بسم الله الرحمن الرحيم

كتاب
مدخلك لتعلم لغة Java Script الخاصة بمحرك الألعاب

Unity 3D 2.6

إعداد

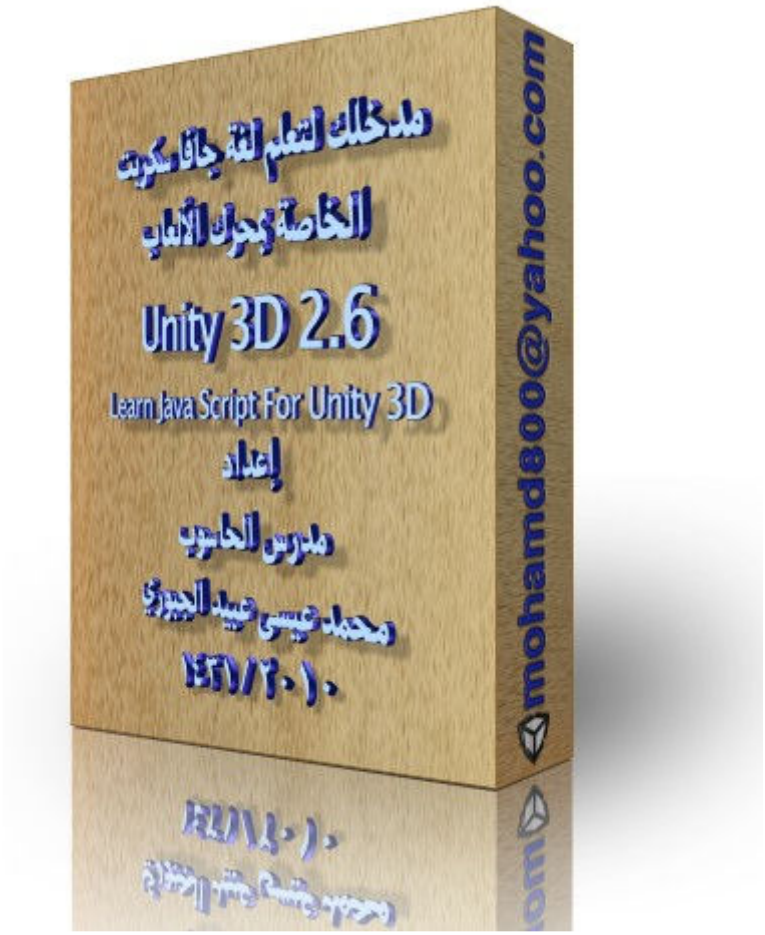
مدرس الحاسوب

محمد عيسى عبيد الجبوري

mohamd800@yahoo.com

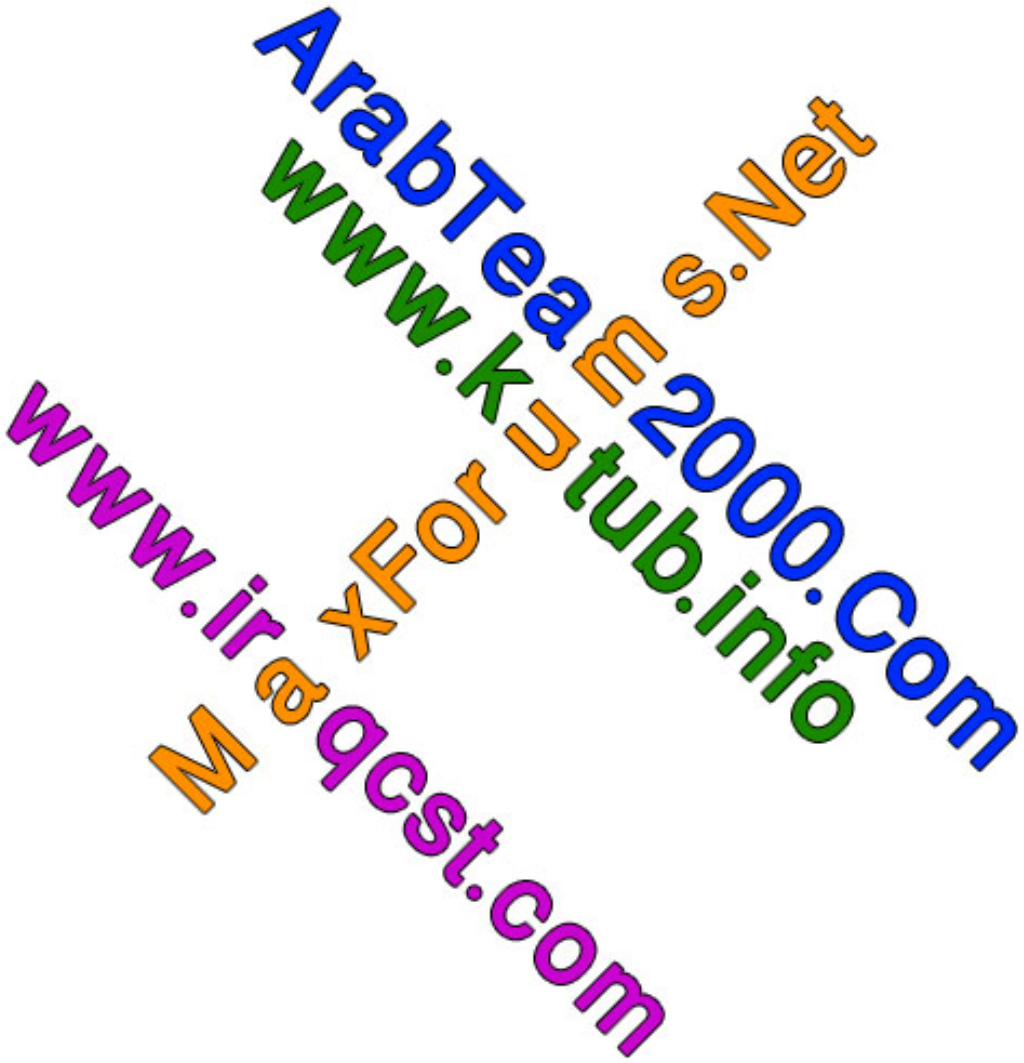
٢٠١٠م / ١٤٣١ هـ

هـذا الكتـاب يعتبـر مـدخلاً لمـن يريـد أن يـتعلم لغـة الجافـا سكربت الخاصـة بمحرك الألعاب Unity3d 2.6 ، وهو لايشرح أساسيات البرمجة بصورة عامة ولا يتطرق إلى أساليبها فقط يوضح مكونات هذه اللغة وطريقة التعامل معها في محرك الألعاب ، كما انـه لا يحتـوي جميـع مكونـات وتفاصـيل هـذه اللغـة ، ويفتـرض ان القارئ يجيد استخدام المحرك المذكور مثلاً معرفة نوافذ البرنـامج وفائدة كل منها ووضع المجسمات واستيرادها وإجراء التحويلات عليهـا ووضـع المكونـات عليهـا مثـل مكونـات الفيزيـاء والـصوت والخامـات وغيرهـا مـن الأمـور ، لأنـك إذا لـم تجيـد التعامـل مـع المحرك فلا فائدة من تعلم البرمجة به.



مراجعة وتدقيق : أحمد ألبنا

تحية شكر وتقدير إلى :



لا تنسونا من صالح دعائكم لي ولوالدي العزيزين ولجميع المسلمين.....

قبل القراءة

الكتاب غير مجاني وثمنه وهو أن تخرج أي مبلغ تستطيع عليه وتعطيه لفقير أو مسجد أو مؤسسة خيرية، حتى لو بعد قراءة الكتاب ولكن يبقى رهانٌ عليك ، أمتثالاً لقول الله تعالى:

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

﴿ يَا أَيُّهَا الَّذِينَ آمَنُوا أَنْفِقُوا مِمَّا رَزَقْنَاكُمْ مِنْ قَبْلِ أَنْ يَأْتِيَ يَوْمٌ لَا بَيْعٌ فِيهِ وَلَا خُلَّةٌ وَلَا شَفَاعَةٌ وَالْكَافِرُونَ هُمُ الظَّالِمُونَ ﴾

سورة البقرة : آية (٢٥٤)

تنبيه : إن لم تكن من الذين في بداية الآية ،فويحك أنت من الذين في نهايتها ،فكر من جديد.....

ولايحق لأي جهة أو شخص أن يتاجر بهذا الكتاب ، ويحق لك طباعته على ورق واستنساخه وتوزيعه بعد أن تدفع ثمنه المذكور أعلاه على أن لا يتم حذف وتغيير أي حرف منه وأن تأخذ كلفة الطبع أو الاستنساخ فقط وليس ثمن الكتاب...

مكونات اللغة:

تتكون هذه اللغة من مجموعة من الكلاسات(Classes) والدوال(Functions) الجاهزة التي تلبي اغلب احتياجات برمجة الألعاب مع امكانية برمجة كلاسات ودوال جديدة من قبل المبرمج إذ لم يجد ما يلبي احتياجه (كما في الألعاب الضخمة والمعقدة) ، حيث يتم استدعاء هذه الكلاسات بأسمائها من ثم الدخول إلى المتغيرات أو الدوال التي بداخل هذه الكلاسات وتمرير البارمترات التي تحتاجها لتنفيذ الأوامر المطلوبة منها ،وبصفة عامة فان طريقة استدعاء هذه الكلاسات والدوال كما يلي:

1- Class.Function();

مثال:

```
transform.Rotate(x ,y ,z);
animation.Play();
```

2- Class.variable;

مثال:

```
transform.position;
```

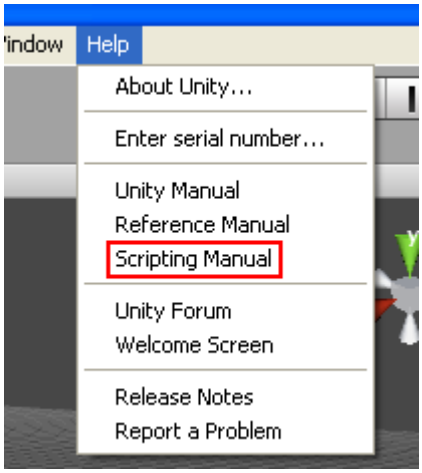
3- Class.variable.variable;

مثال:

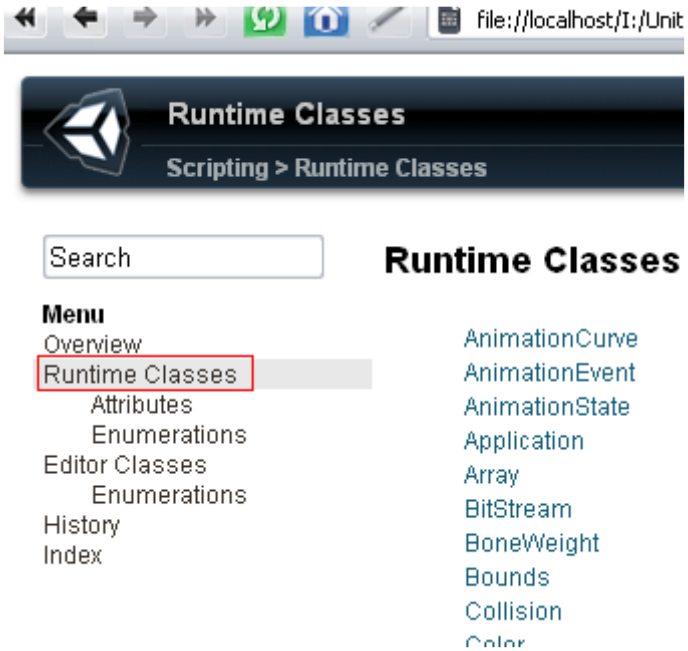
```
transform.position.x;
```

ويمكن القول ان الكلاس هو ما يكتب في البداية ويحتوي على الدوال والمتغيرات وان الدوال هي التي تحتوي على قوسين في نهايتها قد يكونا فارغين أو يمرر بينهما البارمترات، وما عداهم فهو متغير.

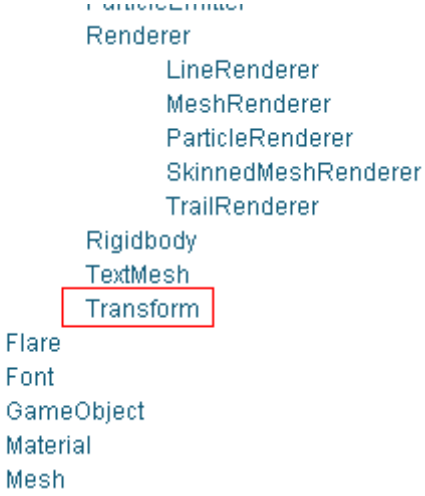
ولمعرفة جميع أسماء الكلاسات المتاحة وما تحتويه من دوال ومتغيرات اتبع التالي:



ثم اختر الكلمة Runtime Classes



سوف تظهر لك جميع الكلاسات المتاحة اختر الكلاس الذي تريد معرفته مثلاً



سوف تظهر لديك جميع الدوال والمغيرات التابعة لهذا الكلاس اختر احدها لتظهر لك نبذة عنه.

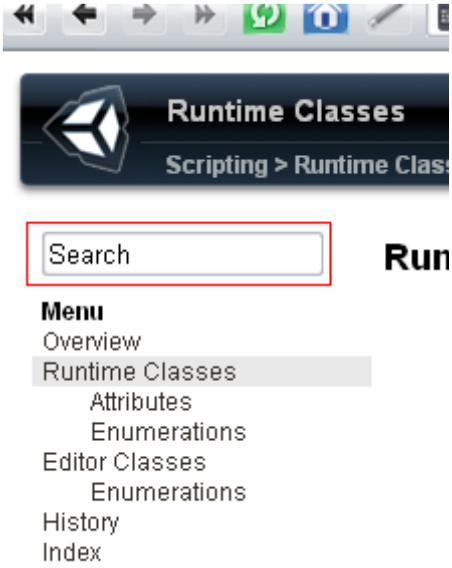
Variables

position	The position of the transf
localPosition	Position of the transform
eulerAngles	The rotation as Euler ang
localEulerAngles	The rotation as Euler ang
right	The red axis of the transf
up	The green axis of the trar
forward	The blue axis of the trans
rotation	The rotation of the transfc
localRotation	The rotation of the transfc
localScale	The scale of the transforr
parent	The parent of the transfor
worldToLocalMatrix	Matrix that transforms a p
localToWorldMatrix	Matrix that transforms a p
root	Returns the topmost tran
childCount	The number of children th
lossyScale	The global scale of the ol

Functions

Translate	Moves the transform in th
Rotate	Applies a rotation of eule
	eulerAngles.y degrees ar
RotateAround	Rotates the transform ab
LookAt	Rotates the transform so
TransformDirection	Transforms direction fror
InverseTransformDirection	Transforms a direction fr
TransformPoint	Transforms position from
InverseTransformPoint	Transforms position from
DetachChildren	Unparents all children.
Find	Finds a child by name an
IsChildOf	Is this transform a child c

ويمكنك أيضاً أن تستخدم البحث عن طريق كتابة اسم الشيء الذي تريد البحث عنه في المربع الخاص بالبحث ثم الضغط على زر Enter



نبذة عن المتغيرات :

ممكن في السكربت الإعلان عن أنواع المتغيرات المعروفة في لغات البرمجة عامة مثل المتغيرات العددية بأنواعها والمتغيرات الحرفية والنصية والمتغيرات المنطقية وغيرها .

كما يمكن الإعلان عن المتغيرات الخاصة بمحرك الألعاب مثلاً يمكن الإعلان عن متغير من نوع كائن لعبة (GameObject) ويتمتع بما يتمتع به أي مجسم ثلاثي في اللعبة وكذلك يمكن الإعلان عن متغير من نوع تحويل (Transform) يمكن إجراء التحويلات الثلاثة عليه (النقل والتدوير والتحجيم) ، وأيضاً يمكن تعريف متغير من نوع Texture للتعامل مع الخامات ، وبصفة عامة أي كلاس يمكن تعريف متغير من نوعه (سنأتي إلى موضوع الكلاسات لاحقاً) ، والغرض منها هو للتعامل مع مجسمات اللعبة من داخل السكربت عن طريق اسناد هذه المتغيرات إلى مجسمات اللعبة ، وسنتعرف على طريقة استخدامها وإسنادها في موضوع الكلاسات عند شرح الأمثلة.

والصيغة العامة لتعريف المتغيرات:

```
var  اسم المتغير  :  نوعه  ;
```

أو:

```
var  اسم المتغير  =  قيمة معينة  ;
```

وفي الطريقة الثانية فان القيمة المسندة إلى المتغير سوف تحدد نوعه.

