THIRD EDITION

آموزش یونیتی

Unity in action

Unity IN ACTION

THIRD EDITION

Multiplatform game development in C#

Joseph Hocking

Foreword by Jesse Schell

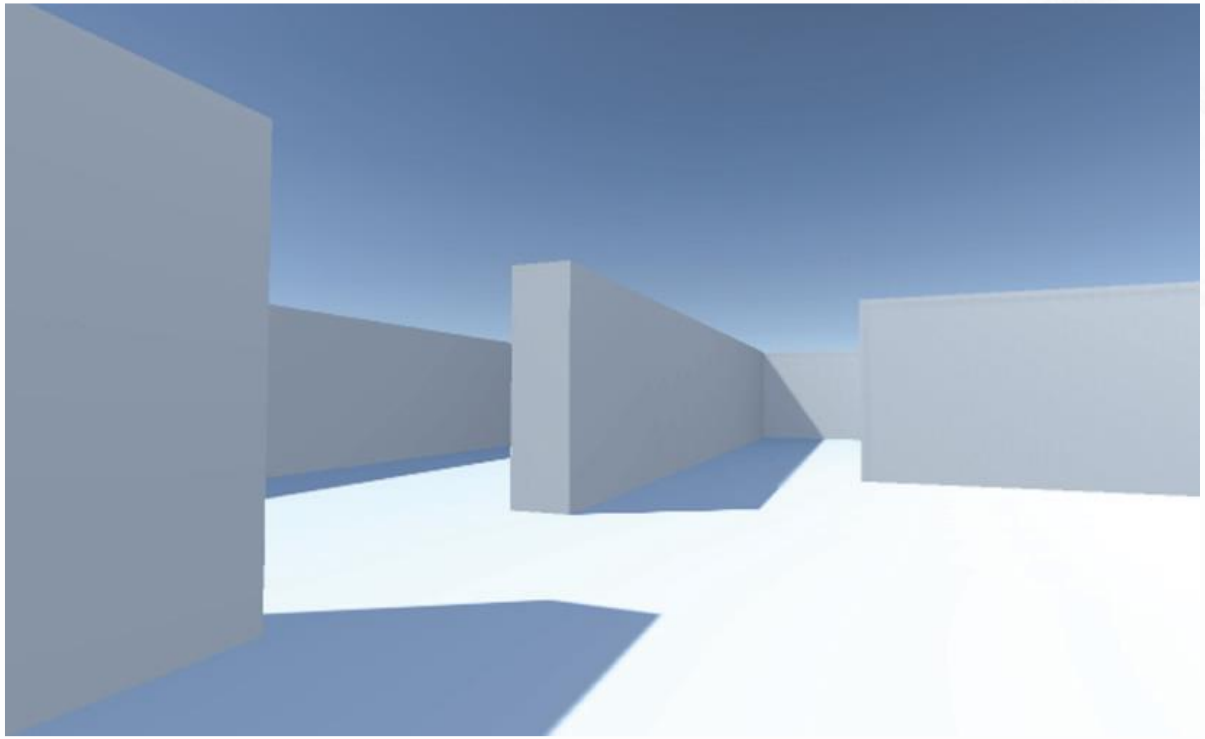
Building a demo that puts you in 3D space

آشنایی با ساخت یک بازی سه بعدی

This chapter covers

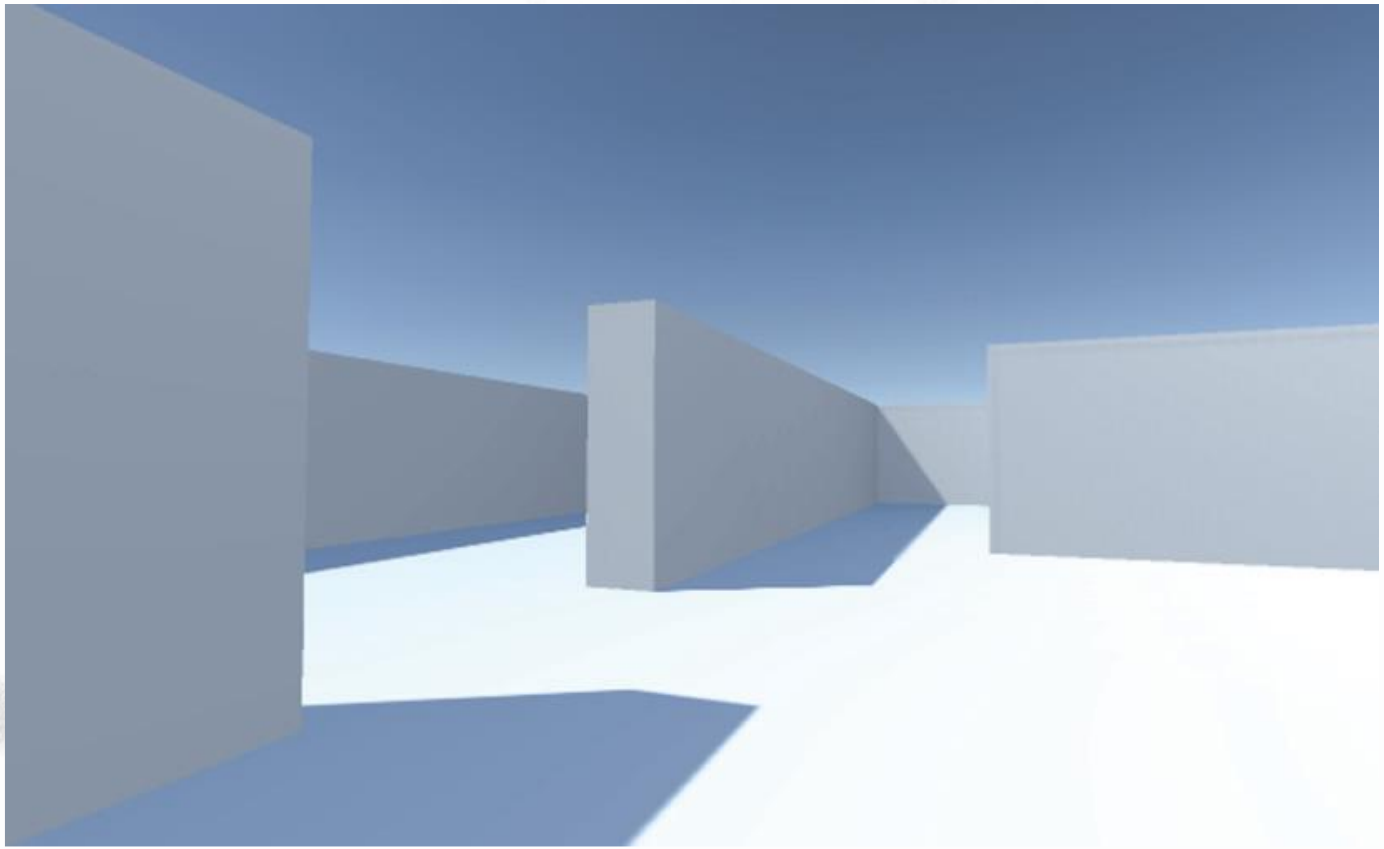
- Understanding 3D coordinate space
- Putting a player in a scene
- Writing a script that moves objects
- Implementing FPS controls

دانلود پروژه ها و مثال های کتاب



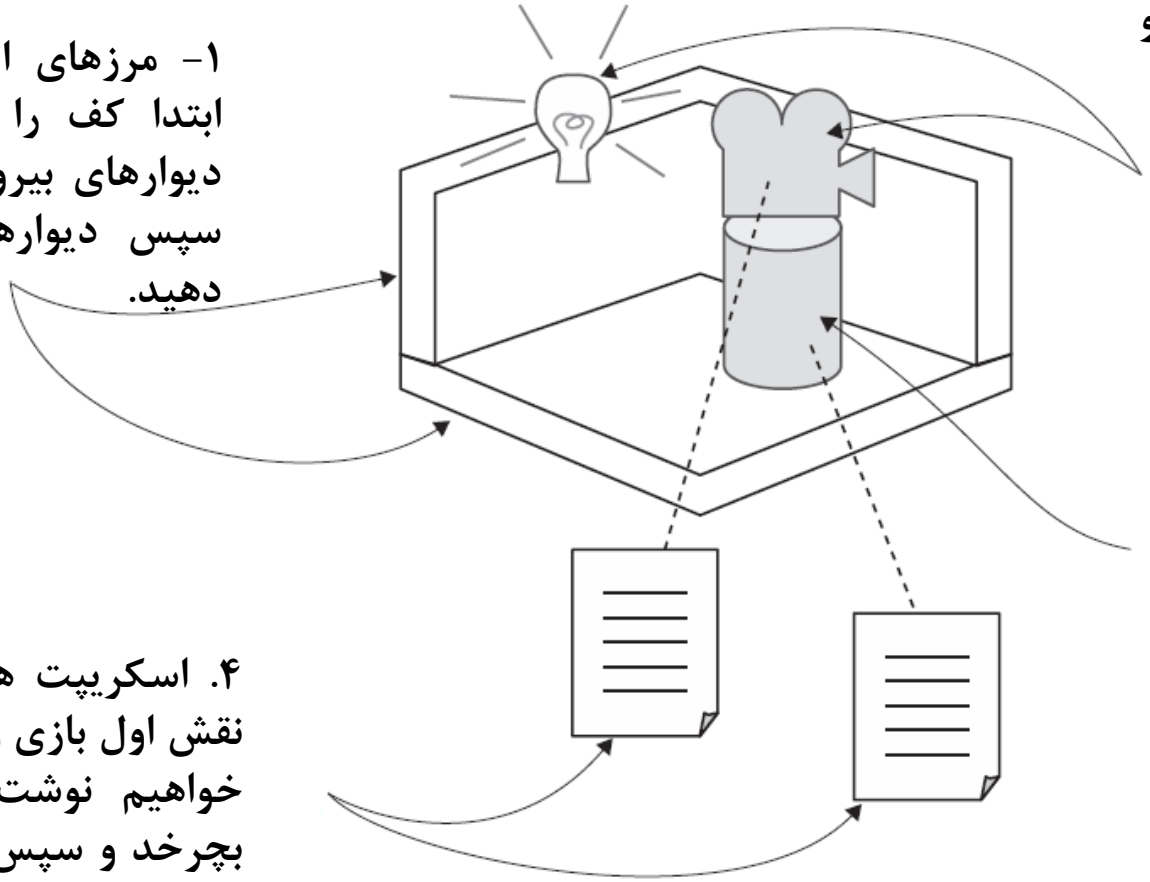
دانلـود (http://mng.bz/VBY5) کتـاب به یاد داشته باشید، پروژه هر فصل را می توان از وب سایت کرد .ابتدا پروژه را در Unityباز کنید و سپس صحنه اصلی)معمولاً فقط Sceneنام دارد (را برای اجرا و بازرسی باز کنید .در حالی که در حال یادگیری هستید، توصیه می کنم تمـام کـدها را خودتـان تایـپ کنید و از نمونه دانلود شده فقط برای مرجع استفاده کنید.

