به نام خدا

کتاب آموزش برنامه نویسی

با

سطح کتاب : پیشرفته

نوشته شده توسط مهرشاد کاوسلا

در پایان این قسمت شما باید بتوانید به سوالات زیر پاسخ دهید :

متغییر ها چیستند ؟

متغییر ها چگونه نوشته می‌شوند ؟

توابع چیستند ؟

توابع چگونه نوشته میشوند ؟

تابع های اصلی یونیتی کدام اند ؟

دستورات شرطی چیستند ؟

نحوه‌ی نوشتن دستورات شرطی چگونه است ؟

متغییرها در زبان برنامه نویسی مثل ظرف هایی هستند که اطلاعات برنامه شما در آنها قرار می‌گیرد. برای مثال شما میخواهید قیمت هر عدد تفنگ در بازی ۵۰۰ تومان باشد و هر روز ۵۰ تومان گران تر شود. برای همین باید یک متغییر بنویسید و مقدار اولیه آن را ۵۰۰ قرار دهید. این متغییر در حافظه خود ذخیره میکند که هر عدد تفنگ ۵۰۰ تومان است و بعد میتوانید در کدنویسی بگویید که هر روز به متغییری که قیمت تفنگ در آن است، ۵۰ تومان اضافه شود.

ساختار کلی متغییر ها در یونیتی به شکل زیر است :

Var + اسم متغییر + نوع متغییر + چیز موجود در متغییر;

برای مثال یک متغییر به نام j که عدد اعشاری است مینویسیم که حاوی عدد ۵,۲ میباشد.

```
Var j : float = 5.2;
Var j = 5.2;
```

امتحان : حالا یک متغییر به نام G بنویسید که حاوی عدد اعشاری ۳,۸ باشد.

خب حالا باید با انواع متغییر ها آشنا شوید :

۱. اعداد صحیح :

این نوع متغییر شامل تمام اعداد طبیعی + اعداد منفی هستند. این نوع متغییر را به شکل زیر می‌نویسند :

```
Var j : int=4;
```

(اعداد صحیح : { ...,-3,-2,-1,0,+1,+2,+3,... }

۲. اعداد اعشاری

این نوع متغییر ها شامل اعداد اعشاری یا همان اعدادی که میتوان آنها را به صورت یک کسر متعارفی نوشت میباشد. مثال :

```
Var j : float = 5.2;
```

۳. رشته

این نوع متغییر شامل هر چیزی که شما دوست دارید هست. از حروف و اعداد گرفته تا اشکال.
کاربردش هم اینجاست که مثلا شما میخواهید یک اسم از بازیکن بگیرید و آنرا در متغییر p ذخیره کنید. مثال :

```
Var p : string = "GameMakerPlugin.ir"
```

همانطور که میبینید در متن بالا آدرس وب سایت گیم میکر پلاگین که داری حروف و نقطه است، قرار دارد.
(البته علامت سوال ، تعجب ، @ و علامات مشابه نیز قابل استفاده اند.)
(به جای p یا j ، هر حرف یا کلمه ای را میتوانید استفاده کنید و اجباری در نوشتن این حروف نیست !)

توجه کنید که متن شما باید بین " قرار بگیرد.

توابع : با توابع می‌شود متغییر ها رو تغییر داد و کار اصلی برنامه نویسی هم ویرایش متغییرها برای اجرای بازی هست.

وقتی شما در یونیتی اسکریپتی ایجاد می‌کنید، به طور خودکار ۲ تابع مهم در صفحه برنامه نویسی وجود دارند که درباره آنها توضیح خواهم داد.

ساختار کلی توابع در یونیتی :

Function +نام تابع+( )+{محل قرار گیری کدها}

مثال :

```
Function Update ()
{
محل قرار گیری کد ها
}
```

توابع اصلی در یونیتی :

1. Start

این تابع هنگام ایجاد اسکریپت نیز وجود دارد. هر کدی که داخل این تابع قرار گیرد، موقع اجرای بازی انجام میگیرد.

2. Update

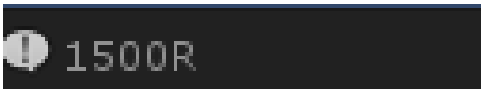
این تابع مهم نیز هنگام ایجاد اسکریپت وجود دارد. این تابع لحظه به لحظه چک می‌کند و کدهای داخل خود را اجرا می‌کند.

مثال : شما متغییری به نام MyMoney تعریف کردید و میخواهید در بازی ببینید که داخل این متغییر چقدر است.

پس باید کد زیر رو بنویسید :

```
var MyMoney = "1500R";
function Update ()
{
print(MyMoney);
}
```

این کدو به دوربین اصلی یا همون Main Camera میدیم و بعد بازی رو اجرا می کنیم.



مشاهده میکنیم که مقدار متغییر ما یعنی 1500 R رو نوشته. حالا به تشریح کد میپردازیم.

در خط اول که یک متغییر از نوع رشته‌ای انتخاب کرده ایم. در خط دوم هم تابع آپدیت رو نوشتیم. اما در خط چهارم یه دستور جدید به نام Print نوشتیم. کار این

دستور نشان دادن ℎ مقداری است که در پرانتز روبرویش قرار دارد.

۳. OnGUI

با این تابع در یونیتی میشه متن و دکمه و موارد مشابه رو درست کرد.

دستورات شرطی یک کد هستند که یه شرط از ما میگیرند. اگه آن شرط درست بود، مجموعه ای از دستوراتی که در زیر آن نوشته شده، اجرا خواهند شد.

ساختار کلی دستورات شرطی :

```
If
{
دستور ۱
دستور ۲
دستور ۳
...
}
```

برنامه‌ی قبلی رو به خاطر دارید ؟ حال میخواهیم در آن برنامه از if استفاده کنیم و بگوییم اگر MyMoney برابر با ۱۵۰۰ بود، اون رو تبدیل به ۱۶۰۰ کنه و بعد به ما نشون بده.

پس باید کد زیر رو بنویسیم.

```
Var My Money = “1500R”;
Function Update ()
{
    If(MyMoney == “1500R”)
   {
      MyMoney == “1600R”;
      Print(MyMoney);
    }
 }
```

در خط چهارم ما یه شرط گذاشتیم که اگه MyMoney مساوی 1500R بود، دستوراتش اجرا بشن. در خط ششم که اولین دستور ما هست، گفتیم MyMoney رو برابر 1600R کنه و در خط بعد گفتیم که این متغییر رو نشون بده.

۴. Bool

این متغییر از ما درست یا نادرست (True , False) رو میگیره. مثال :

Var Game = true; **(عدد یک هم میشه گذاشت)**

Var Game = false; **(صفر هم میشه گذاشت)**

توجه : در مثال بالا، Game متغییر می‌باشد.

ساخت یک پروژه

ما میخواهیم درون بازی یه مکعب قرار دهیم که هر وقت روی صفحه کلیک چپ کردیم، مکعب سه واحد بالا برود. اگر کلیک راست کردیم، ۳ واحد به سمت راست برود. اگر روی غلطک موس کلیک کردیم، ۳ واحد تو برود. (یعنی در محور z باشد)

ابتدا روی Main Camera کلیک کرده و بعد مقادیر زیر رو بهش بدین.

