to make Collider components on different layers aware of each other. This approach uses a Unity feature called Layer-Based Collision Detection.

We'll use this feature so that the player and coin colliders, despite being on different layers, are aware of each other. We'll also configure things so that the enemy colliders aren't aware of the coins because they can't pick them up. If two colliders aren't aware of each other, they won't interact. The enemy will walk right through the coins without picking them up. To see this feature in action, first we need to create and assign Layers to the relevant GameObjects.

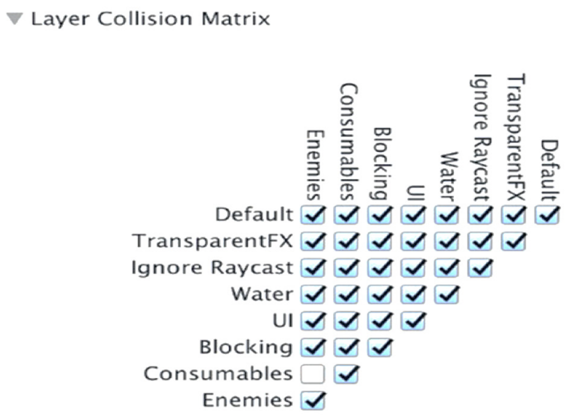
create new Layers

1. Select the CoinObject in the Hierarchy
2. In the Inspector, select the Layer drop-down menu
3. Select: "Add Layer"
4. Create a new Layer called: "Consumables"
5. Create another Layer called: "Enemies"

The Consumables layer will be used for items such as coins, hearts, and other objects that we want the player to consume. The Enemies layer will be used for enemies.

Go to the Edit menu ➤ Project Settings ➤ Physics 2D. Look at the Layer Collision Matrix on the bottom of the Physics2DSettings view. This is where we'll configure the layers to allow the enemies to walk right through coins, power-ups, and whatever else we choose.

By checking and unchecking boxes in the intersection of a column and a row, we can configure which layers are aware of each other and will interact. Colliders on objects from different layers can interact if the box at the intersection of the two layers is checked.



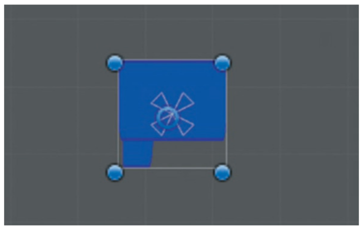
▼ Layer Collision Matrix

	Enemies	Consumables	Blocking	UI	Water	Ignore Raycast	TransparentFX	Default
Default	✓	✓	✓	✓	✓	✓	✓	✓
TransparentFX	✓	✓	✓	✓	✓	✓	✓	
Ignore Raycast	✓	✓	✓	✓	✓	✓		
Water	✓	✓	✓	✓	✓			
UI	✓	✓	✓	✓				
Blocking	✓	✓	✓					
Consumables	☐	✓						
Enemies	✓							

Unit II

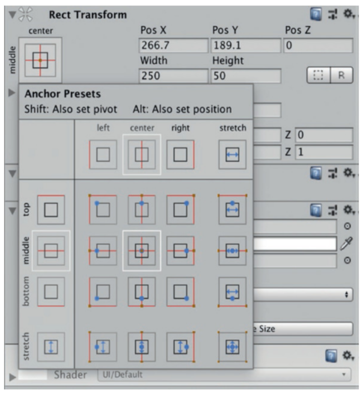
4. a) Demonstrate about Anchor and Adjusting the Anchor Points. CO2,L4 **5M**
   b) Create the Prefabs and Build the Slot Script. CO2,L2 **5M**

a) star-like symbol in the center of Figure 6-2 and in Figure. This symbol is made up of four small

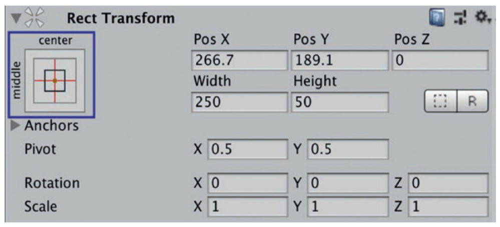


triangular handles representative of a property specific to UI Elements called the Anchor Points.

Each corner of a UI Element will always be rendered with the same distance relative to its respective Anchor Point. This ensures that UI Elements are always in the same location, scene to scene. The ability to set a consistent distance between Anchor Points and UI Elements becomes especially helpful when the size of the Canvas scales along with the size of the screen.