برای این پروژه اول، یک صحنه (first-person shooter (FPS) خواهید ساخت. ما اتاقی برای حرکت ایجاد خـواهیم کـرد و بازیکنـان جهـان بازی را از دید شخصیت می بینند و می توانند با استفاده از ماوس و صفحه کلید شخصیت را کنترل کنند.
تمام پیچیدگی های جالب یک بازی کامل را می توان در حال حاضر حذف کرد تا روی مکانیک اصلی تمرکز کرد(حرکت در یک فضای سه بعدی).



۱- مرزهای اتاق را تعیین کنید. ابتدا کف را ایجاد کنید، سپس دیوارهای بیرونی را ایجاد کنید و سپس دیوارهای داخلی را قرار دهید.

۲. بازیکنان باید بتوانند اتاق را ببینند. چراغ‌ها را در اطراف اتاق قرار دهید و دوربینی را که دید بازیکن است قرار دهید.

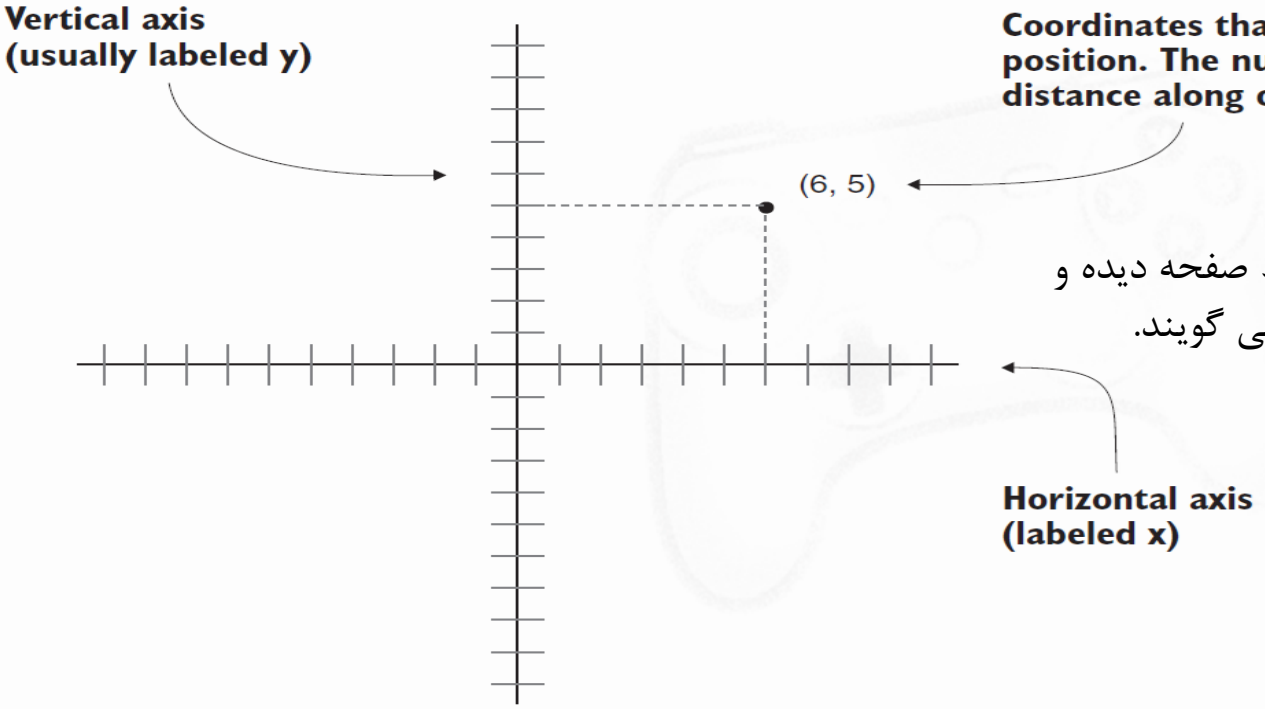


۳. شکل اولیه را برای بازیکن ایجاد کنید. دوربین را به بالای این شکل وصل کنید تا با حرکت این جسم، دوربین با آن حرکت کند.

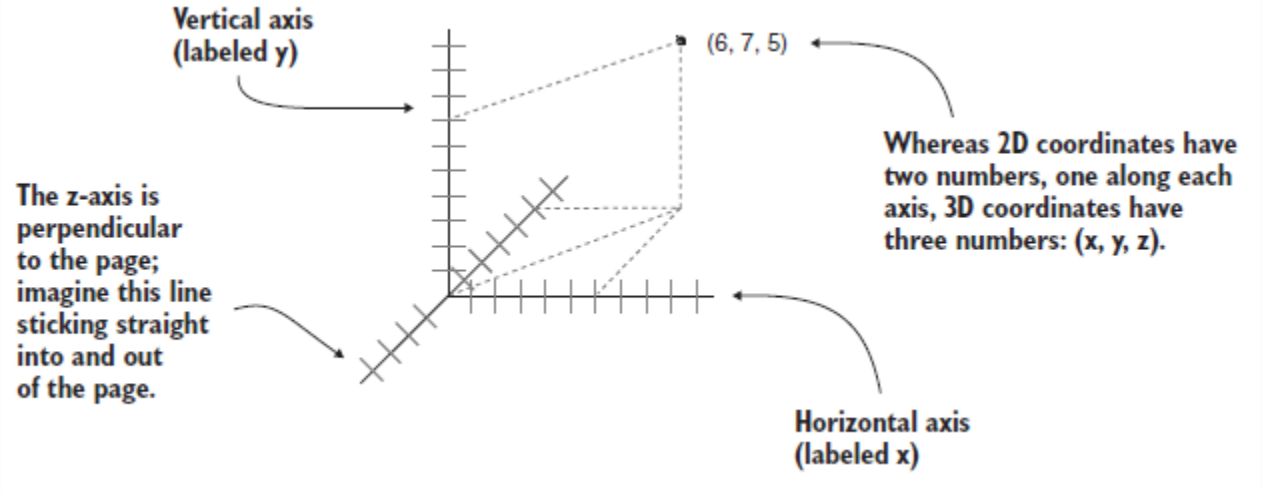
۴. اسکریپت های لازم برای حرکت نقش اول بازی را بنویسید. ابتدا کدی خواهیم نوشت که بتواند با موس بچرخد و سپس کد لازم برای حرکت با صفحه کلید را خواهیم نوشت

این شکل مراحل انجام پروژه را نشان می دهد

Understanding 3D coordinate space



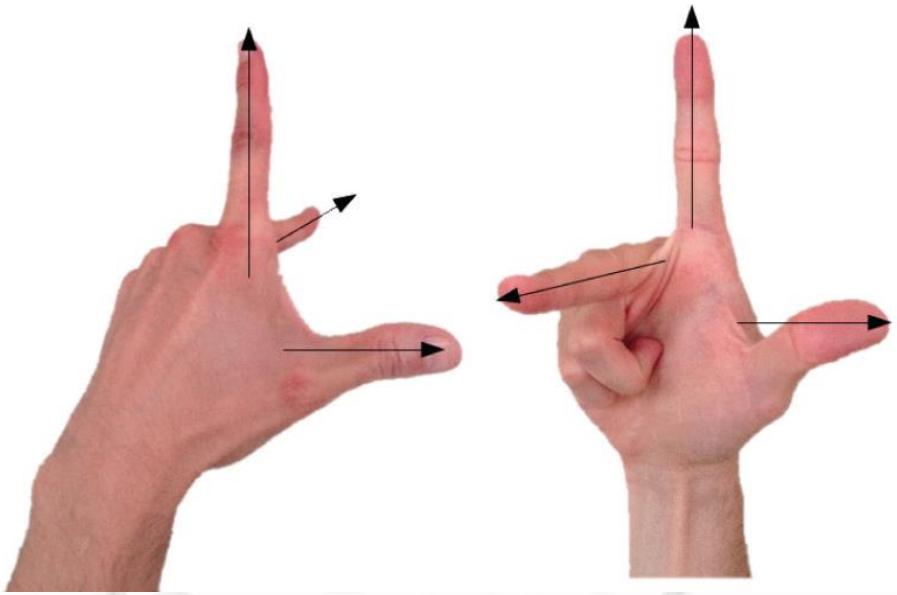
احتمالاً محورهای xو yرا برای اختصاص مختصات به نقاط صفحه دیده و از آنها استفاده کرده اید. به این سیستم مختصات **دکارتی** می گویند.