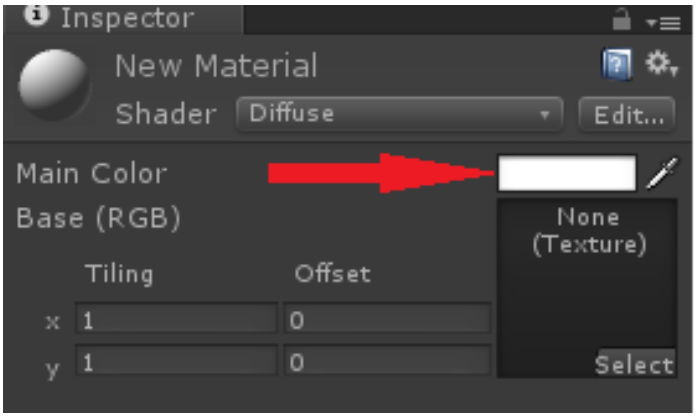
X = 0, Y=0, Z=0

سپس یک مکعب بسازید. (از قسمت Create و Cube)

بعد مقادیر زیر رو بهش بدین :

X = 0, Y=0, Z=20

حالا یک متریال درست کنید و روی قسمت مشخص شده‌ی داخل عکس کلیک کنید و آنرا به مکعب نسبت دهید.



بعد میتونید برای روشن شدن برنامه، point light نیز قرار دهید.

حالا یک اسکریپت جاوا قرار دهید. تابع Start رو همراه با بلاکت هاش پاک کنید.

بعد در تابع آپدیت یه if بزارید و داخل پرانتز هاش کد زیر رو بنویسید :

```
If(Input.GetButtonDown("Fire1"))
{


}
```

حالا در براکت های کد Fire1 این کد رو مینویسیم.

```
Transform.positon.x = transform.position.x+3;
```

حالا کد رو به مکعب بدید و بازی رو اجرا کنید.

فقط کافیه کدو دوبار دیگه بنویسیم :

بار اول :

```
If(Input.GetButtonDown("Fire2"))
{
Transform.positon.y = transform.position.y+3;
}
```

بار دوم :

```
If(Input.GetButtonDown("Fire3"))
{
Transform.positon.z = transform.position.z+3;
}
```

در این آموزش از یونیتی ما می خواهیم نگاهی به اسکریپت نویسی به زبان C# بیاندازیم , به طور کلی اگر چه گزینه های دیگر برنامه نویسی نیز برای توسعه دهندگان بازی در نظر گرفته شده است (چه حرفه ای و یا چه مبتدی) برای پروژه های بزرگتر با رویداد های پیچیده تر c# ارائه و توصیه می شود. مزایای بسیاری از جمله داشتن یک رویکرد مقاوم تر که پتانسیل بروز مشکلات را کمتر می کند و سادگی آن برای برنامه نویسی همه و همه می تواند شما را در انتخاب این زبان برنامه نویسی برای یونیتی یاری رساند.

برای مثال در حالی که سی شارپ در مقابل یونیتی اسکریپت زبانی دشوارتر به شمار میرود اما در کنار همین دشواری مشکلات کد نویسی می توان کد هایی با باگ کمتر و پیچیده تر را تولید نمود

در این مجموعه مقالات در ابتدا در نظر گرفته شده که

خواننده دانش برنامه نویسی و حداقل یک پایه ای از برنامه نویسی را دارا می‌باشد.

در این مقالات مباحث زیر مورد پوشش قرار خواهد گرفت :

مقدمه ای بر سی شارپ

۱.ایجاد یک کلاس

۲.متغییرها و خواص

۳.متد های کلاس

۴.انتخاب و تکرار

۵.تعامل و ارتباط بین اشیای بازی و کلاس ها

مقدمه :

-وی کی پدیا

سی شارپ (C#)زبانی شیء گرا و سطح بالا از خانواده

زبان های چارچوب دات نت شرکت مایکروسافت است.
زبان C#، یک زبان برنامه نویسی چند الگویی است و منظم شدهٔ مدل های تابعی، امری، عمومی، شی گرا و جز گرا می باشد. این زبان توسط مایکروسافت و جزئی از دات نت به وجود آمد و بعدا استانداردهای ECMA و ISOرا نیز در بر گرفت. C# یکی از ۴۴ زبان برنامه نویسی ای است که توسط زمان اجرای زبان مشترک از .NET Framework پشتیبانی می شوند و در همه جا به وسیله مایکروسافت ویژوال استودیو شناخته می شود.

این زبان برپایه سادگی، مدرن بودن، همه منظوره و شی گرا بودن ساخته شد. Anders Hejlsberg، طراح زبان برنامه نویسی دلفی، سرپرستی تیم طراحان زبان C# را بر عهده داشت. این زبان دارای دستوری شی گرا مشابه C++است و به شدت از زبان های جاوا و

دلفی[نیازمند مدرک] تاثیر پذیرفته است. در ابتدا نام این زبان COOL بود که مخفف C like Object Oriented Languageبود، هر چند در جولای ۲۰۰۰، زمانی که مایکروسافت پروژه را عمومی اعلام کرد، اسم آن به C#تغییر پیدا کرد. آخرین نسخه آن نسخه ۵,۰ است که همزمان با دات نت ۴,۵ در آگوست ۲۰۱۲ منتشر شد.

مزایای سی شارپ :

مطئنا مطالبی که در ادامه گفته خواهد شد تمام مزایا یا لیست کامل از مزایای این زبان نمی باشد اما می توان گفت قابل توجه ترین آن ها را می توان شامل اجزای زیر دانست :

۱ـ

شی گرا بودن :

سی شارپ یک زبان با خاصیت شیئ گرا است بنابراین به

طور خودکار اسکریپت ها نیز ساختار شیئ گرایی پیدا می کنند در حالی که ممکن است آن را برای مبتدیان پیچیده تر کنند , البته باید این نکته را نیز در نظر داشته باشید که در توسعه بازی یکی از نیاز های اساسی استفاده از مفاهیم شیئ گرایی است در حقیقت به وسیله شیئ گرایی در کنار اینکه خود برنامه نویسی ساده تر شده و کارآمد و پیچیده تر در کنار آن ساخت بازی هم به طبع آسان تر و کارآمد تر خواهد بود.

۲-

به صورت پیش فرض خصوصی (Private)

تمامی متد ها و متغییر ها در اینجا به صورت پیش فرض از نوع خصوصی می باشند این در حالی است که در یونیتی اسکریپت به صورت پیش فرض این مقادیر به صورت عمومی یا Public می باشند البته مگر آن که در

این دو مورد غیر از آن مشخص شود . در حالی که بسیاری از عموم بدون در نظر گرفتن این مورد همیشه از پیش فرض ها استفاده می کنند ، مزایای خصوصی بودن به صورت خودکار دامنه دسترسی به مجوز متغییر ها را تعیین می کند.

۳ـ امکانات پیشرفت و پشتیبانی از چارچوب دات نت یونیتی اسکریپت یک زبان پیش فرض برای یونیتی است که بر پایه جاوا اسکریپت ساخته شده است , با این حال سی شارپ , سی شارپ واقعی است به این معنی که برای دسترسی به ویژگی های زبان اصلی سی شارپ همراه با چارچوب آن ارائه شده است .

مزایای دیگری نیز وجود دارند اما موارد ذکر شده و تاثیر آن ها با هم باعث میشود که یک زبان ساختار کامل تر و قدرتمند تری داشته باشد.

