UECS3273 Programming with Game Engines Tutorials

Tutorial 1

Questions

Textbook: Sams Teach Yourself Unity® Game Development in 24 Hours, 2nd edition
Hour 1. Introduction to Unity
Hour 2. Game objects

1. A game engine is a software framework designed for the creation and development of video games. Provide 4 key motivations for using a game engine to develop a game, as opposed to developing a game from scratch using low-level graphics libraries.

2. Explain in detail the concept of a game engine as a form of middleware, and describe how this is functionally achieved.

3. State 4 main categories of items that are typically created in the Project View.

4. Briefly differentiate between the term assets and objects in the Unity engine.

5. List the 5 main views that are found in the Unity main IDE window and briefly describe their purpose

6. State and describe the purpose of the two main types of cameras in Unity

7. Distinguish between the specific details and purpose of the world coordinate system and local coordinate system in the Unity environment.

8. Explain in detail how the order of transformation affects the 3 primary transformations of translation, rotation and scaling.

9. Explain in detail how transforms work on nested objects with respect to their parents and their respective coordinate systems.

Answers

1. Answer:

a) Economizes the process of game development by reusing the same game engine to create different games
b) Make it easier to port games to multiple platforms
c) Provide tools such as an IDE to enable rapid development of games
d) Provides all the core functionality needed to develop a game: thus reducing costs, complexities, and time-to-market

2. The game engine functions as middleware by allowing a game to be developed and run on various platforms (game consoles, PC, mobile devices). This is achieved by 3D rendering systems in game engines that are built upon a lower-level graphics API such as Direct3D or OpenGL which abstracts the graphics processing unit (GPU) or video card.

3. Materials, Models, Prefabs, Scripts

4. An asset is any item that exists as a file in the assets folder (such as textures, meshes, sound files, scripts, and so on). Game objects do not result in the creation of a corresponding file

5. Answer

a) Project: Contains all the assets for a project (files, scripts, textures, models, and so on) organized appropriately into folders

b) Hierarchy view: This shows all the items in the current scene instead of the entire project.

c) Inspector view: enables the viewing of all the properties of a currently selected item in the Project or Hierarchy view

d) The Scene View: provides a view of the various game objects involved in the construction of the game

e) Game View: plays the game within the editor by simulating the action of the various objects shown through the view of the current active camera in the Scene view.

6. Answer:

a) Standard game object camera (in scene by default – Main Camera): determines what is seen in the Game View

b) Imaginary camera: determines what is seen in the Scene view

7. Answer:

World coordinate system
- Applies to all objects in the scene
- There is only a single x, y, and z axis and origin shared by all objects

Local coordinate system
- Unique to each object
- Used in reference when performing transformation in relation to parent game objects

8. Answer:

a) Translation: Any changes applied after it won't be affected

b) Rotation:
- Rotation changes the orientation of the local coordinate system axes
- Any translations applied after a rotation would cause the object to move along the new axes.

c) Scaling:
- Scales the size of the local coordinate grid
- Translation on the newly scaled coordinate grid will be scaled as well

9. Answer

a) A child object's coordinate system is relative to the local coordinate system of the parent
b) A nested child object's transform is relative to that of the parent object
c) Transforms applied to parent object cause same effect in child, without actually changing the transforms of the child.

Tutorial 2

Questions

Textbook: Sams Teach Yourself Unity® Game Development in 24 Hours, 2nd edition
Hour 3. Models, Materials and Textures
Hour 4. Terrain
Hour 5. Environment

1. Explain in detail what a mesh is.

2. State the important properties of a triangle that make it very useful in graphics processing.

3. Explain the problem that unwrapped textures can help to solve, and how they solve this problem.

4. Explain the purpose of a shader.

5. Explain the details and function of a heightmap

6. State 6 guidelines that are useful when applying a terrain texture.

7. Explain the warping effect and the purpose for it within the Unity platform

8. Describe in detail what a skybox is and how it is used.

9. Explain what needs to be done when a character controller is added to the current scene, and state the reason for this action.

Answers

Source: Material Texture, Terrain Environment

1. Meshes are series of interconnected triangles that contains all the vertex information that defines the 3D shape of an object, to allow quick processing

2. Answer:
- With a single triangle, need only one more vertex to make another.
- Using strips of triangles, it is possible to model any 3D object.

3. Answer:
- Sometimes textures get applied incorrectly around a 3D model

- Unwrap is a map that shows exactly how a flat texture will wrap back around a model and is common for intricate models

4. A shader determines what kind of textures can be applied on a material and how these texture looks like when it is applied

5. Details and purpose:

- A heightmap is a grayscale 8-bit image (256 shades of grey) that contains elevation information
- The darker shades represent low points, and the lighter shades represent high points which guides Unity on how to render the terrain elevation correctly

6. 6 guidelines

a) Try to make the pattern repeatable.
b) Try to make the texture larger
c) Make the texture square.
d) Try to make the texture dimension a power of two (4,8,16)
e) Keep effects subtle.
f) Elements fade from one element to another without harsh transitions

7. Warping effect:

a) Models that are far away from the viewer, rendered as a much lower-quality billboard
b) When viewer gets closer, model becomes more higher-quality
c) Efficiencies of rendering engine to make project run faster.