برای تعریف فضای سه بعدی از سه محور استفاده می شود. از آنجایی که محور xدر امتداد صفحه به صورت افقی و محور yدر امتداد صفحه به صورت عمودی است ، اکنون محور سومی را تصور می کنیم که مستقیماً به داخل و خارج صفحه می چسبد، عمود بر هر دو محور xوy است شکل روبرو محورهای -x، -yو zرا برای فضای مختصات سه بعدی نشان می دهد. هر چیزی که موقعیت خاصی در صحنه دارد دارای مختصات x، yو zخواهد بود: موقعیت بازیکن، قرار گرفتن دیوار و غیره.

- در نمای Unity’s Sceneمی توانید این سه محور را مشاهده کنید.
- در Inspectorمی توانید سه عدد مورد نیاز برای قرار دادن یک شی را تایپ کنید.
- در اسکریپت نویسی باید موقعیت اشیاء را با سه عدد مشخص کنید
- نه تنها موقعیت اشیاء ، بکله برای جا به جایی یک شی باید مقدار جابه جایی در راستای این سه محور را مشخص کنید

جهت مثبت و منفی هر محور دلخواه است و مختصات بدون توجه به اینکه محورها در کدام جهت قرار می گیرند همچنان کار می کنند. شما به سادگی نیاز به حفظ ثبات در یک ابزار گرافیکی سه بعدی (ابزار انیمیشن، ابزار توسعه بازی و غیره) دارید.



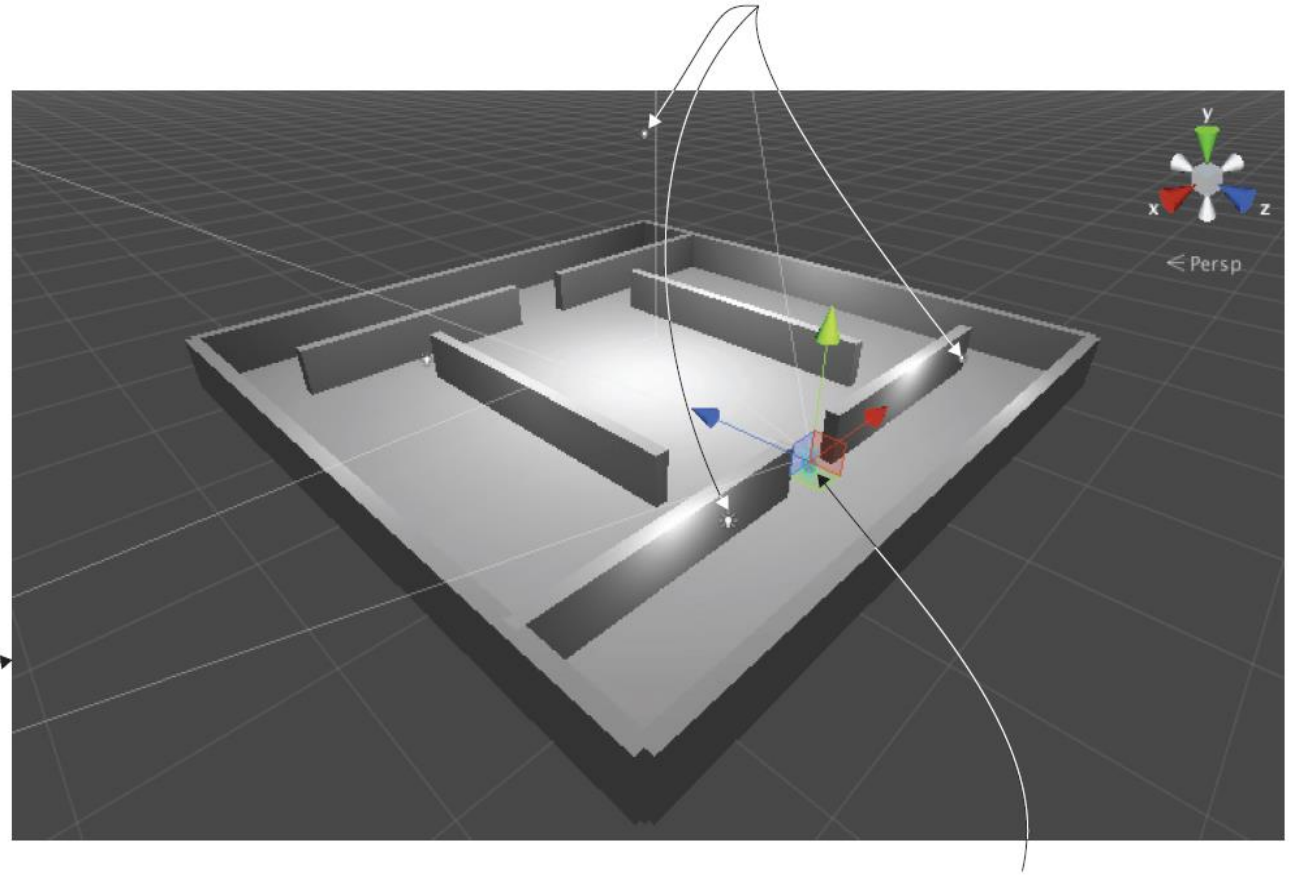
اما تقریباً در همه موارد، xبه سمت راست می رود و yبالا می رود. چیزی که بین ابزارهای مختلف تفاوت دارد این است که zوارد صفحه می شود یا از آن خارج می شود. به این دو جهت چپ دست یا راست دست گفته می شود. همانطور که این شکل نشان می دهد، اگر انگشت شست خود را در امتداد محور xو انگشت اشاره خود را در امتداد محور yقرار دهید، انگشت وسط شما در امتداد محور zقرار می گیرد.

یونیتی مانند بسیاری از برنامه های کاربردی هنر سه بعدی از **یک سیستم مختصات چپ استفاده می کند**. بسیاری از ابزارهای دیگر از سیستم مختصات راست دست استفاده می کنند (برای مثال OpenGL)، بنابراین اگر مسیرهای مختصات متفاوتی را مشاهده کردید، گیج نشوید.

Begin the project: Place objects in the scene

یک پروژه سه بعدی جدید بسازید و صحنه زیر را در آن طراحی نموده و آن را ذخیره کنید

نورها – دو نوع نور(جهتدار و نقطه ایی) در این صحنه هستند.

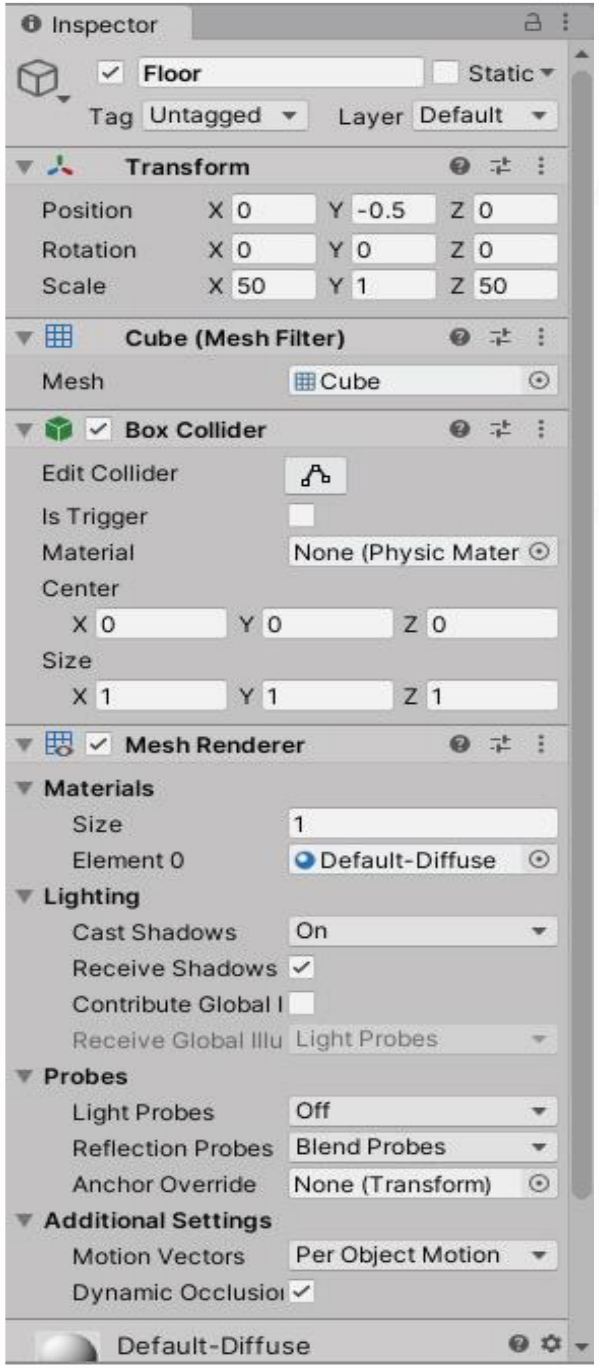


نمای دوربین - شی دوربین درست در بالای Player قرار دارد. این خطوط سفید زاویه دار میدان دید دوربین را نشان می دهد.