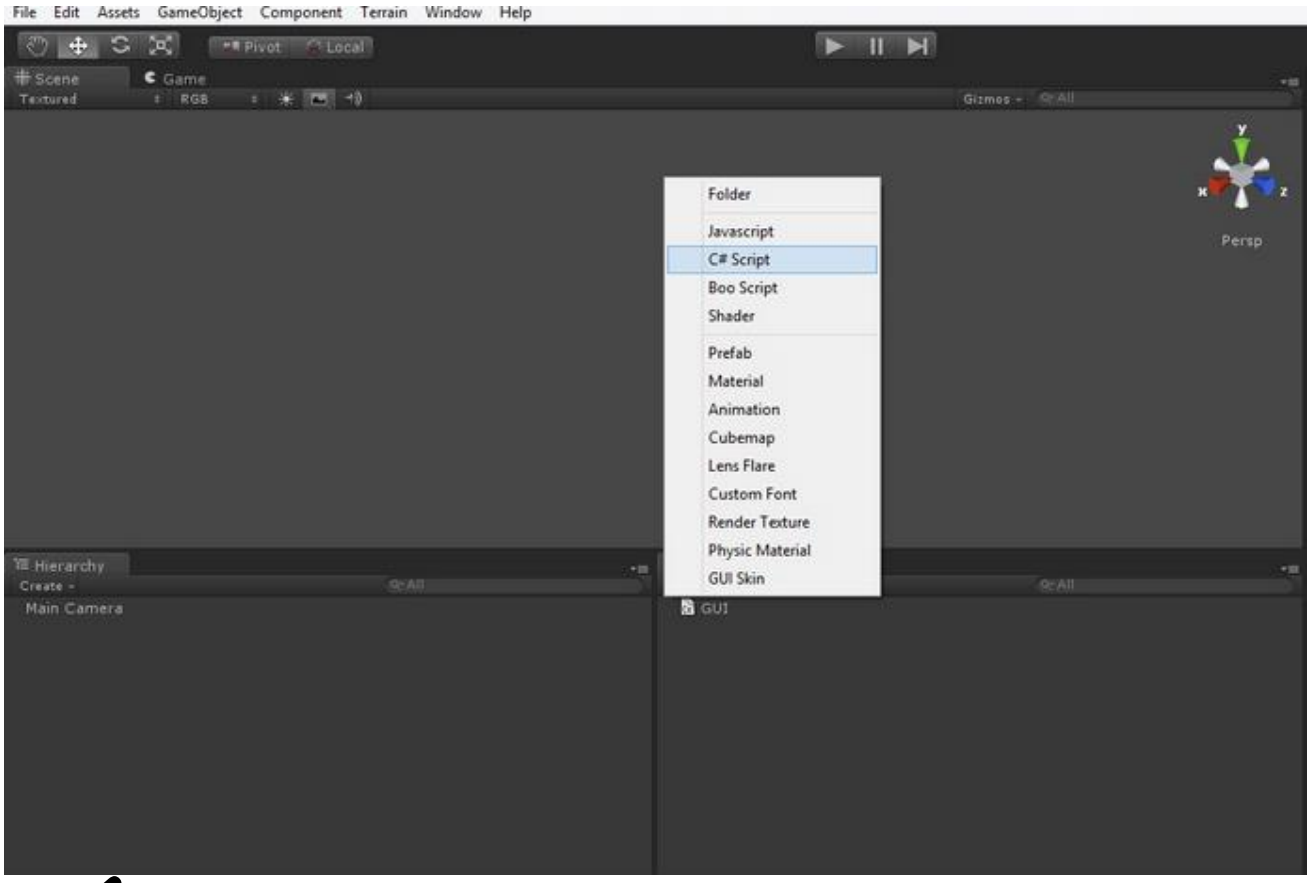
ساختار سی شارپ اسکریپت :

در قسمت اول اموزش سی شارپ در یونیتی ما به معرفی سی شارپ و بحث در مورد بعضی از ویژگی های آن پرداختیم . در ادامه می خواهیم به روند چگونگی ایجاد یک اسکریپت سی شارپ و همچنین بررسی ساختار کلی آن در اینجا بپردازیم . همانطور که گفته شد برعکس زبان های برنامه نویسی دیگر پشتیبانی شده در اینجا ما به

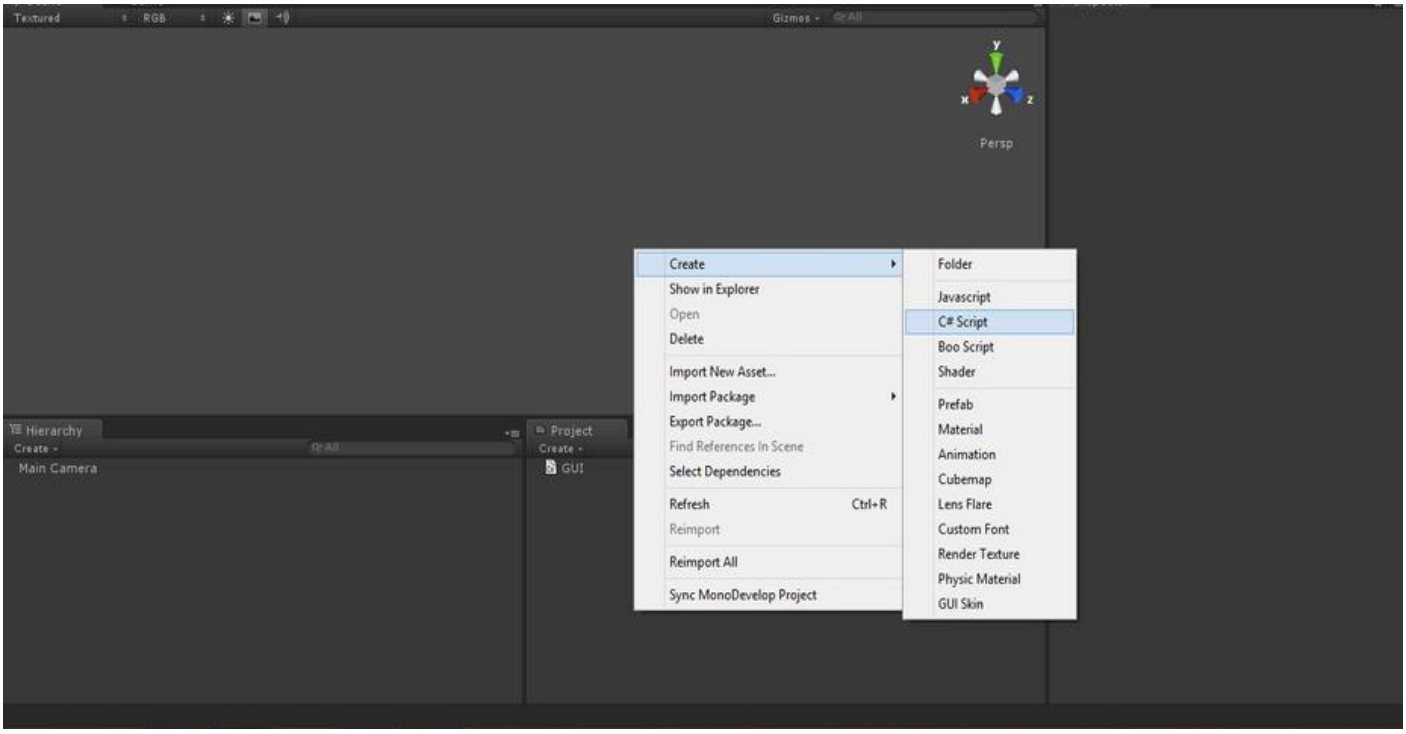
وسیله سی شارپ می توانیم به ویژگی های دانت نت فریم ورک نیز دسترسی داشته باشیم و این مزیت این زبان نسب به زبان های داخلی و زبان های در دسترسی اینجاست.