8. A skybox is a cube consisting of six flat sides enclosing the world of the scene view. It has inward-facing textures to make it look round and infinite. Can use different skyboxes (or no skyboxes) on different cameras to make the world look different to different viewers.

9. Answer:

- Both Main Camera and the camera in the character controller has an audio listener component, which represents the 3D sounds heard from the player's perspective
- Therefore, need to remove one listener component from either one camera so that there is only a single sound source

Tutorial 3

Questions

Textbook: Sams Teach Yourself Unity® Game Development in 24 Hours, 2nd edition
Hour 6. Lights and Cameras
Hour 9: Scripting Part 2

1. State the 3 different types of light, and describe each of them in detail
2. What are halos and why do they occur ?
3. What are layers in the Unity environment ? Describe 2 situations that they are commonly used for.
4. Describe in detail what a script is and its purpose.
5. Briefly explain how implementation of different types of light affects the placing of the camera.
6. `Start()` and `Update()` are two main inherited methods from the `MonoBehaviour` class; which all script classes by default derive from. Describe in detail the purpose of these two methods, and when they are called.
7. Describe 3 possible approaches to find and manipulate a remote Game Object within a script
8. Describe 3 possible approaches to find and manipulate a remote script from a different script.

Answers

1. Answer

a) Point light
- All light is emitted from one central location out in every direction.
- most common type of light for illuminating interior areas

b) Spotlight
- Light begins at a central spot and radiates out in a cone
- illuminates whatever is in front of them

c) Directional light
- Illuminates the entire scene

- light radiates evenly in parallel lines across a scene

2. Halos are glowing circles that appear around lights in foggy conditions. They occur because light is bouncing off small particles around the light source

3. Layers are groupings of similar objects so that they can be treated a certain way. 2 situations:

- items viewable by only certain cameras or illuminated by only certain lights
- collision to occur only between certain types of objects

4. A script is a class that is an attached to a Game Object as a Component. After attachment, the script can be executed to control the behaviour of the various Game Objects in the Scene view.

5. Different light types provide a different degree of light intensity and coverage which in turn affects the amount and coverage of illumination of the game objects. The main camera will then need to be placed in an appropriate position to ensure that poorly illuminated objects can still be seen when the game is played.

6. Purpose of the methods:

    - The Start() method is called once when the GameObject the script is attached to is first used in the Scene view. It is primarily used to initialize the member variables in your script.

    - Update() method is called after each frame depicting the scene is displayed (many times per second). It is useful for detecting input or rendering changing output to the screen

7. 3 possible approaches

a) Drag the GameObject into the public variable in the script
b) Put name of the GameObject in Hierarchy view into variable and use Find method
c) Create tag for one or more GameObjects and put into variable and use FindWithTag or FindGameObjectsWithTag method

8. 3 possible approaches

a) Drag the GameObject holding the remote script into the public variable of the current script
b) Use FindObjectOfType / FindObjectOfTypes method call on the remote script name
c) Get a reference to the GameObject that the remote script is attached to, and then use GetComponent to reference the script itself

Tutorial 4

Questions

Textbook: Sams Teach Yourself Unity® Game Development in 24 Hours, 2nd edition
Hour 9: Scripting Part 2
Hour 10. Collision
Hour 12. Prefabs

1. It is intended to attach a script to a Cube object in the Scene view that will move it along the vertical (y-axis) when the keys corresponding to the Vertical Y input axes are depressed, and along the horizontal (x-axis) when the keys corresponding to the Vertical X input axes are depressed. Show the code statements in the `Update()` method of this script.

2. A Cylinder object is in the scene view as shown in Figure 2. It is required to rotate the Cylinder about the X axis by 10 degrees while simultaneously translating along the X axis by 0.05 units. Write code within the `Update()` method of a script that will be attached to the Cylinder to perform this action.

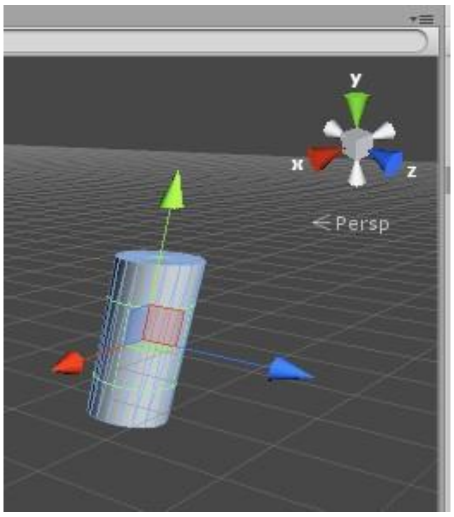


Figure 2

3. As an extension from Q2, it is now required to keep rotating the Cylinder about its local X axis; while at the same time moving this rotating Cylinder along the circumference of an imaginary circle on the XY plane. Describe briefly how this can be accomplished using a Plane object, and write the code for the `Update()` methods in the scripts to be attached to both these objects in order to accomplish the described action.