Adjusting the Anchor Points:
Select the Background object. In the Rect Transform component, press on the Anchor Presets icon highlighted in Figure.



Rect Transform

center	Pos X	Pos Y	Pos Z
middle	266.7	189.1	0
	Width	Height	
	250	50	
▶ Anchors			
Pivot	X 0.5	Y 0.5	
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Pressing on the icon should give you a menu of Anchor Presets, as seen in Figure. By default, the middle-center is selected. This explains why the Background object's Anchors appear in the middle of the Canvas.
We want to anchor the Health Bar relative to the top-right corner of the screen at all times. Select the Anchor Preset setting in the column titled, "right" and the row titled, "top". You'll see a white box surrounding the selected Anchor Preset, as seen in Figure. Press the Anchor Preset icon to close it and notice how the Anchor Points have now moved to the top-right corner of the Canvas.

We've left a little bit of space between the health bar and the corner of the Canvas, and the Anchor Points are all collected in the top-right. Regardless of how much we scale the screen size, the health bar will always be situated in that exact spot.

b) Create a new GameObject in the project view and rename it to CoinObject. Select the four individual coin sprites from the sliced heart-coin-fire spritesheet and drag them onto the CoinObject to create a new animation.

Rename the animation clip to "coin-spin" and save it to the Animations ➤ Animations folder. Rename the generated Controller, "CoinController" and move it to the Controllers folder.

In the Sprite Renderer component, click the little dot next to the "Sprite" form and select a Sprite to use when previewing this component in the Scene view.

Create a new Sorting Layer by selecting the Sorting Layer drop-down menu in the Sprite Renderer component, click "Add Sorting Layer", then add a new layer called, "Objects" between the Ground and Characters layers.

Select the CoinObject again and set its Sorting Layer to: Objects.

To allow the player to pick up coins, we need to configure two aspects of the CoinObject:

1. Some way to detect that the player has collided with the coin
2. A custom Tag on the coin that says it can be picked up

5.	a)	What are Singletons and explain about Creating the Singleton.	CO2,L3	**5M**
	b)	Explain the steps in Building a Spawn Point Prefab.	CO2,L4	**5M**

a) Singletons are used in situations where your application needs one and only one instance of a particular class to be created for the lifetime of the application. Singletons can be helpful when you have a single class that provides functionality used by several other classes in your game, such as coordinating game logic in a Game Manager class.

Although Singletons can provide a unified access point to functionality, this also means that the Singleton holds globally accessible values with indeterminate state. Any piece of code in your entire game can access and set the data inside the Singleton.