ایجاد یک اسکریپت سی‌شارپ:
شما میتوانید این کار را در خود ادیتور یونیتی انجام دهید. برای انجام این کار:

۱- در پنل Project بر روی Create کلیک کرده و از منوی باز شده گزینه C# Script را انتخاب نمایید.



همچنین این کار را نیز می توانید به وسیله راست کلیک در همین قسمت و انتخاب گزینه Create و سپس C# Script همین فایل را ایجاد کنید.



همچنین پس از به وجود آوردن اسکریپت شما می توانید نام آن را در ابتدا تغییر دهید .

به عنوان مثال ما این اسکریپت را Player Character می نامیم .

برای استفاده از آن شیئ جدید را انتخاب کرده و سپس اسکریپت را کشیده و بر روی شیئ جدید بیندازید.

اکنون شما می توانید با دوبار کلیک کردن بر روی آن وارد محیط IDE برنامه نویسی MonoDevelop شوید.

همچنین شما می توانید از ویژوال استدیو نیز برای برنامه نویسی سی شارپ در اینجا استفاده کنید.

ساختار اسکریپت :

به وسیله سی شارپ ما یک ساختار طبقاتی در اختیار داریم ، یونیتی به صورت خودکار یک الگو پایه ای برای شما که موجب صرفه جویی در زمان برای به وجود آوردن اسکریپت می شود را به وجود می آورد . پس از اولین اجرا شما می توانید این ساختار را در IDE برنامه نویسی خود مشاهده کنید :

```
using UnityEngine;
using System.Collections;


public class playerCharacter : MonoBehaviour
{


  // Use this for initialization
  void Start ()
  {
```

```
    }

    // Update is called once per frame
    void Update ()
    {

    }
}
```

توضیح این کدها :

۱- در ابتدای کدهای این کلاس ما می توانیم دستوراتی را مشاهده کنیم , در این مکان دستورات مربوط به تعریف کتابخانه های مورد استفاده که در این کلاس ها از آن استفاده می کنیم نوشته میشود.

۲- در قسمت بعدی ما نام کلاس را داریم و شما می توانید ببینید که به طور مستقیم از MonoBehaviour مشتق شده.

۳- در داخل تعریف کلاس , عنوان عمومی نیز تعریف شدة که باعث میشود شما بتوانید یک کلاس عمومی تعریف کنید

۴- ما در اینجا دو متد داریم

۱- start

۲- update

البته با اینکه فقط این دو متد به عنوان متد های اصلی در اینجا قید شدة اند اما ما در یونیتی ساختار های متعددی نیز داریم

مثلا :

Awake( )

که زمانی که اسکریپت لود می شود اجرا میشود . می توان گفت اولین فراخوان از این متد انجام می شود.

()Start

این متد در مرتبه بعدی اجرا می شود و زمانی انجام میشود که متد اول تمام شدة باشد می توان گفت وظیفه این متد مقدار دهی اولیه می باشد.

() Update

این متد در هربار اجرا بر اساس هر فریم اجرا می شود همچنین دو متد زیر نیز در راستای همین متد عمل می کنند :

()FixedUpdate

()LastUpdate

() OnGUI

برای استفادة از رابط کاربری مورد استفادة قرار می گیرد.

همانطور که گفته شد دو متد اولیه به عنوان دو متداصلی در اینجا شناخته میشوند اما شما در استفاده از آن ها نیز خود مختار هستید و همچنین می توانید از دیگر متد های اصلی نامبرده شده نیز در اینجا استفاده کنید.

ساختار عمومی اسکریپت :

به طور کلی ساختار یک کلاس مشابه قالب زیر می باشد :

```
using UnityEngine;
using System.Collections;
... include directives here, using the "using" directive ...


public class playerCharacter : MonoBehaviour
{
   ... class wide property and variable declarations here ...

   ... constructors here ...

   ... "update" methods here ...
```

```
    ... your methods here ...
}
```

متغییر ها و مشخصه ها :

در قسمت دوم از آموزش اسکریپت نویسی به زبان سی شارپ ما نگاهی به چگونگی ایجاد اسکریپت های سی شارپ انداخته و همچنین ساختار این کلاس ها را در سی شارپ مورد بررسی قرار دادیم . اکنون زمان آن رسیده که نگاهی به متغییر ها و مشخصه ها بیاندازیم . علاوه بر آن ما باید نگاهی نیز به تفاوت این دو یعنی متغییر ها و مشخصه ها نیز بیندازیم و همچنین فرابگیریم که چگونه یک متغییر و یا یک مشخصه را تعریف کنیم , انواع داده ها و دیگر نکاتی که در ادامه آن را مورد بررسی قرار می دهیم .

در سی شارپ ما در حقیقت دو نوع متغییر داریم :

متغییر ها یا فیلد ها

مشخصه ها (خاصیت ها)

با توجه به زمینه استفادة آن می تواند در برخی از زبان ها متفاوت باشد، بسته به زبان مورد نظر آن ها معمولا مشخصه – متغییر – فیلد و یا صفات نامیدة می شوند. همه اینها به یک چیز اشارة می کنند. ظرف ذخیرة سازی وجود دارد که در داخل کلاس وجود داشته و برای ذخیرة سازی دادة ها استفادة می شود

صفات متغییر ها :

در سی شارپ به طور کلی ما چند نوع صفات یا مشخصه برای متغییر ها داریم:

1- Public

متغییر های عمومی از هر نقطه در داخل برنامه قابل دسترسی هستند به طوری که در حقیقت کلاس های دیگر از هر نقطه به طور مستقیم قادر به مشاهده و دسترسی و تعیین محتوای متغییر های تعریف شده هستند . اگر چه این متغییر ها اجازه دسترسی آسان را می دهند اما بلاخره مشکلاتی دارند که به خوبی مشهود می باشند.

Private-۲

در صورتی که بخواهید از این متغییر در مکان های دیگر کلاس استفاده کنید و یا اعمالی که در نوع بالا انجام می دادید را اینجا نیز تکرار کنید کامپایلر به شما خطا خواهد داد چون این متغییر به خصورت خصوصی بوده و فقط در همان کلاس و مکان تعریف شده قابل استفاده است . البته برای استفاده از آن روش هایی

نیز وجود دارد اما در جای خود می تواند مشکل دسترسی کامل را کنترل کند

۳-Protected

متد یا متغییر های محافظت شده در تمامی کلاس و زیر کلاس های تعریف شده قابل دسترسی هستند . یا به طور ساده متد یا متغییر محافظت شده فقط در همان کلاس یا کلاس هایی که از آن مشتق می شوند قابل دسترس می باشند و خارج از کلاس نمی توان به آن دسترسی داشت.

انواع داده ها :

در سی شارپ تمامی متغییر ها باید یک نوع داشته باشند . شما باید به کامپایلر بگویید که چه نوع داده شما در چه نوع متغییری ذخیره خواهد شد .

انواع متغییر ها :

رشته ای (String)

یک رشته از کارکتر های یونیکد که به طور معمول الفبایی بوده ومعمولا برای کلمات و متن مورد استفاده قرار می گیرد.