4. It is intended to attach a script `CubeScript` to a Cube object in the Scene view that will be used to scale the Cube in an equal magnitude along all 3 axes (X, Y and Z). An instance variable `scaleIncrement` is used for the purpose of controlling the magnitude of this scaling. Below are the expected results when the following keys are pressed:

   - D or A is pressed: the scale values for all 3 axes are either incremented (if D is pressed) or decremented (if A is pressed) by the current value in `scaleIncrement`. The Cube will then be scaled to these new values so that its appearance changes in the Scene view.
   - W or Q is pressed: the value contained within the variable `scaleIncrement` is increased (if W is pressed) or decreased (if Q is pressed) by 0.005 respectively. The Cube remains the same in the Scene view.
   - T is pressed: the current value in `scaleIncrement` is displayed to the console window

   Implement `CubeScript` to accomplish the above functionality, ensuring that you include the declaration and / or initialization of all public / local variables used.

5. A Cube and Sphere game objects are placed in the Scene view. Both these objects have being renamed to user-defined names in the Hierarchy view (for e.g. Dog and Cat). It is intended to create a script `MoveSphere` that will be attached to the Cube that causes the following action:

   - Each time the T key is pressed, the Sphere will appear at one of 4 possible different positions in the Scene view (see Figure 5a) – Figure 5d).
   - The Sphere will initially be placed at the 1st position. It will appear at the 2nd position when T is pressed, at the 3rd position when T is pressed again and so on. If the T key is pressed while the Sphere object is at the last (4th) position, it will appear back again at the first position and this cyclic appearance order is repeated.
   - The coordinates for these 4 positions are (2, 0.5, -2), (2, 0.5, 2), (-2, 0.5, -2) and (-2, 0.5, 2) respectively.
   - The Cube is initially placed at (0, 0, 0)

   Implement `MoveSphere` to accomplish the above functionality. Make use of the `enumerated` data type in your code. As the Sphere will have a user-defined name, this name should be provided to the script via a `public` variable that the user can specify in the Inspector panel.

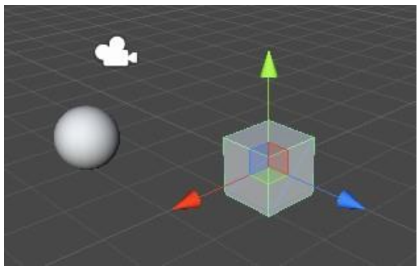


Figure 5a)

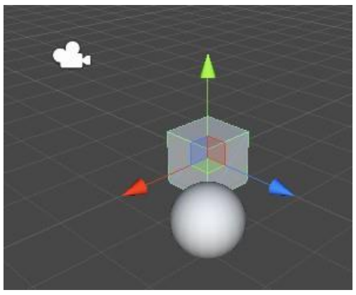


Figure 5b)

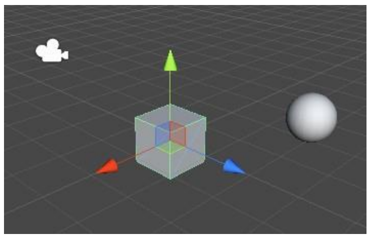


Figure 5c)

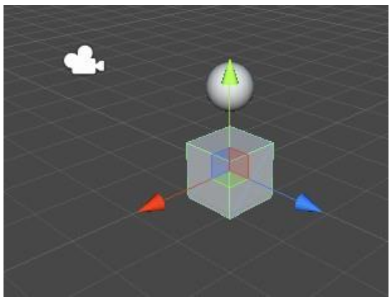


Figure 5d)

6. Write an alternative script solution for Q5 which does not involve the use of an enumerated type. The solution should use one or more arrays to store the coordinates for the 4 possible positions that the Sphere can appear at.

7. A Cube and Sphere game objects are placed in the Scene view. Both these objects can have customized names in the Hierarchy view. The Sphere has two scripts (`MainScript`, `SecondScript`) attached to it, and is placed at location (4, 0, -5). The Cube has a single script (`CubeScript`) attached to it, and is placed at location (-4, 0, -5). Note that both the Cube and Sphere both have the same y and z coordinate values, and only differ in terms of their x coordinate values.

   - `SecondScript` has a single public global variable within it (`float incrementValue`)
   - `CubeScript` has a method with the signature:

```
 public void MoveToNewPosition(Vector3 newPosition, float
incrementValue)
```

   - The function of `MainScript` is to obtain the value within `incrementValue` and call the `MoveToNewPosition` method within `CubeScript`, passing as parameters the current position of the Sphere and the value of `incrementValue`
   - The function of `CubeScript` is to translate the Cube on the X-axis in increments given by `incrementValue` until the Cube is approximately 2 units away from the Sphere on the X axis

   Implement `CubeScript` and `MainScript` to accomplish the above functionality. You may include additional methods (if necessary) in the scripts beyond the standard `Update()` and `Start()` methods.

8. Create any 3 Sphere objects and place them randomly within the Scene view. Create another 3 Cube objects and place them randomly within the Scene view as well. Tag all the Sphere objects with the tag `SphereTag` and the Cube objects with the tag `CubeTag`. A script `MoveObjects` is attached to the Main Camera. Write code for `MoveObjects` so that when the Q key is held down, all the Cubes start rotating about the Y axis; and when the W key is held down, all the Spheres start moving up on the Y axis. To perform this functionality, you will need to use the `GameObject.FindGameObjectsWithTag("XXX");` statement which will return an array of `GameObjects` which have the tag `"XXX"` applied to them.