Create a new GameObject in the Hierarchy and rename it: "RPGGameManager". Then create a new folder under Scripts called: "Managers". Create a new C# script called "RPGGameManager" and move it to the Managers folder. Add the script to the RPGGameManager object. Open the RPGGameManager script in Visual Studio and use the following code to build out the RPGGameManager class:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class RPGGameManager : MonoBehaviour
{
        public static RPGGameManager sharedInstance = null;
        void Awake()
        {
            if (sharedInstance != null && sharedInstance != this)
            {
            Destroy(gameObject);
            }
            else
            {
            sharedInstance = this;
            }
        }
        void Start()
        {
        SetupScene();
        }
public void SetupScene()
{
// empty, for now
}
}
```

b)
```
using UnityEngine;
public class SpawnPoint : MonoBehaviour
{
public GameObject prefabToSpawn;
public float repeatInterval;
        public void Start()
        {
                if (repeatInterval > 0)
                {
                InvokeRepeating("SpawnObject", 0.0f, repeatInterval);
                }
        }
        public GameObject SpawnObject()
        {
                if (prefabToSpawn != null)
                {
                return Instantiate(prefabToSpawn, transform.
                position, Quaternion.identity);
                }
      return null;
      }
}
```

This could be any prefab that we want to spawn once or at a consistent interval. We'll set this to be the player or enemy prefab in the Unity Editor. If we want to spawn the prefab at a regular interval, we'll set this property in the Unity Editor. If the repeatInterval is greater than 0 then we're indicating that the object should be spawned repeatedly at some preset interval. Because the repeatInterval is greater than 0, we use InvokeRepeating() to spawn the object at regular, repeated intervals. The method signature for InvokeRepeating() takes three parameters: the method to call, the time to wait before invoking the first time, and the time interval to wait between invocations.

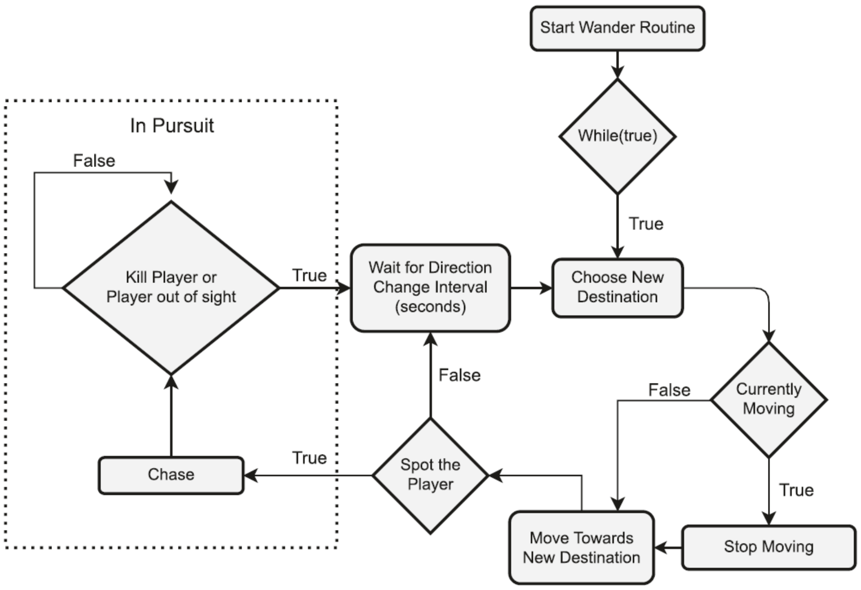
SpawnObject() is responsible for actually instantiating the prefab and "spawning" the object. The method signature indicates that it will return a result of type: GameObject, which will be an instance of the spawned object. We set the access modifier of this method to: public, so that it can be called externally.

Check to make sure we've set the prefab in the Unity Editor before we instantiate a copy to avoid errors. Instantiate the prefab at the location of the current SpawnPoint object. There are a few different types of Instantiate methods used to instantiate prefabs. The specific method we're using takes a prefab, a Vector3 indicating the position, and a special type of data structure called a Quaternion. Quaternions are used to represent rotations, and Quaternion. identity represents "no rotation." So we instantiate the prefab at the position of the SpawnPoint and without a rotation. If the prefabToSpawn is null, then this Spawn Point was probably not configured properly in the editor. Return null;

Unit III

6.	a)	Demonstrate about the Wander Algorithm.	CO3,L3	**5M**
	b)	Illustrate Enemy Walk Animation	CO3,L2	**5M**

a)



Select the Enemy prefab and drag it into the scene to make our lives easier. Select the EnemyObject and add a CircleCollider2D component to it. Check the Is Trigger box on the Circle Collider and set the radius of the collider to be: 1. This Circle Collider represents how far the Enemy can "see." In other words, when the Player's collider crosses the Circle Collider, the Enemy can see the Player.

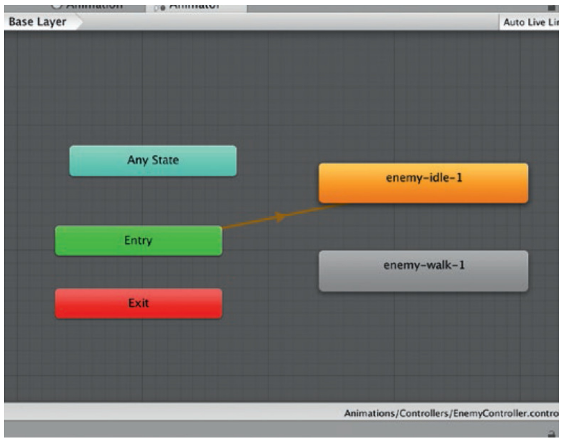
Ensure that whatever GameObject we attach the Wander script to in the future has a Rigidbody2D, a CircleCollider2D, and an Animator.

