Unity C# basics

This is an introduction to some of the key commands and objects found in Unity.

There is a game loop that happens within Unity. Each object can be scripted to take advantage of this loop by telling it to do something or to check something each time the loop cycles.

The two most used functions for this game loop are:

Update:
This function is called before rendering a frame. This is where most game behavior code goes, except physics code.

This is like the draw() function in Processing.

```
void Update(){

}
```

Also LateUpdate can be used instead as it is executed right before rendering

FixedUpdate:
This function is called once every physics time step. This is the place to do physics-based game behavior. It differs in its timing from the Update function as it loops on the physics time step.Physics time step can be set in Edit>Project Settings>Time.

```
void FixedUpdate(){

}
```

There are other functions with predetermined execution times that can be used as well.

Start:
This is called when a prefab is instantiated or when the level is first run and it is on screen. You can also get the same behavior by putting code outside of other functions in a script attached to an object.

```
void Start(){

}
```

Awake:
This is executed at the very beginning of scene execution.

```
void Awake(){

}
```

All of these functions have an order of execution that is followed by Unity. This can be important if you are curious about whether something is done before you may want to access a result in another script.

All Awake calls

All Start Calls

while (stepping towards variable delta time)

All FixedUpdate functions

Physics simulation

OnEnter/Exit/Stay trigger functions

OnEnter/Exit/Stay collision functions

Rigidbody interpolation applies transform.position and rotation

OnMouseDown/OnMouseUp etc. events

All Update functions

Animations are advanced, blended and applied to transform

All LateUpdate functions

Rendering

Time:
Time is the object used to manage time based values in the game.

Time.deltaTime :

Time passed since last update. This allows for time-based animation that is free from the frame rate of the machine. If you change values without this in Update you would be subject to the fluctuations in the speed of the looping.

amount*Time.deltaTime = the change per update where amount is the amount per second.

Time.time

Time expired since the beginning of the game in seconds. Used to create timers by setting to a variable and then comparing.

```
private float  startTime=Time.time;
private float maxTime=30;

if (Time.time-startTime>maxTime){

            //Times up do something and reset totalTime

}
```

Variables

C# is a typed language

Variables added to top of script are global to the script. By default they are public and can be changed in the inspector in the interface. You drag content onto slot in inspector to add or change value. This makes it easy to pass other gameObjects to a function. This visual way of passing parameters makes it easy to adjust the values for a sketch. You can use private to protect variables and prevent them from showing in the inspector.

public int walkSpeed =5;   //You can give it an initial value as well

private int multiplier =60; //would not be visible in the inspector

Public variables are always exposed in the inspector. Their values can be changed in the inspector and this breaks the connection with the script. You must reset the component to return to the original value.

Vector3

Vector3 is a class used to describe three dimensional vectors. Vectors are used to describe a position in space. They are also used to hold values that may be used to alter a position such as velocity or acceleration. Even when working with Unity in 2D mode we must often use Vector3.

Vector3(x,y,z)

transform.position=Vector3(20,15,-30);

This sets the current gameObject to the position x=20, y=15, and z=-30.

Vector3(0,1,.5)

This is a custom direction that could be added to a position.

There are several existing vectors to speed up your coding.

Vector3.up, Vector3.down, Vector3.right, Vector3.left, Vector3.forward

(.up is in y, .forward is in z, and .right is in x)

Vector3.Distance(onevector, othervector)

The Distance method calculates the distance between to vectors or points in space. Great for proximity checking.

Vector2

Vector2 is a class used to describe two dimensional vectors.

Vector2(x,y)

Vector2.MoveTowards(Vector2 current, Vector2 target, float maxDistanceDelta);

This is essentially the same as Vector2.Lerp but instead the function will ensure that the speed never exceeds `maxDistanceDelta`. Negative values of `maxDistanceDelta` pushes the vector away from `target`.

Components
Components are attached to gameobjects within Unity. They add functionality and when accessed can change the attributes of a gameobject. The most prevalent component is the transform component. It is used to track the position, rotation and scale of a gameobject. We can set these absolutely or use methods to change the value.

Capital letters distinguish between object and variable.

Transform=object type,
transform=instance of the component on a gameobject.

Position

transform.position(x,y,z)

transform.position=Vector3(20,15,-30);

var zposition = transform.position.z;

Rotation

transform.rotation(x,y,z) – does not rotate only sets to a particular rotation.

transform.rotation= Quaternion.identity; //Quaternion.identity will set the rotation to no rotation. Aligned to world axis.

Methods used to change position and rotation.

transform.Translate(x,y,z);

trasform.Translate(Vector.forward*(mvAmount*time.deltaTime));

This moves the object forward at a rate of mvAmount per second

transform.Rotate(x,y,z);

transform.Rotate(0, rotSpeed*time.deltaTime, 0);

Rotates around the y axis at a speed of rotSpeed.

Uses a z,x,y rotation order. Rotation is in degrees.

Accessing Components

GetComponent<Component>()

GetComponent lets us drill down to the components of a gameobject.

Used for other components such as scripts

```
public ExampleScript someScript;

someScript = GetComponent <ExampleScript>);

someScript.DoSomething ();
```

Prefabs

Destroy(object);

This will destroy an existing gameobject from the scene. It is often used to remove objects after they have been collided with.

Instantiate(object, position, rotation)

This will instantiate a prefab into the scene at the position and rotation specified

```
var chest: GameObject;

Instantiate(chest, Vector3(0,0,0), Quaternion.identity);
```

Other functions

GameObject.Find("Name")

Finds a gameobject in the scene by its name.