بازیکن - این یک شیء کپسولی اساسی است.

- منـــوی GameObject را در بـــالای صفحه انتخاب کنیـد و سـپس روی 3D Object نگه دارید تا آن منوی کشویی را ببینید.
- Cube را برای ایجاد یک شـی مکعـب جدید در صحنه انتخـاب کنیـد (بعـداً از اشـــکال دیگـــری ماننـــد Sphereو Capsuleاستفاده خواهیم کرد).
- موقعیت و مقیاس این جسم و همچنـین نام آن را برای ساختن کف تنظیم کنید.

اجزای باقی‌مانده که نمای را پر می‌کنند بـا یـک شـی Cubeجدیـد ارائه می‌شوند اما در حال حاضر نیازی به تنظیم ندارند. این اجزا شامل فیلتر مش (برای تعریف هندسه جسـم)، (Mesh Rendererبـرای تعریف ماده روی جسم) و (Box Colliderبه طـوری کـه بتـوان در حین حرکت با جسم برخورد کرد).



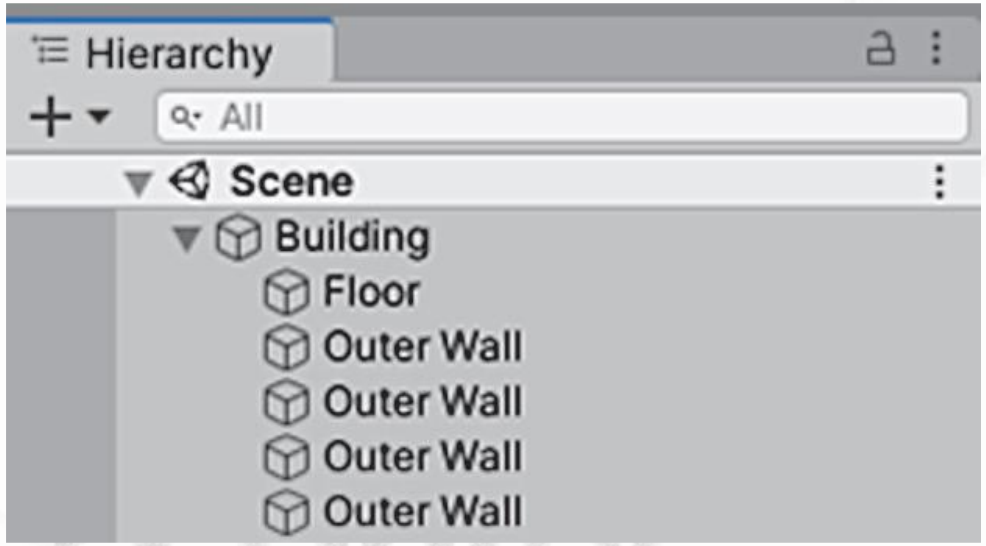
در بالا، می توانید یک نام برای شی تایپ کنید. به عنوان مثال، شی طبقه را Floorبنامیم.

مکعب را در موقعیت و مقیاس قرار دهید تا کفی برای اتاق ایجاد کنید. پس از کشیده شدن با مقادیر مختلف مقیاس در محورهای مختلف، دیگر شبیه یک مکعب نخواهد بود.

در همین حال، موقعیت کمی پایین می آید تا ارتفاع را جبران کند. ما مقیاس Yرا روی ۱ قرار می دهیم و جسم در اطراف مرکز خود قرار می گیرد.

در نمای سلسله مراتبی اشیا را روی هم بکشید تا ارتباط بین آنها ایجاد شود. به اشیایی که اشیاء دیگر به آنها متصل است والدین گفته می شود. اشیاء متصل به اشیاء والدین به عنوان فرزند شناخته می شوند. هنگامی که شی والد جابه جا می شود (یا می چرخد یا کوچک می شود)، اشیاء فرزند همراه با آن تغییر شکل می دهند.

شما همچنین می توانید اشیاء خالی بازی (empty game objects) ایجاد کنید تا از آنها برای سازماندهی صحنه استفاده کنید. از منوی GameObject، Create Emptyرا انتخاب کنید. با پیوند دادن اشیاء قابل مشاهده به یک شی ریشه، فهرست سلسله مراتب آنها را می توان مشاهده کرد. به عنوان مثال، در شکل ۲.۷، دیوارها همه فرزندان یک شی ریشه خالی (به نام Building) هستند، به طوری که فهرست سلسله مراتب سازمان یافته به نظر می رسد.



What is GameObject?

تمام اشیاء صحنه نمونه هایی از کلاس GameObjectهستند، مشابه روشی که همه اجزای اسکریپت از کلاس MonoBehaviourبه ارث می برند. این واقعیت در مورد شی خالی به نام GameObjectواضح تر بود، اما صرف نظر از اینکه نام شی Floor، Cameraیا Playerباشد، همچنان صادق است.

GameObject در حقیقت محفظه ای برای مجموعه ای از اجزا است. هدف اصلی GameObject ارائه چیزی برای اتصال به MonoBehaviour است. اینکه شی دقیقاً در صحنه چیست بستگی به این دارد که کدام مؤلفه به آن Game- Object اضافه شده است. اجسام مکعبی دارای مولفه مکعب هستند، اشیاء کره دارای جزء کره هستند و غیره.

Lights and cameras

به طور معمول، شما یک صحنه سه بعدی را با یک نور جهت دار و سپس یک سری نور نقطه ای روشن می کنید.

اضافه کردن نورهای مختلف به صحنه:
GameObject > Light

انواع نورهای موجود در یونیتی

شما می توانید انواع مختلفی از منابع نور ایجاد کنید که با نحوه و مکان انتشار پرتوهای نور تعریف می شوند. سه نوع اصلی عبارتند از: نقطه ای (point)، مخروطی(spot) و جهت دار (directional).

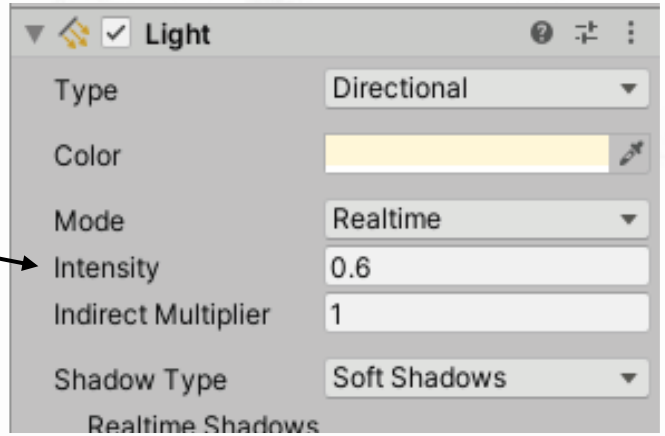
نور نقطه ایی : در چراغ‌های نقطه‌ای، تمام پرتوهای نور از یک نقطه منشأ می‌گیرند و مانند یک لامپ در دنیای واقعی در همه جهات پخش میشوند. نور از نزدیک روشن تر است زیرا پرتوهای نور به صورت دسته جمعی هستند.

نور مخروطی : در نورهای مخروطی، تمام پرتوهای نور از یک نقطه منشأ می گیرند، اما تنها در یک مخروط محدود بیرون می آیند. هیچ نور مخروطی در پروژه فعلی استفاده نمی شود، اما این چراغ ها معمولا برای برجسته کردن قسمت های یک سطح استفاده می شوند.

نور جهتدار : در نورهای جهت دار، تمام پرتوهای نور موازی هستند و به طور یکنواخت بیرون می آیند و همه چیز را در صحنه به یک شکل روشن می کنند. این مثل خورشید در دنیای واقعی است.

مکان نور جهت‌دار بر نوری که از آن می‌تابد تأثیر نمی‌گذارد، فقط جهتی که منبع نور دارد موثر است، بنابراین از نظر فنی، می‌توانید آن نور را در هر جایی از صحنه قرار دهید. من توصیه می‌کنم نور جهت‌دار را در بالای اتاق قرار دهید تا به طور شهودی مانند خورشید احساس شود و وقتی در حال دستکاری بقیه صحنه هستید، این نور دستکاری نشود.

