آموزش ساخت بازی اول شخص در Unity

قسمت دوم (سطح متوسط)

مترجم و مجری: علی زنجیران (AliyerEdon)

آبان-آذر 1388

مقدمه

سلام. سلامی داغ مخصوص این روزهای زمستانی!! البته پاییزی!! امید دارم بهترین لحظه ها رو تجربه می کنید و بسیار مهیج هستید تا قسمت دوم از آموزش رو بخونید. همونطور که می دونید این دومین قسمت از آموزش ساخت بازی اول شخص شوتر با یونیتی ه و بعد از این یک قسمت دیگه باقی مونده. من بهتون قول می دم با خوندن این آموزش ها دیگه یه مستر برای خودتون می شید و هر نوع بازی ای که خواستید با یونیتی می سازید. برای طرفدارهای آنریل انجین 3 هم یه چیزی رو بگم. اینکه اگه شما این آموزش ها رو بخونید و تو یونیتی اجرا کنید، تکنیک های ساخت بازی های اول شخص رو یاد می گیرید و می تونید از این تکنیک ها تو هر انجین دیگه ای استفاده کنید. الان من چون اصول ساخت بازی رو توسط یونیتی یاد گرفتم، پس با هر انجینی می تونم چیزی که تو یونیتی ساختم رو پورت کنم. چون آموزش های نوشته شده در یونیتی تکنیک ها واصول ساخت بازی رو آموزش داده که این آموزش ها رو من تو هیچ انجین دیگه ای ندیدم. پس پیش به سوی خوش ساخت ترین انجین دنیا!! یو-نی-تی !!!!

یک بازی اول شخص توسعه یافته در Unity

این قسمت از آموزش که دارای سطح متوسط است،قسمت اول آمـوزش (مقـدماتی) را بـا اضـافه کردن عناصر جدید تر مانند چندین اسلحه، سیستم آسیب پذیری و دشمنان توسعه می دهد.

زمان برای تکمیل : 3-4 ساعت

نویسنده : گراهام مک آلیستر

فهرست:

1. هدفهای آموزش

این قسمت از آموزش در مورد جزئیات اضافه کردن قابلیت ها ی جدید به بازی اول شخص، مانند چندین اسلحه، سیستم آسیب پذیری و دشمنان پایه (اسلحه ی نگهبان)، توضیح می دهد.

2. پیش نیاز ها

شما باید از قبل با مفاهیمی که در قسمت قبلی آموزش توضیح داده شد آگاهی داشته باشید.

نکته : در هر متنی که با حرف "-" شروع می شود، باید عملی توسط شما انجام شود.

قبل از شروع - تنظیم مرحله

- پروژه ی جدیدی ایجاد کنید (همانطور که در قسمت قبل آموزش داده شد). مطمئن شوید که محتویات دانلود کرده را Import کرده اید.

- صحنه ی جدیدی را ایجاد کرده و FPS Controller را به آن اضافه کنید.

توجه : در این قسمت از آموزش ما اسکریپت جدیدی ایجاد نخواهیم کرد و از اسکریپت های دانلود شده استفاده می کنیم.

3. تعویض اسلحه

قبل از اینکه ما در مورد چگونگی ساخت هر اسلحه صحبت کنیم، ابتدا باید مقداری کد بنویسیم تا اسلحه های ما را مدیریت کند و با کلید های 1، 2 و ... آنها را تعویض کند. بیایید به کد جاوااسکریپت PlayerWeopens.js نگاهی بیاندازیم:

```
function Awake () {
     // Select the first weapon
     SelectWeapon(0);
}


function Update () {
     // Did the user press fire?
     if (Input.GetButton ("Fire1"))
          BroadcastMessage("Fire");

     if (Input.GetKeyDown("1"))
     {
          SelectWeapon(0);
     }
     else if (Input.GetKeyDown("2"))
     {
          SelectWeapon(1);
     }
}
function SelectWeapon (index : int) {
     for (var i=0;i<transform.childCount;i++)
     {
          // Activate the selected weapon
          if (i == index)
               transform.GetChild(i).gameObject.SetActiveRecursively(true);
          // Deactivate all other weapons
          else
               transform.GetChild(i).gameObject.SetActiveRecursively(false);
     }
}
```

1. این تابع اسلحه ی 0 را به عنوان اسلحه ی پیش فرض انتخاب می کند.

2. این تابع ورودی کیبورد را بررسی می کند. کلید Fire (چپ ماوس)، کلید 1 برای اسلحه ی اول، کلید 2 برای اسلحه ی دوم. اسلحه ها باید فرزند (Childe) یا زیرمجموعه ی شی Main Camera باشند.

3. فعال کننده ی اسلحه ی مناسب بر اساس ورودی کیبورد.