9. A Sphere and Cube game objects are placed in the Scene View. Both these objects have scripts attached to them (`SphereScript` and `CubeScript`). The `SphereScript` has code within it to move the Sphere around based on user input, while the `CubeScript` has code to perform specific functionality whenever any other game object collides with the Cube. Implement code within `CubeScript` to make the Cube rotate clockwise around the Y axis when the Sphere contacts with it, and to rotate anti-clockwise around the Y axis when the Sphere withdraws from contact with it.

10. A Sphere, Cylinder and Cube are placed in the Scene View. All these 3 objects have scripts attached to them (`SphereScript`, `CylinderScript` and `CubeScript`). Both `SphereScript` and `CylinderScript` have code within them to move the Sphere and Cylinder around based on user input, while `CubeScript` has code to perform specific functionality whenever any other game object collides with the Cube. Implement `CubeScript` to accomplish the functionality below:

    - The Cube rotates around the Y axis if the Sphere enters it from the top
    - The Cube rotates around the Z axis if the Sphere enters it from the side
    - The Cube stops rotating the moment the Sphere moves out from contact with it after having previously entered it
    - The Cylinder disappears from the Scene View the moment it enters the Cube

11. The scenario in Q10 is repeated. Change the code in `CubeScript` to make the Cube rotate around the Y axis or Z axis in proportion to the speed (magnitude of translation) at which the Sphere collides with it. This means the angle of rotation will be greater if the speed of the colliding Sphere is higher at the point of collision, and vice versa. The script controlling the movement of the Sphere is shown below.

```
public class SphereScript : MonoBehaviour {

     public float hVal, vVal;

     void Start () {     }

     void Update () {

          hVal = Input.GetAxis("Horizontal") / 10;
          vVal = Input.GetAxis("Vertical") / 10;
          transform.Translate(hVal, vVal, 0);

     }
}
```

12. An empty game object is placed in the scene view at (0, 0, 0) and a script `GenerateCubes` is attached to it. A Cube prefab is created and a script `MoveCube` is attached to the prefab. `GenerateCubes` has a `public` variable `prefab` that is initialized to refer to the Cube prefab from the Inspector panel. Complete the code for the both these scripts so that the following functionality is achieved:

- Pressing any of the following keys: Q, W, E or R causes a new Cube object to appear in the game view.
- The first Cube object appears at the same location as the empty object. The second Cube object appears 2 units to the left of the first Cube, the third Cube appears 2 units to the left of the second Cube and so on.
- If the Q key was pressed, the new Cube that appears will rotate on the Y axis
- If the W key was pressed, new Cube that appears will rotate on the Z axis
- If the E key was pressed, the new Cube that appears will move up on the Y axis
- If the R key was pressed, the new Cube that appears will move down on the Y axis

13. An empty game object is placed in the scene view at (6, 0, 0) with a script `GenerateScript` attached to it. A Cube is placed at (-6, 0, 0), scaled to (1, 6, 3) and renamed to Wall. A Rigidbody component is added to it and the `Use Gravity` option is unchecked. The `isTrigger` property for the Box Collider of the Wall is checked. Three prefabs are created based on the Cube, Sphere and Capsule game objects. `GenerateScript` has 3 `public` variables that are initialized to these prefabs. Each of these prefabs have the scripts `CubeScript`, `SphereScript` and `CapsuleScript` attached to them respectively. Implement all these 4 scripts so that the following functionality is achieved:

    - When the Q, W or E key is pressed, a Cube, Sphere or Capsule will appear. The Cube will appear 2 units above the empty game object, while the Sphere or Capsule will appear 2 units below it.
    - These objects will start moving towards the Wall after appearing
    - When the Cube collides with the wall it will stop moving and starting rotating about the Y axis
    - When the Sphere collides with the wall, it will disappear
    - When the Capsule collides with the wall, it will start moving backwards from the Wall, while the Wall moves backwards from it (i.e. both these objects move away from each other).

14. The scenario in Q13 is repeated. This time however, all of the prefabs will have a single script, `MoveScript`, attached to them as shown below. The Wall has a single script, `WallScript`, attached to it while the empty game object still has `GenerateScript` attached to it with the same implementation as in Q12. Implement `WallScript` so that the same functionality in Q12 is achieved.

```
public class MoveScript : MonoBehaviour {

    public float rotateY = 0f;
    public float moveX = -0.1f;
    void Start () {

    }

    void Update () {

         transform.Rotate (0f, rotateY, 0f);
         transform.Translate (moveX, 0f, 0f);

    }
}
```

Answers

1. Answer:

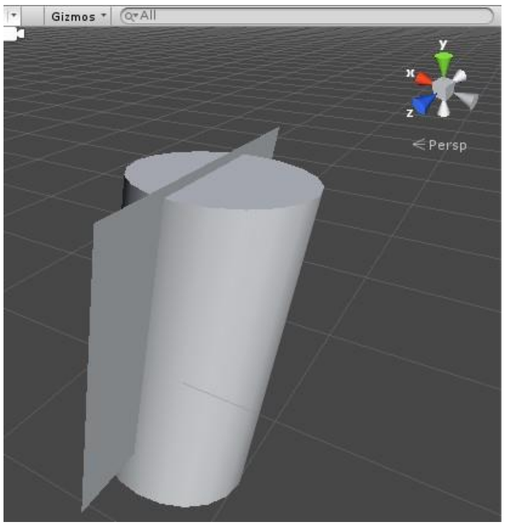
```
         float hVal = Input.GetAxis("Horizontal");
         float vVal = Input.GetAxis("Vertical");
         transform.Translate(hVal, vVal, 0);
```

2. Answer:

```
    void Update () {
         transform.Rotate (10f, 0, 0);
         transform.Translate (0.05f, 0, 0);
    }
```

3. Answer:

- Create a Plane object that intersects the cylinder on the yz plane. Nest the Cylinder within the Plane in the Hierarchy view so that the Plane is now the parent object, while the Cylinder is the child object.