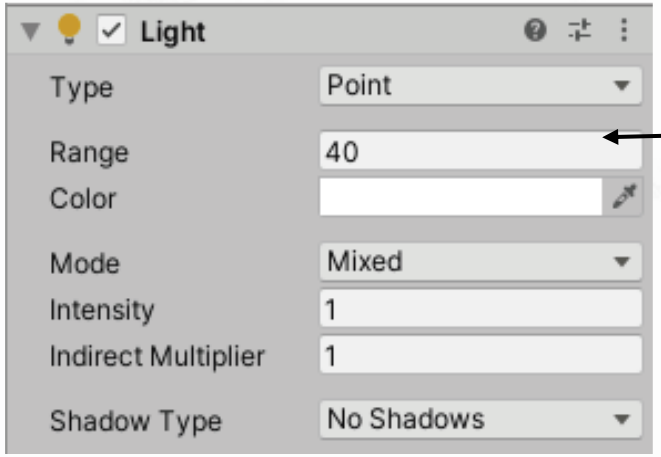
تنظیمات باقیمانده در حال حاضر نیازی به تنظیم ندارند. این تنظیمات شامل رنگ نور، سایه های ایجاد شده توسط نور، و ...



این خصوصیت شدت روشنایی نور را مشخص میکند و می تواند از ۰ تا یک تغییر کند.

در مورد چراغ‌های نقطه‌ای، با استفاده از منوی یکسان، چندین نور ایجاد کنید و آنها را در نقاط تاریک اطراف اتاق قرار دهید تا مطمئن شوید که تمام دیوارها روشن هستند. تعداد نورها، زیرا اگر بازی دارای نورهای زیادی باشد، عملکرد می تواند کاهش یابد.

نور جهتدار به تمام اشیاء نور را به صورت یکسان می تابند
در سایر نورها با نزدیک شدن جسم به منبع نوری شدت نور افزایش می یابد این فاصله توسط خصوصیت Range قابل تنظیم است.



شـما محـدوده نـور را بـا واحدهای مشابه موقعیت و مقیـاس کنتـرل مـی کنید.

به غیر از Range، تنظیمات برای چراغ های نقطه ای مانند چراغ های جهت است.

(اگر خطایی در مورد پشتیبانی نشدن بیدرنگ مشاهده کردید، کافی است آن را نادیده بگیرید یا حالت را به Mixedتغییر دهید.)

دوربین: شئ که برای تماشای صحنه استفاده می شود

در یونیتی امکان ایجاد چند دوربین وجود دارد اما همیشه یک دوربین به عنوان دوربین اصلی در نظر گرفته میشود

در بازی اول شخص ما از یک دورین در بالای سر بازی کننده استفاده می کنیم که صحنه را به کاربر نشان می دهد.

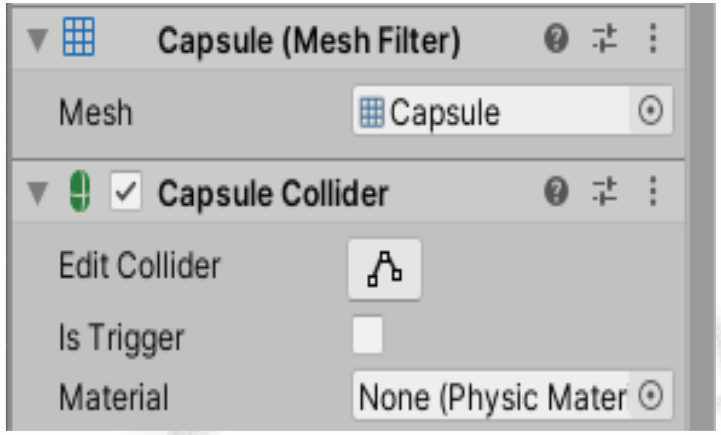
از یک کپسول بصورت انتزاعی برای نشان دادن بازی کننده در صحنه استفاده می کنیم

این کپسول را در مکان مورد نظر و مقداری بالاتر از سطح قرار دهید

این کپسول دارای خصوصیت (کامپوننت) ناحیه برخورد است این ناحیه بصورت پیش فرض برای همه اجسام وجود دارد

ما باید این کامپوننت را با کامپوننت کنترل کاربر جابه جا کنیم

با کلیک بر روی نماد منو در سمت راست بالای آن خصوصیت، برخورد دهنده کپسول را بردارید



به جای برخورد دهنده کپسولی، یک کنترل کننده کاراکتر به این شی اختصاص می دهیم. در پایین Inspectorدکمه ای با عنوان Add Componentوجود دارد. روی آن دکمه کلیک کنید تا منوی اجزایی که می توانید اضافه کنید باز شود. در بخش Physicsاین منو، Character Controllerرا خواهید دید. آن گزینه را انتخاب کنید همانطور که از نام آن مشخص است، این مولفه به شی اجازه می دهد تا مانند یک کاراکتر رفتار کند.

برای تنظیم شیء بازیکن باید یک مرحله آخر را انجام دهید: این مرحله شامل اتصال دوربین است

شیء دوربین را روی کپسول (بازیکن) بکشید تا دوربین به آن وصل شود.

اکنون دوربین را به گونه ای قرار دهید که مانند چشم بازیکن به نظر برسد (کتاب موقعیت 0 ، 0.5 ، 0 را پیشنهاد می کند).
در صورت لزوم ، چرخش دوربین را به (0, 0, 0) تنظیم کنید.

Make things move: A script that applies transforms

برای اینکه بازیکن در صحنه قدم بزند، اسکریپت های حرکتی را به آن متصل کنید. به یاد داشته باشید که کامپوننت ها توابع و قابلیت هایی هستند که به اشیا اضافه می کنید و اسکریپت ها نوعی کامپوننت هستند. در نهایت، آن اسکریپت‌ها به ورودی صفحه‌کلید و ماوس پاسخ می‌دهند، اما ابتدا بازی‌کننده (کپسول) را در جای خود قرار می‌دهید.

این شروع ساده به شما یاد می دهد که چگونه جابه جایی ها را در کد اعمال کنید. به یاد داشته باشید که سه نوع جابه جایی عبارتند از Translate، Rotate و Scale.
به یاد داشته باشید چرخش با تغییر حالت حول محورهای مختصات انجام می شود

Visualizing how movement is programmed

توضیح نحوه برنامه نویسی حرکت

برای متحرک سازی یک شی (مثل چرخاندن آن) در هر فریم مقدار کمی آن را حرکت دهید و در حالی که فریم ها پشت سر هم پخش می شوند در نهایت یک انیمیشن خواهیم داشت.

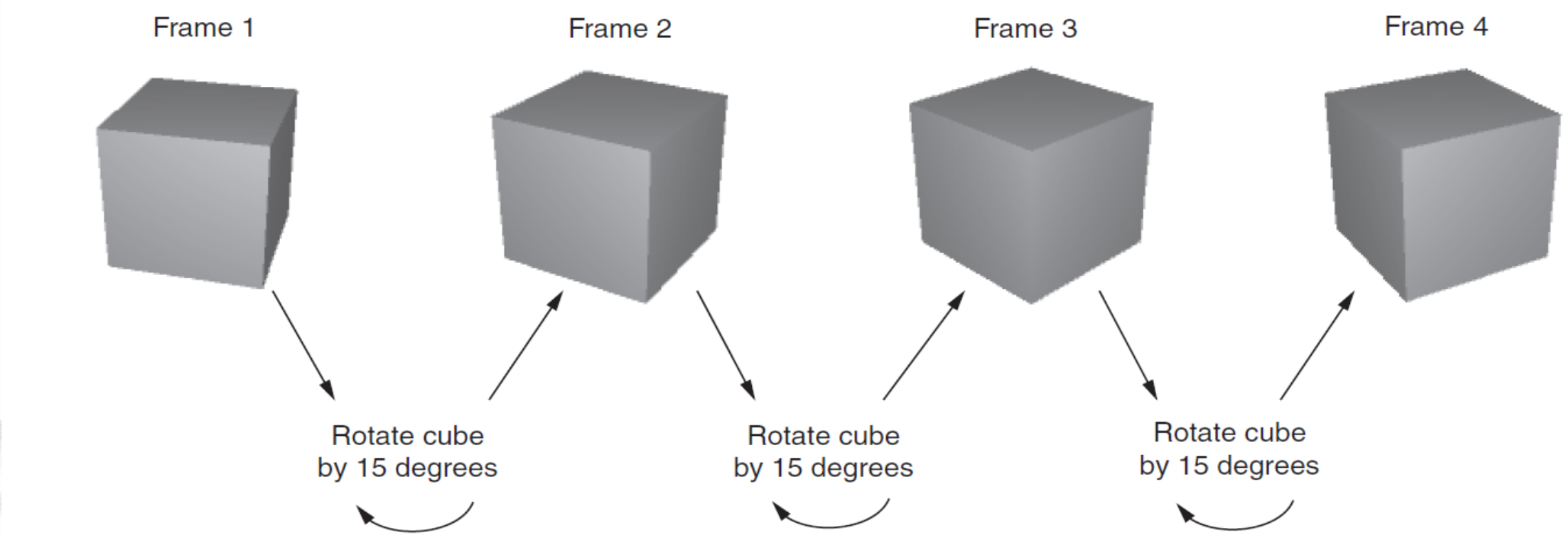


Figure 2.11 The appearance of movement: a cyclical process of transforming between still pictures

به یاد بیاورید که اجزای اسکریپت دارای یک متد Update() هستند که هر فریم را اجرا می کند. برای چرخاندن مکعب، کدی را درون Update()اضافه کنید که مکعب را کمی می‌چرخاند. این کد بارها و بارها در هر فریم اجرا می شود.