عددی (int)

یک متغییر عدد صحیح به عنوان یک متغییر ها از نوع ۳۲ بیتی دارای مقادیر از

۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷ می باشد

شناور (Float)

یک عدد اعشاری برای مثال : ۳,۱۰۰ یا ۱۰۰,۲

بولین (Boolean)

یک مقدار درست یا نادرست ـ ۱ یا ۰

short

یک مقدار صحیح عددی ۱۶ بیتی با رنج : ۳۲,۷۶۸- تا ۳۲۷۶۷

long

یک مقدار صحیح عددی ۶۴ بیتی با رنج :

۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۸-

تا

۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۸

byte

یک مقدار صحیح کوچک ۸ بیتی با رنج ۲۵۵- تا ۲۵۵

char

یک مقدار کوچک ۱۶ بیتی یونیکد کارکتر از ۰ تا ۶۵۵۳۵

مقادیر دیگری نیز وجود دارند که شما در آینده فراخواهید گرفت . علاوه بر این ها شما باید از ارایه ها و لیست ها نیز استفاده کنید که ما در آینده آن را مورد بررسی قرار می دهیم

اعلان متغییر ها :
با استفاده از مقادیری که تا اینجا فراگرفته ایم می خواهیم این متغییر ها را تعریف کنیم . این کار را در

همان فایل کلاس قبلی که ایجاد کرده ایم انجام می دهیم.

```
using UnityEngine;
using System.Collections;


public class playerCharacter : MonoBehaviour
{
  // Class properties
  public string playerName;
  private int playerScore;
  private int playerLives = 3;
  int playerHealth;


  // Use this for initialization
  void Start () {

  }

  // Update is called once per frame
  void Update () {

  }
}
```

ما در اینجا ۴ نوع متغییر از ۴ نوع ویژگی تعریف کرده ایم . در حالت کلی همانطور که قبلا نیز به آن اشاره شد تعریف یک متغیر مراحل زیر را دارد :

۱ـ حوزه اختیارات

۲ـ نوع داده

۳ ـ نام متغییر

۴ ـ مقدار پیش فرض

که البته گزینه های بخش ۱ و ۴ به صورت اختیاری می باشند.

اعلان های ما در واقع کاملا به هم ریخته و از یک ساختار مشخص و مشترک پیروی نمی کند ما با آنها را برای استفاده تنظیم کنیم :

```
// Class properties
private string playerName;
private int playerLives = 3;
private int playerHealth = 100;
private int playerScore = 0;
```

همانطور که قبلا نیز بیان شد قسمت مقدار دهی اختیاری بوده شما می توانید این کار را نیز در متد ها انجام دهید .در متد های پیش فرض ـ متد ابتدایی و یا

در متد آغازین . تمام اینها بر اساس نیازهای شما در طول برنامه تغییر خواهد کرد.

متغییر های محلی :

این متغییر ها در داخل متد ها تعریف میشوند و نحوه دسترسی به انها نیز از طریق همین متد ها می باشد

این متغییر ها همیشه از طبیعت خصوصی بودن پیروی می کنند.

می توان آنها را در مثلا به وجود آوردن شمارنده های داخلی نیز استفاده کرد .

برای مثال :

```
private void method1()
{
```

```
    // This is a local variable called myVar
    string myVar = "something";

    string mySecondVar = method2(myVar);
}


private string method2(string myValue)
{
    // This method has received myVar from method1 as "myValue"
    return myValue;
}
```

در همین مثال نیز می توانید متغییر های تعریفی در داخل متد ها را نیز مشاهده کنید که اینها متغییر های بازگشتی خود متد هستند یعنی با تعریف این متغییر ها ما می دانیم که این متد مقداری را در قالب این متغییر باز می گرداند.

ارایه ها :

شما همچنین می توانید یک سری ارایه و یا یک لیست از داده ها ایجاد کنید . یک داده یک نوع از انواع داده

می باشد برای مثال ارایه ای از مجموعه ای نوشته ها و یا اعداد صحیح.

مثال :

```
string[ ] someNames;
int[ ] someNumbers;
```

لیست ها :

لیست ها نیز یک نوع شبیه به ارایه ها می باشند که ما می توانیم در داخل ان ها به صورت دینامیک ایتم اضافه و یا کم کنیم .

مثال :

```
List<int> myList = new List<int>();
```

به عنوان مثال برای اضافه کردن به این لیست :

```
List<string> myList = new List<string;( )<
```

```
myList.Add("Jim;("
myList.Add("Bob;("
myList.Add("Jane;("
myList.Add("Sue");
```

همچنین با استفاده از دستور زیر می توانید در مکان مورد نظر خودتان در لیست ایتم اضافه کنید.

```
myList.Insert(1, "Mark");
```

حذف کنید :

```
myList.Remove("Jim");
```

تا اینجا آشنایی ابتدایی با متغییر ها پیدا کرده ایم در آینده بیشتر در این بخش کار خواهیم کرد.

در بخش قبلی از آموزش برنامه نویسی یونیتی به زبان سی شارپ ما نگاهی به چگونگی اعلان متغییر ها و استفاده از آنها به همراه مشخصه هایشان در یونیتی

پرداختیم . حالا ما می خواهیم نگاهی به متدهای کلاس بیندازیم .

ما باید انواع مختلفی از متد هایی که می توان در سی شارپ برای یونیتی استفاده کرد ، همچنین چگونگی استفاده از آن در قوانین ساخت آنها، تبادل داده ها بین متد ها ، برگرداندن داده ها از متد ها و تعداد دیگری از نکات مهم مورد نیاز را پوشش دهیم.

متد چیست ؟

متد ها در برنامه نویسی شیئ گرایی توابعی هستند که در داخل یک کلاس وجود دارند . آنها غالبا با عنوان کلاس متد ها اشاره می شوند .متد ها می توانند

صفات و قوانین دسترسی همانند Public و یا Private نیز بگیرند .

تعریف یک کلاس متد دقیقا به صورت زیر است :

```
private void someFunction( )
}
 ...   The code to do something
}
```

در بخش اول دامنه کد را تعریف می کنیم . که در این مورد خصوصی می باشد . همچنین اگر دامنه را حذف کنید به صورت پیش فرض خصوصی خواهند بود.

در بخش دوم ما مقدار بازگشتی را داریم که به صورت خالی است یعنی ما نمی توانیم هیچ مقدار بازگشتی از آن داشته باشیم.

و در نهایت نام متد قرار دارد که متد نیز به وسیله آن فراخوانی میشود

مثلا برای فراخوانی متد به صورت زیر می توان عمل کرد:

```
MethodName( )
```

```
private void setLives(int newLives)
{
  playerLives = newLives;
}
```

کد بالا یک مقدار عدد صحیح با نام NewLive را دریافت می کند .

همانطور که گفته شد صدا زدن یک متد در اینجا با آوردن نام آن انجام می شود

مثلا در اینجا هنگامی که متد دارای یک مقدار می باشد می توان آن را با مقدار مورد نظر برگشت داشت.

مثلا :

```
setLives(5);
```

نکته : متد ها می توانند یک مقدار را برگردانند اما اجباری برای ان نیست !