بیایید کد بالا را استفاده کنیم:

- شی جدیدی را ایجاد کنید و نام آن را Weopens بگذارید. آن را درگ کرده و بر روی شی Main Camera رها کنید تا فرزند آن شود (در داخل FPS Controller و در زیرمجموعه ی Main Camera). اسلحه های ما در زیرمجموعه ی این شی قرار خواهند گرفت.

- اسـکریپت PlayerWeopens.js را بـه شـی Weopens در زیـر مجموعـه ی Main Camera اضافه کنید.

حال ما اولین اسلحه ی خود را می سازیم.

4. موشک انداز

این قسمت آموزش می دهد که چگونه اسلحه هایی در سبک موشک انداز بسازید.

موشک انداز

موشک انداز (RocketLauncher) مسئول ایجاد یک راکت و دادن نیروی اولیه به آن است. راکت نیز به سمتی که دوربین به آنجا نگاه مـی کنـد پرتـاب شـده و بعـد از برخـورد بـا شـی ای منفجر شده و نابود می شود.

- یک شی خالی ایجاد کنید و نام آن را RocketLauncher بگذارید. مکـان آن را جـایی قـرار دهید که دست های FPS Controller در آنجا قرار می گیرد.

- شی RocketLauncher را فرزند شی Weopens در زیـر مجموعـه ی Main Camera کنید. این کار باعث می شود ک شی موشک انداز ما در جایی که دوربین نگاه می کند قرار گیـرد و با حرکت FPS Controler بـا آن حرکـت کنـد.(چـون شـی Main Camera فرزنـد شـی FPS Controller است).

- مـــــدل Objects/Weopens/RocketLauncher را بـــــه عنـــــوان فرزنـــــد

RocketLauncher اضافه کنید.

- مدل اسلحه را تغییر اندازه، چرخش و انتقال دهید تا در جای مناسبی قرار گیرد. دقت کنید که شی خالی RocketLauncher را حرکت ندهید و مدل اسلحه را تکان دهید.

- کد اسکریپت RocketLauncher.js به شرح زیر است:

```
var projectile : Rigidbody;
var initialSpeed = 20.0;
var reloadTime = 0.5;
var ammoCount = 20;
private var lastShot = -10.0;

function Fire ()
{
    // Did the time exceed the reload time?
    if (Time.time > reloadTime + lastShot && ammoCount > 0)
    {
        // create a new projectile, use the same position and rotation as the
        // Launcher.
        var instantiatedProjectile : Rigidbody = Instantiate (projectile,
            transform.position, transform.rotation);
        // Give it an initial forward velocity. The direction is along the z-axis of
        // the missile launcher's transform.
        instantiatedProjectile.velocity = transform.TransformDirection(Vector3 (0, 0,
            initialSpeed));

        // Ignore collisions between the missile and the character controller
        Physics.IgnoreCollision(instantiatedProjectile.collider,
            transform.root.collider);

        lastShot = Time.time;
        ammoCount--;
    }
}
```

1. این کد مطمئن می شود که اسلحه تند تر از زمان reloadTime نمی تواند شلیک کند. همچنین بررسی می کند تا مطمئن شود بازیکن مهمات کافی برای شلیک دارد یا خیر.

رفتار موشک انداز شبیه قسمت قبل آموزش است و فقط در قسمت های reloadTime و تعداد مهمات متفاوت است.

- اسکریپت RocketLauncher.js را به شی RocketLauncher اضافه کنید.

راکت (Rocket) :

حال ما یک راکت در صحنه می سازیم و مدل نهایی آن را به داخل یک Prefab آپلود می کنیم.

- مدل Objects/Weopens/rocket را به داخل Scene View درگ کنید.
- اسکریپت WeopensScript/Rocket را به آن اضافه کنید.
- یک Box Collider به آن اضافه کنید. اندازه ی آن را بزرگتر کنید تا بهتر برخورد کند. چون اشیاء کوچک با توجه به سایز و سرعتشان ممکن است برخوردشان نادیده گرفته شود. پس محور Z این Collider را بزرگتر کنید تا از برخورد آن مطمئن شویم.
- یک Particle System جدید ایجاد کنید (GameObject/Create Other/Particle System).
- مقدار elipsoid ه XYZ آن را برابر 0,1 کنید.
- مقدار Rnd Velocity را برای هر محور 0,1 قرار دهید.
- مقدار Min Size و Max Size را 0,5 قرار دهید.
- مقدار Min Emit و Max Emit را 100 کنید.
- متریال ParticleEffects/Smoke.mat را بر روی Particle System درگ کنید.
- در قسمت Particle Animator مقدار هر محور را 0,5 قرار دهید.
- در قسمت RigidBody راکت، خاصیت Use Gravity را غیر فعال کنید. این باعث می شود راکت از جاذبه تاثیر نگیرد.
- مقدار Size Grow را 3 قرار دهید.
- خاصیت auto destruct را فعال کنید. این باعث می شود وقتی که راکت نابود شد، Particle System از کار بیافتد.
- Particle System را در قسمت Hierarchy View درگ کرده و فرزند Rocket کنید. transform راکت را Reset کنید تا در مرکز راکت قرار گیرد. سپس آن را به پشت راکت منتقل کنید.
- راکت را در Hierarchy View انتخاب کرده و با حرکت دادن آن در Scene View مطمئن شوید که دود آن را دنبال می کند.