```
public IEnumerator WanderRoutine()
{
        while (true)
        {
        ChooseNewEndpoint();
                if (moveCoroutine != null)
```

```
                                        {
                                        StopCoroutine(moveCoroutine);
                                        }
                              moveCoroutine = StartCoroutine(Move(rb2d, currentSpeed));
                              yield return new WaitForSeconds(directionChangeInterval);
                              }
                }
```

We want the Enemy to wander indefinitely, so we'll use while(true) to loop through the steps indefinitely. The ChooseNewEndpoint() method does exactly what it sounds like. It chooses a new endpoint but doesn't start the Enemy moving toward it. We'll write this method next. Check if the Enemy is already moving by checking if moveCoroutine
is null or has a value. If it has a value then the Enemy may be moving, so we'll need to stop it first before moving in a new direction. Stop the currently running movement Coroutine. Start the Move() Coroutine and save a reference to it in moveCoroutine. The Move() Coroutine is responsible for actually moving the Enemy. We'll write it shortly. Yield execution of the Coroutine for directionChangeInterval seconds, then start the loop over again and choose a new endpoint.

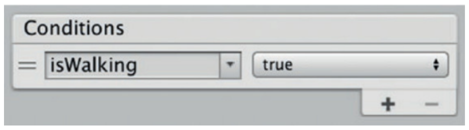
b) Select the Enemy prefab then open the Animation window



If the Idle state is the default state, it will be colored Orange. If it isn't the default state, right-click on the "enemy-idle-1" state and select: Set as Layer Default State. As you can see, the enemy-walk-1 state exists, with an animation clip, but isn't being used at the moment. The plan is to create an Animation Parameter and use that parameter to switch between the idle and walking state.

Click on the plus-symbol in the Parameters section of the Animator and select Bool . Name this parameter: "isWalking", Our Wander script will use this parameter to switch the Enemy's animation state between idle and walking. To keep things simple, the walking animation will serve as a stand-in for running, when in pursuit of the Player, as well as leisurely walking. Right-click on enemy-idle-1 state and select: Make Transition. Create a transition between the idle state and the walking state. Then create another transition between the walking state and the idle state.

Click on the transition from enemy-walk-1 to enemy-idle-1 and configure it. Set up each transition to use the Animation Parameter: isWalking, that we just created. Set the condition: isWalking to true, in the transition from enemy-idle-1 to enemy-walk-1.

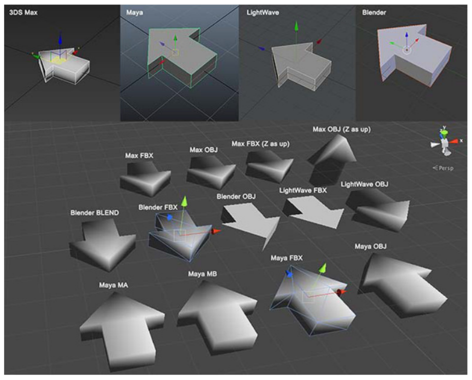

Conditions
isWalking true

Set isWalking to false, in the enemy-walk-1 to enemy-idle-1 transition. To use the new animation state, we just need to change isWalking to true, in our Move() Coroutine.

7.	a)	Explain Setting up a scene in Maya.	CO3.L1	**5M**
	b)	Write about exporting FBX file from Maya and configuring it after import in unity.	CO3,L2	**5M**

a) The first point to keep in mind when setting up a scene in Maya is that the standard unit in Maya is 1 cm, while the standard unit in Unity is 1 m So, whenever you export an FBX file from Maya into Unity, Unity scales it down to 0.01 percent of its original size.

Another very relevant point is that Maya and Unity are affected by strange kinds of idiosyncrasies that put them on opposing sides, with regard to what left or right and front or bottom mean. This is not something that only happens between Maya and Unity. Many 3D software disagree about the concepts of right and up.



As you can see, the red arrow, representing the left-right axis in the 3D world, may point to the left or right on different software or file formats, and the green and blue axes may switch to alternatively point to the forward or upward directions. With Maya and Unity, what happens is that the front in Maya is the back in Unity. So you model the front of a character in Maya, and when you import it into Unity, it shows its back. How do we deal with this? There's more than one option available, and turning the camera by 180 degrees in Unity is not the only one.