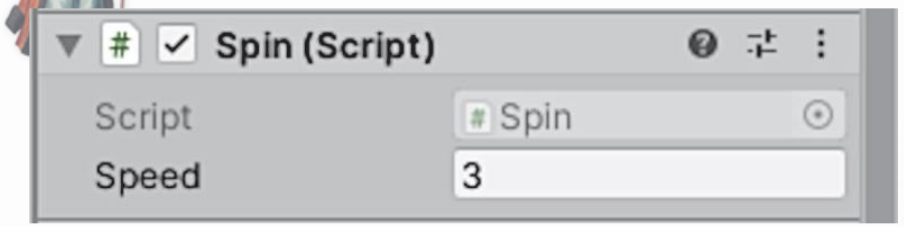
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Spin : MonoBehaviour {
public float speed = 3.0f;
void Update() {
transform.Rotate(0, speed, 0);
}
}
```

کلاس های مورد نیاز

یک متغیر public برای سرعت چرخش تعریف کنید.

دستور چرخش که در هر فریم اجرا میشود

متغیرهای عمومی (Public) در Inspector در دسترس هستند تا بتوانید پس از افزودن یک کامپوننت به یک شی بازی، مقادیر مؤلفه را تنظیم کنید. این به عنوان سریال سازی (*serializing*) مقدار نامیده می شود، زیرا Unity وضعیت تغییر یافته متغیر را ذخیره می کند.



تابع Rotate : این تابع در داخل Update() قرار دارد تا در هر فریم اجرا شود. Rotate()متدی از کلاس Transformاست.

به یاد بیاورید که سه محور در فضای سه بعدی وجود دارد که دارای برچسب x، yو zهستند. درک چگونگی ارتباط این محورها با موقعیت ها و حرکات کاملاً شهودی است، اما می توان از این محورها برای توصیف چرخش ها نیز استفاده کرد. در علم هوانوردی چرخش ها را به روشی مشابه توصیف می کند، بنابراین برنامه نویسانی که با گرافیک سه بعدی کار می کنند اغلب از مجموعه ای از اصطلاحات وام گرفته شده از علوم هوانوردی استفاده می کنند: pitch، yaw، و roll
گام چرخش حول محور x، انحراف چرخش حول محور yو رول چرخش حول محور zاست.

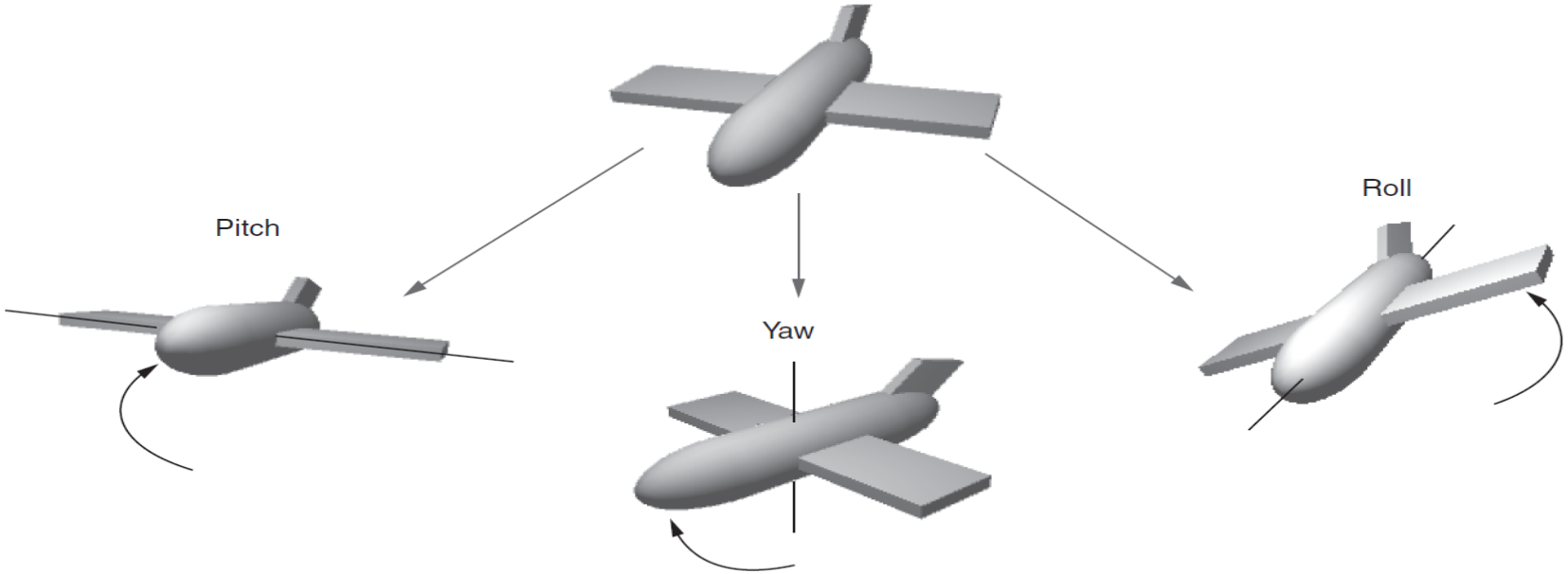


Figure 2.13 Illustration of pitch, yaw, and roll rotation of an aircraft

با توجه به اینکه می‌توانیم چرخش‌های حول محورهای x، yو zرا توصیف کنیم، به این معنی که سه پارامتر Rotate()چرخش X، Yو Zهستند. از آنجایی که می‌خواهیم بازیکن فقط به طرفین بچرخد، برخلاف چرخش به سمت بالا و پایین، فقط برای چرخش Yباید یک عدد داشته باشیم و برای چرخش Xو Zعدد 0 داده شود.

Understanding local vs. global coordinate space

مختصات جهانی (سراسری) با استفاده از پارامتر چهارم و با نوشتن عبارت Space.Self یا Space.World تنظیم می شود مانند: Rotate(0, speed, 0, Space.World).

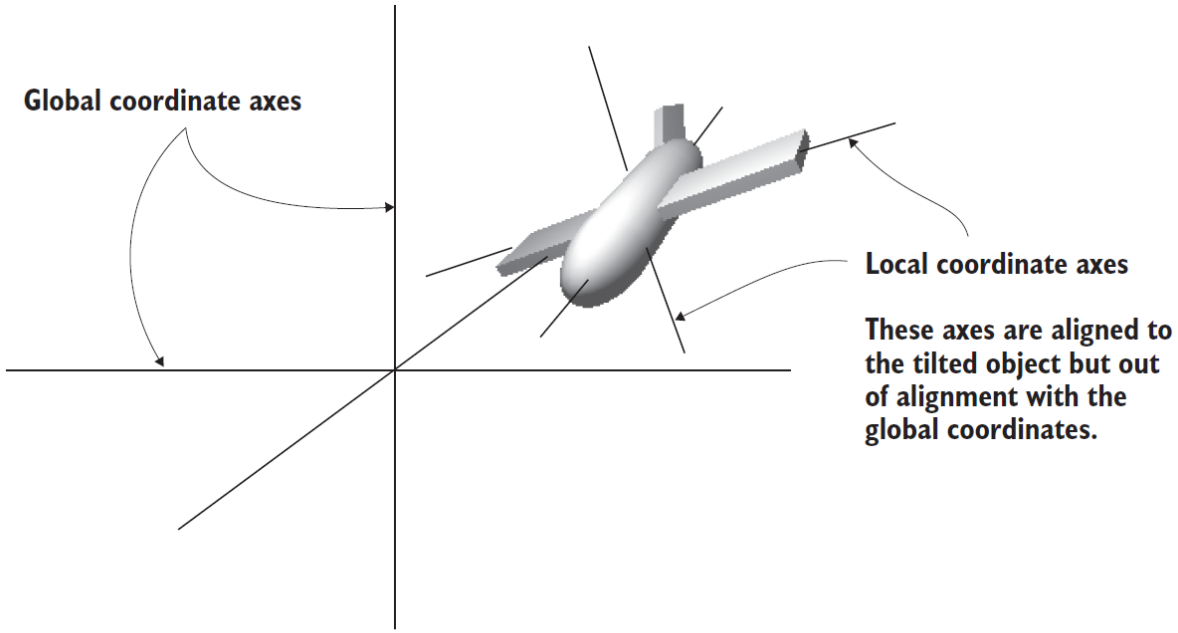


Figure 2.14 Local versus global coordinate axes

- به نظر می رسد که هر جسم ، نقطه مبدا خود را دارد و همچنین جهت خود را برای سه محور دارد و این سیستم مختصات با جسم حرکت می کند. این مختصات محلی (*local coordinates*) نامیده می شود.
- صحنه کلی سه بعدی نیز نقطه مبدا و جهت خود را برای سه محور دارد و این سیستم مختصات هرگز حرکت نمی کند. به این مختصات جهانی (*global coordinates*) می گویند.

Script component for looking around: MouseLook

می خواهیم با توجه به حرکت موس بازی کننده بچرخد و به اطراف نگاه کند یا کنترل زاویه دید با استفاده از موس

ابتدا بازیکن فقط به این طرف و آن طرف می چرخد و سپس بازیکن فقط بالا و پایین می چرخد. در نهایت، بازیکن می‌تواند در تمام جهات به اطراف نگاه کند (در یک زمان به صورت افقی و عمودی می‌چرخد)، رفتاری که به آن *mouse-look* می‌گویند.

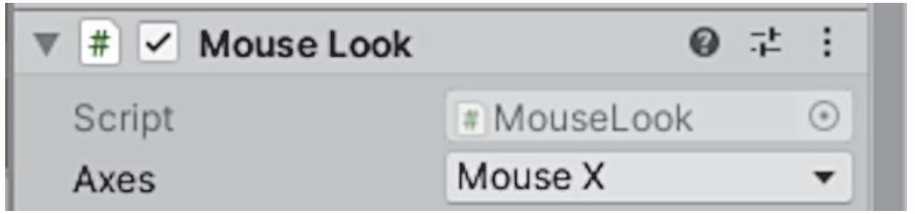
یک اسکریپت C#جدید ایجاد کنید، نام آن را MouseLook بگذارید

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class MouseLook : MonoBehaviour {
public enum RotationAxes {
MouseXAndY = 0,
MouseX = 1,
MouseY = 2
}
public RotationAxes axes = RotationAxes.MouseXAndY;
void Update() {
if (axes == RotationAxes.MouseX) {
// کد مربوط به چرخش افقی
}
else if (axes == RotationAxes.MouseY) {
// کد مربوط به چرخش عمودی
}
else {
// کد مربوط به چرخش افقی و عمودی همزمان
}
}
}
```