حال ما راکت را کامل کردیم و باید آن را به داخل یک Prefab آپلود کنیم.

- ابتدا یک Prefab خالی بسازید و نام آن را rocket بگذارید.
- Rocket را در Hierarchy View انتخاب کرده و آن را به داخل Prefab ه rocket در

Project View درگ کنید.

- پوشه ی جدیـدی در Project View سـاخته و نـام آن را Weopen Prefabs بگذاریـد و Prefab ه rocket را به داخل آن درگ کنید.

در زیر کد جاوااسکریپت Rocket.js را می بینید:

```
// The reference to the explosion prefab
var explosion : GameObject;
var timeOut = 3.0;

// Kill the rocket after a while automatically
function Start () {
     Invoke("Kill", timeOut);
}


function OnCollisionEnter (collision : Collision) {
     // Instantiate explosion at the impact point and rotate the explosion
     // so that the y-axis faces along the surface normal
     var contact : ContactPoint = collision.contacts[0];
     var rotation = Quaternion.FromToRotation(Vector3.up, contact.normal);
    Instantiate (explosion, contact.point, rotation);

     // And kill ourselves
     Kill ();
}

function Kill ()
{
     // Stop emitting particles in any children
     var emitter : ParticleEmitter= GetComponentInChildren(ParticleEmitter);
     if (emitter)
          emitter.emit = false;

     // Detach children - We do this to detach the trail rendererer which should be set
     // up to auto destruct
     transform.DetachChildren();

     // Destroy the projectile
     Destroy(gameObject);
}


@script RequireComponent (Rigidbody)
```

1

1. تابع Kill ابتدا Particle Emmiter (ساتع کننده ی ذرات) را در زیر مجموعه ی شی Rocket پیدا می کند و آن را خاموش می کند. سپس تمام فرزندان شی Rocket را جدا کرده و بعد Smoketrail و راکت را از بین می برد.

2. مهمترین قسمت transform.DetachChildren است.این خط فرزندان شـی Rocket را جدا می کند و سپس Rocket را نابود می کند ولـی دیگـر Smoketrail ه اضـافه شـده بـه آن فرزند آن نیست.

3.دستور "@Script" مطمئن می شود که شی Rocket دارای کامپوننـت Rigidbody اسـت.

زمانی که راکت ایجاد شد و با اشیای دیگر برخورد کرد، ما می خواهیم که شی Rocket را نابود کنیم. پس چون Smoketrail به راکت اضافه شده است (فرزند آن است)، پس وقتی راکت نابود شد، آن نیز ناگهان ناپدید می شود. این چیزی نیست که در واقعیت اتفاق می افتند و ما باید Smoketrail را قبل از اینکه راکت نابود شود از آن جدا کنیم.

دقت کنید که راکت می تواند با دو راه از بین برود. اول اینکه سه ثانیه از ایجاد شدن آن بگذرد (به هوا شلیک شود)، دوم اینکه با چیزی برخورد کند.

- کد جاوا اسکریپت بالا را به شی Rocket اضافه کنید.
- شی راکت را به متغیر مربوطه در RocketLauncher (در Inspector View) درگ کنید.
- بازی را اجرا کنید. حال اگر راکتی را شلیک کنید، دودی آن را دنبال می کند.

انفجارها

شما باید دقت کرده باشید که وقتی که راکت با شی ای برخورد می کند انفجاری ایجاد نمی شود.

- Prefab ه Small Exlosion را به داخل متغیر Explosion در شی Rocket (در Inspector View) درگ کنید.