- Create and attach a script for the Cylinder so that it continues to rotate on its local x-axis.
- Create and attach a script for the Plane so that it traces the circumference of an imaginary circle on the XY plane.

Script for Cylinder

```
     void Update () {
               transform.Rotate (10f, 0, 0);
     }
```

Script for Plane

```
     void Update () {
          transform.Rotate (0, 0, 1f);
          transform.Translate (0.05f,0, 0);
     }
```

4. Answer:

```
public class CubeScript  : MonoBehaviour {

     // this must be public so that it can be changed from the
inspector view
     public float scaleIncrement = 0;
     private float scaleX, scaleY, scaleZ;


     // Use this for initialization
     void Start () {
          // necessary to initialize this to 1,
          // as 1 represents the actual normal size prior to
          // any increment value being applied
          scaleX = scaleY = scaleZ = 1f;
     }

     void Update () {

          if (Input.GetKeyUp (KeyCode.W))
               scaleIncrement += 0.005f;

          if(Input.GetKeyUp (KeyCode.Q))
               scaleIncrement -= 0.005f;

          if(Input.GetKeyUp (KeyCode.T))
               print ("Scale increment : " + scaleIncrement);

          if (Input.GetKeyUp (KeyCode.D)) {
               scaleX += scaleIncrement;
               scaleY += scaleIncrement;
               scaleZ += scaleIncrement;
               transform.localScale  =  new  Vector3  (scaleX,
scaleY, scaleZ);
          }

          if (Input.GetKeyUp (KeyCode.A)) {
               scaleX -= scaleIncrement;
               scaleY -= scaleIncrement;
               scaleZ -= scaleIncrement;
               transform.localScale  =  new  Vector3  (scaleX,
scaleY, scaleZ);
          }

     }
}
```

5. Answer:

```
public class MoveSphere : MonoBehaviour {

     // must be a public variable so that the name of the
     // sphere can be specified here in the Inspector view
     public string nameOfObject;
     private GameObject target;

     enum  SpherePositions  {topleft,  bottomleft,  bottomright,
topright};
     SpherePositions currentPosition;

     Vector3 topleftposition = new Vector3(2f, 0.5f, -2f);
     Vector3 bottomleftposition = new Vector3(2f, 0.5f, 2f);
     Vector3 bottomrightposition = new Vector3(-2f, 0.5f, 2f);
     Vector3 toprightposition = new Vector3(-2f, 0.5f, -2f);


     // Use this for initialization
     void Start () {

          // first get a reference to the sphere based on its
name in
          // the hierarchy view
          target = GameObject.Find (nameOfObject);
          currentPosition = SpherePositions.topleft;
          if (target != null)
               target.transform.position = topleftposition;
     }

     void Update () {

          if (Input.GetKeyUp (KeyCode.T)) {

               currentPosition++;

               // at position 4, if T key is pressed again
               // go back to original position
               if ((byte)currentPosition > 3)
                         currentPosition                    =
SpherePositions.topleft;

               if (currentPosition == SpherePositions.topleft)
                         target.transform.position          =
topleftposition;
               else        if        (currentPosition        ==
SpherePositions.bottomleft)
                         target.transform.position          =
bottomleftposition;
               else        if        (currentPosition        ==
SpherePositions.bottomright)
                         target.transform.position          =
bottomrightposition;
               else        if        (currentPosition        ==
SpherePositions.topright)
                         target.transform.position          =
toprightposition;
               }

     }
}
```

6. Answer:

```
public class MoveSphereNoEnum : MonoBehaviour {

     // must be a public variable so that the name of the
     // sphere can be specified here in the Inspector view
     public string nameOfObject;
     private GameObject target;

     int currentPosition;

     float [] xcoordinate = {2f, 2f, -2f, -2f}; // different
for all 4 positions
     float [] zcoordinate = {-2f, 2f, 2f, -2f}; // different
for all 4 positions
     float ycoordinate = 0.5f; // fixed for all 4 positions

     // Use this for initialization
     void Start () {

          // first get a reference to the sphere based on its
name in
          // the hierarchy view
          target = GameObject.Find (nameOfObject);
          currentPosition = 0;
          if (target != null)
               target.transform.position         =         new
Vector3(xcoordinate[currentPosition],             ycoordinate,
zcoordinate[currentPosition]);

     }

     void Update () {

          if (Input.GetKeyUp (KeyCode.T)) {

               currentPosition++;
               // at position 4, if T key is pressed again
               // go back to original position
               if (currentPosition > 3)
                    currentPosition = 0;

               target.transform.position         =         new
Vector3(xcoordinate[currentPosition],             ycoordinate,
zcoordinate[currentPosition]);
          }

     }
}
```