Also GameObject.FindWithTag("TagName");

```
SceneManager.LoadLevel("LevelName");
SceneManager.LoadLevel(SceneNumber);
```

Loads a scene named LevelName or by a number. Must add scenes into the build settings before you can refer to them by number or by name.

Input

Unity has an input manager which allows you to change the settings for the project. Edit>Project Settings>Input. By selecting each of the inputs you can change which key or mouse button is used to trigger input event. There are positive and negative values returned by the GetAxis methods.

Input.GetAxis("Horizontal");

Return a value from -1 to 1, depending on which key was pressed.

Create a simple driving using a block and the GetAxis with horizontal and vertical. See reference for example code.

Input.GetAxis(“Mouse X”) and “Mouse Y”

Range is not -1 to 1. Instead uses Sensitivity setting in Input settings for Mouse X. Also this is a change/delta not an absolute.

Try to replace Horizontal with Mouse X in your car example.

Input.GetKeyDown(“space”)

This allows you to track to see if a key has been pressed.

```
if( Input.GetKeyDown("space") ){
}
```

Check the GetKeyDown inside of the update to see if a certain key has been pressed.

Audio

Audio can be added to a scene by adding AudioClips in scripts, var mySound:AudioClip or by adding an AudioSource component to an existing gameobject. Audio files used must be an aif, .wav, .mp3, or an .ogg. There can only be one Audio Listener component in the scene, this is typically attached to the main camera. Using a listener in your project automatically generates 3d Audio.

AudioSource component for a gameobject allows you to play the sound on Awake(). You can also set looping from the component. Volume, pitch and rolloff can be set as well. Audio clip must be dragged on in inspector for the sound to play.

In scripting, you must declare a clip. Make it public so you can drop it on from the project window.

```
public AudioClip soundToPlay;
```

Then you can use either Play or PlayOneShot.

```
audio.clip = soundToPlay;
```

```
audio.Play(); //Plays audioclip assigned to audio.clip
```

```
audio.loop=true; //sets the loop to true
```

For one time sounds use PlayOneShot

```
audio.PlayOneShot(soundToPlay);
```

Add this line at the end of your script to ensure you don’t get errors.

@script RequireComponent(AudioSource)

Note: no semicolon at end of this line when used in your script

Physics components
Colliders

Colliders are component needed to track collisions in Unity. They have basic shapes or they can be derived from the shape of the mesh. Mesh colliders are more processor intensive and sometimes complex shapes can be mimicked with compound colliders where multiple simple colliders are combined to encompass the shape. These are created by making the colliders the children of your gameobject.

You can also have a static collider. A Static Collider is a GameObject that has a Collider but not a Rigidbody. Static Colliders are used for level geometry which always stays at the same place and never moves around.

Collision requires that some gameobjects have a rigidbody component.

Rigidbodies

Rigidbody components allow your objects to be controlled by physics. You can add velocity or torque to an object that has a rigidbody component. Mass, drag and whether it is controlled by gravity can be set for each rigidbody. We can remove an object from the influence of the physics engine by using isKinematic. This will allow collisions, since rigidbody components are required for collisions, but keep the object stationary. They can be moved with a Constant Force component as well.

Triggers

Triggers are typically invisible objects, that is objects with their mesh renderer disabled, that are used to trigger events in a game. They can also be empty gameobjects that have only a collider component attached. Triggers must be set by selecting the Is Trigger option on the collider of the object. Triggers are ignored by the physics engine.

Physics methods
OnCollisionEnter(collision : Collision)

This is called when rigidbody/collider intersects with another rigidbody/collider. Collision class objects contain more information than Collider objects. Collision points, relative velocity as well as collider, rigidbody, gameobject, and transforms are returned.

See reference for good examples.

Also OnCollisionExit2D and OnCollisionStay2D

OnTriggerEnter2D(Collider2D hit)

This is called when collider enters a trigger. It passes a Collider2D object which has transform, rigidbody and gameobject information.

Also OnTriggerExit2D and OnTriggerStay2D

Ray casting
Rays can be cast into the space of the environment. This allows you to check if any collisions occur along this ray. It can be used for lines of sight or to determine if you have clicked on an object.

RaycastHit2D hit = Physics2D.Raycast(transform.position, Vector2.right, distance);

This will cast a ray of a specified distance into our scene and return any collider that it hits. There are many versions of the method so check the reference. Depending on which version you use, you may get back a Boolean result that will indicate if something was hit or a more complex object that will return information about what was hit.

hit.point, hit.distance can be checked on the returned object.

More importantly we can use hit.transform.gameobject to get the object and compare it to an object found by GameObject.Find("Name") or GameObject.FindWithTag("TagName");

Picking Objects
Convert 2d screen to ray with

```
camera.ScreenPointToRay (Input.mousePosition);
```

Tags
Tags are used to create a group of objects or to label an object. Objects can only have one tag and can be found with tags(GameObject.FindWithTag("TagName");). Tags are assigned in the inspector. Choosing Add New Tag will open the Tag Manager.

Example:If we hit an object of a certain tag then Destroy it.

```
if(gameObject.tag == "Player"){

        Destroy(gameObject);

}
```

Arrays
Arrays are extremely fast and is typed. This array is a fixed size and can't be resized. You must recreate the array at a larger size and copy into it.

To declare "built in" arrays use:

```
public float[] myArray;

myArray = new float[20];
```

Individual values can be set:

```
myArray[0]=3.4;
```

To iterate through the array use "in"

```
for(var somevalue in myArray){
  print(somevalue);
}
```

Arrays created at top of script can be populated in the inspector and can be changed in the inspector.

Calling methods from other scripts
script = GetComponent<"ScriptName">(); is the name of another C# script.

script.DoSomething (); //where DoSomething() is a function within the script Scriptname