واسم المتغير لايجوز أن يبدأ برقم أو أن يحتوي احد الرموز الخاصة مثل ($ ، # ، @ ، ! ، ...الخ) ماعدا الشارحة (_) فيجوز أن يبدأ بها أو أن يحتويها.

أمثلة على تعريف المتغيرات:

```
var  v1 : int ;
var  v1 = 5 ;
```

في الطريقتين تم تعريف متغير نوع عدد صحيح لايقبل الكسر العشري

```
var  v2 : float ;
var  v2 = 5.0 ;
```

في الطريقتين تم تعريف متغير نوع عدد نسبي يقبل الكسر العشري

```
var  v3 : double ;
```

تم تعريف متغير نوع عدد نسبي طويل يقبل الكسر العشري

```
var  v4 : boolean ;
var  v4 = false ;
```

في الطريقتين تم تعريف متغير منطقي يقبل القيمتين true ، false

```
var  v5 : GameObject ;
```

تم تعريف متغير نوع كائن لعبة له كافة الخصائص والصفات لأي مجسم ثلاثي الأبعاد.

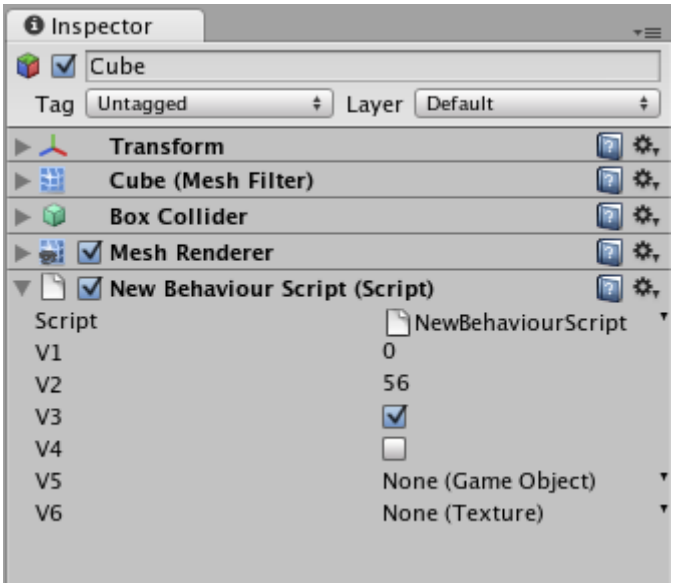
```
var v6 : Transform ;
var  v6 = Vector3( 5 , 0 , 6 );
var  v6 = Vector3.zero;
```

في الطرق الثلاثة تم تعريف متغير من نوع تحويل يمتلك كافة الدوال اللازمة لإجراء التحويلات المختلفة.

وهكذا مع بقية أنواع المتغيرات.....

وهذه المتغيرات بعد تعريفها وحفظ السكربت ووضعه على احد الكائنات في اللعبة تظهر أسفل السكربت في نافذة Inspector لهذا الكائن ،وعند تغيير قيمها من هذا المكان سوف يأخذ المتغير هذه القيمة بدلاً عن القيمة الموجودة في تعريف المتغير داخل السكربت ، وهذا مثال لسكربت يحتوي على متغيرات متنوعة وصورة توضح ظهورها في نافذة Inspector كما ذكرنا حيث يمكن تغير قيمها من هذا المكان:

```
var  v1 : int ;
var  v2 = 5.6 ;
var  v3 = true ;
var  v4 = false ;
var  v5 : GameObject ;
var  v6 : Texture ;

function Update () {

}
```



لاحظ ان المتغيرات الخاصة بمحرك الالعاب كالمتغيرين الاخيرين في هذا السكربت يكونا فارغين عند تعريفهم ، وسنتطرق الى كيفية استخدامهم عند وضع امثلة في موضوع الكلاسات كما ذكرنا.

ملاحظة لو عرفنا متغير لنتحكم من خلاله بالسرعة مثلاً

```
var speed = 5;
```

واردنا عند الضغط على زر معين زيادة هذا المتغير بمقدار 0.1

```
speed = speed + 0.1;
```

ففي هذه الحالة لايتم زيادة أي شيء الى المتغير والسبب اننا عرفنا المتغير على انه عدد صحيح لايقبل الكسور العشرية ولو كتبنا

```
speed = speed + 0.7 ;
```

فسوف يتم زيادة المتغير بمقدار واحد ،أي ان العملية هي تقريب إلى اقرب عدد صحيح وليس قطع الكسر العشري ، ولهذا يجب أن ننتبه إلى نوع المتغير عند التعامل معه في السكربت ويجب أن نكتب:

```
var speed =5.0;
```

مبدأ عمل جملة الشرط IF :

```
if ( شرط معين ) {

اذا تحقق الشرط يتم تنفيذ الأوامر المكتوبة هنا

}
```

والشرط يكون عبارة عن عمليات مقارنا بين قيمتين (< ، > ، == ، <> ، ...الخ) ، أو قيمة منطقية يتم الحصول عليها من فحص أمر معين وتكن True أو False ، وسنستخدم الطريقتين في أمثلة المواضيع القادمة.

ملاحظة : إذا كان الأمر المراد تنفيذه عند تحقيق شرط جملة IF هو أمر واحد فمن الممكن الاستغناء عن الأقواس { } .

ملاحظات حول كتابة الاكواد بلغة الجافا سكربت:

١- هذه اللغة حساسة لحالة الأحرف الصغيرة والكبيرة يعني لو كتبنا Var بدلاً من var سوف يظهر خطأ وكذلك لو كتبنا update() بدلاً من Update() وكذلك عند تعريف متغير يجب أن ننتبه إلى حالة أحرفه عند استخدامه في الكود.

٢- يمكن وضع تعليقات داخل الكود وهي لا تؤثر على عمله أي انه سوف يتجاهلها ، وذلك بكتابة علامتي سلاش // في بداية السطر ثم نكتب التعليق مثلاً نبذة عن وضيفة كل كواد ، ولو كنت تستخدم محرر الاكواد الافتراضي للمحرك وأردت الكتابة باللغة العربية يجب أن تذهب إلى

File → Encoding → UCS-2 Big Endian .

٣- كل سطر يجب أن ينتهي بالفارزة المنقوطة ; وهناك استثناءات مثلاً بعد جملة الشرط والدوال فلا توضع الفارزة المنقوطة.

٤- عند وجود أخطاء إملائية في السكربت فان المحرك سوف يكتشفها يظهر الخطأ ويشير إلى رقم السطر الموجود فيه الخطأ ، ويمكن إظهار أرقام الأسطر بالذهاب إلى

View → Line Numbers

٥- سكربت الجافا إذا كان فارغاً لايظهر أي خطأ بعكس سكربت ال C# وال Boo المستخدمين في اليونيتي فلو مسحت ما بداخلهم وجعلتهم فارغين فسيظهر خطأ.

٦- عند تكوين ملف سكربت جافا Assets → Create → JavaScript فان الدالة Update() يتم إضافتها تلقائياً لأهميتها حيث أنها تنفذ في كل فريم من فريمات اللعبة ولكتابة دوال أخرى فإنها تضاف خارج هذه الدالة وليس داخلها.

شرح بعض الكلاسات والدوال التابعة لها:

1- Input : الإدخال

بدأنا بهذا الكلاس لأهميته واستخدامه يعتبر من أهم واكبر الفروقات بين محركات الألعاب وبرامج التصميم الثلاثي الأبعاد التي تفتقر إلى إمكانية التعامل مع وحدات الإدخال بعد إنتاج الفيلم ، ويقصد بوحدات الإدخال مثلاً : لوحة المفاتيح ، الماوس ، عصا اللعب ،وغيرها...

ويتضمن الكلاس Input الدوال التالية:

1-1 Input.GetKey

```
Input.GetKey ("اسم الزر");
```

وتستخدم لمعرفة الزر المضغوط من لوحة المفاتيح ،وغالباً تكون ضمن شرط جملة if ليتم الفحص إذا كان الزر مضغوط أم لا.

مثال: تدوير مجسم 45 درجة على محور Y عند الضغط على الزر K

```
function Update () {

   if (Input.GetKey ("k")) {

         ransform.Rotate(0 , 45 , 0);
      }

}
```

فلو ضغطنا على الزر K فسوف يتم تحقيق الشرط وبالتالي سيتم تدوير المجسم وفي حالة استمرار الضغط على الزر فان المجسم سيستمر بالدوران أي ان الشرط يبقى متحققاً.

1-2 Input.GetKeyDown

```
Input.GetKeyDown ("اسم الزر");
```

وهي تشبه سابقتها ولكن الفرق عند استمرار الضغط على الزر لايبقى الشرط متحققاً أي ان الشرط يتحقق مرة واحدة عند الضغط على الزر

مثال:

```
function Update () {

   if (Input.GetKeyDown("k")) {

        ransform.Rotate(0 , 45 , 0);
    }
}
```

فلو ضغطنا على الزر K فسوف يتم تحقيق الشرط وبالتالي سيتم تدوير المجسم وفي حالة استمرار الضغط على الزر فلا يستمر دوران المجسم أي ان الشرط لا يبقى متحققاً.

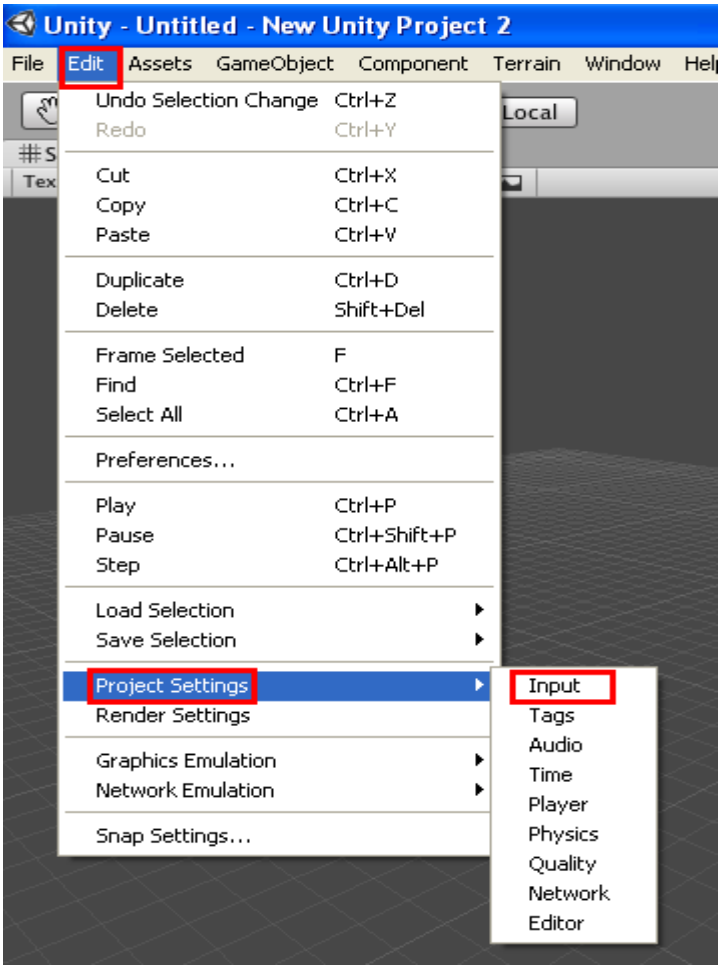
1-3 Input.GetKeyUp

```
Input.GetKeyUp ("اسم الزر");
```

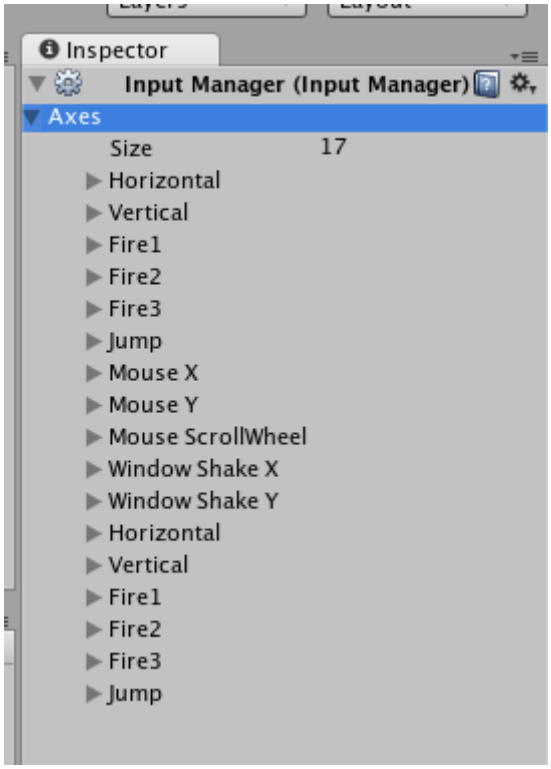
وتستخدم لمعرفة الزر الذي رفع عنه الضغط من لوحة المفاتيح ، ويتم تحقيق الشرط بعد رفع الضغط عن الزر.

1-4 Input.GetButton , Input.GetButtonDown , Input.GetButtonUp

هذه الدوال تعمل عمل الدوال الثلاثة السابقة ولكن الفرق الوحيد هو في اسماء الأزرار ،حيث ان الدوال الثلاثة السابق تستخدم أسماء الأزرار الاعتيادية في لوحة المفاتيح ،أ ما هذه الدوال تستخدم أسماء افتراضية لتشير إلى أزرار معينة من لوحة المفاتيح أو عصا اللعب أو الماوس ويتم تحديد هذه الاسماء ومايقابلها من ازرار كالتالي:

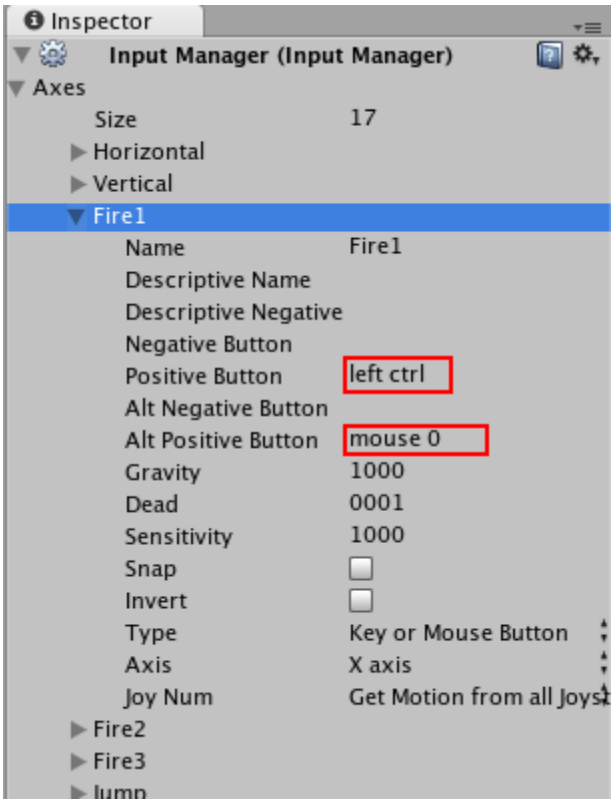


سوف تظهر في نافذة (Inspector) القائمة التالية الخاصة بالادخال



ستلاحظ أن هناك أسماء مكررة والسبب هو لاستخدام نفس الاسم ليقابل زر من أزرار لوحة المفاتيح وعصا اللعب في نفس الوقت. بمعنى أخر إمكانية التحكم باللعبة بواسطة لوحة المفاتيح وعصا اللعب في نفس الوقت.

