Unity Cheat Sheet

[Creating a new Cube](#_mygrecq2qwz3)

[Changing colour to a cube](#_w8abf6tlt1hv)

[Creating a new script](#_lcz16z9ghkxx)

[Action triggered with a mouse click](#_lwqym4b18x2s)

[Action triggered on collision](#_i9qhobn8ma78)

[Action triggered on collision with a specific object](#_3yt2fdsirdsh)

[Action triggered with the keyboard](#_39r97pky51hi)

[Action triggered by left / right / up / down](#_p1olioze0clt)

[Change the axes](#_l4du3b42xxt6)

[Moving an object to a new position](#_hnm908afysim)

[Moving an object at a constant speed](#_scsvb29zk16l)

[Meters per second](#_ct16q3fefas5)

[Moving at an angle](#_asaug7ss1ez8)

[Moving an object with the keyboard](#_t91du22c5fi7)

[Making an object (dis)appear](#_xg7pn2c8krp9)

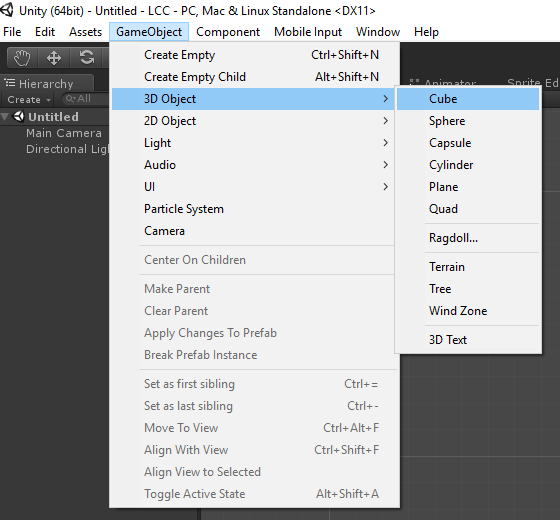
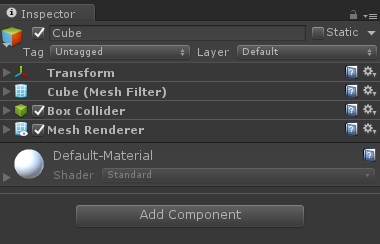
[Via code:](#_a5o0wnqyl7cf)

[From the editor](#_8ajfchorsn4p)

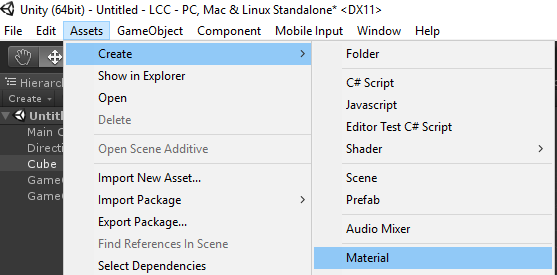
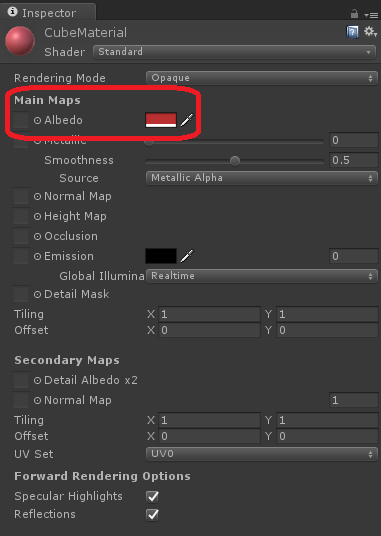
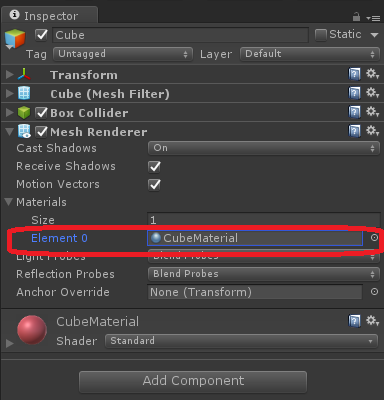
[Changing colour to an object (via code)](#_gkl70difhisb)

[Generating random numbers](#_egqjk89kvo30)

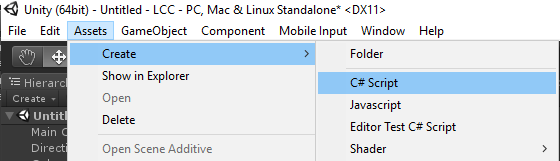
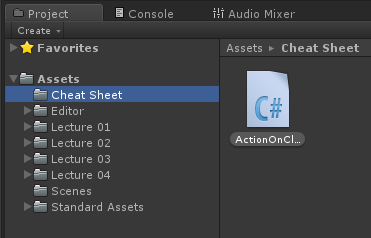
# Creating a new Cube

1. From the menu: **GameObject > 3D Object > Cube**
2. Find the cube in the Hierarchy (left) and select it, to see its properties in the Inspector (right).  
     