مثال :

فراخوانی یک متد در متدهای اصلی برنامه ! :

در این مثال ما می خواهیم متد مورد نظرمان را به وسیله متد اصلی start فراخوانی کنیم . در اینجا همانطور که قبلا نیز گفته شد متد های اصلی به صورت خودکار صدا زده می شوند و باعث اجرا متد ما می شوند :

```
using UnityEngine;
using System.Collections;


public class playerCharacter : MonoBehaviour
{
  // Class properties
  private string playerName;
  private int playerLives = 3;
  private int playerHealth = 100;
```

```
  private int playerScore;


  // Start() is called once automatically when the script
has loaded
  void Start()
  {
    print ("Initializing PlayerCharacter Class");

    initializePlayerCharacter();
  }


  // A private method that will initialize the player
character
  private void initializePlayerCharacter()
  {
    // Some code to initialize the player character
    print("Initializing Character");
  }
}
```

همچنین شما می توانید به وسیله کلید واژه this متد ها را از کلاس خود فراخوانی کنید .

تا اینجا به برخی از این مفاهیم اشاره کرده ایم . توجه کنید که این مفاهیم جزئی مفاهیم اولیه سی شارپ بوده و مختص به یونیتی نیست . برای ادامه شما باید مفاهیم را با استفاده از منابع دیگر کامل کنید ! به خاطر اینکه هدف یادگیری یونیتی است و نه سی شارپ این بخش را ادامه نمی دهیم.

در این قسمت می خواهیم نگاهی مختصر به نحوه استفاده از انتخاب گر ها و تکرار کننده ها داشته باشیم.

از انجا که هدف در این سری از اموزش ها این نیست که بخواهیم سی شارپ را از ابتدا اموزش دهیم . بخش های ابتدایی را به سرعت طی می کنیم

انتخاب گر ها را برای این دلیل به این نام صدا می کنیم که پس از انجام آن دستور جدیدی انتخاب می شود
تکرار کننده ها را نیز می توان نوعی از انتخاب گر ها نام برد که تا زمان به وقوع پیوستن شرط انتخاب دائما اجرا می شود. در این قسمت از آموزش ها ما می خواهیم این سری از دستورات را مورد بررسی قرار دهیم

IF , ELSE :

یکی از معروف ترین دستورات در زبان های برنامه نویسی که معمولا ما شاهد این هستیم که در بیشتر زبان های برنامه نویسی نیز وجود دارد این دستور می باشد. قالب این دستور به صورت زیر می باشد :
اگر (شرط مورد نظر برابر با صحیح بود) آنگاه عمل را انجام بده :

IF(true)

{

}

همانطور که مشاهده می کنید قالب بسیار ساده دارد هنگامی که شرط داخل پرانتز دستور برابر با صحیح باشد انگاه دستورات داخل حلقه اجرا می شود که معمولا این دستور را به وسیله عملگرهای مقایسه ای یا منطقی به وجود می آیند . البته باید دقت کنید به صورت پیشفرض در صورتی که مقداری را درون این پرانتز ها بنویسید صحیح خواهد بود.

: Else

زمانی که شرط انجام میشود زمانی پیش میاد که ما می خواهیم حالت غیر از حالت شرط را در نظر بگیریم به طور سادة مثلا :

اگر (شرط داخل پرانتز انجام شد) آنگاة دستورات را انجام بدة ، در غیراینصورت این دستورات را انجام بدة :

```
if(true)
{

}
else
{
}
```

این قالب به صورت صریح به دستور کنترلی اعلام می کند که زمانی که شرط برابر بود انجام بشود و اگر نبود دستورات دیگری را انجام بدهد اما در این قالب شرط نقض فقط بر اساس شرط شروع قالب دستور کنترلی

ایجاد خواهد شد مثلا اگر شما بگویید PlayerHealth برابر با ۶۰ باشد دستورات را انجام بدهد و اگر نبود دستورات شرط نقض را انجام بدهد اما اگر بخواهیم شروط بیشتری را نیز اعمال کنیم چه ؟ البته می توان به وسیله عملگرهای منطقی این کار را انجام داد اما خب حجم کار در همان یک نقطه نیز بالاتر می رود اینجاست که دستور else if به میدان میاید به وسیله این دستور می توانیم شرایط نقض دستور قبلی را بر اساس شرط جدید اعلام کنیم

یعنی :

اگر (دستورات داخل پرانتز صحیح بود) دستورات را انجام بدهد

در غیر صورت اگر (دستورات داخل پرانتز صحیح بود) دستورات ادامه را انجام بدهد

و الی اخر

همانطور که مشاهده می کنید بسیار کار را برای برنامه نویس راحت تر می کند.

مثال :

```
// A method to color a health bar based on the characters health
private void colorHealthBar()
{
  if(playerHealth >= 60)
  {
    // Color the health bar green
    print("Color the health bar green");
  }
  else if(playerHealth >= 40 && playerHealth <= 59)
  {
    // Color the health bar yellow
    print("Color the health bar yellow");
  }
  else if(playerHealth >= 1 && playerHealth <= 39)
  {
    // Color the health bar red
    print("Color the health bar red");
  }
```

```
  else
  {
    // Health bar is empty or in - figures
    print("Health has been consumed; so do something else");
  }
}
```

هر چقدر هم که این دستور برای ما کارآمد باشد اما در مقایسه های بزرگتر ما دچار مشکل خواهیم شد اگر مثلا بخواهیم یک دنباله را ایجاد کنیم نمی توان به همین سادگی با if آن را انجام داد البته می شود انجام داد اما حجم کار بسیار بالا خواهد رفت . برای همین دستور دیگری وجود دارد با نام switch case که در آن دستور مقایسه شرط با مقدار case ها مقایسه شده و سپس در صورتی که برابر با هر case بود مقدار آن را بر می گرداند.

مثال :

```
int month = 1;

switch (month)
```

```
{
  case 1:
    print("It is winter");
    break;
  case 2:
    print("It is winter");
    break;
  case 3:
    print("It is spring");
    break;
  case 4:
    print("It is spring");
    break;
  case 5:
    print("It is spring");
    break;
  case 6:
    print("It is summer");
    break;
  case 7:
    print("It is summer");
    break;
  case 8:
    print("It is summer");
    break;
```

```
  case 9:
    print("It is autumn");
    break;
  case 10:
    print("It is autumn");
    break;
  case 11:
    print("It is autumn");
    break;
  case 12:
    print("It is winter");
    break;

  default:
    print("Month is out of range");
    break;
}
```

اگر مثلا می خواستیم همین دستور را با استفاده از if بنویسیم :

```
if(month == 12 || month == 1 || month == 2)
{
  print("It is winter");
}
else if(month == 3 || month == 4 || month == 5)
```

```
{
  print("It is spring");
}
else if(month == 6 || month == 7 || month == 8)
{
  print("It is summer");
}
else if(month == 9 || month == 10 || month == 11)
{
  print("It is autumn unless you are American in which case it is fall");
}
else
{
  print("The month " + month + " is invalid!");
}
```

اگر چه انجام شد اما ظاهر چندان مناسبی ندارد.

گفته شد که علاوه بر این گونه دستورات نوع دیگری نیز وجود دارند که شباهت به نوع اول داشته اما با فرق اینکه تا زمانی که شرط مورد نظر انتخابی انجام نشده باشد به حرکت و چرخش ادامه می دهد و

دستورات داخلی خود را تا زمان وقوع شرط ادامه می دهد.

while loop

این دستور به صورت عمل می کند که تا زمانی که شرط انتخابی توقفش انجام نپذیرد می چرخد و اگر شرط داخلی انجام نشود این چرخش تا بینهایت ادامه خواهد داشت

مثال :

```
var i = 0;


while(i <= 10)
{
  print(i);
  i++;
}
```

نوع دستور دیگری نیز در همین بخش وجود دارد که به وسیله آن می توان یک حلقه را به طور واقعی اجرا کرد .