b) Exporting FBX file from Maya and configuring it after import in unity

1. Open the scene with the model in Maya.
2. From the outliner panel, select the root node of your model. Be sure that the model is at the 0,0,0 position with 0,0,0 rotation.
3. In the top menu window, navigate to File | Export Selection and the Maya exporter panel will open.
4. Be sure that FBX export is selected from the drop-down menu at the bottom of the panel.
5. Put a name you like in the File name field.
6. From the Options... panel on the right-hand side, let's examine the first group of settings. Edit the General Options, Reference Options, and Include Options tab
7. Now we can move to the next group of settings. In File Type Specific Options, make sure that the Include and Geometry settings are configured
8. Next comes the animation-related group of properties. Since we are not importing animations with an FBX file, unflag the Animation option entirely. This action will disengage all the following properties
9. Unflag Cameras, Lights, and Embed Media; we don't need any of them either.
10. Flag Input Connections in the Connections tab.

Unit IV

8.	a)	Discuss about Setting materials' names in Maya.	CO4,L3	**5M**
	b)	Explain about Texture atlases 2D gamming.	CO4,L2	**5M**

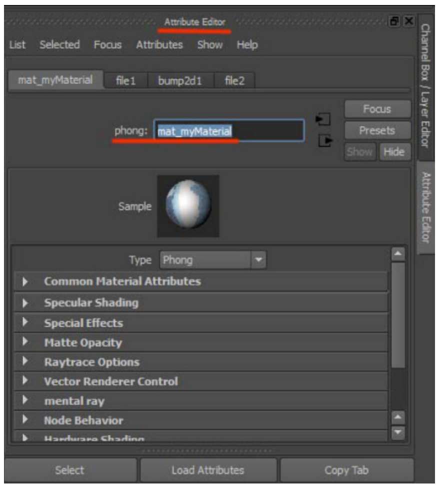
a) Setting materials' names in Maya

When Unity imports a model, it also imports the materials linked to the model in the 3D editor. By default, the materials are named based on the names of the textures used in the 3D editor to build them.

If you remember from the previous chapter, upon importing a model into Unity, we set an option in the Model tab to pick material names from a model's materials, instead of the texture names.

We adopt this solution because it helps keep our project clean and has each asset named correctly. As our goal is to have the materials named with a meaningful convention that can help us keep materials and their textures well separated.

1. Open Maya or your 3D editor of choice, then open the model file in the editor.
2. From the main menu, navigate to Window | RenderingEditors | Hypershade to open Hypershade, the panel where materials and their properties are displayed.
3. Select a material in Hypeshade to display its properties in the Attribute Editor panel.



phong: mat_myMaterial

4. In the text field, type a name you want to assign to that material. Refer to the following screenshot to check whether you are in the right panel/text field:

When this model is exported into Unity and the materials' naming property is correctly set in Inspector, a Materials folder will be automatically created to store the materials named with the convention we set in Maya.

A drawback of this practice is that you end up with two instances of each material: one named tex_TexName, which is automatically created by Unity, and another named mat_matName, which is the name you get from applying the settings to the Inspector window.

b)

An arrangement of images and parts of images that are all nicely deployed to better fill the whole space available. This is technically called a texture atlas. A texture atlas is in fact a way to optimize memory management as you feed your project with 2D textures. Instead of having an image for any differently colored mesh or mesh part, an artist can align several chunks on a single texture, saving memory that would otherwise go wasted.

Actually, in 2D gaming, texture atlases are extremely useful for backgrounds. To improve the perspective illusion of 2D static backgrounds, it is a good practice to actually build them as is done in theaters, by putting several screens on stage at different depths (distance from the audience), each with its own piece of background.

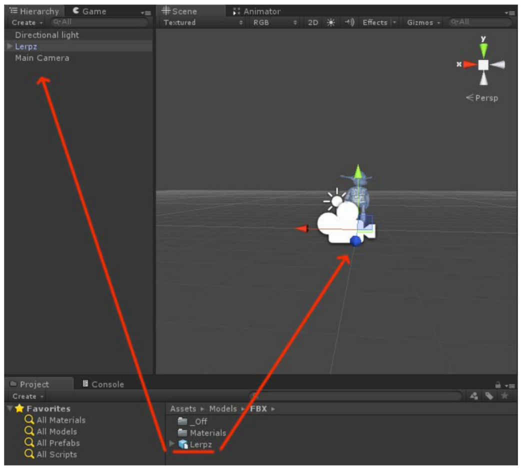
Likewise, artists create pre-rendered backgrounds by putting several images on different planes (or quads, more likely) that they scatter around on the game stage. On each of those quads (quads, by the way, are very simple, single-faced 3D shapes, made of a rectangular plane divided into two triangles), artists put an image selected from a texture atlas, which contains all the images required to actually build up that background.