   Each GameObject has several components. A standard Unity cube has:
   * A **name**, a **tag** and a **layer** (the last two are used for collisions)
   * **Transform**. To change its position, rotation and scale
   * **Mesh Filter**: Contains the 3D model of the cube
   * **Box Collider**: Used to collide with other objects. Remove it and you can walk through it.
   * **Mesh Renderer**: Used to render it into the game. Remove it, and you won’t see it.

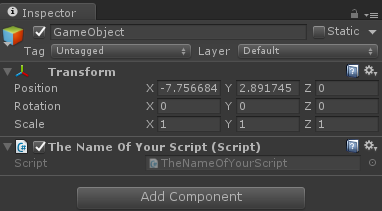
# Changing colour to a cube

1. To change the aspect of an object, you need to create a new Material.  
   You can do it from **Assets > Create > Material**.
2. Locate the material in the Project window and select it to access its properties in the Inspector.
3. You can change the colour of the material by using the **Albedo** colour picker:  
   
4. Assign this new material to the cube. To do so, drag the material onto the cube.
   1. Alternatively, select the cube in the Inspector, expand the **Mesh Renderer** component by clicking on the little triangle on the left.
   2. Expand the **Materials** lists
   3. Click on the little circle next to **Element 0**:  
      
   4. Select your new material from the window the appears.

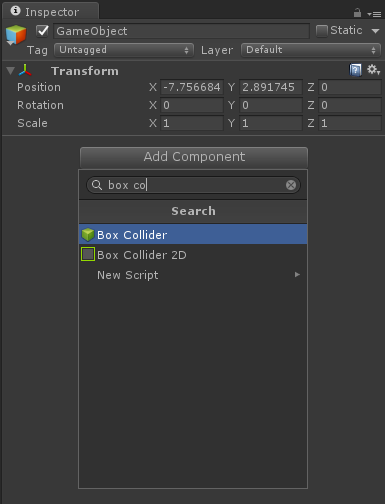
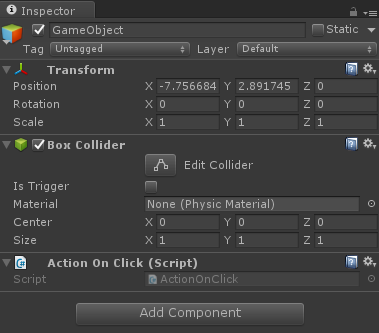
# Creating a new script

1. Create a new Script: **Assets > Create > C# Script**  
   Rename it accordingly and open it.  
   Script names should contain no spaces, and start with a capital letter.  
   
2. Locate the script in the Project window.  
   Double click to open it.  
   
3. Make sure the name of the class is the same of the name of the file.  
   If not, the script won’t work.  
   The script should look like this:

|  |
| --- |
| using UnityEngine;  using System.Collections;  public class **TheNameOfYourScript** : MonoBehaviour {  // Use this for initialization  void Start () {    }    // Update is called once per frame  void Update () {    }  } |

1. Scripts represent custom behaviours. If you want this behaviour to affect an object, simply drag the script from the Project windows onto the object itself.  
   

# Action triggered with a mouse click

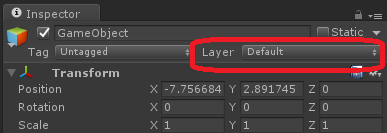
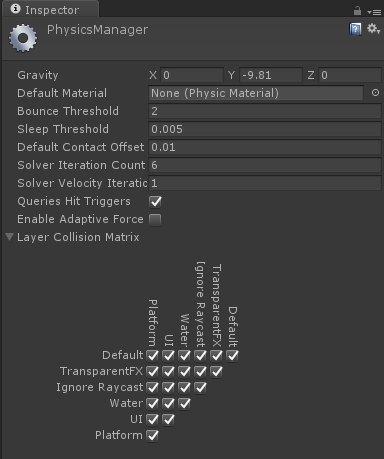
1. Select the object you want to react to mouse clicks
2. In the inspector, make sure that there is a **Collider**.
   1. If not, click on **Add Component > Box Collider** (or the shape you need)  
      The colliders will be used later in the script to catch the mouse click  
      
3. Create a new script (for instance **ActionOnClick**) and drag it onto the object you want to react:  
   
4. Open the script and create a method called **OnMouseDown**.Unity will invoke it when the player clicks on the object collider.