یک ساختار داده enum برای مرتبط کردن نام ها با تنظیمات تعریف کنید.

یک متغیر عمومی برای تنظیم در ویرایشگر Unityاعلام کنید.

توجه داشته باشید که از enumبرای انتخاب چرخش افقی یا عمودی برای اسکریپت Mouse- Look استفاده می شود. تعریف یک ساختار داده enumبه شما این امکان را می دهد که مقادیر را با نام تنظیم کنید، نه اینکه اعداد را تایپ کنید و سعی کنید معنی هر عدد را به خاطر بسپارید.



Horizontal rotation that tracks mouse movement

```
...
public RotationAxes axes = RotationAxes.MouseXAndY;
public float sensitivityHor = 9.0f;
void Update() {
if (axes == RotationAxes.MouseX) {
transform.Rotate(0, sensitivityHor, 0);
}
...
```

یک متغیر برای سرعت چرخش اعلام کنید.

دستور Rotateرا در اینجا قرار دهید تا هر فریم اجرا شود.

دریافت ورودی از موس برای چرخش بازی کن بصورت افقی

برای این کار از تابع `Input.GetAxis()` استفاده می کنیم.

- کلاس Input مجموعه ای از متدها برای مدیریت دستگاه های ورودی (مانند ماوس) دارد
- تابع GetAxis متناسب با حرکت موس در جهت مشخص شده عددی بین ۱- تا ۱ را برمی گرداند این تابع نام محور مورد نظر را به عنوان ورودی دریافت می کند برای مثال محور افقی `GetAxis("Mouse X")`
- اگر مقدار بازگشتی تابع `GetAxis` را در سرعت چرخش ضرب کنیم آن گاه چرخش متناسب با حرکت موس اتفاق می افتد
- با توجه به حرکت ماوس کاهش می‌یابد و به صفر می‌رسد یا حتی جهت معکوس را کاهش می‌دهد.

```
transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityHor, 0);
```

Vertical rotation with limits

برای چرخش افقی، ما از متد Rotate() استفاده کرده‌ایم، اما با چرخش عمودی رویکرد متفاوتی خواهیم داشت. اگرچه این روش برای اعمال انتقال و حرکت راحت است، اما به نوعی غیر قابل انعطاف است. این فقط برای افزایش چرخش بدون محدودیت مفید است، که برای چرخش افقی خوب بود، اما چرخش عمودی به محدودیت‌هایی در مورد میزان کج شدن بازی کن (کپسول) به بالا یا پایین نیاز دارد.

```
...
public float sensitivityHor = 9.0f;
public float sensitivityVert = 9.0f;
public float minimumVert = -45.0f;
public float maximumVert = 45.0f;
private float verticalRot = 0;
void Update() {
if (axes == RotationAxes.MouseX) {
transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityHor, 0);
}
else if (axes == RotationAxes.MouseY) {
verticalRot -= Input.GetAxis("Mouse Y") * sensitivityVert;
verticalRot = Mathf.Clamp(verticalRot, minimumVert, maximumVert);
float horizontalRot = transform.localEulerAngles.y;
transform.localEulerAngles = new Vector3(verticalRot, horizontalRot, 0);
}
...
```

متغیرهای مورد استفاده برای چرخش عمودی را اعلام کنید.

یک متغیر خصوصی برای زاویه عمودی تعریف کنید.

زاویه عمودی را بر اساس حرکت ماوس افزایش دهید.

محدود کردن زاویه چرخش

دریافت زوایه فعلی

تنظیم یک مقدار جدید برای زاویه چرخش جسم (در این جا از تابع Rotate استفاده نکردیم)

این کد چندین مفهوم جدید را معرفی می کند که نیاز به توضیح دارد.

- اول از همه، ما این بار از Rotate()استفاده نمی کنیم، بنابراین به متغیری نیاز داریم که زاویه چرخش را در آن ذخیره کنیم (این متغیر در اینجا verticalRotنامیده می شود و به یاد داشته باشید که چرخش عمودی حول محور xمی رود).
- تابع Rotate() چرخش جاری (زاویه فعلی جسم) را افزایش می دهد، در حالی که این کد زاویه چرخش را مستقیماً تنظیم می کند. این تفاوت بین گفتن "افزودن ۵ به زاویه" و "تنظیم زاویه ۳۰" است.
- با استفاده نکردن از Rotate()، می‌توانیم زاویه چرخش را به روش‌های مختلف دستکاری کنیم و فقط آن را افزایش دهیم.
- مقدار چرخش را در Input.GetAxis()ضرب می شود، همانطور که در کد چرخش افقی است، با این تفاوت که اکنون ماوس Yرا درخواست می کنیم زیرا این محور عمودی ماوس است.

- ما از Mathf.Clamp()برای حفظ زاویه چرخش بین حداقل و حداکثر محدودیت استفاده می کنیم. این محدودیت‌ها متغیرهای هستند که قبلاً در کد اعلام شده‌اند و تضمین می‌کنند که بازیکن فقط ۴۵ درجه به بالا یا پایین متمایل می‌شود.
- متد Clamp() مختص چرخش نیست، اما عموماً برای نگه داشتن متغیر عددی بین محدودیت‌ها مفید است.
- برای دیدن تاثیر این دستور می توانید یکبار آن را حذف کنید

DEFINITION A *vector* is multiple numbers stored together as a unit. For example, a `Vector3` is three numbers (labeled x, y, z).

اخطار دلیل اینکه ما نیاز به ایجاد یک Vector3جدید به جای تغییر مقادیر در بردار موجود در transform داریم این است که این مقادیر فقط خواندنی هستند. این یک اشتباه رایج است که می تواند شما را غافلگیر کند.

Horizontal and vertical rotation at the same time

در این جا نیز از Rotate() استفاده نمیشود،
به همین دلیل: در چرخش عمود باید زاویه چرخش را محدود کنیم و به همین دلیل در این جا نیز مجبور هستیم بطور دستی هر دور زاویه عمودی و افقی را محاسبه کنیم.

```
...
else {
verticalRot -= Input.GetAxis("Mouse Y") * sensitivityVert;
verticalRot = Mathf.Clamp(verticalRot, minimumVert, maximumVert);

float delta = Input.GetAxis("Mouse X") * sensitivityHor;
float horizontalRot = transform.localEulerAngles.y + delta;

transform.localEulerAngles = new Vector3(verticalRot, horizontalRot, 0);
}
...
```

دلتا مقدار تغییر چرخش را مشخص میکند.

در نهایت، هر دو زاویه، عمودی و افقی، برای ایجاد یک بردار جدید که به ویژگی زاویه کامپوننت transform تخصیص داده می شود، استفاده می شود.

Keyboard input component: First-person controls

- نگاه کردن به اطراف در پاسخ به ورودی ماوس بخش مهمی از کنترل‌های اول شخص است، اما شما فقط در نیمه راه هستید. بازی کننده همچنین باید در پاسخ به ورودی صفحه کلید حرکت کند.
- بیایید یک جزء کنترل صفحه کلید بنویسیم تا جزء کنترل ماوس را تکمیل کند.
- یک اسکریپت C#جدید به نام FPSInput ایجاد کنید و آن را به پخش کننده متصل کنید (در کنار اسکریپت MouseLook). در حال حاضر، مؤلفه MouseLookرا فقط روی چرخش افقی تنظیم کنید.

نکته کنترل های صفحه کلید و ماوس که در اینجا توضیح داده شده اند به اسکریپت های جداگانه تقسیم می شوند. شما مجبور نیستید کد را به این شکل ساختار دهید و می توانید همه چیز را در یک اسکریپت واحد قرار دهید. اما یک سیستم کامپوننت (مانند سیستم یونیتی) زمانی که عملکردها به چندین مؤلفه کوچکتر تقسیم شده است، انعطاف پذیرتر و در نتیجه بیشترین کاربرد را دارند.