9.	a)	Describe about Coding the Boolean-based transitions.	CO4,L2	**5M**
	b)	Explore the operations required to set up a 2D image as a Sprite type in Unity.	CO4,L1	**5M**

a) With the transition between Idle and Jump configured in the Animator window, we can script a piece of code to trigger it.

1. Access the Scripts folder in your Project panel and double-click on the newly created script to open it in Monodevelop, the default Unity script editor.
2. Let's begin by creating an Animator type variable to store the reference to the character animator and add the following line at the top of the script: private Animator charAnimator;

3. Get inside the *Start()* function; here we need to address the charAnimator variable we created to the actual animator that we will attach to the character. We do that by adding the following line to the script:
charAnimator=this.GetComponent<Animator>();
4. Now we define an event to trigger the Jump clip. In the Update() function, add the following lines to intercept the pressing of the bar and set tJump:
if(Input.GetKey(KeyCode.Space)){
charAnimator.SetBool ("bJump", true);
}



5. Now, we need to add this script and the Animator controller to the character, but to do that, we first need to complete two steps:

The first is to instantiate the character itself into the game scene, which we haven't done yet. Select Character in the project panel and drag it into the hierarchy or directly into the game scene.

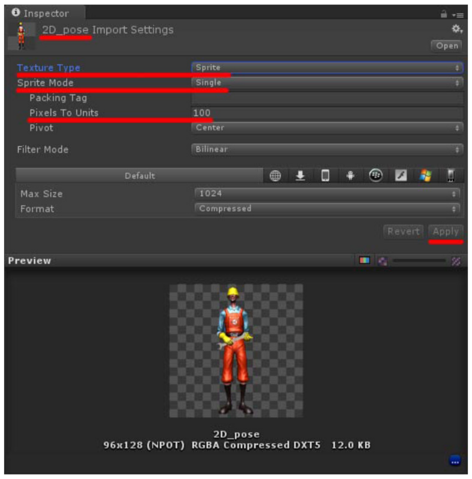
The second step is to add the Animator component to the character in the scene. You can do that from the top menu or in Inspector (we go for the first solution). With Character selected in the scene, go to the menu and navigate to Component | Miscellaneous | Animator.

6. Now drag Packt_Animator from the Animator folder in the project panel in the Controller field of the Animator component into the Inspector window, which should display None in the Controller option field right now.

7. Next, drag Char_Animator from the Scripts folder onto the character in the scene. If you did things right, with the character selected, you should see both components displayed in the Inspector panel.

8. Press the Play button to start your game scene in the editor. If you press the spacebar, the character jumps, playing the correct clip.

b) operations required to set up a 2D image as a Sprite type in Unity.


2D_pose Import Settings

Texture Type Sprite
Sprite Mode Single
Packing Tag
Pixels To Units 100
Pivot Center
Filter Mode Bilinear
Default
Max Size 1024
Format Compressed

2D_pose
96x128 (NPOT) RGBA Compressed DXT5 12.0 KB

1. Import the 2D_Sprites package we have provided with the contents of this book to your project. If you don't remember how to do it, right-click on the Sprites folder you just created and select Import Package/Custom Package... from the menu.

2. The package contains two images named 2D_pose and 2D_walk. Ensure that both are selected in the Import panel and then hit Import.

3. Once the importing process is complete, access the Sprites folder and select the first picture, 2D_pose.

4. Go to the Inspector window and set the Texture Type property to Sprite.

5. A Sprite Mode property should appear, right below Texture Type. Select Single from the drop-down menu. The Pixel to Units property defines how many pixels in the sprite are there in one unit (1

unit is 1 meter in Unity) in the game world. We don't need to edit this property; you can leave it at its default value of 100. Remember, however, that this gets important if you plan to use physics in your game.

6. The base sprite for our 2D character is ready. Create new Empty GameObject in Scene and name it 2DCharacter.

7. Now you can drag the sprite named 2D_pose onto 2DCharacter in the scene. Once you do so, a Sprite Renderer component is automatically attached to 2DCharacter, and a Default-Sprite material is created, with the 2D_pose sprite as the Material entry of the component

Signature of the faculty

Prof. N Sivaram Prasad
Head of the Department
Information Technology