|  |
| --- |
| using UnityEngine;  public class ActionOnClick : MonoBehaviour {  **void OnMouseDown()**  **{**  **Debug.Log("Click!");**  **}**  } |

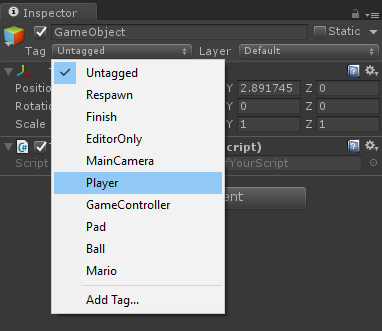
# Action triggered on collision

1. Select the object you want to react to collision
2. In the inspector, make sure that there is a **Collider**.
3. Create a new script (for instance **ActionOnCollision**) and assign it to object
4. Open the script and create the following:

|  |
| --- |
| using UnityEngine;  using System.Collections;  public class ActionOnCollision : MonoBehaviour {  void **OnCollisionEnter** (Collision collision)  {  Debug.Log("Collision with: " + collision.collider)  }  } |

1. Make sure that two objects CAN collider with each other.  
   This means that they are both in layers that can collider.  
   You can check the layer of an object from the inspector:  
   
2. Make sure that the two layers you want to collide can collider with each other.  
   To do so, go in **Edit > Project Settings > Physics** to adjust the **layer collision matrix**:  
   

# Action triggered on collision with a specific object

1. To limit collision only to certain object, you can either change the layers and collision matrix, or change the script (continue reading).
2. Take the object you want to collide with and assign a specific tag to it.  
   
   1. If you need a different tag (such as “Bullet” or “Water”) click on **Add Tag…** to create it. Once added, don’t forget to go back to assign it to the object.
3. Change the script so that it reacts only if the object colliding has the matching tag.  
   Unity has a function called **CompareTag** that can be used for this:

|  |
| --- |
| using UnityEngine;  using System.Collections;  public class ActionOnCollision : MonoBehaviour {  void OnCollisionEnter (Collision collision)  {  **if (collision.collider.CompareTag("Player"))**  {  Debug.Log("Collision with: " + collision.collider)  }  }  } |

# Action triggered with the keyboard

1. Create a new script and attach it to the object you want to react.
2. Use the **Input** class to detect inputs from the keyboard:

|  |
| --- |
| using UnityEngine;  using System.Collections;  public class ActionOnKeyboard : MonoBehaviour {  void Update ()  {  if (**Input.GetKeyDown(KeyCode.E)**)  {  Debug.Log("Key pressed");  }  }  } |

Unity offers a variety of methods you can use:

* **Input.GetKeyDown**: you have just pressed the key
* **Input.GetKey**: a key is being pressed
* **Input.GetKeyUp**: a key has just been released

# Action triggered by left / right / up / down

Unity abstracts the moving controllers in a way that is independent of the controller. If you want your code to work both on keyboard and XBOX controller, use this method.

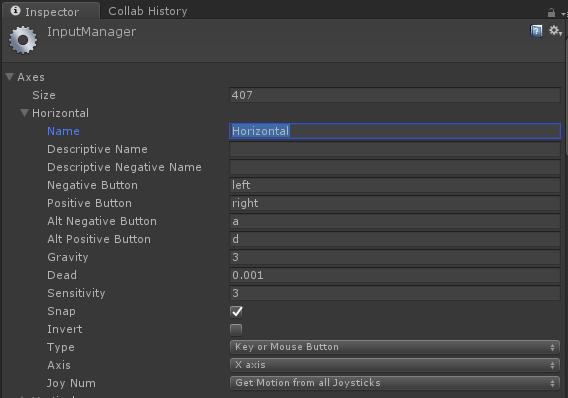
Use the GetAxis method to get the movement on a specific axis:

|  |
| --- |
| if (**Input.GetAxis(“Horizontal”)** > 0) {  Debug.Log(“Moving left”);  } else if (**Input.GetAxis(“Horizontal”)** < 0)  {  Debug.Log(“Moving right”);  } |

You can use the following:

* **Input.GetAxis(“Horizontal”)**: left / right; A / D; analog stick, joystick; …
* **Input.GetAxis(“Vertical”)**: up / down; W / S; analog stick, joystick; …

## Change the axes

To define your own (for instance **Input.GetAxis(“MyAxis”)**), go to **Edit > Project Settings > Input**:

# Moving an object to a new position

To move an object to a new position you have to change the **position** property of its **Transform**. This instantly teleports the object to a new location.