حال ما نیاز داریم تا رفتار انفجارمان را تعیین کنیم. فایل جاوا اسکریپت جدیدی ایجاد کرده و نام آن را Explosion-Simple بگذارید. کد زیر را در آن وارد کنید:

```
var explosionRadius = 5.0;
var explosionPower = 10.0;
var explosionDamage = 100.0;

var explosionTime = 1.0;

function Start () {

     var explosionPosition = transform.position;
1    var colliders : Collider[] = Physics.OverlapSphere (explosionPosition,
     explosionRadius);


     for (var hit in colliders) {
          if (!hit)
               continue;

          if (hit.rigidbody)
          {
2              hit.rigidbody.AddExplosionForce(explosionPower, explosionPosition,
                    explosionRadius, 3.0);

               var closestPoint = hit.rigidbody.ClosestPointOnBounds(explosionPosition);
               var distance = Vector3.Distance(closestPoint, explosionPosition);

               // The hit points we apply decrease with distance from the hit point
3              var hitPoints = 1.0 - Mathf.Clamp01(distance / explosionRadius);
               hitPoints *= explosionDamage;

               // Tell the rigidbody or any other script attached to the hit object
               // how much damage is to be applied
4              hit.rigidbody.SendMessageUpwards("ApplyDamage", hitPoints,
                    SendMessageOptions.DontRequireReceiver);

          }
     }

    // stop emitting ?
    if (particleEmitter) {
        particleEmitter.emit = true;
          yield WaitForSeconds(0.5);
          particleEmitter.emit = false;
    }

    // destroy the explosion
     Destroy (gameObject, explosionTime);
}
```

1. **این قسمت از کد آرایه ای از** Collider **هایی که در شعاع یک کره قرار می گیرند را بـر مـی گرداند.**
2. **این قسمت نیرویی به سمت بالا به تمام** Rigidbody **هایی که در شعاع انفجار (شعاع کـره) قرار می گیرند وارد می کند که باعث ایجاد یک انفجار زیبا می شود.**
3. **این قسمت میزان صدمه ی وارد شده به** Rigidbidy **هایی که تحت تاثیر انفجار قـرار گرفتـه اند را محاسبه می کند. میزان صدمه بستگی به فاصله ی آنها از مرکز انفجار دارد.**
4. **این قسمت پیامی برای تعیین میزان صدمه به** Rigidbody **ها می فرستد.**

- اسکریپت بالا را به عنوان کامپوننتی به Prefab ه Smal Explosion اضافه کنید.

اسکریپت انفجار را می توان به عنوان اسکریپت عمومی انفجار برای هر انفجاری استفاده کرد.

برای رسیدن به انفجار های دلخواه و متفاوت، میزان متغیر های زیر را تغییر دهید:

1. explosionPower : میزان نیرویی که به اشیاء تحت انفجار وارد می شود.
2. explosionDamage : میزان صدمه ی وارد شده به شی.
3. explosionRadius : شعاعی که انفجار می تواند به اشیاء تاثیر بگذارد.

اسکریپت انفجار خیلی شبیه چیزی که در قسمت قبل آموزش آورده شده است و تنها تفاوت آن در نقاط برخورد (hiPoints) است. متغیر hitPoints محاسبه کننده ی میزان explosionDamage بر اساس فاصله ی اشیاء از مرکز انفجار است. یعنی هر شی ای که دورتر از مرکز انفجار باشد، طبیعتا صدمه ی کمتری می بیند. این به این معنی است که حال شما می توانید به اشیاء صدمه وارد کنید. چگونگی استفاده از hitPoint ها برای اشیاء در آینده با جزئیات بیشتری آورده شده است.

- بازی را اجرا کنید.

5. اسلحه ی تیر بار (MachineGun):

اسلحه ی از نوع تیربار قابلیت شلیک سریعتری را نسبت به موشک انداز دارد و همچنین گلوله های آن کمتر از راکت صدمه وارد می کند.

- شی دیگری ایجاد کرده و نـام آن را MachineGun بگذاریـد. حـال آن را بـه عنـوان فرزنـد شی Weopens در قسمت Hierarchy View کنید.

- مدل Objects/Weopens/machineGun را به شی MachineGun اضافه کنید.

- اسکریپت MachineGun.js را به شی MachineGun اضافه کنید.

- muzzleFlash (در زیرمجموعـه ی machineGun) را بـه متغیـر Muzzle Flash در قسمت Inspector View ه مربوط به MachineGun درگ کنید.

در زیر کد کامل مربوط به MachineGun.js را می بینید:

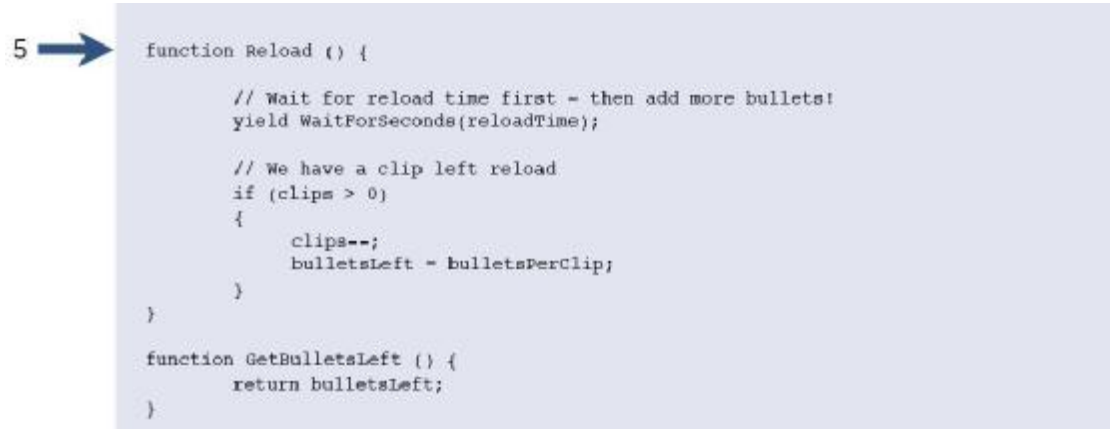
```
var range = 100.0;
var fireRate = 0.05;
var force = 10.0;
var damage = 5.0;
var bulletsPerClip = 40;
var clips = 20;
var reloadTime = 0.5;
private var hitParticles : ParticleEmitter;
var muzzleFlash : Renderer;

private var bulletsLeft : int = 0;
private var nextFireTime = 0.0;
private var m_LastFrameShot = -1;

1 → function Start ()
{
        hitParticles = GetComponentInChildren(ParticleEmitter);

        // We don't want to emit particles all the time, only when we hit something.
        if (hitParticles)
             hitParticles.emit = false;
        bulletsLeft = bulletsPerClip;
}

2 → function LateUpdate()
{
        if (muzzleFlash)
        {
             // We shot this frame, enable the muzzle flash
             if (m_LastFrameShot == Time.frameCount)
             {
                  muzzleFlash.transform.localRotation = Quaternion.AxisAngle
                       (Vector3.forward, Random.value);
                  muzzleFlash.enabled = true;
             }
             // We didn't, disable the muzzle flash
             else
             {
                  muzzleFlash.enabled = false;
                  enabled = false;
             }
        }
}
```

3

```
function Fire ()
{
        if (bulletsLeft == 0)
             return;

        // If there is more than one bullet between the last and this frame
        // Reset the nextFireTime
        if (Time.time - fireRate > nextFireTime)
             nextFireTime = Time.time - Time.deltaTime;

        // Keep firing until we used up the fire time
        while( nextFireTime < Time.time && bulletsLeft != 0)
        {
             FireOneShot();
             nextFireTime += fireRate;
        }
}
```

4

```
function FireOneShot ()
{
        var direction = transform.TransformDirection(Vector3.forward);
        var hit : RaycastHit;

        // Did we hit anything?
        if (Physics.Raycast (transform.position, direction, hit, range))
        {
             // Apply a force to the rigidbody we hit
             if (hit.rigidbody)
                  hit.rigidbody.AddForceAtPosition(force * direction, hit.point);

             // Place the particle system for spawing out of place where we hit the
             // surface!
             // And spawn a couple of particles
             if (hitParticles)
             {
                  hitParticles.transform.position = hit.point;
                  hitParticles.transform.rotation =
                            Quaternion.FromToRotation (Vector3.up,
                            hit.normal);
                  hitParticles.Emit();
             }

             // Send a damage message to the hit object
             hit.collider.SendMessageUpwards("ApplyDamage", damage,
                  SendMessageOptions.DontRequireReceiver);

        }

        bulletsLeft--;

        // Register that we shot this frame,
        // so that the LateUpdate function enabled the muzzleflash renderer for one
        // frame
        m_LastFrameShot = Time.frameCount;
        enabled = true;

        // Reload gun in reload Time
        if (bulletsLeft == 0)
             StartCoroutine("Reload");
}
```



```
5 → function Reload () {

        // Wait for reload time first - then add more bullets!
        yield WaitForSeconds(reloadTime);

        // We have a clip left reload
        if (clips > 0)
        {
             clips--;
             bulletsLeft = bulletsPerClip;
        }
}

function GetBulletsLeft () {
        return bulletsLeft;
}
```

1. تابع ()Start فقط Particle Emitter (bulletSpark) را مقدار دهی اولیه می کند که در اینجا آن را خاموش می کند.

2. تابع ()LateUpdate به صورت اتوماتیک بعد از تابع Update فراخوانی می شود. دقت کنید که تابع Update در فایل اسکریپت PlayerWeopens فراخوانی می شود که قبلا به شی Weopens اضافه کرده ایم. (شی والد MachineGun).

عموما تابع LateUpdate زمانی استفاده می شود که بخواهیم برای عمل انجام شده در تابع Update واکنش نشان دهیم. در این مورد بازیکن تیر را در تابع Update شلیک می کند و در تابع LateUpdate فلش انفجار نمایش داده می شود.