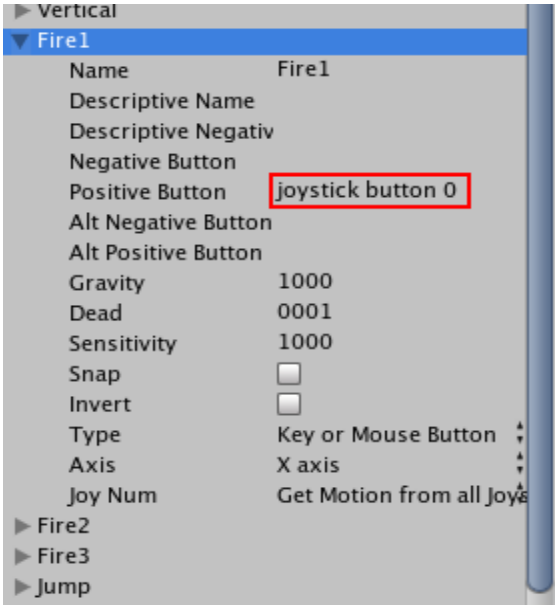
لنأخذ مثال الادخال (Fire1)



لاحظ ان الزر المقابل لهذه الاسم هو زر "كنترول يسار" والزر البديل هو "الزر الايسر للماوس"

ملاحظة: ممكن تغيير هذه الازرار حسب الرغبة وذلك عن طريق كتابة اسم الزر المرغوب بدل من الازرار الافتراضية المكتوبة أعلاه.
(سأضع قائمة بأسماء الأزرار في نهاية شرح دوال ال Input)

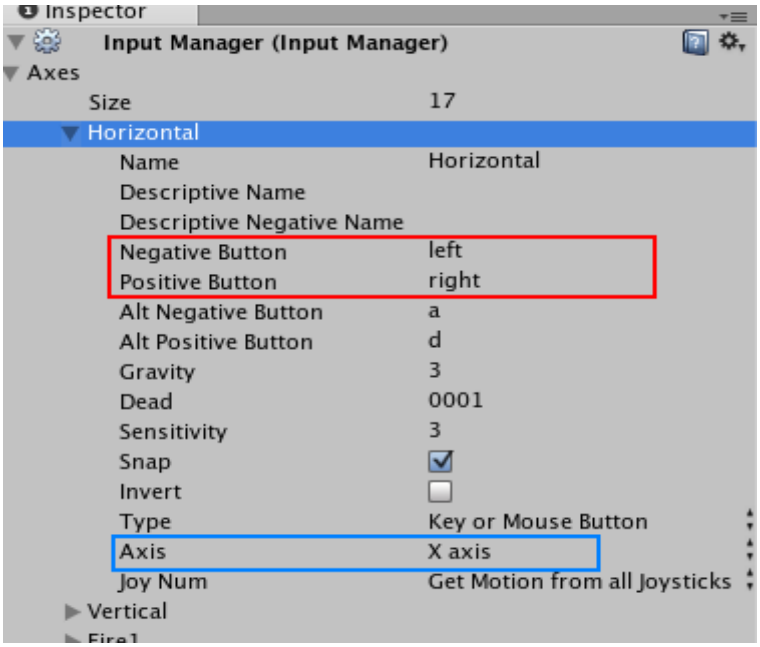
انزل للأسفل الى Fire1 الثانية ستجدها خاصة بعصا اللعب حيث تم تحديد الزر رقم صفر من أزرار عصا اللعب (لاتسألني الزر رقم صفر مإذا يقابل بالفعل ،لا اذكر في حياتي لمست عصا اللعب الخاص بالحاسوب ،فقط الخاص بالاتاري أيام زمان...)



فمثلاً لو استخدمنا ال Fire1 في السكربت لتنفيذ امر معين مثلاً فسيكون من الممكن استخدام الثلاثة أزرار المذكورة أعلاه وكلها تحقق الشرط
وطريقة الاستخدام:

```
function Update () {

   if (Input.GetKeyButton("Fire1")) {

      الأمر المراد تنفيذه//
    }
}
```

لنأخذ الإدخال Horizontal لان هناك فائدة أخرى لم تستخدم في المثال أعلاه



قد عرفنا الفائدة الأولى من استخدام هذه الاسم ليشير الى أزرار في لوحة المفاتيح أو غيرها من وحدات الإدخال وهنا قد أشار إلى زر الاتجاه الأيسر و زر الاتجاه الأيمن وبدائلهما الزرين a و d على التوالي ،والفائدة الثانية انه كما ترى زر الاتجاه الأيسر مخصص للقيمة السالبة (Negative) وزر الاتجاه الأيمن مخصص للقيمة الموجبة (Positive) هذه القيم الموجبة والسالبة تعود لأحد المحاور الثلاثة X أو Y أو Z يتم تحدد المحور المرغوب من خاصية Axis في الأسفل حيث انها حالياً مخصصة لمحور X ،كما يمكن التحكم بحساسية تلك القيمة ، فلو تم الضغط على زر الاتجاه الأيمن (أو مايقابله: a) فسنحصل على قيمة موجبة على محور X وإذا ضغطنا على زر الاتجاه الأيسر(أو مايقابله: d) فسنحصل على قيمة سالبا على محور X.

وطريقة الاستخدام لو أردنا تحريك مجسم على محور X فبدلا من أن نستخدم الدالة GetKey مرتين في جملتين if واحدة للضغط على الاتجاه الأيمن والتحريك لليمين و الأخرى للضغط على الاتجاه الأيسر والتحريك لليسار يعني كالتالي:

```
function Update () {

   if(Input.GetKey("right")) {

        transform.Translate( 1 , 0 , 0 );
      }

  if(Input.GetKey("left")) {

       transform.Translate( -1 , 0 , 0 );
     }

}
```

نستخدم بدلاً عن ذلك الإدخال Horizontal لكي نختصر السكربت كالتالي:

```
function Update () {

transform.Translate(Input.GetAxis("Horizontal") ,0,0);

}
```

علاوة على ذلك فان هذا السكربت يستخدم الأزرار البديلة وعصا الألعاب كما مذكور أعلاه.

وما ذكر عن Horizontal ينطبق على Vertical ولكنا تستخدم محور Y وغالباً ما تستخدم للتحريك على محور Z ،يعني لو أردنا كتابة سكربت للتحريك على الاتجاهات الأربعة (يمين ،يسار ، أمام ، خلف) يكون الكود :

```
function Update () {

transform.Translate(Input.GetAxis("Horizontal") ,0,0);

transform.Translate(0 ,0, Input.GetAxis("Vertical"));

}
```

مع إمكانية تقليل السرعة وذلك أما بتقليل حساسية قيمة هذين الادخالين عن طريق خاصية (sensitivity) الموجودة في الصورة أعلاه ، أو عن طريق الضرب في معامل الوقت.

```
transform.Translate(Input.GetAxis("Horizontal") * Time.deltaTime, 0,0);
```

وذلك ليتم حساب قيمة التحريك لكل ثانية وليس لكل فريم من فريمات اللعبة ، يعني سيتحرك كذا وحدة بالثانية وليس بالفريم.

1-5 Input.GetMouseButton

```
Input.GetMouseButton ("رقم زر الماوس");
```

وتستخدم لمعرفة الزر المضغوط للماوس،وغالباً تكون ضمن شرط جملة if ليتم الفحص إذا كان الزر مضغوط أم لا.

*قيم أزرار الماوس :

الزر الأيسر للماوس = 0 ،
الزر الأيمن للماوس = 1 ،
الزر الوسط للماوس = 2 .

مثال: عند الضغط على الزر الأيسر للماوس يتحرك المجسم للأمام وعند الضغط على الزر الأيمن للماوس يتحرك المجسم للخلف:

```
function Update () {

      if ( Input.GetMouseButton(0) ){

        transform.Translate(0 ,0 , 0.1);
      }

   if ( Input.GetMouseButton(1) ){

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

عند استمرار الضغط على أزرار الماوس يستمر تحقيق الشرط بالتالي يستمر التحريك.

1-6 Input.GetMouseButtonDown

```
Input.GetMouseButtonDown ("رقم زر الماوس");
```

أيضاً تستخدم لمعرفة الزر المضغوط للماوس ،ولكن عند استمرار الضغط على زر الماوس لايستمر تحقيق الشرط ،أي ان الشرط يتحقق مرة واحدة عند ضغط الزر ، استخدمها في المثال السابق لترى الفرق.

1-7 Input.GetMouseButtonUp

```
Input.GetMouseButtonUp ("رقم زر الماوس");
```

وتستخدم لمعرفة الزر المرفوع عنه الضغط من الماوس ، ويتم تحقيق الشرط بعد رفع الضغط عن الزر ، ممكن أن تستخدمها في المثال السابق لتشاهد الفرق بينها وبين سابقتها.

وهذه رموز الأزرار كما موجودة في ملف المساعدة للبرنامج:-

الأزرار الاعتيادية يجب كتابتها صغيرة. ... Normal keys: "a", "b", "c"

أزرار الأرقام الاعتيادية Number keys: "1", "2", "3", ...

أزرار الاتجاهات الأربعة Arrow keys: "up", "down", "left", "right"

الأزرار التي تعمل مع (Num Lock)

Keypad keys: "[1]", "[2]", "[3]", "[+]", "[equals]"

واضحين من أسمائهم ما عدا أخر اثنين فهم خاصين بلوحة مفاتيح نظام التشغيل ماكنتوش
Modifier keys: "right shift", "left shift", "right ctrl", "left ctrl", "right alt", "left alt", "right cmd", "left cmd"

أزرار الماوس Mouse Buttons: "mouse 0", "mouse 1", "mouse 2", ...

أزرار عصا اللعب
Joystick Buttons (from any joystick): "joystick button 0", "joystick button 1", "joystick button 2", ...

Joystick Buttons (from a specific joystick): "joystick 0 button 0", "joystick 0 button 1", "joystick 1 button 0", ...

الأزرار الخاصة وهي واضحة من أسمائها
Special keys: "backspace", "tab", "return", "escape", "space", "delete", "enter", "insert", "home", "end", "page up", "page down"

أزرار الوظائف
Function keys: "f1", "f2", "f3", ...

2- Transform:

ويستخدم هذا الكلاس لإجراء التحويلات على المجسمات ويقصد بالتحويلات ، النقل(Translate) ، والتدوير(Rotate) ، والتحجيم (Scale) .

2-1 Rotate : التدوير

```
transform.Rotate ( x , y , z );
```

وتستخدم لتدوير المجسم على المحاول الثلاثة ،فلو أردنا تدوير المجسم على محور معين نضع القيمة المطلوبة في مكان المحور ونضع صفر في مكان المحاور الأخرى ، وهذه القيمة تعتبر زاوية الدوران (وهي زاوية دائرية وليس ستينية يعني لاحاجة للتحويل من الستيني إلى الدائري).

مثال : اجعل المجسم يدور 50 درجة حول محور Y عند الضغط على زر R

```
function Update () {

if (Input.GetKeyDown("r") ) {

    transform.Rotate(0 , 50 , 0);

        }
}
```

فعند الضغط على الزر R سوف يتم إضافة 50 درجة إلى تدوير المجسم على محور Y بغض النظر هل تم تدويره سابقاً أم لا يعني لو ضغطنا زر R مرة أخرى سوف يتم إضافة ٥٠ درجة أخرى فيصبح تدوير المجسم على محور Y 180 درجة وهكذا بالتراكم ، ومعنى ذلك انه يقابل التدوير عن طريق ال offset بالثري دي ماكس وليس عن طريق ال absolute.

ولو أردنا التدوير بالعكس نجعل قيمة الزاوية بالسالب:

```
transform.Rotate(0 , -50 , 0);
```

و لاستخدام التدوير المطلق absolute بمعنى لو أردنا تدوير المجسم 50 درجة من الصفر أي إلغاء التدوير السابق وتطبيق التدوير الجديد نستخدم الصيغة التالية لحل المثال السابق:

```
function Update () {

if (Input.GetKeyDown("r") ) {

    transform.eulerAngles.y = 50 ;

        }
}
```

ولكي تتضح الصورة في الفرق بين ال Rotate و ال eulerAngles قم بتدوير المجسم يدوياً من نافذة التصميم على محور X وغير الأمر في المثال السابق إلى

```
transform.Rotate(0 , 0 , 0);
```

شغل اللعبة واضغط على زر R سوف لا يحصل أي تدوير ويبقى المجسم على التدوير اليدوي السابق ،

غير الأمر إلى التالي

```
transform.eulerAngles.x = 0 ;
```

شغل اللعبة واضغط على زر R سوف يتم إلغاء تدوير المجسم السابق على محور X .

ولو أردنا إلغاء تدوير المجسم على جميع المحاور نستخدم:

```
transform.eulerAngles =Vector3(0 ,0 , 0);
```

أو :

```
transform.eulerAngles =Vector3.zero ;
```

ملاحظة: عند استخدام هذه الطريقة للتدوير على محور واحد مثل:

```
transform.eulerAngles =Vector3 (0 , 45 ,0);
```

فانه سيتم الغاء التدويرات السابقة على المحاور الأخرى. ولكن استخدامها يحل لنا بعض مشاكل التدوير إذا حصلت في الطريقة الاعتيادية Rotate.

وممكن الحصول من طريقة ال eulerAngles على قيمة زاوية التدوير للمجسم :

مثال: قم بجعل المجسم يدور على محور Z وإذا وصلت درجة الدوران اكبر من 90 درجة يتم ارجاع التدوير 50 درجة للخلف على نفس االمحور.

```
function Update () {

transform.Rotate(0 , 0 , 1);

if (transform.eulerAngles.z > 90 ) {

    transform.Rotate(0 , 0 , -50);

        }
}
```

شغل اللعبة ..... حركة حلوة ^_^ .

والسؤال الأن كيف اجعل ال eulerAngles تعمل عمل ال Rotate الطريقة بسيطة وذلك بان نضيف قيمة التدوير القديم إلى التدوير الجديد كالتالي

```
transform.eulerAngles.z = transform.eulerAngles.z + 30;
```

واختصاراً

```
transform.eulerAngles.z += 30;
```

(انتبه لاتترك مسافة بين علامتي =+ فسيظهرخطأ)

وهذا يكافئ الأمر

```
transform.Rotate(0 , 0 , 30);
```

ولو كان التدوير بالعكس نكتب :

```
transform.eulerAngles.z -= 30;
```

وهذا يكافئ الأمر

```
transform.Rotate(0 , 0 , -30);
```

والأن انظر إلى المثال التالي:

```
function Update () {

    transform.Rotate(0 , 0 , 5);

}
```

عند تنفيذ هذا السكربت سيتم تدوير المجسم 5 درجات على محور Z في كل فريم بشكل مستمر ، ولو أردنا ان يتم التدوير 5 درجات في كل ثانية وليس في كل فريم نضرب في معامل الوقت كما في التالي:

```
function Update () {

    transform.Rotate(0 , 0 , 5 * Time.deltaTime);

}
```

واخيراً هناك طريقة للتدوير transform.rotation لكن معقدة وتدخل في مشاكل كثيرة ان لم تستخدمها بدقة.

2-2 Translate : النقل (التحريك)

transform.Translate(x , y , z);

وتستخدم لتحريك المجسم على المحاور الثلاثة ،فلو أردنا تحريك المجسم على احد المحاور نضع القيمة المطلوبة في مكان المحور ونضع صفر في مكان المحاور الأخرى.

وهي تقابل التحريك عن طريق ال offset بالثري دي ماكس وليس عن طريق ال absolute ،يعني يتم اضافة القيمة المعطاة إلى القيمة القديمة لموقع المجسم وهكذا بالتراكم.

مثال: اجعل المجسم يتحرك على الاتجاه الامامي لمحور Z عند الضغط على زر الاتجاه الاعلى أو زر W وعلى الاتجاه الخلفي لمحور Z عند الضغط على زر الاتجاه الاسفل أو زر S ، ويدور لليمين حول محور Y عند الضغط على زر الاتجاه الايمن أو زر D ،ويدور لليسار حول محور Y عند الضغط على زر الاتجاه الايسر أو زر A .

```
function Update () {

  if(Input.GetKey("up") || Input.GetKey("w")) {

      transform.Translate(0, 0 , 0.1 );
    }

 if(Input.GetKey("down") || Input.GetKey("s")) {

     transform.Translate( 0 , 0 , -0.1 );
   }

  if(Input.GetKey("right") || Input.GetKey("d")) {

      transform.Rotate(0, 1 , 0 );
    }

 if(Input.GetKey("left") || Input.GetKey("a")) {

     transform.Rotate( 0 , -1 , 0 );
   }

}
```

هذا السكربت يعمل بصورة صحيحة ولكنه ليس حلاً ذكياً ، أتذكر الحل الذكي؟ فقد تطرقنا له سابقاً وهو كالتالي:

```
function Update () {

transform.Translate( 0 , 0 , Input.GetAxis("Vertical") );

transform.Rotate( 0 , Input.GetAxis("Horizontal") , 0);

}
```

وسبب كتابتي الحل الاول لكي اتطرق إلى موضوع اداة الربط (أو ||) و اداة الربط (و &&) .

اداة الربط (أو ||) : إذا تحقق احد الشرطين أو كلاهما يتم تحقيق شرط جملة IF وبالتالي تنفيذ ما بداخلها ، بمعنى اخر الحالة الوحيدة التي لايتم الدخول إلى جملة IF هي عندما يكون الشرطين غير متحققين.

والحل الأول للمثال السابق هو مثال على اداة الربط " أو ||".

اداة الربط (و &&) : إذا لم يتحقق احد الشرطين أو كلاهما لايتم الدخول إلى جملة IF ،بمعنى اخر الحالة الوحيدة لدخول جملة IF هي ان يكو الشرطين متحققين.

مثال: عند الضغك على زري كنترول يمين و X معاً يتم تدوير المجسم على محور X .