در این دستور یک حلقه در این دستور با یک مقصد و منبع مشخص انجام میشود. یعنی مثلا حلقه ما ده بار تا زمان وقوع شرط انجام می شود . از مزایای این نوع دستور این است که می توان علاوه بر تعداد چرخش گام حلقه را نیز تعیین کرد.

```
for(int i=0; i <= 10; i++)
{
  print(i);
}
```

Foreach

در سی شارپ حلقه ای به اسم Foreach نیز وجود دارد که این حلقه تک تک عناصر یک مجموعه را به ترتیب از ابتدا تا انتها بررسی می کند. یک مجموعه شامل گروهی از اشیا می باشد. در سی شارپ چندین نوع مجموعه وجود دارند که یکی از آنها آرایه ها می باشد.

شکل کلی این دستور به صورت

foreach (type intem in collection) statement;

می باشد.

در اینجا type item مشخص کنندهٔ اسم و نوع یک متغییر است که این متغییر عنصر بعدی مجموعه را در هر بار که حلقه می شود در یافت می کند این متغییر در اصطلاح iteration varible نامیدهٔ می شود . بنابراین type یاجنس آرایه یکسان یا سازگار باشد .

هنگامی که حلقه foreach شروع به کار میکند اولین عنصر ارایه به متغییر item اختصاص دادهٔ می شود و در هربار تکرار حلقه عنصر بعدی آرایه گرفته شدهٔ و به item اختصاص دادهٔ می شود و تکرار حلقه تا زمانی که عنصری در آرایه وجود داشته باشد ادامه می یابد.

مثال :

```
string[] inventory =
{
  "apples",
  "pears",
  "bananas",
  "blackberries",
  "strawberries"
};
```

```
[CPP]
foreach(string item in inventory)
{
  print("Inventory contains: " + item);
}
```

در اینجا مبحث این بخش نیز به پایان رسید.

ارتباط بین کلاس ها و اشیا :

در قسمت قبلی از مجموعه آموزشی برنامه نویسی سی شارپ در یونیتی ما نگاهی به چگونگی کار با دستورات انتخاب و تکرار داشتیم . حالا ما می خواهیم بررسی کنیم که چگونه می توانیم بین کلاس ها و اشیا ارتباط بر قرار کنیم .

به منظور انجام این کار ما نیاز به پوشش چند موضوع مختلف داریم تا آن ها را به عنوان یک راهنمای گام به گام برای تنظیمات اشیا، اضافه کردن کلاس ها به آن ها و ارتباط بین آنها داشته باشیم در حقیقت ما میخواهیم کار با فراخوانی متد ها از یک کلاس به کلاس های دیگر را فرا بگیریم .

برای این بخش یک پروژه جدید ایجاد می کنیم

ابتدا از منوی :

یک Game Object - > Create Other -> Cube
مکعب ایجاد و آن را در وسط صفحه می آوریم نام آن را
به GameMakerPluginObject1 تغییر دهید
مکعب دومی را به همین ترتیب به سکانس وارد کرده و
نام آن را GameMakerPluginObject2 قرار دهید.
به وسیله :
Game Object - > Create Other - > Direction Light
یک نور به سکانس اضافه کنید.
تمام آن ها را چه از دوربین تا مکعب ها را در مکان
مناسب قرار دهید.
یک EmptyGameObject با نام PlayerCharacter
نیز بسازید.
سپس یک فایل سی شارپ با نام PlayerCharacter

و مجددا فایل دیگری با نام GameMakerPlugin2ObjectClass بسازید.

اسکریپت ایجاد شدهٔ PlayerCharacter را کشیده و بر روی PlayerCharacter قرار دهید تا اسکریپت به آن نسبت دادهٔ شود.

اسکریپت GameMakerPluginObjectClass را بر روی آبجکت اول یعنی GameMakerPluginObject1 انداخته تا به آن نیز نسبت دادهٔ شود و این کار را برای GameMakerPluginObject2 نیز تکرار کنید.

کلاس ای که ما بر روی مکعب خود احمال کردهٔ ایم در حقیق شامل رفتاری است که مکعب ما باید داشته باشد

کلاس The GameMakerPluginObjectClass

کدهای این کلاس را به صورت زیر وارد کنید :

```
using UnityEngine;
using System.Collections;


public class GameMakerPluginObjectClass :
MonoBehaviour
{
   private bool rotateCube = false;    // Flag for
whether the cube should rotate



   // Update is called once per frame
   void Update()
   {
      // If rotateCube is true, start the rotation
      if(this.rotateCube == true)
      {
         transform.Rotate(Vector3.up,
Time.deltaTime * 100, Space.World);
      }
   }
```

```
  // This method sets the rotateCube to true, thus
starting cube rotation
  public void startRotation()
  {
    print("Starting Rotation");
    this.rotateCube = true;
  }
}
```

هدف این مثال این است که ما مکعب یا مکعب ها را به چرخش در بیاوریم

توضیح کد :

در خط اول ما یک متغییر با نام rotateCube داریم که به صورت پیش فرض به ان مقدار false نسبت داده شده است . این مورد به عنوان یک پرچم برای شروع و یا توقف چرخش مکعب به کار می رود .

ما در اینجا متد Update را داریم . در مباحث قبلی خودمان نیز اشاره کرده بودیم که این متد در هر فریم

صدا زده می شود این می تواند بسیار کاربردی برای اکشن های ادامه دار مانند حرکت های اسکریپتی باشد. در این متد ما یک شرط نیز داریم که با استفاده از آن نحوه حرکت مکعب را مشخص می کنیم

اگر مقدار rotateCube برابر با true باشد آنگاه transform.Rotate( ) مکعب را می چرخاند. پارامتر های اضافه شامل مسیر – چرخش و سرعت چرخش می باشد.

و در آخر ما یک متد عمومی تحت عنوان startRotation داریم که به وسیله آن می توانیم چرخش را فعال کنیم . توجه کنید که ما در اینجا می خواهیم این متد را به وسیله یک کلاس دیگر در این کلاس فعال کنیم.

همچنین ما در دستور صدا زده شده transform مقدار Rotate را داریم که سه مقدار به آن اضافه شده است.

توجه کنید Rotate سه مقدار از نوع Float قبول می کند که شما می توانید محور چرخش را در آن تعیین کنید. مثلا در اینجا یکی بر اساس زمان بازی و یکی دیگر بر اساس مختصات جهان بازی تعیین شده است.

کلاس PlayerCharacter :

اکنون ما می خواهیم در این کلاس با استفاده از یک متد دیگر و به وسیله ارتباط با کلاس اولی متد اجرا کننده این چرخش را صدا بزنیم.

برای این کار کدهای زیر را به کلاس مربوط به PlayerCharacter اضافه کنید :

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

```
public class playerCharacter : MonoBehaviour
{

  // Start() is called once automatically when the script
has loaded
  void Start()
  {
    print ("Initializing");

    this.interactWithCube(1);
  }


  // A private method that will handle interaction with
other game objects
  private void interactWithCube(int cubeNumber)
  {
    print("About to interact with cubes");



    // Find GameMakerPluginObject1 in the scene and
give it a reference variable (cubeObject)
```

```
    GameObject cubeObject =
GameObject.Find("GameMakerPluginObject" +
cubeNumber);
    GameMakerPluginObjectClass otherObject =
(GameMakerPluginObjectClass)
cubeObject.GetComponent(typeof(GameMakerPluginObj
ectClass));

    otherObject.startRotation();
  }
}
```

اول از همه ما متد start را داریم که به وسیله آن می خواهیم یک اسکریپت داخلی را اجرا کنیم . این متد initializing را برای کنسول چاپ کرده و سپس متد interactwithcube( ) را صدا می زند. صدا زدن عدد یک در اینجا بدین معناست که کدامین مکعب باید واکنش صدا زده شده را اجرا کند .