3. تابع Fire محاسبه می کند که بازیکن باید بر اساس میزان سرعت شلیک اسلحه (Fire rate) با اسلحه شلیک کند.

4. تابع FireOneShot با انداختن اشعه ای (Ray) در مرکز دید اسلحه، محاسبه می کند تا مشخص شود که تیر با چیزی برخورد کرده است یا خیر. ما فاصله ی برد تیر را به مقدار مشخصی محدود کرده ایم.

اگر اشعه (Ray) با یک شی Rigidbody برخورد کرد، نیرویی در جهت جلو به آن وارد می کنیم. (نیروی کمی که مشخص باید از تیر وارد می شود).

سپس Particle ه BulletSpark در محل برخورد ایجاد می شود. مرکز ایجاد Particle بر اساس نرمال سطح برخورد شده با آن است.

سپس پیام میزان صدمه (Damage) وارد شده به شی با استفاده از فرستادن پیام Damage

به آن شی تعیین می شود.

5. تابع Reload خشاب گلوله ها را تعویض مـی کنـد (البتـه اگـر در دسـترس باشـد!). میـزان طول کشیدن تعویض خشاب را می توان در Inspector View مشخص کرد.

تنظیم کردن Particle Emitter

اسلحه ی تیر بار نیاز به یک افکت جرقه هنگام برخـورد گلولـه بـا یـک شـی Rigidbody دارد. پس بیایید یکی از آن را بسازیم. دو راه برای ساختن افکت جرقـه وجـود دارد. یـک راه سـاده و یک راه سخت تر که نیاز به تنظیمات بیشتری دارد. بیایید به هردو نگاهی بیاندازیم:

راه ساده:

- Prefab ه Sparks را از Standar Assets/Particles درگ کـرده و در Hierarchy View فرزند شی **m**achineGun کنید (نه **M**achineGun).

- خوبه بازی را اجرا و تست کنید.

راه سخت تر:

- یـک Particle System جدیـد بـسازید (Gameobject/Create Other/Particle System) و نام آن را BulletSpark بگذارید. آن را به عنـوان فرزنـد شـی **m**achineGun قرار دهید (درگ کنیـد). سیـستم ذرات پـیش فـرض زیـاد مناسـب نیـست. حـال مـا چگـونگی ویرایش پارامتر های Particle System خود را برای ساخت جرقه نشان می دهیم:

- BulletSpark را انتخـاب کـرده و مطمـئن شـوید کـه مقـدار Emit در Inspector View انتخاب شده باشد (برای دیدن نتیجه ی کار به صورت RealTime).

- در قـسمت Ellipsoid Particle Emitter، میـزان ellipsoid بـرای محـور هـای xyz را X=0 Y=1 Z=0 قرار دهید.

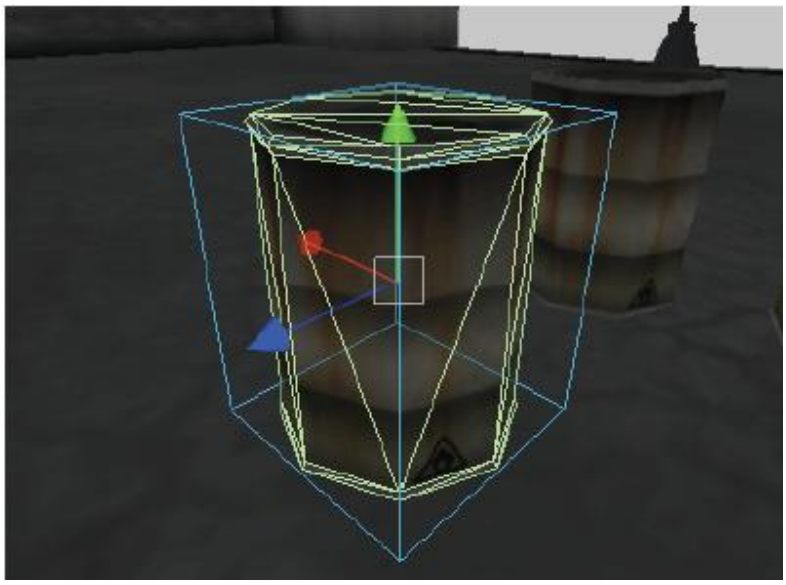
- در قسمت Local Velocity مقدار XYZ را X=0 Y=1 Z=0 قرار دهید.

- مدت زمان عمر ذرات خیلی زیاد است. مـا بـا کـم کـردن مقـدار پـارامتر Max Energy مـی توانیم آن را کمتر کنیم. (1,0 یا کمتر باید خوب باشد).