```
function Update () {

  if(Input.GetKey("right ctrl") && Input.GetKey("x")) {

     transform.Rotate(1, 0 , 0 );
   }
  }
```

لاحظ إذا تم الضغط على احد الزرين فقط لايتم تحقيق الشرط ،الا إذا تم الضغط على الزرين.

ملاحظة : إذا شغلت اللعبة من داخل المحرك اضغط على زر X ثم كنترول يمين ، اما إذا شغلتها من الملف التنفيذي فلا يفرق بأي زر تبدأ.

ولو أردنا استخدام النقل المطلق يعني نقل المجسم إلى موقع معين وليس تحريكه مثلاً لو أردنا نقل الجسم إلى النقطة (1 , 3 , 5) نستخدم التالي:

```
transform.position=Vector3( 5 , 3 , 1 ) ;
```

ولو أردنا النقل على محور معين مثلاً محور X نكتب :

```
transform.position.x = 5 ;
```

ولا نكتب :

```
transform.position=Vector3( 5 , 0 , 0 );
```

لان ذلك سوف يجعل محوري Y و Z كليهما صفر (في حالة رغبنا في ذلك فلا مانع).

والسؤال البديهي كيف اجعل طريقة ال position تعمل عمل Translate ، والجواب بسيط وذلك بان نجمع القيمة الجديدة مع القيمة القديمة:

```
transform.position.z = transform.position.x + 0.1 ;
```

واختصاراً

```
transform.position.z += 0.1 ;
```

وهذا يكافىء الامر

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

ولو كان التحريك بعكس المحور نكتب:

```
transform.position.z -= 0.1 ;
```

وهذا يكافىء الامر

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

في بعض الاحيان هذه الطريقة تحل لنا مشاكل وخصوصاً إذا استخدمنا التدوير والتحريك معاً على نفس المجسم كما في عجلات السيارة.

ولتوضيح الفرق بين ال Translate وال position ، ضع مجسماً وغير موقعه يدوياً بحيث لايكون في الموقع صفر (نقطة الأصل) ونفذ الامرين:

```
transform.Translate( 0 , 0 , 0 );
```

سوف لايتم تغيير موقع المجسم لاننا قلنا له اضف القيمة صفر إلى قيم موقع المجسم ومن المعروف ان الصفر لايؤثر على عملية الاضافة أو الجمع.

```
transform.position=Vector3( 0 , 0 , 0 );
```

أو

```
transform.position=Vector3.zero ;
```

سوف تلاحظ تغيير موقع المجسم لاننا قلنا له انقل المجسم إلى نقطة الأصل الصفر.

وممكن الحصول من طريقة ال position على قيم موقع المجسم :

مثال: لو كان لديك مجسم يتحرك على محور X وإذا تجاوز النقطة 15 على نفس المحور يتم ارجاعه للنقطة صفر على نفس المحور :

```
function Update () {

transform.Translate(0.5 , 0 , 0);

if (transform.position.x > 15 ) {

    transform.position.x = 0 ;

      }
}
```

والأن قم بنسخ المجسم عدة نسخ واجعلهم واحد خلف الاخر على محور X وشغل اللعبة ^_^ .

* انظر إلى المثال التالي:

```
function Update () {

 if ( Input.GetKey("up") ) {
    transform.Translate(0 , 0 , 3);
    }
}
```

عند تنفيذ هذا السكربت سيتم تحريك المجسم بمقدار 3 وحدات على محور Z في كل فريم من فريمات اللعبة ، ولو أردنا ان يتم التحريك ٣ وحدات في كل ثانية وليس في كل فريم نضرب في معامل الوقت كما في التالي:

```
function Update () {

 if ( Input.GetKey("up") ) {
    transform.Translate(0 , 0 , 3 * Time.deltaTime);
    }
}
```

2-3 Scale : التحجيم (التكبير والتصغير)

في هذا التحويل طريقة ال offset غير جاهزة كما في التحويلين السابقين ،ويوجد طريقة لاستخدامها ولكن بعد ان نتطرق إلى الطريقة المطلقة absolute ، والتحجيم سيكون محلياً فقط (Locale) وليس عاماً (Global) لانه يفي بالغرض.

```
transform.localScale=Vector3( x , y , z );
```

وتستخدم لتحجيم المجسم على المحاور الثلاثة ،فلو أردنا تغيير حجم المجسم على احد المحاور نضع القيمة المطلوبة في مكان المحور ونضع واحد في مكان المحاور الأخرى.

فإذا أردنا التكبير نضع قيما موجبة اكبر من الواحد وإذا أردنا التصغير نضع قيمة موجبة اصغر من الواحد ، وإذا وضعنا واحد لايتم تغيير أي شيء.

فالتحجيم كما هو معروف في برامج التصميم الثلاثي مثل الماكس يعتمد على النسبة المئوية بالنسبة لحجم المجسم الحالي ، يعني يعتبر الحجم الحالي هو 100% فلو أردنا التكبير نضع قيمة اكبر مثلاً 150% ،وإذا أردنا التصغير نضع قيمة اصغر مثلاً 50% ،ولو وضعنا 100% لايتم تغيير أي شيْ.

هنا نفس الشيء ولكن بدل التعامل مع النسبة 100 يتم التعامل مع النسبة 1 .

مثال: عند الضغط على الزر S يتم تكبير المجسم بمقدار ضعف على كل المحاور.

```
function Update () {

if (Input.GetKeyDown("s") ) {

transform.localScale =Vector3( 2 , 2 , 2 );
}
```

فعند الضغط على زر S سيتم تكبير المجسم بقدر حجمه ، أي سيصبح حجمه ضعف حجمه قبل التكبير، وعند الضغط مرة أخرى على الزر لايتم تغيير أي شيء ،فنحن قلنا ان هذه الطريقة مطلقة وليس تراكمية.

ولو أردنا تصغير المجسم إلى نصف حجمه نكتب:

```
transform.localScale =Vector3( 0.5 , 0.5 , 0.5 );
```

مإذا يحدث لو كتبنا :

```
transform.localScale =Vector3( 1 , 1 , 1 );
```

الجواب معروف انه لايتم تغيير أي شيء.

ولو كان المطلوب التكبير بمقدار نصف حجمه نكتب:

```
transform.localScale =Vector3( 1.5 , 1.5 , 1.5 );
```

والأن مإذا يحدث لو كتبنا:

```
transform.localScale =Vector3( 0, 0, 0);
```

سوف يختفي المجسم لأننا جعلنا حجمه صفر ،ولهذا يجب التنبيه لو أردنا مثلاً تكبير المجسم بنسبة 0.2 على محور Y فقط فمن الغلط أن نكتب:

```
transform.localScale =Vector3( 0 , 1.2 , 0 );
```

ففي هذه الحالة سيختفي المجسم لان تم جعل التكبير على محوري X و Z صفر ، فيجب ان نكتب:

```
transform.localScale =Vector3( 1 , 1.2 , 1 );
```

أو بطريقة أمن كالتالي:

```
transform.localScale.y = 1.2;
```

ولو أردنا التصغير بنسبة 0.2 على محور Y نكتب :

```
transform.localScale.y = 0.2;
```

والأن نرجع إلى طريقة ال offset للتحجيم ، والتي أسميتها بالتراكمية لأنه لايتم إلغاء التحجيم السابق للمجسم بل يتم إضافة القيم الجديدة إليه بشكل تراكمي والطريقة هي بإضافة التحجيم الجديد إلى التحجيم القديم كالتالي:

```
transform.localScale +=Vector3( 0.1 , 0.1  , 0.1 );
```

انتبه هنا القيمة اصغر من واحد ولكن لايتم تصغير الحجم بل تكبيره لأننا أضفنا هذه النسبة إلى الحجم الحالي فبالتالي سيتم تكبيره.

ولو أردنا تصغير الحجم بهذه الطريقة نكتب :

```
transform.localScale -= Vector3( 0.1 , 0.1  , 0.1 );
```

ولو كتبنا :

```
transform.localScale +=Vector3( 0 , 0 , 0 );
```

لايتم تغيير أي شيء لأننا قمنا بإضافة صفر إلى الحجم القديم وبالتالي لايؤثر شيء.

مثال : يتم تكبير المجسم بنسبة 0.1 باستمرار على محور Y وإذا وصل إلى نسبة اكبر من أو تساوي 5 يتم إرجاعه إلى حجمه الأصلي.

```
function Update () {

transform.localScale += Vector3( 0 , 0.1 , 0 );

                    // transform.localScale.y += 0.1;  // أو

if (transform.localScale.y > = 5) {

transform.localScale  = Vector3( 1 , 1 , 1 );

                    // transform.localScale.y = 1;  // أو
    }
}
```

2-4 localRotation , localPosition

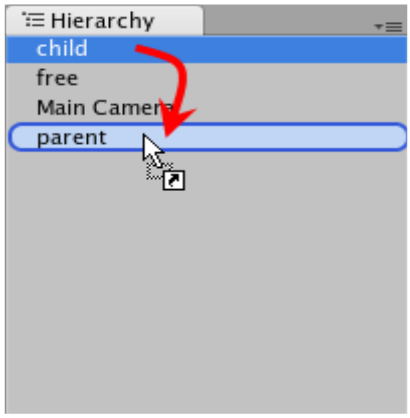
في الموضوع السابق شرحنا ال localScale وبقى لنا أن نتطرق إلى التحويلات (localRotation ، localPosition) من حيث الفرق الجوهري عند استخدامهم بالسكر بت عن استخدام التحويلات (position ، eulerAngles).

ملاحظة : لايشمل هذا الشرح التحويلات العامة (Global) والمحلية (Local) المعروفة فهذا موضوع اخر.

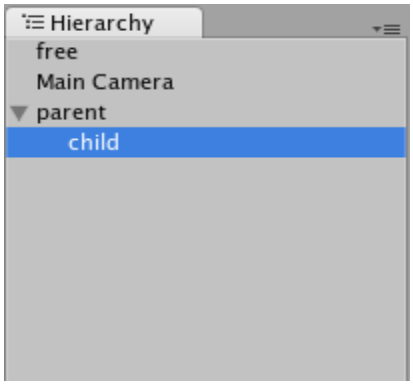
الفرق الجوهري عند استخدام الطريقتين في السكربت يحدده فيما إذا كان المجسم صاحب السكربت تابع (أبن) لمجسم أخر أم لا.

فلو كان المجسم غير تابع لمجسم أخر فلا يوجد فرق بين استخدام الطريقتين في التحويلات ، أما في حال كان المجسم تابع لمجسم أخر فعند استخدام التحويلات Local ستعتبر نقطة الأصل (الصفر) لهذا المجسم (الابن) هي النقطة التي تمثل موقع المجسم الأخر (الأب) ، أما عند استخدام التحويلات العادية فلا يتم اعتبار هذه الشيء.

مثال: ضع مكعب في نافذة التصميم وسميه "free" ،وضع مكعب اخر وسميه "parent" وغير موقعه من خاصية position في نافذة inspector إلى التالي (3 ، 0 ، 5) ، ضع مكعب اخر وسميه "child" ، الأن في نافذة Hierarchy اضغط على المكعب "child" ومع استمرار الضغط اسحبه فق المكعب "parent" وأفلته كما في الصورة



الأن أصبح المكعب "child" تابع للمكعب "parent" كما في الصورة:



الأن اكتب الكود التالي:

```
function Update () {

if (Input.GetKeyDown("m") ) {

   transform.position=Vector3.zero;

   }
}
```

ضع السكربت على المكعبين "free" و "child" ،شغل اللعبة واضغط على زر M ستلاحظ المكعبين تم نقلهما إلى نقطة الأصل (الصفر) ولكن يبدون كمكعب واحد لأنهم فوق بعض ولكي تتأكد من ذلك ، مع استمرار تشغيل اللعبة اذهب إلى نافذة Scene لتظهر نافذة التصميم ثم من نافذة Inspector اختر المكعبين لكي ترى موقعهم.

الأن غير الكود إلى:

```
function Update () {
if (Input.GetKeyDown("m") ) {

   transform.localPosition=Vector3.zero;
   }
}
```

شغل اللعبة واضغط على زر M سوف يتم نقل المكعب "free" إلى نقطة الأصل (الصفر) ،أما المكعب "child" فيتم نقله إلى موقع المكعب "parent" لأنه تابع له و كما قلنا عند استخدام التحويلات المحلية Local تعتبر نقطة الأصل للمجسم هي موقع المجسم الأب إن وجد.

الأن غير في الكود :

```
transform.localPosition.z = 1;
```

ستلاحظ ان المكعب "free" تم وضعه على بعد وحدة واحدة من نقطة الأصل (الصفر) بالنسبة لمحور Z ، بينما المكعب "child" تم وضعه على بعد وحدة واحدة عن المكعب الأب "parent"

مثال : ضع المكعبات الثلاثة في المثال السابق ، ولكن قبل أن تجعل المكعب "child" تابع للمكعب "parent" قم بجعل زاوية دوران المكعب "parent" 45 درجة على محور X ثم اجعل المكعب "child" تابع للمكعب "parent" ثم اكتب السكربت التالي :

```
 function Update () {

if (Input.GetKeyDown("m") ) {

   transform.eulerAngles =Vector3.zero;

   }
}
```

ضع السكربت على المكعبين "free" و "child" ،شغل اللعبة واضغط على زر M ،لايتم تغيير أي شيء في المكعبين والأن غير السكربت إلى:

```
function Update () {

if (Input.GetKeyDown("m") ) {

   transform.localEulerAngles =Vector3.zero;

   }
}
```

شغل اللعبة واضغط على الزر M سوف تلاحظ ان المكعب "free" لايتم تغيير أي شيء فيه ،أما المكعب "child" فان سوف يأخذ تدويراً يشبه تدوير المكعب "parent" أي سيستدير 45 على محور X لأننا استخدمنا التدوير المحلي.

2- 5 RotateAround : التدوير حول نقطة معينة

وتستخدم لدوران المجسم حول نقطة معينة وعلى محور معين

transform.RotateAround(Position , Axis , Angle) ;

Position : هي نقطة مركز الدوران ،أي التي سيدور المجسم حولها.

ويتم إدخالها بطريقتين
Vector3(x ,y ,z) : حيث يتم إدخال القيم المرغوبة بدل ال (x , y , z).
translate.position : في حالة رغبنا أن يكون مركز الدوران هو موقع مجسم معين.

Axis : محور الدوران ، بعد تحديد نقطة مركز الدوران يجب تحديد المحور الذي سوف يدور عليه المجسم حول نقطة المركز.

Angle : زاوية الدوران المرغوبة.

* طريقة تعيين محور الدوران Axis :

١- محور X :

بالاتجاه الطبيعي : (1 , 0 , 0) Vector3 أو Vector3.right
بالاتجاه العكسي :(-1 , 0 , 0) Vector أو Vector3.left

٢- محور Y :

بالاتجاه الطبيعي : (0 , 1 , 0) Vector3 أو Vector3.up
بالاتجاه العكسي :(0 , -1 , 0) Vector3

٣- محور Z :

بالاتجاه الطبيعي : (0 , 0 , 1) Vector3 أو Vector3.forward
بالاتجاه العكسي :(0 , 0 , -1) Vector3

٤- عدة محاور :

على محوري X و Y : (1 , 1 , 0) Vector3

على محوري X و Y ولكن حول محور Y اقرب مماهو حول محور X: (1 , 5 , 0) Vector3
يعني بشكل مائل.

وهكذا توضع القيم المرغوبة في مكان المحاور المرغوبة.

مثال : اجعل المجسم يدور حول نقطة الأصل (الصفر) بزاوية 90 درجة في الثانية على محور X .