The following code moves the object to world coordinates (10, 5, 6):

|  |
| --- |
| void Start ()  {  transform.position = new Vector3(10f,5f,6f);  } |

If you want to move an object in respect to its current position:

1. Retrieve the current **transform.position**
2. Change it
3. Assign it back

|  |
| --- |
| // Correct  Vector3 newPosition = transform.position;  newPosition.x += 10f;  transform.position = newPosition;  // Wrong  transform.position.x += 10f; |

There are alternative ways in which this can be implemented:

|  |
| --- |
| // Method 1  Vector3 newPosition = transform.position;  newPosition.x += 10f;  transform.position = newPosition;  // Method 2  transform.position = transform.position + **new Vector3(10f, 0, 0)**;  // Method 3  transform.position = transform.position + **Vector3.right \* 10f**;  // Vector3.right is (1, 0, 0)  // Method 4  transform.position **+=** Vector3.right \* 10f; |

# Moving an object at a constant speed

If you want to move an object over a period of time, you need to move it a little bit each frame. This is best done in the Update function.

The following code moves the object by 1 meter per second to the right:

|  |
| --- |
| public float speed = **1f**; // meters per second  void Update ()  {  Vector3 newPosition = transform.position;  newPosition.x += speed \* **Time.deltaTime**;  transform.position = newPosition;  } |

## Meters per second

The Update function is invoked each frame. If the game runs at 60 frames per second, the Update function is called 60 times a second. This means that every change you make to the position is also done 60 times a second. If you write:

|  |
| --- |
| newPosition.x += speed; |

you are moving 1 meter each frame, meaning 60 meters a second.

To change this from “*meters per frame*” into “*meters per second*”, you have to multiply **speed** by **Time.delaTime**.

## Moving at an angle

If you want to move your object at an angle, you can change multiple axes at the same:

|  |
| --- |
| **public float speedX = 1f; // meters per second**  **public float speedY = 0.5f; // meters per second**  **public float speedZ = 1.4f; // meters per second**  void Update ()  {  Vector3 newPosition = transform.position;  **newPosition.x += speedX \* Time.deltaTime;**  **newPosition.y += speedY \* Time.deltaTime;**  **newPosition.z += speedZ \* Time.deltaTime;**  transform.position = newPosition;  } |

This could also be written as:

|  |
| --- |
| **public Vector3 speed = new Vector3(1f, 0.5f, 1.4f);**  void Update ()  {  Vector3 newPosition = transform.position;  **newPosition += speed \* Time.deltaTime;**  transform.position = newPosition;  } |

Remember that if you are doing this, the actual velocity (in meters per second) you are moving is given by **speed.magnitude**.

# Moving an object with the keyboard

1. Create a new script (for example **KeyboardController**) and assign it to the object you want to control
2. Open the script:

|  |
| --- |
| using UnityEngine;  public class KeyboardController : MonoBehaviour  {  public Vector3 speed = new Vector3(1f, 0f, 1f);    void Update ()  {  Vector3 newPosition = transform.position;  **newPosition.x += speed.x \* Time.time \* Input.GetAxis(“Horizontal”);**  **newPosition.z += speed.z \* Time.time \* Input.GetAxis(“Vertical”);**  transform.position = newPosition;  }  } |

The horizontal input (left/right) controls the movement on the X axis.

The vertical input (up/down) controls the movement on the Z axis.

This moves the object on the flat plane.

# Making an object (dis)appear

While an object is not active:

* Its scripts won’t be executed
* Is invisible
* It won’t collide
* It won’t move

## Via code:

* Disappear:

|  |
| --- |
| gameObject.SetActive(false); |

* Appear

|  |
| --- |
| gameObject.SetActive(true); |

## From the editor

Toggle this checkbox next to the object name:  


# Changing colour to an object (via code)

To change the colour of an object via code, you need to change the colour of its material. You can access the material that an object uses by accessing its Renderer.  
This is done with a method called **GetComponent**.

|  |
| --- |
| GetComponent<Renderer>.material.color = new Color(r,g,b); |

When creating a colour, **r**, **g** and **b** are the red, green and blue components to use. They go from zero to one.

They are **floating point** values, so you have to use an **f** after them:

|  |
| --- |
| Color c1 = new Color(0.5**f**, 0.1**f**, 0.8**f**); // right  Color c2 = new Color(0.5, 0.1, 0.8 ); // wrong |

# Generating random numbers

You can generate random numbers with the function **Random.Range**.

The following piece of code moves an object to a random position.

|  |
| --- |
| void Start ()  {  transform.position = new Vector3  ( **Random.Range(-10,+10),**  **Random.Range(-10,+10),**  **Random.Range(-10,+10)**  );  } |

If you feed the function with integers, the result will be an integer as well.

If you want decimal numbers as well, you need to provide floats instead:

|  |
| --- |
| Random.Range(-10, +10 ) // returns an integer such as 1,2,3, ...  Random.Range(**-10f, +10f**) // returns a float such as 1.4, 3.4, -2.4, ... |