7. Answer:

MainScript

```
public class MainScript : MonoBehaviour {

     SecondScript secondScript;
     public string objectToLookFor;
     CubeScript cubeScript;

     void Start () {

          secondScript = GetComponent<SecondScript> ();
          GameObject       target       =       GameObject.Find
(objectToLookFor);
          cubeScript = target.GetComponent<CubeScript> ();
          cubeScript.MoveToNewPosition     (transform.position,
secondScript.incrementValue);

     }

     void Update () {

     }
}
```

CubeScript

```
using UnityEngine;
using System.Collections;
using System;

public class CubeScript : MonoBehaviour {

     private bool isMoving;
     private float incrementX, incrementValue;
     private Vector3 newPosition;

     void Start () {
          isMoving = false;
     }

     void Update () {
          // The purpose of this is to make sure that
          // movement does not happen until the
          // MoveToNewPosition method has being called
          if (isMoving) {

               if  ((newPosition.x  -  transform.position.x)  <=
2.0)
                    incrementX = 0f;
               else
                    incrementX = incrementValue;
               transform.Translate (incrementX, 0, 0);
          }
     }

     public void MoveToNewPosition(Vector3 newPosition, float
incrementValue)
     {
          this.newPosition = newPosition;
          this.incrementValue = incrementValue;
          isMoving = true;
     }
}
```

8. Answer

```
public class MoveObjects : MonoBehaviour {

     public GameObject[] allCubes;
     public GameObject[] allSpheres;

     void Start () {
          allCubes                                              =
GameObject.FindGameObjectsWithTag("CubeTag");
          allSpheres                                            =
GameObject.FindGameObjectsWithTag("SphereTag");
     }

     void Update () {
          if (Input.GetKey (KeyCode.Q)) {

               foreach (GameObject cube in allCubes)
                    cube.transform.Rotate (0f, 1f, 0f);

          } else if (Input.GetKey (KeyCode.W)) {

               foreach (GameObject cube in allSpheres)
                    cube.transform.Translate (0f, 0.1f, 0f);
          }
```

```
	}
}
```

9. Answer

```
using UnityEngine;
using System.Collections;

public class CubeScript : MonoBehaviour {

	public float rotation;

	void Start () {
		rotation = 0f;
	}

	void Update () {
		transform.Rotate (0f, rotation, 0f);
	}

	void OnTriggerEnter(Collider other) { rotation = 1f; }

	void OnTriggerExit (Collider other) { rotation = -1f; }

}
```

10. Answer

```

using System;

public class CubeScript : MonoBehaviour {

	float rotateYAngle = 0f;
	float rotateZAngle = 0f;

	void Update () {
		gameObject.transform.Rotate    (0f,    rotateYAngle,
rotateZAngle);
	}

	void OnTriggerEnter(Collider other) {
		if (other.gameObject.name.Equals ("Cylinder"))
			Destroy (other.gameObject);
		else if (other.gameObject.name.Equals ("Sphere")) {
			Vector3 cubeLocation = transform.position;
			Vector3             sphereLocation            =
other.gameObject.transform.position;
   // Math.Abs gives absolute value, regardless of sign
			float  diffX  =  Math.Abs(  cubeLocation.x  -
sphereLocation.x);
               float  diffY  =  Math.Abs  (  cubeLocation.y  -
sphereLocation.y);

     // Comparing the difference will let us know
     // whether the contact is made from the top or the side
               if (diffX < diffY)
                    rotateYAngle = 1f;
                else
                    rotateZAngle = 1f;
     }

     void OnTriggerExit(Collider other) {
          rotateYAngle = 0;
          rotateZAngle = 0;
     }
}
```

11. Answer

```
using System;

public class CubeScript : MonoBehaviour {

     float rotateYAngle = 0f;
     float rotateZAngle = 0f;

     void Update () {
          gameObject.transform.Rotate    (0f,    rotateYAngle,
rotateZAngle);
     }

     void OnTriggerEnter(Collider other) {
          if (other.gameObject.name.Equals ("Cylinder"))
               Destroy (other.gameObject);
          else if (other.gameObject.name.Equals ("Sphere")) {
               Vector3 cubeLocation = transform.position;
               Vector3             sphereLocation            =
other.gameObject.transform.position;
               float  diffX  =  Math.Abs(  cubeLocation.x  -
sphereLocation.x);
               float  diffY  =  Math.Abs  (  cubeLocation.y  -
sphereLocation.y);
```

```
               SphereScript            theScript            =
other.gameObject.GetComponent<SphereScript>();
               // use the appropriate values of either vVal or
hVal
               // to determine the magnitude of the rotation
               if (diffX < diffY)
                    rotateYAngle = theScript.vVal * 10;
                else
                    rotateZAngle = theScript.hVal * 10;
          }
     }

     void OnTriggerExit(Collider other) {
          rotateYAngle = 0;
          rotateZAngle = 0;
     }
}
```