```
function Update () {

transform.RotateAround(Vector3.zero,Vector3.right , 90 * Time.deltaTime);

}
```

مثال : اجعل المجسم يدور حول مجسم أخر اسمه "center" بزاوية 5 درجات على محور Z.

الطريقة الأولى :

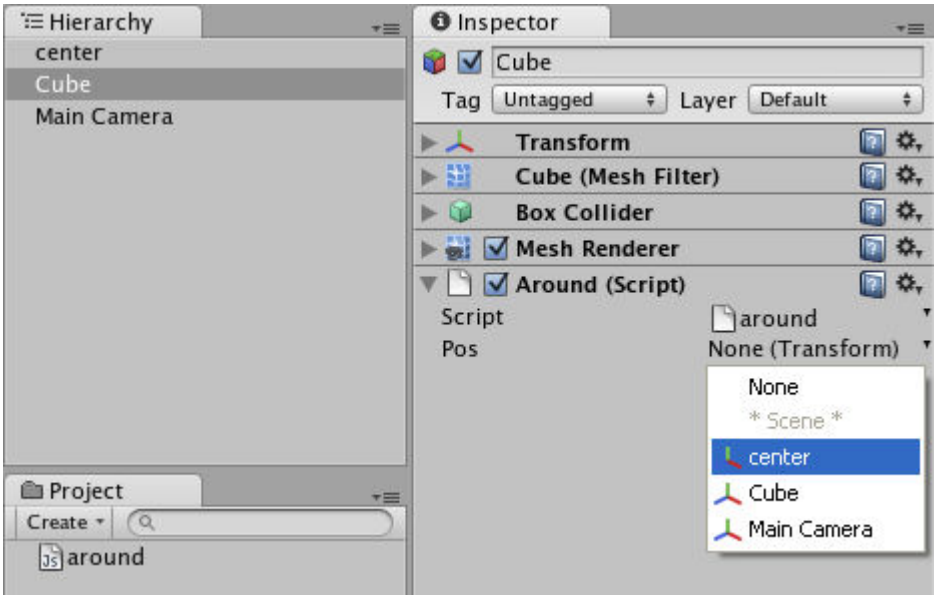
```
function Update () {

  var pos = gameObject.Find("center").transform.position;

  transform.RotateAround( pos ,Vector3. forward , 5 );

}
```

ضع السكربت على المجسم وشغل اللعبة سيدور حول المجسم "center".
ملاحظة اجعل المجسمين في موقعين مختلفين بالنسبة لمحور Z وإلا ستلاحظ ان المجسم يدور حول نفسه.

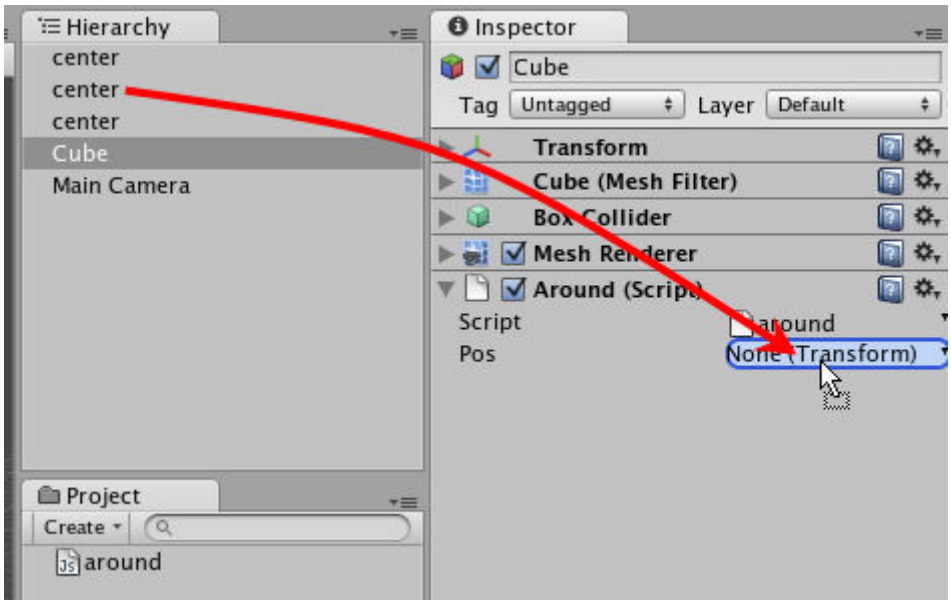
الطريقة الثانية :

```
var pos : Transform;
function Update () {

transform.RotateAround(pos.position,Vector3. forward , 5 );

}
```

ضع السكربت على المجسم ومن نافذة Inspector للمجسم اذهب للسكربت ستجد المتغير pos أسفل السكربت اسند إليه المجسم "center" كما في الصورة :



ربما يقول احدهم مالفرق بين الطريقتين ،إنا أرى الطريقة الأولى أسهل : الجواب ، لو كان يوجد أكثر من مجسم اسمه "center" ففي الطريقة الأولى ربما يأخذ السكربت المجسم الذي لانريده نحن ، ففي الطريقة الثانية نستطيع تحديد المجسم المطلوب حتى لو كان هناك مجسمات بنفس اسمه حيث ان هناك طريقة اخرى للإسناد للمتغير غير التي في الصورة أعلاه وهي السحب والإفلات كما في الصورة أدناه:



2-6 LookAt : التدوير باتجاه نقطة معينة

وتستخدم لجعل المجسم يتجه إلى نقطة معينة ،بمعنى جعل الاتجاه الأمامي لمحور Z للمجسم يتجه إلى تلك النقطة.

Transform.LookAt (position);

وطريقة إدخال النقطة position هي مثل طريقة إدخال ال position لل AroundRotate المشروحة أعلاه.

مثال : على نفس المثال السابق لل AroundRotate اكتب السكربت التالي :

```
var lookat : Transform;

function Update () {

transform.LookAt(lookat.position);

}
```

ضع السكربت على المجسم "center" وليس على المجسم الذي يدور حوله "Cube" ، في نافذة ال Inspector لل "center" اسند المكعب للمتغير lookat أسفل السكربت كما تعلمنا بطريق السحب والإفلات ، شغل اللعبة ستلاحظ ان الوجه الأمامي Z للمجسم "center" يتجه إلى المكعب وهو يدور حوله.

وممكن استخدام ال LookAt مع الكامرة لكي تنظر إلى مجسم معين أين ما تحرك.

3- Distance المسافة بين نقطتين

Vector3.Distance (position1 , position2)

تستخدم لمعرفة المسافة بين نقطتين في الفضاء الثلاثي الأبعاد وهي قيمة مطلقة أي لاتكون سالبة في أي حال من الاحوال.

مثال: يتم حساب المسافة بين المجسم صاحب السكربت ومجسم أخر إذا كانت اقل أو تساوي 3 وحدات يتم إعادة تحميل المرحلة الحالية من جديد

```
var other : Transform;
function Update () {

  if (Vector3.Distance(transform.position , other.position ) <= 3 ) {

      Application.LoadLevel(Application.loadedLevel);
   }
}
```

4 - Renderer

يستخدم هذا الكلاس للتحكم بمظهر المجسم .

4-1 renderer.enabled

renderer.enabled = false;

لإخفاء المجسم ولا يعني ذلك حذفه من الذاكرة بل يعتبر موجوداً فمثلا لو كان هذه المجسم جداراً واخفي بهذه الطريقة فلا يمكن اختراقه بل يعتبر موجود ويتم حساب التصادم له ولكنه لايرى.

renderer.enabled = true; لإظهار المجسم إذا كان مخفي

if (renderer.enabled) {

إذا كان المجسم غير مخفي يتم تنفيذ الكود الموجود هنا

}

if (renderer.enabled == false) {

إذا كان المجسم مخفي يتم تنفيذ الكود الموجود هنا

}

ملاحظة المساواة = في جملة IF تكتب مرتين == وبدون مسافة بينهم.

4-2 renderer.material

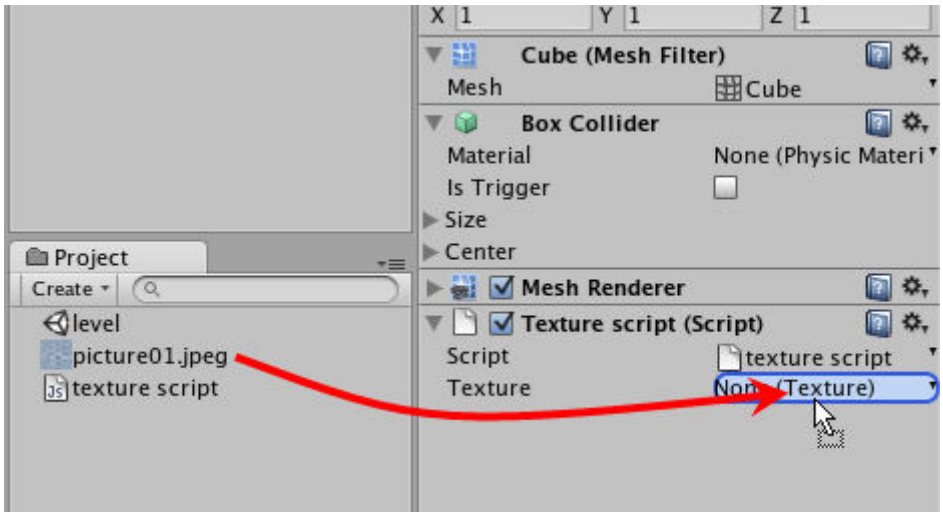
تستخدم للتحكم بال Texture وال Shader للمجسم ويفضل وضع material على المجسم ثم استخدام السكربت.

renderer.material.color = Color.red ; لتغيير اللون

ولتغيير ال Texture للمجسم عن طريق السكربت يجب أن نعرف متغير من نوع Texture ليتم وضع الصورة إليه ثم يسند إلى material كالتالي:

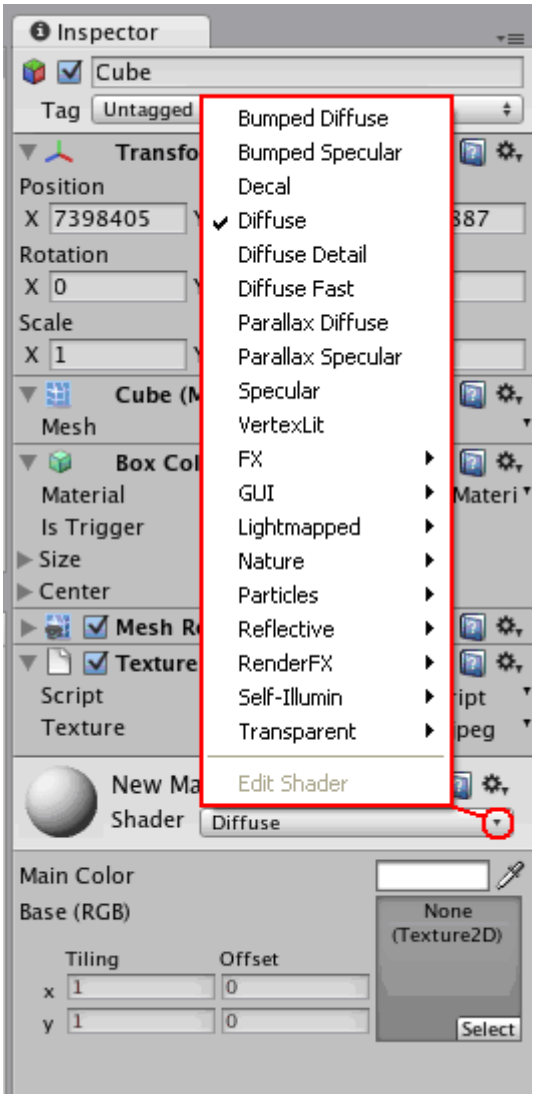
```
var texture : Texture;
function Update () {

  if ( Input.GetKeyDown("t") ) {

     renderer.material.mainTexture = texture;

    }
}
```

ضع السكربت على المجسم ثم اسند صورة إلى المتغير texture كما في الصورة:



شغل اللعبة واضغط على زر T ..

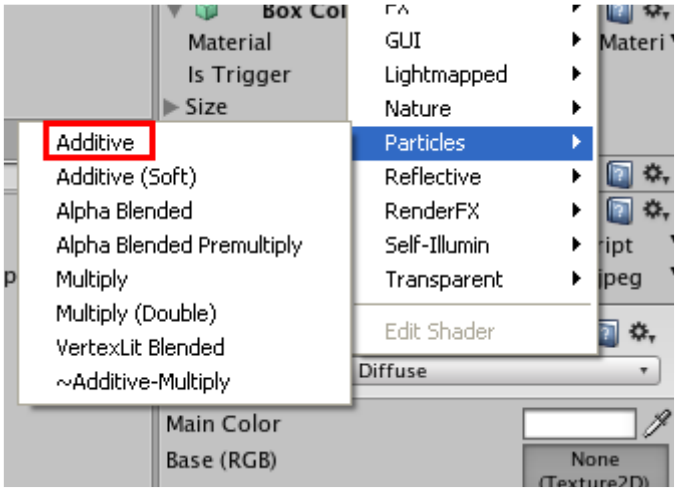
ولتطبيق ال shaders على المجسم يجب ان يكون ال shader موجود في قائمة ال shaders في ال material



ويجب أن تكتب بنفس اسمها في هذه القائمة وكلها تبدأ بحرف كبير مثلاً لو أردنا تطبيق shader ال Specular عند الضغط على زر T على فرض انك قد وضعت صورة في ال material :

```
function Update () {

  if ( Input.GetKeyDown("t") ) {

      renderer.material.shader = Shader.Find( "Specular" );


    }
}
```

وإذا أردنا أن نختار shader من قائمة فرعية مثل :



فنكتب التالي:

```
renderer.material.shader = Shader.Find( "Particles/Additive" );
```

والتحكم بال tiling للصورة في ال material يتم من خلال

```
renderer.material.mainTextureScale = Vector2 ( X , Y );
```

ونضع القيم المرغوبة بدل ال X وال Y.

والتحكم بال offset للصورة في ال material يتم من خلال

```
renderer.material.mainTextureOffset = Vector2( X , Y );
```

ونضع القيم المرغوبة بدل ال X وال Y.

مثال : ضع مجسم وضع عليه material وضع في ال material صورة واضبط ال tiling بشكل مناسب ، الأن سنكتب سكربت لتغيير ال offset الافقي x للصورة لنحصل على Texture متحرك .

```
var offset = 0.0;
function Update () {

 renderer.material.mainTextureOffset = Vector2 ( offset , 0 );

 offset = offset + 0.01;

}
```

والأن لو كان المجسم يحتوى على أكثر من material فان الأوامر السابقة تنفذ على أول واحد فقط وربما لايظهر الناتج على المجسم فيجب ان نستخدم:

renderer.materials[0] الأول

renderer.materials[1] الثاني

.....
.....

وهكذا وبنفس الاستخدامات السابقة..

طريقة الوصول من سكربت موضوع على مجسم معين إلى المجسمات الأخرى:

هذه الموضوع قد تطرقنا له ولكن هناك طريقة أخرى لم نذكرها لتصبح لدينا ثلاث طرق،

١ـ الطريقة الأولى :

في هذه الطريقة يجب ان نعرف اسم المجسم ويكتب كما هو من حيث الأحرف الكبيرة والصغيرة:

```
gameObject.Find("اسم المجسم")
```

فمثلاً لو كان السكربت موضوع على مجسم ونريد تدوير مجسم أخر اسمه "Cube" نكتب:

```
gameObject.Find("Cube").transform.Rotate(0 , 5 , 0 );
```

وكما ذكرنا سابقاً يكمن عيب هذه الطريقة بأن لو كان هناك عدة مجسمات بأسم Cube فان السكربت سيأخذ أول واحد فيهم فقط ويهمل البقية وربما يكون ليس هو المطلوب تدويره.

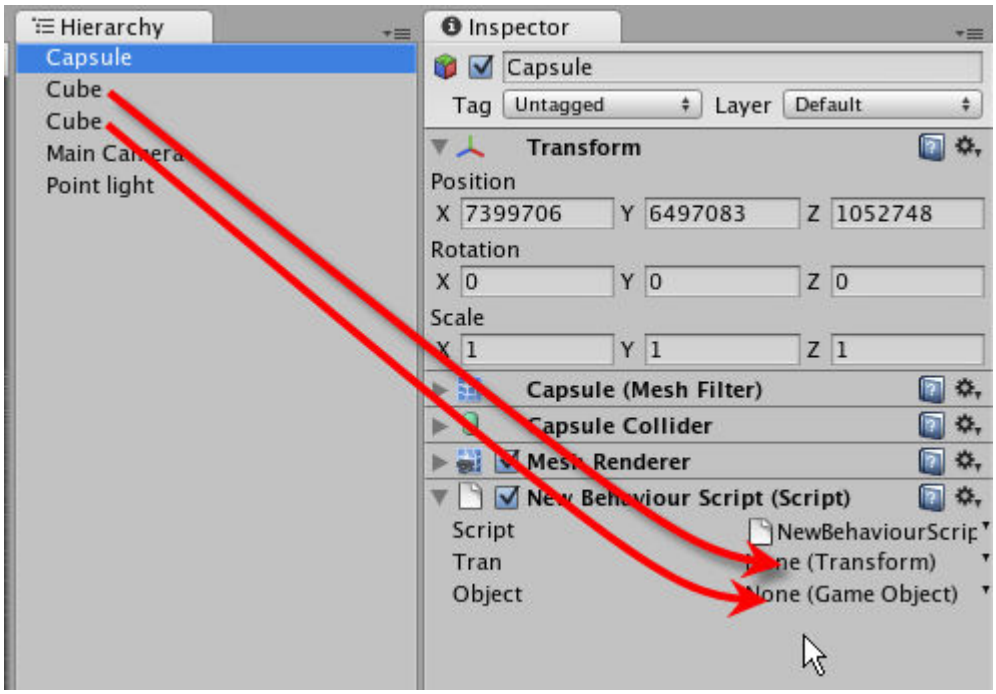
٢ـ الطريقة الثانية :

هذه الطريقة تحل لنا مشكلة تشابه أسماء المجسمات ،حيث يتم تعريف متغير من نوع GemeObject أو من نوع Transform إذا كنا محتاجين إجراء التحويلات فقط ، سنعرف متغيرين من هذين النوعين لتدوير مجسمين ليتضح الفرق:

```
Var tran : Transform;
Var object : GameObject;

function Update () {

tran.Rotate( 0 , 5 , 0 );
object.transform.Rotate( 5 , 0 , 0 );

}
```

الأن اتضح الفرق بين استخدام المتغيرين حيث ان النوع GameObject هو كائن لعبة حاله حال أي مجسم ويحتوي على جميع الكلاسات والدوال المشروحة سابقاً ، بينما النوع Transform فهو يستخدم لإجراء التحويلات فقط .

ضع السكربت على مجسم (Capsule) ثم من نافذة Inspector لهذا المجسم قم بإسناد مجسمين مختلفين (Cube) إلى المتغيرين tran و object



والأن أي النوعين من المتغيرين السابقين نستخدم ، بمعنى اخر مالفرق في استخدام النوعين:

الجواب : ان المتغيرات تحجز مساحة من الذاكرة وعند استخدامها تحجز من ذاكرة كارت الفيديو أيضاً ومن الواضح ان المتغير من نوع GameObject يأخذ مساحة اكبر لأنه يتعامل مع كلاسات ودوال كثيرة ،بينما المتغير من نوع Transform يتعامل مع كلاس ودوال التحويلات فقط ، ولهذا لو كنا نحتاج المتغير لإجراء التحويلات فقط كما في المثال السابق استخدمنا التدوير فقط فيكون المتغير من نوع Transform وهو الاختيار الأن سب ، أما إذا أردنا الوصول من المتغير إلى عدة أشياء مثلاً لإجراء تحويلات وتغيير اللون أو ال material وأشياء أخرى فنستخدم النوع GameObject.

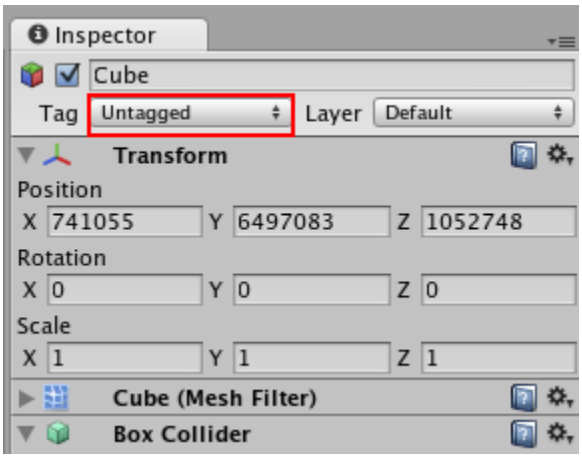
٣- الطريقة الثالثة :

لو كان عندنا مثلاً خمسة أبواب في اللعبة ، وكلهم يحتوون على Animation يحاكي فتح الباب وإغلاقه ، وأردنا أن نكتب سكربت لتشغيل ال Animation عند دفع الباب من قبل اللاعب ، فمن الغير عملي واحترافي أن نكتب خمس سكربتات يعني لكل باب سكربت لان احتمال أثناء تطوير اللعبة سيزيد من عدد الأبواب وبالتالي عدد السكربتات.

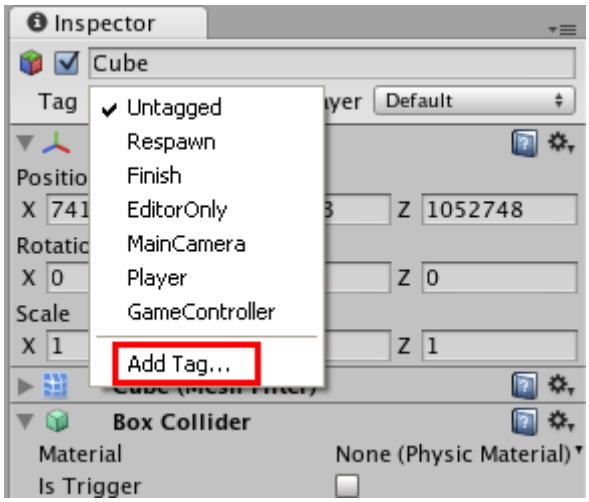
إذن سنكتب سكربت واحد ونضعه على مجسم اللاعب ، الطرق السابقة لاتنفع لان في الطريقة الأولى سيزيد عدد اسطر الكود ومشكلة أسماء المجسمات ،وفي الطريقة الثانية يجب أن نعرف متغير لكل باب إضافة إلى طول الكود .

أما الطريقة الذكية فانه يوجد هناك مايسمى بال Tag كنوع من التنظيم والتصنيف حيث ان كل مجسم في اللعبة يمكن ان يصنف إلى Tag معين له اسم معين ولو أردنا التعامل مع كل المجسمات المصنفة على Tag معين فاننا نتعامل مع اسم هذا ال Tag ليعوض عن كل هذه المجسمات.

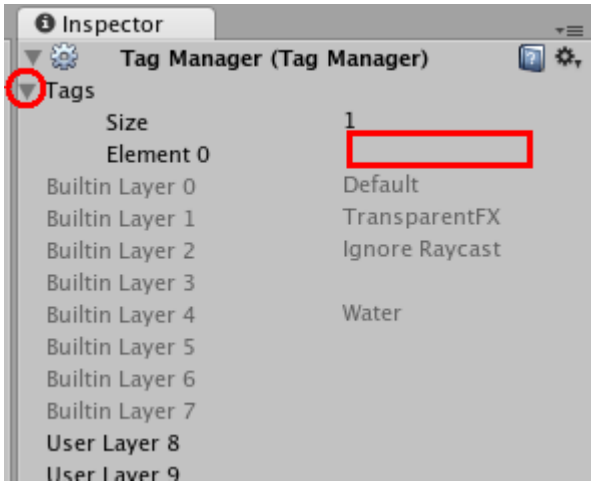
وخاصية ال Tag موجودة في نافذة ال Inspector وكل مجسم جديد يتم إضافته إلى اللعبة يكون بدون Tag أي يكون في صنف untagged كما في الصورة :



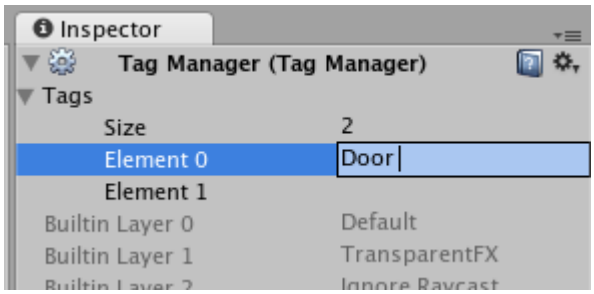
الأن لنأتي إلى طريقة إضافة Tag جديد بأسم Door لكي نصنف الأبواب على أساسه ، اضغط على كلمة untagged المؤشر عليها في الصورة أعلاه سوف تظهر قائمة بال Tags المتوفرة :



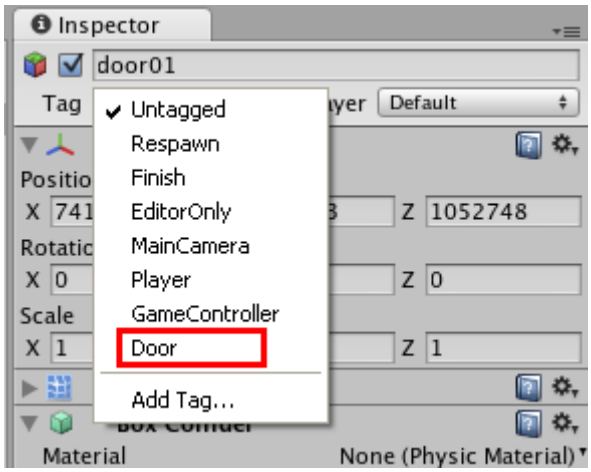
اختر Add Tag ستظهر النافذة التالية :



ربما يكون المثلث المؤشر عليه بدائر مغلق قم بالنقر عليه ليفح.
الأن قم بالنقر في المنطقة المؤشر عليها بمستطيل احمر واكتب كلمة Door بداخلها كما في الصورة:



اضغط انتر ، الأن تم إضافة Tag جديد ، اذهب إلى الأبواب واختار احدها ثم اذهب إلى نافذة ال Inspector وغير ال Tag إلى Door كما في الصورة :



غير ال Tag للأبواب الأخرى إلى Door كما في الصورة أعلاه ، واكتب السكربت التالي:

```
OnControllerColliderHit (hit : ControllerColliderHit) {
if ( hit.gameObject.tag == "Door" ){
hit.gameObject.animation.Play();
}
}
```

وإذا لم تكن تستخدم شخصية بل مجسم متحرك اكتب هذا السكربت :

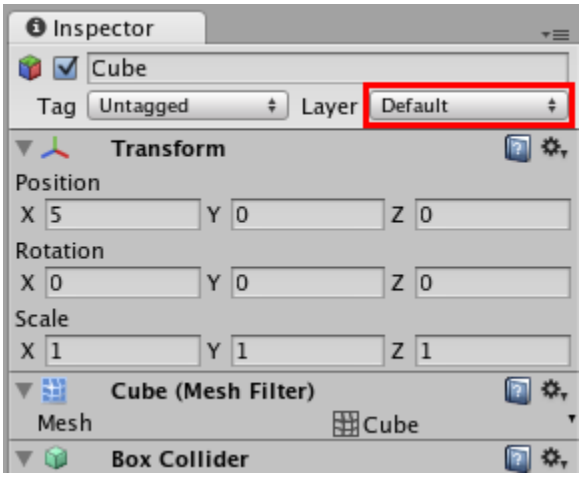
```
OnCollisionEnter (collision : Collision) {
if ( collision.gameObject.tag == "Door" ){
collision.gameObject.animation.Play();
}
}
```

ضع السكربت على الشخصية وشغل اللعبة ثم اذهب واصطدم بأي باب سوف يتم تشغيل ال Animation الخاص به ، وعند إضافة أبواب أخرى إلى اللعبة ماعليك سوى جعل ال Tag لها إلى Door وسيعمل السكربت معها.

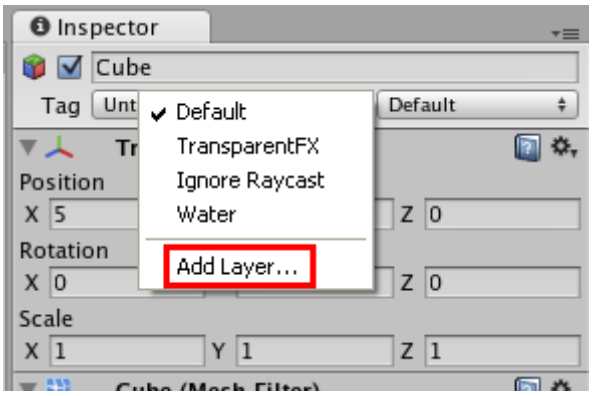
استخدامات الطبقات Layers

ال Layer هي أيضاً نوع من التنظيم والتصنيف فبعض الخصائص للكائنات تحتوي على خيار يتم من خلاله اختيار الطبقات (Layers) التي سيطبق عليها تأثير تلك الخاصية ، وكل مجسم جديد يتم إضافته إلى اللعبة تكون خاصية ال Layer له هي Default ، وسنأخذ مثال واحد يوضح فكرة إضافة Layer جديدة واستخدامها.

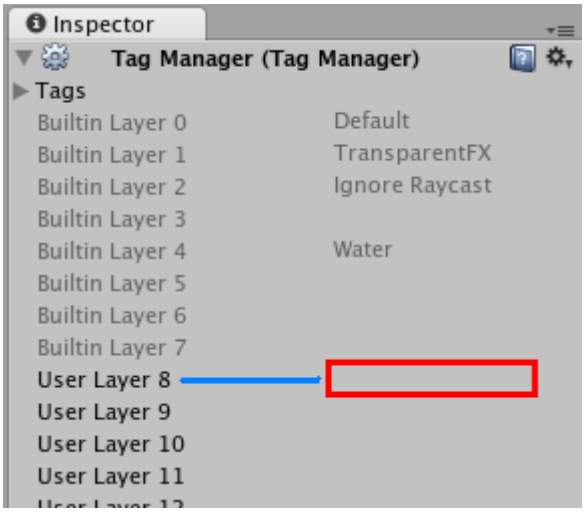
مثال : طريقة جعل بعض المجسمات لاتتأثر بالضوء المنبعث من كائن ضوء معين:
ضع عدة مجسمات ومصدر ضوء (مثلاً Point light) بحيث يغطي ضوئها كل المجسمات،
اختر أي مجسم ومن نافذة Inspector اضغط على خاصية ال Layer كما في الصورة:



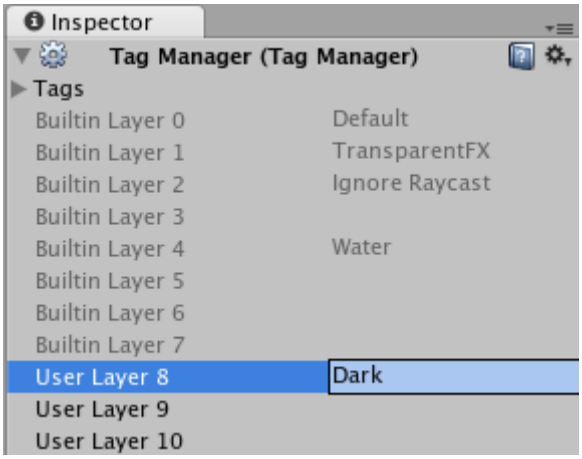
ستظهر قائمة بال Layers المتوفرة ، اختار Add Layer كما في الصورة:



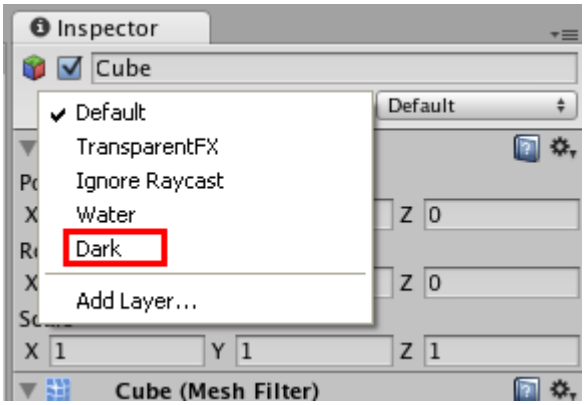
ستظهر القائمة التالية التي تحتوي على عدة طبقات غير مستخدمة:



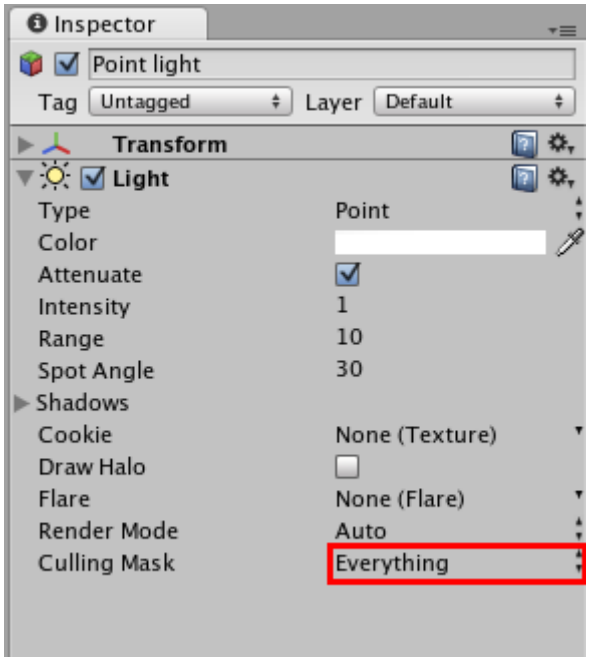
انقر على المنطقة المؤشر عليها بمستطيل احمر (تستطيع أن تختار غيرها) ، ثم اكتب اسم للطبقة كما في الصورة:



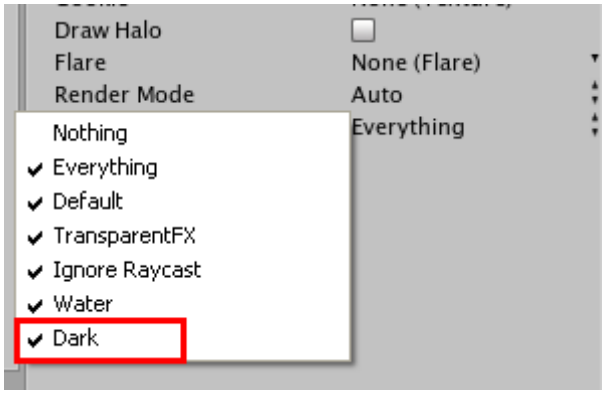
اضغط انتر ، الأن تم إضافة Layer جديدة ، اختر احد المجسمات ثم اذهب إلى نافذة ال Inspector وغير خاصية ال Layer إلى Dark كما في الصورة :



تأكد من ان الضوء من مصدر الضوء يصل هذا المجسم ، الأن نأتي إلى الخطوة المهمة ، أختر مصدر الضوء واذهب إلى نافذة Inspector ثم انقر على الخيار المؤشر عليه بالصورة التالية في خاصية ال Light :



ستظهر قائمة بالطبقات المتوفرة كما في الصورة:



ازل علامة الصح من كلمة Dark وذلك بالنقر عليها ،ستلاحظ ان المجسم الذي جعلنا خاصية ال Lyaer له Dark قد أصبح معتماً ولايتأثر بالضوء من هذه المصدر الضوئي ،وأي مجسم تريد ان لايتأثر بمصدر الضوء هذا اجعل خاصية ال Layer له Dark ،الأن اظهر القائمة في الصورة السابقة وضع علامة صح على كلمة Dark وذلك بالنقر عليها ، سيتأثر المجسم السابق بالضوء من هذا المصدر.

لو تم جعل المجسم لايتأثر بمصدر ضوء معين كما فعلنا أعلاه فانه يتأثر بمصادر الضوء الأخرى ان وجدت، إذا لم نغير فيها كما غيرنا بمصدر الضوء السابق.

++++ أضنك تقول الحمد لله انتهى المثال بدون كتابة سكربتات ^_^ ++++

مكونات ملف السكربت :

يتكون ملف السكربت بشكل رئيسي من دوال السلوك أو ما يسمى (Behaviour Functions) أي ان كل دالة مختصة بسلوك معين فتنفذ ما بداخلها إذا حدث هذا السلوك ،ويقصد بالسلوك أو الحدث مثلاً الضغط على زر الماوس ،أو حدوث التصادم ،أو بدء تحميل اللعبة ،أو تحديث الفريمات أيضاً يعتبر سلوكاً ،أو التعامل مع مايسمى ال (GUI) أي التعامل مع الأزرار وصناديق الحوار ومربعات الاختيار والنصوص على شاشة اللعبة يعتبر سلوكاً ايضاً.

ملاحظة: سكربت الجافا إذا كان فارغاً لايظهر أي خطأ بعكس سكربت ال C# وال Boo المستخدمين في اليونيتي فلو مسحت ما بداخلهم وجعلتهم فارغين فسيظهر خطأ.

بعض دوال السلوك (Behaviour Functions)

هذه الدوال مستقلة عن بعضها يعني كل دالة لها بداية ونهاية ولايجوز كتابة احدها بداخل الأخرى:

```
function func1( ) {

الأوامر التي سوف تنفذ عند استدعاء هذه الدالة
……..
……...
}

function func2( ) {

الأوامر التي سوف تنفذ عند استدعاء هذه الدالة
……...
……
}

وهكذا...
```

ومن هذه الدوال:

1- function Update( )

تستدعى هذه الدالة في كل فريم من فريمات اللعبة ،واغلب الأوامر تكتب بداخلها لأنها مستمرة مع أحداث اللعبة حتى نهايتها.

2- function Start( )

تستدعى هذه الدالة مرة واحدة في بداية تنفيذ السكربت ،فمثلاً لو أردنا تنفيذ أمر معين مرة واحدة أو أعطاء قيم ابتدائية للمتغيرات عند بدء اللعبة ،فمن غير المنطقي وضع هذه الأوامر في دالة ال Update()،بل نضعها في هذه الدالة.

3- function OnMouseDown( )

تستدعى هذه الدالة عند الضغط على الزر الأيسر للماوس على المجسم صاحب السكربت وليس على أي مكان كما في الطريقة (Input.GetMouseButton) التي شرحناها في موضوع الكلاسات.
(يشترط ان يحتوي هذا المجسم على خاصية ال Collider لكي تعمل هذه الدالة).

سنأخذ مثالين لتوضيح الفرق المذكور اعلاه :

مثال :عند الضغط على الزر الايسر للماوس على أي مكان في اللعبة يتم تدوير المجسم حول محور Y :

```
function Update () {

    if ( Input.GetMouseButton(0) ){

      transform.Rotate(0 ,1 , 0);
    }
}
```

مثال :عند الضغط على الزر الايسر للماوس على المجسم صاحب السكربت فقط يتم تدويره حول محور Y :

```
function OnMouseDown( ) {

    transform.Rotate(0 ,1 , 0);

}
```

4- function OnMouseUp( )

تستدعى هذه الدالة عند رفع الضغط عن الزر الأيسر للماوس على المجسم صاحب السكربت (يشترط ان يحتوي هذا المجسم على خاصية ال Collider).

5- function OnCollisionEnter (collision : Collision)

تستدعى هذه الدالة عند حدوث تصادم بين المجسم صاحب السكربت(يشترط ان يحتوي هذا المجسم على خاصية rigidbody و Collider) ،وأي مجسم أخر (يشترط ان يحتوي على خاصية ال Collider).
هذه الدالة تحتوي على بارمتر collision وفائدته معرفة خصائص المجسم الذي لامس المجسم صاحب السكربت مثلاً نستفاد منه لمعرفة إذا كان التصادم مع مجسم معين افعل كذا وإذا مع غيره لا تفعل شيء)

6- function OnControllerColliderHit (hit : ControllerColliderHit)

تستدعى ايضاً عند حدوث التصادم ولكن يتم استخدامها مع الشخصيات المتحركة التي تحتوي على مكون ال Character Controller لكون بعض الشخصيات لايحتوي على خاصية Collider وإنما يتم حسابها من خلال خاصية ال Character Controller للشخصية ، وايضاً ممكن الاستفادة من البارمتر hit كما ذكرنا في سابقتها.

7- function OnCollisionExit(collisionInfo : Collision)

تستدعى هذه الدالة عند الخروج من التصادم مع مجسم أخر ويجب أن تتوافر نفس شروط دالة OnCollisionEnter وايضاً ممكن الاستفادة من البارمتر collisionInfo.

8- function OnCollisionStay(collisionInfo : Collision)

تشبه دالة OnCollisionEnter لكن الفرق بينهما ان هذه الدالة تستمر بالاستدعاء وتنفيذ ما بداخلها مادام التلامس بين المجسمين مستمر أما الدالة OnCollisionEnter تنفذ مرة واحدة في بداية حدوث التلامس.

9- function OnGUI ( )

تستخدم هذه الدالة لوضع الازورار وصناديق الحوار والنصوص والصور(مثلاً خريطة اللعبة) ومربعات الاختيار والزلاقات(slider) (مثلاً للتحكم بمستوى الصوت).

هذه هي أهم الدوال السلوك وهناك دوال أخرى كثيرة ممكن أن تتطلع عليها من صفحات المساعدة للبرنامج.

*شرح مبسط لاستخدام الدالة ( ) OnGUI :

ملاحظة جميع الأوامر الخاصة بهذه الدالة يجب كتابتها بداخلها حيث أن الدوال الأخرى لاتتعرف عليها:

```
function OnGUI ( ) {
……..
……..
……..
}
```

ومن هذه الأوامر:

1- Button : الأزرار

```
GUI.Button (Rect ( Left ,Top ,Width, Height) , "String")
```

Rect : هي المنطقة التي سوف يرسم بها الزر وهي اختصار لكلمة (Rectangle:مستطيل) وتحتوي على التالي

Left : وهو عدد يمثل موقع الزر بالنسبة ليسار الشاشة وكلما زاد هذا العدد اتجه الزر لليمين

Top : وهو عدد يمثل موقع الزر بالنسبة لأعلى الشاشة وكلما زاد هذا العدد اتجه الزر للأسفل

Width : وهو عدد يمثل عرض الزر

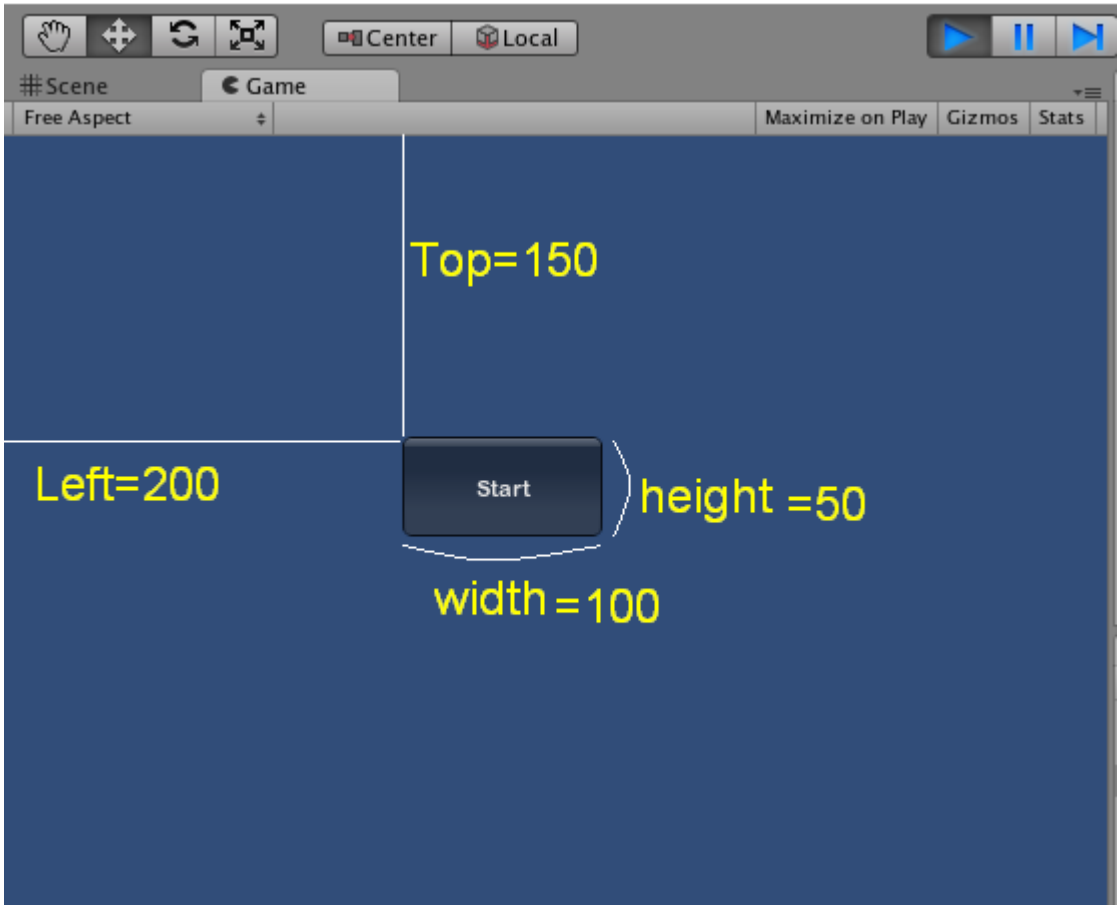
Height : وهو عدد يمثل ارتفاع الزر

"String" : هو النص الذي يظهر مكتوباً على الزر ويجب أن يكون بين علامتي اقتباس ""

مثال :

```
function OnGUI ( ) {

   GUI.Button (Rect (200,150,100,50), "start");

}
```

اكتب السكربت السابق وضعه على أي مجسم في اللعبة مثلاً على الكامرة شغل اللعبة سوف تشاهد التالي



هذا السكربت في حالته هذه لاينفع شيء! لأننا لو ضغطنا على الزر لايتم تنفيذ أي شيء، إذا ماهو التعديل لكي يصبح السكربت ذا فائدة؟: يجب أن نضع شرط (جملة if) ونضع كود الزر داخل شرطها لكي تقوم بالفحص هل تم الضغط على الزر ،إذا نعم فتنفذ ما بداخلها من أوامر، وإلا فلا تنفذ أي شيء:

```
function OnGUI ( ) {

  if ( GUI.Button (Rect (200,150,100,50), "start") ) {

  // اكتب الأوامر التي تريدها أن تنفذ عند الضغط على هذا الزر

  }
}
```

والسؤال هنا لو كان عندنا زرين ما العمل؟ :

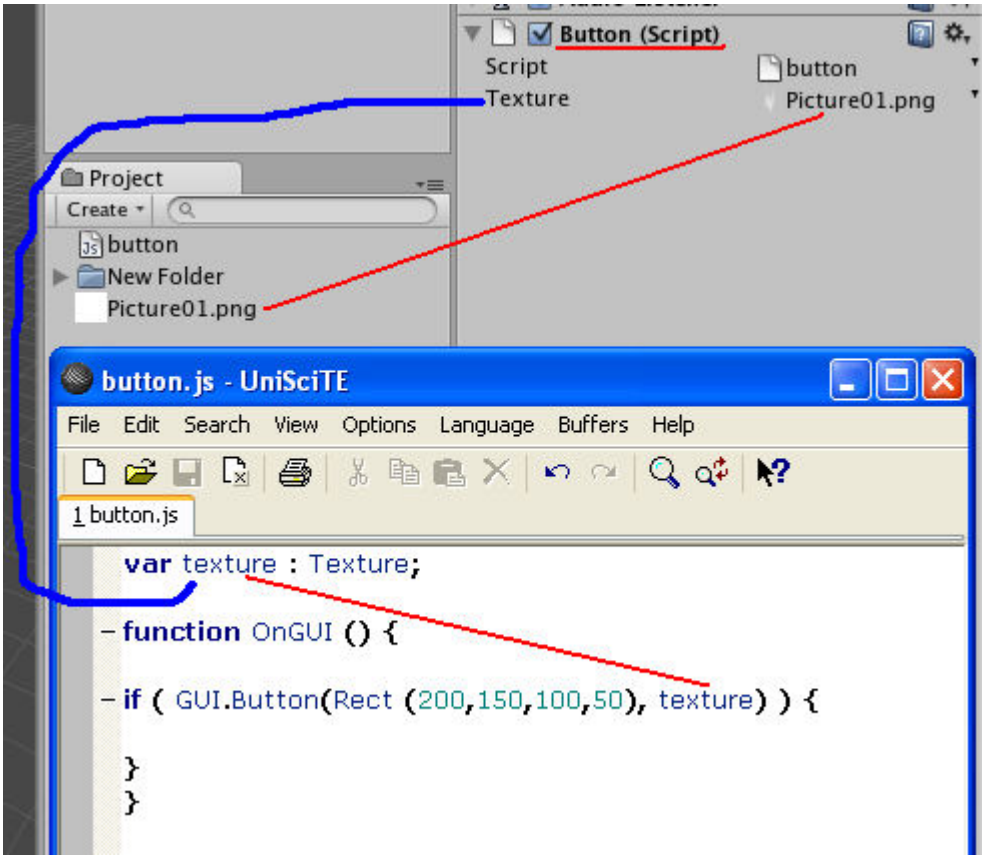
```
function OnGUI ( ) {

  if ( GUI.Button (Rect (200,150,100,50), "Start") ) {
      Application.LoadLevel ("Level1");
  }

 if ( GUI.Button (Rect (200,250,100,50), "Exit") ) {

    Application.Quit( );
  }

}
```

لاحظ اننا غيرنا قيمة ال Top للزر الثاني لكي لايتم وضعه فوق الزر الأول وجعلنا أمر التنفيذ عند الضغط عليه هو الخروج النهائي من اللعبة ، اكتب السكربت السابق وشغل اللعبة ثم اضغط على الزر الثاني سوف لايتم الخروج من اللعبة!! لماإذا ؟؟؟؟ لان هذا الأمر يعمل مع الملف التنفيذي للعبة فقط (أي انه لايستطيع إيقاف محرك الألعاب إذا كنت قد شغلت اللعبة منه).

ومن الممكن ان نضع صورة على الزر بدلاً من النص (لايجوز الاثنين معاً) وذلك بتعريف متغير من نوع (Texture) ومن ثم نسند إليه صورة ، حيث ان هذا المتغير سيظهر اسفل السكربت في نافذه (Inspector) للمجسم الذي وضعنا السكربت عليه كما في الصورة أدناه:

مثال:

```
var textur : Texture;

function OnGUI ( ) {

  if ( GUI.Button (Rect (200,150,100,50), textur) ) {

  }
}
```



ممكن الاستفادة من هذه الخاصية ،بتصميم صورة صغيرة ببرنامج الفوتوشوب أو غيره بحيث تحتوي على نص مكتوب باللغة العربية مثلاً كلمة "ابدأ" وحذف خلفية الصورة وحفظها بصيغة (.png) واستيرادها لليونيتي ووضعها للزر ،لان اليونيتي لايدعم الكتابة باللغة العربية.

2- Box: صندوق حوار أو لوحة(panel)

GUI.Box (Rect (Left ,Top ,Width, Height) , "String")

ما ذكر في الأزرار ينطبق على الصناديق من حيث المعاملات واستبدال النص بصورة ،ماعدا انها لا تستخدم جملة الشرط if لان الصناديق لاتحتوي على حدث الضغط بالماوس ،ولها فوائد عديدة مثلاً استخدامها كحاوية للأزرار وباقي الأدوات ، أو استخدامها كصناديق محاورة مع اللاعب.

مثال:

```
function OnGUI ( ) {

   GUI.Box (Rect (200,150,300,300), "Main Menu");

}
```

3- Labels: العناوين

GUI. Label (Rect (Left ,Top ,Width, Height) , "String")

ما ذكر في الأزرار ينطبق على الصناديق من حيث المعاملات واستبدال النص بصورة ،ماعدا انها لاتستخدم جملة الشرط if لأنها لاتحتوي على حدث الضغط بالماوس ،وتستخدم لوضع العناوين ووصف الأدوات.

مثال:

```
function OnGUI ( ) {
   GUI. Label (Rect (200,150,50,25), "Sound Setting");
}
```

4- DrawTexture : رسم صورة

GUI.DrawTexture (Rect (Left ,Top ,Width, Height) , Texture)

وتستخدم لعرض صورة على الشاشة مثلاً خريطة اللعبة أو مؤشر سدادة السلاح ،أو الأدوات المتوفرة مع اللاعب وغيرها...

مثال:

```
var textur : Texture;
function OnGUI ( ) {
  GUI.DrawTexture (Rect (200,150,100,100), textur) ;
}
```

5- TextField : حقل النصوص

GUI. TextField (Rect (Left ,Top ,Width, Height) , "String")

وتستخدم لإدخال النصوص من قبل المستخدم (يعني الذي يلعب اللعبة) مثلاً ان يدخل اسمه وغيرها من الأمور،وهنا سيكون النص المكتوب في السكربت هو الافتراضي وممكن تغييره من قبل المستخدم ،وممكن تركه فارغاً وذلك بوضع علامتي اقتباس ""، وطريقة الاستخدام يجب ان نعرف متغير من نوع نص ليكون مرجع لنص هذه الأداة.

مثال:

```
var str = "player";
 function OnGUI ( ) {

   str = GUI. TextField(Rect (200,150,50,25) , str );

}
```

في المثال السابق عدد الأحرف غير محددة ،فلو أردنا تحديد عدد الأحرف المدخلة ١٠ أحرف فقط فيصبح الكود:

```
GUI. TextField(Rect (200,150,50,25), str , 10);
```

ففي هذه الحالة لايستطيع المستخدم إدخال أكثر من ١٠ أحرف.

6- TextArea : النص متعدد الاسطر

GUI. TextArea (Rect (Left ,Top ,Width, Height) , "String")

في الاداة TextField يكتب النص في سطر واحد فقط ،أما في هذه الأداة فيمكن الكتاب في أكثر من سطر.

مثال:

```
var str = "Information about me: ";
 function OnGUI ( ) {

   str = GUI.TextArea(Rect (200,150,200,200) , str );

}
```

شغل اللعبة واكتب في هذه الأداة واضغط Enter ليتم النزول إلى سطر جديد وايضاً يمكن تحديد عدد الأحرف المدخلة كما فعلنا في الأداة TextField.

7- PasswordField : النص المشفر

GUI.PasswordField(Rect(Left,Top,Width,Height), "String", "maskchar" , no)

يستخدم في إدخال الرمز السري "PassWord" :

"String" : هو النص المطلوب إدخاله من قبل المستخدم

"maskchar" : هو الرمز الذي يظهر بدلاً من الحروف المدخلة

no : عدد الأحرف المدخلة المسموح به

مثال: ضع هذا السكربت على الكامرة

```
var pass = "";
 function OnGUI ( ) {

   pass = GUI.PasswordField (Rect (200,150,100,25) , pass , "*"[0] , 10);

  if ( GUI.Button (Rect (200,250,100,50), "Start") ) {
       if (pass == "maxforums" ) {
          camera.backgroundColor = Color.green;
       }
     }
}
```

* لماذا استخدمنا المعامل [0] مع النص "*" ، خذ الأمثلة التالية لكي تعرف

```
pass = GUI.PasswordField(Rect (200,150,100,25) , pass , "*$#"[0] , 10);
pass = GUI.PasswordField(Rect (200,150,100,25) , pass , "*$#"[1] , 10);
pass = GUI.PasswordField(Rect (200,150,100,25) , pass , "*$#"[2] , 10);
```

الأن هل عرفت؟ هذا الرقم يشير إلى مكان الرمز المطلوب إظهاره من السلسلة النصية"#$*"
(لمن يعرف المصفوفات ،النص عبارة عن مصفوفة حرفية تبدأ فهرستها من الصفر).

8- Toggle: (Check Box) مربعات الاختيار

GUI.Toggle(Rect(Left ,Top ,Width, Height) , Boolean , "String")

وتستخدم في عرض خيارات ليتم التأشير عليها من قبل المستخدم لتفعيل خاصية أو عدم تفعيلها (مثلاً تفعيل الصوت أو عدم تفعيله)

Boolean :
هو متغير منطقي يأخذ True أو False ويتم من خلاله معرفة هل المربع مختار أم لا

"String" : النص الذي يكتب أمام المربع لوصف عمله

مثال: هذه السكربت يوضع على الكامرة ويحدد إمكانية سماع الأصوات أم لا إذا كانت الرؤية من خلالها وذلك عن طريق مربع الاختيار الذي سوف يضعه السكربت .

```
var sound : boolean;
 function OnGUI ( ) {

sound = GUI.Toggle(Rect (200,150,100,20) , sound , "Enable Sound");
GetComponent(AudioListener).enabled = sound;

}
```

9- Sliders : الزلاقات

يوجد نوعين من الزلاقات ،أفقية (HorizontalSlider) ،وعمودية (VerticalSlider) ،ولمن لايعرف ماهي الزلاقات ،فهي شبيهة بتلك التي نتحكم من خلالها بمستوى الصوت في مشغلات الملتميديا.

GUI.HorizontalSlider(Rect(Left,Top,Width,Height),Value, Start , End)

GUI.VerticalSlider(Rect(Left,Top,Width,Height),Value, Start , End)

Value : هي القيمة الواقف عندها المزلاق وتتغير مع تحريكه
Start :قيمة تمثل بداية المزلاق

End : قيمة تمثل نهاية المزلاق

مثال: لنفرض انه عندنا مجسم بندقية اسمه "gun" ويحتوي على صوت بحيث يصدر عند اطلاق النار ونريد أن نكتب سكربت ونضعه على الكامرة ليتحكم بمستوى هذا الصوت.

```
var volume: 0.5;
 function OnGUI ( ) {

volume = GUI.HorizontalSlider(Rect (200,150,100,20) , volume, 0 ,1);
gameObject.Find("gun").audio.volume = volume ;

}
```

مثال: وضع مزلاق عمودي على يسار الشاشة يستخدم لتقريب وإبعاد الرؤية للكامرة (أي نعمل zoom). يوضع هذا السكربت على الكامرة.

```
 var zoom = 60;
 function OnGUI ( ) {

zoom = GUI.VerticalSlider(Rect (20,200,20,100) , zoom, 20 ,100);
GetComponent(Camera).fieldOfView = zoom;
}
```

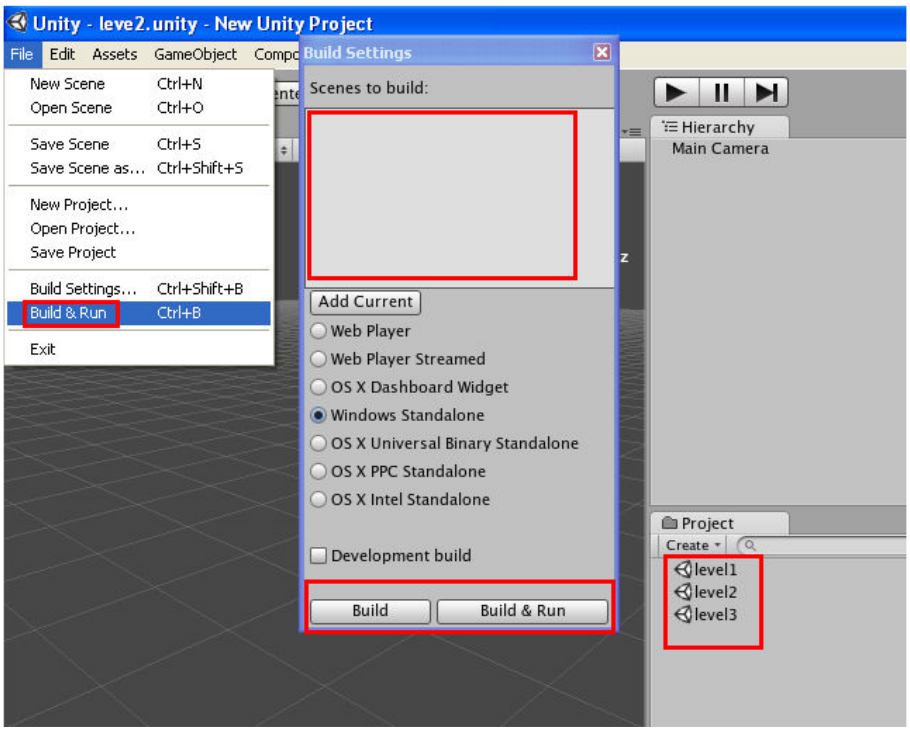
وان أردنا أن لا يظهر مزلاق الزوم إلا إذا ضغط اللاعب على زر Z من الكيبورد وعند الضغط مرة أخرى عليه يختفي المزلاق ، فيصبح السكربت كالأتي:

```
  var zoom: 60;
  var show=false;
 function OnGUI ( ) {
    if(show) {
     zoom = GUI.VerticalSlider(Rect (200,150,100,20) , zoom, 20 ,100);
     GetComponent(Camera).fieldOfView = zoom;
     }
}

function Update () {
 if ( Input.GetKeyDown("z")) {

      show = !show;
    }
}
```

ملاحظة علامة التعجب (!) تستخدم لعكس القيمة المنطقية يعني إذا كانت false تعملها true وبالعكس.

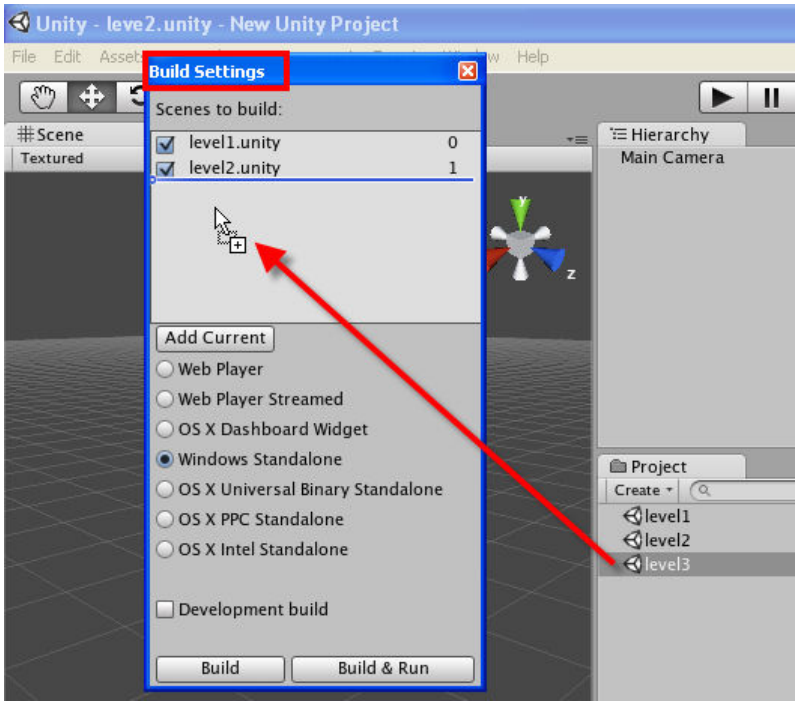
التعامل مع مراحل اللعبة:

عند تصميم لعبة مكونة من عدة مراحل ومن ثم عمل ملف تنفيذي للعبة



سوف يتم بناء وتضمين المرحلة الحالية (المفتوحة للعمل) في الملف التنفيذي فقط ولانستطيع الوصول الى المراحل الاخرى وتحميلها.

ولكي نستطيع التعامل مع مراحل اللعبة جميعها يجب ان نضيفها الى قائمة المراحل التي سوف يتم بنائها وتضمينها ، عن طريق الضغط على المرحلة في نافذة Project ومع استمرار الضغط نسحبها الى قائمة المراحل المذكورة ومن ثم نفلتها كما في الصورة:



ولتحميل احد المراحل نستخدم الامر :

```
Application.LoadLevel ("اسم المرحلة");
```

أو :

```
Application.LoadLevel (رقم المرحلة);
```

رقم المرحلة: هو الرقم الوجود مقابل اسم المرحلة ي قائمة المراحل في الصورة اعلاه ويبدأ بالرقم صفر.

مثال: عند الضغط على زر مرسوم في احد المراحل يتم تحميل المرحلة "level2" :

```
function OnGUI ( ) {

  if ( GUI.Button (Rect (200,150,100,50), "Load Level_2") ) {
      Application.LoadLevel ("Level2");
  }
}
```

واذا اردنا ان نستخدم رقم المرحلة بدل اسمها نكتب:

```
Application.LoadLevel ( 1 );
```

ولو اردنا اعادة تحميل المرحلة الحالية نكتب :

```
Application.LoadLevel (LoadedLevel );
```

حيث ان ال LoadedLevel التي بين القوسين ترجع رقم المرحلة الحالية .

ولو اردنا الحصول على أسم المرحلة الحالية نستخدم :

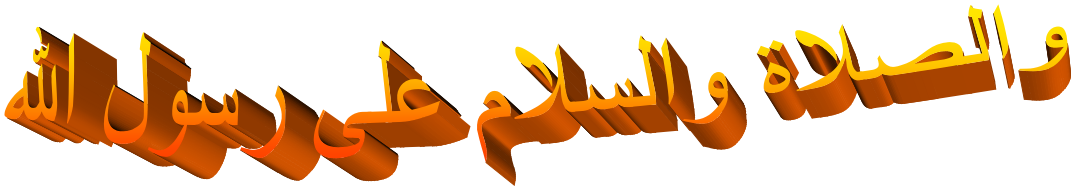
```
Application.LoadedLevelName
```

مثال:في كل مرحلة من المراحل ضع اداة GUI Text من قائمة:
GameObject → Create Other ثم اكتب السكربت وضعه على ال GUI Text:

```
function Start ( ) {

guiText.text = Application.loadedLevelName;

}
```

فعند تنفيذ كل مرحلة سوف تشاهد اسمها في اداة GUI Text .


ثم بحمد الله


والصلاة والسلام على رسول الله