- بازی را اجرا کنید و مطمئن شوید که جرقه ها خوب شده اند. فراموش نکنید با کلید های 1 و 2 می توان بین اسلحه ها سوییچ کرد.

5. نقاط برخورد (hitPoints):

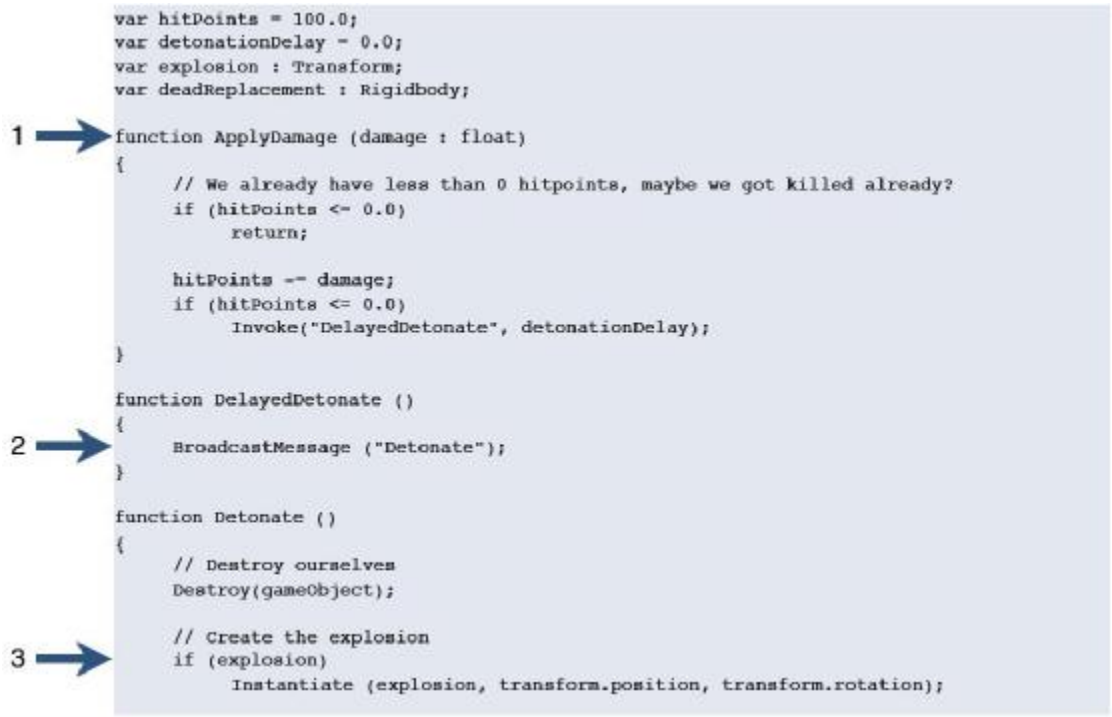
در توضیحات اسکریپت های Explosion و MachineGun توضیح دادیم که چگونه میزان صدمات حاصل از تیر و راکت را به اشیاء مربوطه بفرستیم ولی هنوز به اشیاء نگفته ایم که چگونه آن را مدیریت کنند.



اشیاء می توانند با استفاده از متغیر hitPoints میزان سلامتی خود را بدانند. هر شی ای به صورت اولیه میزان سلامتی مربوط به خود را دارد (بر اساس مقاومتش). هر شی که بخواهد در مورد صدمه دیدن خود تصمیم گیری کند باید دارای تابع ApplyDamage باشد(دقت کنید که این تابع از اسکریپت های Explosion و MachineGun برای وارد کردن صدمه فراخوانی می شوند). این تابع میزان سلامتی شی را کم می کند و تصمیم می گیرد که وقتی که میزان سلامتی به 0 رسید چه تابعی فراخوانی شود (معمولا تابع مردن یا انفجار).
قسمت بعدی نشان می دهد که چگونه از hitPoint و ApplyDamage استفاده کرد.

بشکه های انفجاری

کد جاوااسکریت نوشته شده عمومی است و می توانید به هر شی که می خواهید آن را اضافه کنید. کد کامل DamageReseiver.js به شرح زیر است:



```
var hitPoints = 100.0;
var detonationDelay = 0.0;
var explosion : Transform;
var deadReplacement : Rigidbody;

1 → function ApplyDamage (damage : float)
{
    // We already have less than 0 hitpoints, maybe we got killed already?
    if (hitPoints <= 0.0)
        return;

    hitPoints -= damage;
    if (hitPoints <= 0.0)
        Invoke("DelayedDetonate", detonationDelay);
}

function DelayedDetonate ()
{
2 →  BroadcastMessage ("Detonate");
}

function Detonate ()
{
    // Destroy ourselves
    Destroy(gameObject);

    // Create the explosion
3 →  if (explosion)
        Instantiate (explosion, transform.position, transform.rotation);
```