در داخل متد ها ما دو چیز مهم داریم.

1. GameObject CubeObject = GameObject.Find("GameMakerPluginObject" + CubeNumber);

در این کد ما به وسیله GameObject.Find می توانیم مستقیما شیئ ایجاد شده را به شیئ موجود در سکانس اشاره دهیم.

در اینجا این قسمت به وسیله یک نام GameMakerPluginObject و به همراه نام مکعب CubeObject که برای قابلیت انتخابی بودن آن است تعیین شده.

GameMakerPluginObjectClass OtherObject = (GameMakerPluginObjectClass)CubeObject.GetComponent(Typeof)GameMakerPluginObjectClass));

همچنین ما به وسیله این متغییر جدید که از کلاس مربوط به GameMakerPluginObjectClass ما

ساخته شدة است مقدار ورودی و دسترسی به متد نام برده شدة را ایجاد می کنیم.

otherObject.StartRotation( );

اخرین کد نیز startRotation را در معکب GameMakerPluginObjectClass با استفادة از متغییر OtherObject به عنوان اشارة به کلاس صدا میزند.

درکل با استفادة از interactWithCube ما می توانیم مکعبی را که می خواهد چرخش کند را مشخص کنیم . اگر در این حالت آن را به ۲ تغییر دهیم خواهید دید که مکعب دوم به جای اول چرخش می کند .

یک راه برای اینکه بررسی کنیم ایا یک شیئ قبل از انجام کار ما وجود دارد. استفادة از متد

interactwithCube و تنظیم در کلاس PlayerCharacter به صورت زیر است :

```
private void interactWithCube(int cubeNumber)
{
  print("About to interact with cubes");


  // Find GameMakerPluginObject1 in the scene and give it a reference variable (cubeObject)
  GameObject cubeObject = GameObject.Find("GameMakerPluginObject" + cubeNumber);

  if(cubeObject != null)
  {
    GameMakerPluginObjectClass otherObject = (GameMakerPluginObjectClass) cubeObject.GetComponent(typeof(GameMakerPluginObjectClass));

    otherObject.startRotation();
  }
  else
  {
```

```
    print("The object GameMakerPluginObject" +
cubeNumber + " does not exist in the scene");
  }
}
```

به وسیله شرط ما می توانیم مشخص کنیم که آیا آبجکت وجود دارد یا خیر. اکنون می توانیم سکانس را اجرا کنیم تا نتیجه را مشاهده کنیم.

هوش مصنوعی دشمن (خون و حمله)

به زبان سی شارپ

برای اینکه بتونیم از این کد ها استفاده کنیم ابتدا باید دوتا شی ایجاد کنیم . مثلا دو مکعب ایجاد می کنیم. به نام Enemy و Player

حالا یک فایل سی شارپ با نام EnemyAI ایجاد می‌کنیم.(توجه کنید که حتما اسمش این باشه). کد زیر رو توش کپی می کنیم :

```
using UnityEngine;using System.Collections;


public class EnemyAI : MonoBehaviour {
   public Transform target;
   public int moveSpeed;
   public int rotationSpeed;
   public int maxdistance;

   private Transform myTransform;

   void Awake(){
      myTransform = transform;
   }
```

```
  void Start () {
    GameObject go =
GameObject.FindGameObjectWithTag("Player");

    target = go.transform;

    maxdistance = 2;
  }



  void Update () {
    Debug.DrawLine(target.position,
myTransform.position, Color.red);



    myTransform.rotation =
Quaternion.Slerp(myTransform.rotation,
Quaternion.LookRotation(target.position -
myTransform.position), rotationSpeed *
Time.deltaTime);
```

```
    if(Vector3.Distance(target.position,
myTransform.position) > maxdistance){
    //Move towards target
    myTransform.position += myTransform.forward *
moveSpeed * Time.deltaTime;

    }
  }
}
```

فایل رو سیو می کنیم و به شی Enemy مرتبطش می کنیم و شی Player رو به عنوان هدف بهش معرفی کرده و مقادیر سرعت حرکت سرعت چرخش و میزان فاصلشو با شی مورد نظر مشخص می کنیم.

خب حالا یک فایل C# دیگه با نام PlayerHealth ساخته و کد زیر رو درش کپی می کنیم.

```
using UnityEngine;using System.Collections;


public class PlayerHealth : MonoBehaviour {
```

```
public int maxHealth = 100;
public int curHealth = 100;

public float healthBarLength;



void Start () {
healthBarLength = Screen.width / 2;
}



void Update () {
AddjustCurrentHealth(0);

}


void OnGUI(){
GUI.Box(new Rect(10, 10, healthBarLength, 20),
curHealth + "/" + maxHealth);
}
```

```
public void AddjustCurrentHealth(int adj) {
 curHealth += adj;

  if(curHealth < 0)
    curHealth = 0;

  if(curHealth > maxHealth)
    curHealth = maxHealth;

  if(maxHealth < 1)
    maxHealth = 1;

  healthBarLength = (Screen.width / 2) *
(curHealth / (float)maxHealth);
  }
}
```

حال فایل رو سیو می کنیم و به شی Player متصلش می کنیم و میزان سقف خون ، میزان خون حال حاضر و طول میزان خونی که در صفحه نمایش نمایش داده میشه که البته نیازی به تعیین این نیست و خودش اتوماتیک تعیین

میشه.

خب حالا نوبت به حمله دشمن میرسه . یک فایل با نام EnemyAttack می سازیم و کد زیر رو درش کپی می‌کنیم.

```
using UnityEngine;using System.Collections;


public class EnemyAttack : MonoBehaviour {
   public GameObject target;
   public float attackTime;
   public float coolDown;




   void Start () {
      attackTime = 0;
      coolDown = 2.0f;

   }
```

```
  void Update () {
    if(attackTime > 0)
      attackTime -= Time.deltaTime;

    if(attackTime < 0)
      attackTime = 0;


  if(attackTime == 0) {
    Attack();
    attackTime = coolDown;
    }

  }

  private void Attack() {
    float distance =
Vector3.Distance(target.transform.position,
transform.position);


    Vector3 dir = (target.transform.position -
```

```
transform.position).normalized;
    float direction = Vector3.Dot(dir,
transform.forward);


    if(distance < 2.5f) {
      if(direction > 0) {
     PlayerHealth eh =
(PlayerHealth)target.GetComponent("PlayerHealth");
    eh.AddjustCurrentHealth(-10);
      }
    }
  }
}
```

بعد از اینکار فایل رو ذخیره کرده و به شی دشمن نسبت میدیم و شی هدف که همون Player هست رو بهش معرفی کرده و مقدار زمان حمله و زمان آماده سازی رو براش تعیین می کنیم.

حالا بازی رو اجرا می کنیم و میبینیم که به دنبال پلیر میفته . (تا نگیردشم ول کن نیستش . )

برای دانلود آموزش های بیشتر کافیه به وبسایت گیم میکر پلاگین مراجعه کنید :

GameMakerPlugin.IR