- در این قسمت به جای تابع Rotate از تابع `Translate()` استفاده میکنیم

یک اسکریپت ساده که در آن جسم با یک سرعت ثابت و مشخص حرکت می کند

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class FPSInput : MonoBehaviour {
public float speed = 6.0f;
void Update() {
transform.Translate(0, speed, 0);
}
}
```

Responding to keypresses

```
...
void Update() {
float deltaX = Input.GetAxis("Horizontal") * speed;
float deltaZ = Input.GetAxis("Vertical") * speed;
transform.Translate(deltaX, 0, deltaZ);
}
...
```

Horizontalو Vertical نام های غیرمستقیم برای نگاشت صفحه کلید هستند.

- این نام ها انتزاعی برای Input در Unityهستند. اگر به منوی Editدر قسمت تنظیمات پروژه نگاه کنید و سپس به قسمت Input Managerنگاه کنید، لیستی از نام‌های ورودی انتزاعی و کنترل‌های دقیق نگاشت شده با آن نام‌ها را خواهید یافت.
- هر دو کلیدهای جهت دار چپ و راست و حروف Aو Dبه صورت افقی نگاشت می شوند، در حالی که هر دو کلید جهت دار بالا و پایین و حروف Wو S به صورت عمودی نگاشت می شوند.

Setting a rate of movement independent of the computer's speed

کامپیوترهای مختلف دارای سرعت پردازش مختلف هستند و این باعث خواهد شد که سرعت حرکت اجسام بازی در کامپیوترهای مختلف متفاوت باشد

- تصور کنید این پروژه را روی دو کامپیوتر اجرا می کنید، یکی با سرعت ۳۰ فریم در ثانیه (فریم بر ثانیه) و دیگری با سرعت ۶۰ فریم در ثانیه. این بدان معناست که Update() دو برابر در رایانه دوم فراخوانی می شود.
- اگر میزان حرکت در هر فراخوانی ۶ واحد باشد در ۳۰ فریم در ثانیه، سرعت حرکت ۱۸۰ واحد در ثانیه و حرکت در ۶۰ فریم بر ثانیه ۳۶۰ واحد در ثانیه خواهد بود.

راه حل این است که کد حرکت را تنظیم کنید تا از نرخ فریم (Frame rate) مستقل شود. این سرعت حرکت به نرخ فریم بازی بستگی ندارد. راه دستیابی به این هدف، استفاده نکردن از مقدار سرعت یکسان در هر Frame rate است. در عوض، بسته به سرعت کارکرد کامپیوتر، مقدار سرعت را بیشتر یا کمتر کنید. این کار با ضرب مقدار سرعت در مقدار دیگری به نام deltaTime حاصل می شود.

```
...
void Update() {

float deltaX = Input.GetAxis("Horizontal") * speed;
float deltaZ = Input.GetAxis("Vertical") * speed;

transform.Translate(deltaX * Time.deltaTime, 0, deltaZ * Time.deltaTime);
}
...
```

Moving the CharacterController for collision detection

- بازیکنی که تا این ساخته شده توانایی تشخیص برخورد را ندارد. برای اعمال تشخیص برخورد، کاری که می‌خواهیم انجام دهیم استفاده از CharacterControllerاست، کامپوننت که باعث می‌شود جسم بیشتر شبیه یک شخصیت در بازی حرکت کند، از جمله برخورد با دیوارها.

- به یاد بیاورید که زمانی که پخش کننده را راه اندازی کردیم، یک CharacterController را وصل کردیم، بنابراین اکنون از آن جزء با کد حرکت در FPSInput استفاده می کنیم.

```
...
private CharacterController charController;
void Start() {
charController = GetComponent<CharacterController>();
}
void Update() {
float deltaX = Input.GetAxis("Horizontal") * speed;
float deltaZ = Input.GetAxis("Vertical") * speed;
Vector3 movement = new Vector3(deltaX, 0, deltaZ);
movement = Vector3.ClampMagnitude(movement, speed);
movement *= Time.deltaTime;
movement = transform.TransformDirection(movement);
charController.Move(movement);
}
...
```

متغیر برای دستری و ارجاع به CharacterController

دسترسی به کامپوننت متصل به جسم

محدود کردن مقدار بردار حرکت با توجه به speed
(مقدار حرکت مورب می تواند حداکثر به اندازه speed باشد)

بردار حرکت را از مختصات محلی به سراسری تبدیل کنید.

به CharacterController بگویید بر اساس آن بردار حرکت کند.

متغیر charController در ابتدا تهی است ، بنابراین قبل از اینکه بتوانید از آن استفاده کنید، باید یک شی را به آن اختصاص دهید تا به آن رجوع کند. از تابع Start استفاده خواهیم کرد. متد GetComponent() سایر اجزای متصل به همان GameObjectرا برمی گرداند. به جای ارسال یک پارامتر در داخل پرانتز، شما از دستور سی شارپ برای تعریف نوع درون براکت ها، <> استفاده می کنید.

- هنگامی که از CharacterController استفاده می کنید، می توانید Move()را برای جابه جا کردن جسم فراخوانی کنید. ورودی این تابع مشابه تابع چرخش می باشد که قبلا از آن استفاده کردیم.
- همچنین برای جلوگیری از حرکت زیاد از Vector3.ClampMagnitude()برای محدود کردن بزرگی بردار حرکت استفاده کنید. علت استفاده از این تابع این است که می خواهیم بزرگی اندازه بردار حرکت به درستی تنظیم شود.
- همیشه حرکت در راستای محورهای مختصات نیست و ما برای داشتن مقدار مشخصی تغییر در اندازه بردار، مقدار تغییرات در راستای محورهای مختلف یکسان نیست
- دقت داشته باشید که این محاسبات جابه جایی ما با توجه به محور مختصات محلی انجام می شود که مرکز آن روی Player قرار دارد و ممکن است با محور مختصات سراسری متفاوت باشد به عبارت دیگر ما دلتای حرکت را محاسبه می کنیم ولی هنوز مختصات نهایی جسم بر طبق محور مختصات سراسری حساب شود
- ما باید یک بردار حرکت تعریف شده در محور مختصات جهانی را به متد () Moveارسال کنیم، بنابراین باید بردار محور مختصات محلی را به بردار محور مختصات سراسری تبدیل کنیم. برای این کار تابع TransformDirection() فراخوانی می کنیم.

با استفاده از کنترل های صفحه کلید می توانید به طور کامل به اطراف صحنه نگاه کنید و در اطراف صحنه پرواز کنید. اگر بخواهید بازیکن در اطراف صحنه پرواز کند، بسیار عالی است، اما اگر بخواهید بازیکن به جای پرواز راه برود، چه؟

Adjusting components for walking instead of flying

اکنون که تشخیص برخورد کار می‌کند، اسکریپت می‌تواند جاذبه را شبیه سازی کند و بازیکن روی زمین راه برود. یک متغیر جاذبه را تعریف کنید و از آن مقدار برای محور y استفاده کنید.

```
...
public float gravity = -9.8f;
...
void Update() {
...
movement = Vector3.ClampMagnitude(movement, speed);
movement.y = gravity;
movement *= Time.deltaTime;
...
```

- اکنون یک نیروی رو به پایین ثابت روی بازیکن (کپسول) وجود دارد،
- این نیرو همیشه رو به پایین است ولی بازیکن با حرکت موس به سمت بالا و پایین جابه جا می شود و وجود نیروی جاذبه سبب مشکل و زمین خوردن جسم می شود.
- خوشبختانه، همه چیزهایی که ما نیاز به تعمیر داریم که در حال حاضر وجود دارد، بنابراین فقط باید تنظیمات جزئی را در نحوه تنظیم اجزا دربازیکن انجام دهیم. ما باید از دو کامپوننت MouseLook استفاده کنیم . یکی بر روی **کپسول بازیکن** که فقط دارای **حرکت افقی** است و دیگر بر روی دوربین بازیکن و آن را طوری تنظیم کنیم که فقط حرکت عمودی داشته باشد
- ابتدا مولفه MouseLookرا روی شی پخش کننده فقط روی چرخش افقی تنظیم کنید. مولفه MouseLookرا به شی دوربین اضافه کنید و آن را فقط روی چرخش عمودی تنظیم کنید. درست است؛ شما دو شی خواهید داشت که به ماوس پاسخ می دهند!

بهبود دادن اسکریپت نهایی

- از ویژگی RequireComponentاستفاده کنید تا مطمئن شوید که سایر اجزای مورد نیاز اسکریپت نیز پیوست شده اند.
- RequireComponent را به بالای اسکریپت اضافه کنید و کامپوننت مورد نیاز را به عنوان پارامتر در داخل پرانتز قرار دهید.

- به طور مشابه، اگر ویژگی AddComponentMenuرا به بالای اسکریپت های خود اضافه کنید، آن اسکریپت به منوی مؤلفه در ویرایشگر Unityاضافه می شود.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
[RequireComponent(typeof(CharacterController))]
[AddComponentMenu("Control Script/FPS Input")]
public class FPSInput : MonoBehaviour {
...
```