12. Answer

GenerateCubes. Make sure that the prefab variable is initialized to the Cube prefab before running.

```
public class  GenerateCubes : MonoBehaviour {

     public GameObject prefab;
     float coordX, coordY, coordZ;

     void Start () {
          coordX = transform.position.x;
          coordY = transform.position.y;
          coordZ = transform.position.z;
     }

     void Update () {

          float rotateY, rotateZ, moveY;
         rotateY = rotateZ = moveY= 0f;
          Vector3 newPosition;
          CubeScript script;

          if (Input.GetKeyUp (KeyCode.Q))
               rotateY = 1f;
          else if (Input.GetKeyUp (KeyCode.W))
               rotateZ = 1f;
          else if (Input.GetKeyUp (KeyCode.E))
               moveY = 0.1f;
          else if (Input.GetKeyUp (KeyCode.R))
               moveY = -0.1f;

          if (rotateY != 0 || rotateZ != 0 || moveY != 0) {
               newPosition  =  new  Vector3  (coordX,  coordY,
coordZ);
               GameObject  newCube  =  Instantiate  (prefab,
newPosition, transform.rotation) as GameObject;

               // Get a reference to the CubeScript attached
               // to the newly generated cube in order to
               // change its public variables
               script = newCube.GetComponent<CubeScript> ();
               script.rotateY = rotateY;
               script.rotateZ = rotateZ;
               script.moveY = moveY;

               // decrement so that the next generated cube
               // is 2 units away from the current cube
               coordX -= 2f;
          }

     }
}
```

`MoveCube`. Attach this to the Cube prefab

```
public class MoveCube : MonoBehaviour {

     public float rotateY, rotateZ, moveY;

     void Start () {     }

     void Update () {

          transform.Rotate (0f, rotateY, rotateZ);
          transform.Translate (0f, moveY, 0f);

     }
}
```

13. Answer

`GenerateScript`

```
public class GenerateScript : MonoBehaviour {

     public     GameObject     cubePrefab,     spherePrefab,
capsulePrefab;

     float coordX, coordY, coordZ;
     Vector3 newPosition;

     void Start () {
          coordX = transform.position.x;
          coordY = transform.position.y;
          coordZ = transform.position.z;
     }

     void Update () {

          if (Input.GetKeyUp (KeyCode.Q)) {
               newPosition = new Vector3 (coordX, coordY + 2f,
coordZ);
               Instantiate      (cubePrefab,      newPosition,
transform.rotation);
          } else if (Input.GetKeyUp (KeyCode.W)) {
               newPosition = new Vector3 (coordX, coordY - 2f,
coordZ);
               Instantiate     (spherePrefab,     newPosition,
transform.rotation);
          } else if (Input.GetKeyUp (KeyCode.E)) {
               newPosition = new Vector3 (coordX, coordY - 2f,
coordZ);
               Instantiate    (capsulePrefab,    newPosition,
transform.rotation);
          }

     }
}
```

CubeScript

```
public class CubeScript : MonoBehaviour {

     bool rotateSphere = false;

     void Start () { }

     void Update () {

          if (rotateSphere)
               transform.Rotate (0f, 1f, 0f);
          else
```

```
                    transform.Translate (-0.1f, 0f, 0f);

          }

          void OnTriggerEnter(Collider other) {
               rotateSphere = true;
          }
}
```

CapsuleScript

```
public class CapsuleScript : MonoBehaviour {

          // use to keep track of whether the capsule
          // has collided with the Wall yet
          GameObject collidedObject = null;

          void Start () {

          }

          void Update () {

               if (collidedObject != null) {
                    transform.Translate (0.1f, 0f, 0f);
                    collidedObject.transform.Translate  (-0.1f,  0f,
0f);
               }
               else
                    transform.Translate (-0.1f, 0f, 0f);
          }

          void OnTriggerEnter(Collider other) {
               collidedObject = other.gameObject;
          }

}
```

SphereScript

```
public class SphereScript : MonoBehaviour {

          void Start () { }

          void Update () {
               transform.Translate (-0.1f, 0f, 0f);
          }

          void OnTriggerEnter(Collider other) {
               Destroy (gameObject);
```

```
      }
}
```

14. Answer

```
public class WallScript : MonoBehaviour {

     public float myMove = 0f;

     void Start () { }

     void Update () {
          transform.Translate (myMove, 0f, 0f);
     }

     void OnTriggerEnter(Collider other) {
          MoveScript script;

          if (other.gameObject.name.Equals ("Sphere(Clone)"))
               Destroy (other.gameObject);
          else         if         (other.gameObject.name.Equals
("Cube(Clone)")) {
               script                                        =
other.gameObject.GetComponent<MoveScript> ();
               script.moveX = 0f;
               script.rotateY = 1f;
          }
          else         if         (other.gameObject.name.Equals
("Capsule(Clone)")) {
               script                                        =
other.gameObject.GetComponent<MoveScript> ();
               script.moveX = 0.1f;
               myMove = -0.1f;
          }

     }
}
```

Tutorial 5

Questions

Textbook: Sams Teach Yourself Unity® Game Development in 24 Hours, 2nd edition
Hour 10. Collision
Hour 12. Prefabs

1. Define the term collision with respect the Unity game engine. Explain in detail the function of a collider with respect to a collision.

2. Define what a mesh collider is and state the main problem related to its use, as well as an alternative to using it.

3. Briefly describe what a trigger is and what is necessary for it to work.

4. Explain in detail what raycasting is and what it is used for.

5. Explain what a prefab is and its purpose in constructing a game in Unity.

6. Define the following terms with regards to a prefab: instance, instantiate, inheritance

7. State two ways to change a prefab

Answers

1. Collision occurs when the border of one object comes into contact with another object. A collider is a component necessary for to detect collision, and is perimeter that is projected around an object, usually the same shape as it.

2. Mesh collider has the exact shape of a 3D model. There is a limit to the number of polygons allowed in it because it reduces game performance. Alternative: compose complex object with several basic colliders

3. A trigger is used to detect collisions and requires a rigidbody component

4. Raycasting is the act of sending out an imaginary ray from an object to see what it hits based on line of sight. It is useful in games for aiming, checking distance, etc

5. A prefab is a special type of asset that bundles up other game objects. This allows easy generation in game view during runtime

6. Answer

- Instance: An actual object of the prefab in the Scene / Game view
- Instantiate: Process of creating an instance of a prefab
- Inheritance: All instances of a prefab are linked to the prefab so any changes in it are reflected in all its instances

7. Answer

a) Expand the prefab to access its children and make changes directly in the Project view
b) Change an instance in the Hierarchy / Scene view and drag it back into the Prefab in the Project View

Tutorial 6

Questions

Textbook: Sams Teach Yourself Unity® Game Development in 24 Hours, 2nd edition
Hour 13. 2D Games Tools
Hour 14. User Interfaces

1. State 3 different design challenges associated with 2D games.

2. Explain in detail the effect of orthographic cameras on depth appearance and discuss the method used in Unity to work with this.

3. Discuss in detail 3 ways to ensure that sprites appear properly in a 2D scene.

4. Explain in detail the following elements that are used in building a UI: Canvas and EventSystem game object.

5. Explain in detail the relevance of anchors and how they function in making UI elements work.

6. List the two modes of an anchor and describe them in detail.

7. List and describe in detail the 3 possible options for deciding how UI is rendered to the screen

Answers

1. 3 different design challenges

- The lack of depth makes immersion harder to achieve.
- A lot of game types don't translate well to 2D.
- 2D games are generally unlit, so the sprites have to be carefully drawn and arranged.

2. Orthographic cameras do not show perspective distortion. All objects appear the same size and shape regardless of distance. A sorting layer is a way of providing depth appearance to sprites in order to determine which sprites draw in front of each other. If the sprites are sorted from front-to-back properly, an impression of depth can be given.

3. 3 ways

a) Group them into different Sorting Layers, where the lowest layer in the list is drawn on top
b) In the same sorting layer, different sprites can be given different Order in layer numbers, with higher numbers drawing on top of lower numbers
c) If the Sorting Layer and Order in Layer is not set, than the z-coordinate can be used to affect draw order.

4. A canvas is the basic building block for a UI. All the UI elements you add to the scene will be child objects of the Canvas in the Hierarchy. The EventSystem game object is always added when you add a canvas. It allows users to interact with the UI through pressing buttons or dragging elements.

5. Every UI element comes with an anchor which is used to find its place in the world in relation to the rect transform of its parent. These anchors determine how the element is resized and repositioned when the Game window changes size and shape.

6. 2 modes

a) Together mode: The anchor is a single point which fixes the UI element in place relative to that spot. So if the canvas changes size, the element won't.
b) Split mode: The anchor causes the element to fix its corners relative to the corners of the anchor. If the canvas resizes, so will the element.

7. 3 possible options.

a) Screen-space overlay. This is the default mode and it draws over the top of everything on the screen, regardless of the camera settings, or the position of the camera. The UI will show in the scene view at a fixed position, with the bottom-left at (0, 0, 0) in the world.

b) Screen-Space Camera. The UI is rendered by a camera of your choice and stays in a fixed position relative to this chosen camera. Effects such as lighting affect the UI, and objects can even pass between the camera and UI.
c) World Space. In this mode, the canvas is becomes a game object in the world and is not drawn over the rest of your game. It can be a part of, or blend with, the rest of the scene objects.

Tutorial 7

Questions

Textbook: Sams Teach Yourself Unity® Game Development in 24 Hours, 2nd edition
Hour 17. Animations
Hour 18. Animators

1. Briefly explain what a rig is and describe its role in animation.

2. Briefly explain what an animation is and how it is used with rigs.

3. Briefly explain the process of creating a complete 2D animation from scratch in Unity.

4. Identify the 3 key parts of Unity's animation system, Mecanim, and describe their roles briefly.

5. Briefly explain the purpose of states and blend trees in the Animator View in Mecanim in Unity.

Answers

1. A rig allows Unity to determine which parts of a 3D model are capable of movement and the manner in which they can move. The rig dictates the parts of a model that are rigid (bones) and the parts which can bend (joints).

2. An animation is a series of instructions that define the movement of a rig. They can be played, paused and stepped through. They can also be applied to different models to achieve the same movement effect.

3. It involves creating key frames for the animation sequence and then choosing appropriate values for them. Unity then creates all the frames in between these key frames and interpolate values to smoothly transition between them. These

interpolated values can also be changed manually to change the nature of transition between them.

4. Unity’s animation system (Mecanim) comprises three parts: the animation clip, an animator controller, and an animator component. The animation clips are the various motions are imported or created in Unity. The Animator Controller contains the animations and determines which clips are playing at any given moment. Complex models have an Avatar which acts as the translator between the animator controller and the rigging of the model. Both the animator controller and the avatar are put together on the model using an Animator Component.

5. States refer to the statuses that the model is currently in, and this define the animation that is playing. A state can have multiple animations. A blend tree is used to seamlessly blend one or more animations based on some parameter.