1. این تابع میزان صدمه ای را به شی ای که تحت انفجار یا تیر قرار می گیرد وارد می کند. اگر از قبل میران hitPoints 0 باشد، پس هیچ کاری صورت نمی گیرد. در غیر اینصورت، میزان صدمه از سلامتی (hitPoints) کم می شود. اگر بعد از کم کردن میزان صدمه از سلامتی میزان سلامتی کمتر از 0 شد، حال تابع DelayedDetonate فراخوانی می شود (فقط برای بهتر کردن انفجار است و دلیل دیگری ندارد).
2. این قسمت متد Detonate را روی شی و فرندان آن فراخوانی می کند.
3. اگر Prefab ه انفجار به بشکه (barrel) اضافه شده باشد، آن را زمانی که سلامتی بشکه به صفر رسید نشان می دهد.
4. اگر شی دارای جایگزین مرده باشد(در این مورد بشکه ی سوخته)، بشکه ی سوخته را جایگزین بشکه ی اولیه می کند. ما نیروهایی که به بشکه ی اولیه وارد شده است را به بشکه ی سوخته نیز وارد می کنیم.

حال بیایید در بازی مان از اسکریپت DamageReseiver.js برای بشکه استفاده کنیم.
بیایید با اضافه کردن مقداری محتوا(Asset) شروع کنیم:
- بشکه را از Objects/crateAndBarrel/barrel داخل صحنه درگ کنید.
- کامپوننت Rigidbodyرا به بشکه اضافه کنید.
- یک Box Collider به بشکه اضافه کنید. حال باید اندازه ی Box Collider را با اندازه ی بشکه ست کنید. این کار را در قسمت Inspector View انجام دهید.
- اسکریپت DamageReseiver را به بشکه اضافه کنید.
- Prefab ه Explosionرا در متغیر Explosion درگ کنید.
- Prefab ه جدیدی با نام Barrel بسازید.
- شی بشکه(barrel) موجود در Hierarchy View را به Prefab ه Barrel در Project View درگ کنید.
- چندین بشکه(Barrel) به صحنه اضافه کنید(از Dublicate استفاده کنید. Ctrl + D).
- بازی را اجرا کنید.
دقت کردید که زمانی که بشکه منفجر می شود، فقط ناپدید می شود. حال ما یک بشکه ی سوخته اضافه خواهیم کرد.
- Prefab ه دیگری با نام Barrel-dead ایجاد کنید.
- یک بشکه ی معمولی به آن اضافه کنید.
صبر کنید! هر دوی بشکه ها دارای یک تکسچر هستند!(barrel1).
ما می خواهیم تکسچر Barrel-dead با Barrel تفاوت داشته باشد تا بازیکن بفهمد که ترکیده است. طوری که شبیه بشکه ی سوخته باشد. اگر ما تکسچر barrel1 را دستکاری کنیم، تکسچر بشکه ی اول نیز تغییر می کند چون Barrel-dead و Barrel هردو از یک تکسچر مشترک استفاده کرده اند. ما برای حل این مسئله باید تکسچر جدیدی ایجاد کرده و آن را بر روی Barrel-dead قرار دهیم. برای این کار ما یک کپی از تکسچر barrel1 می گیریم و آن را اصلاح می کنیم.
- Barrel-dead را انتخاب کرده و بر روی barrel1 در زیر قسمت Mesh Renderer در Inspector View کلیک کنید. خط زرد رنگی در Project View نشان داده می شود که

نشان می دهد تکسچر barrel1 در کجای Project View قرار دارد.

- از متریـال barrel1 کپـی بگیریـد(در فولـدر Materials) و نـام کپـی را barrel-dead بگذارید.

- حال ما این تکسچر را ویرایش می کنیم تـا شـبیه بـشکه ی سـوخته شـود. مطمـئن شـوید کـه متریـال barrel-dead انتخـاب شـده اسـت و سـپس بـر روی Main Color در Inspector View کلیک کنید. حال تمام اسلایدر های رنگ را به سمت چپ درگ کنید تا رنگ سیاه حاصـل شود و سپس پنجره را ببندید. حال می بینید که تکسچر به رنگ سوخته در آمده است.

- متریال جدید را بر روی Barrel-dead قرار دهید. برای این کار متریال را بـر روی آن درگ کنید.

- بـرای تـست بـشکه هـا هـردو ی Barrel و Barrel-dead را بـه داخـل Scene Vie درگ کنید و مطمئن شوید که با هم تفاوت دارند. حال Barrel-dead را پاک کنید چـون قـرار اسـت وقتی بشکه منفجر شد ایجاد شود.

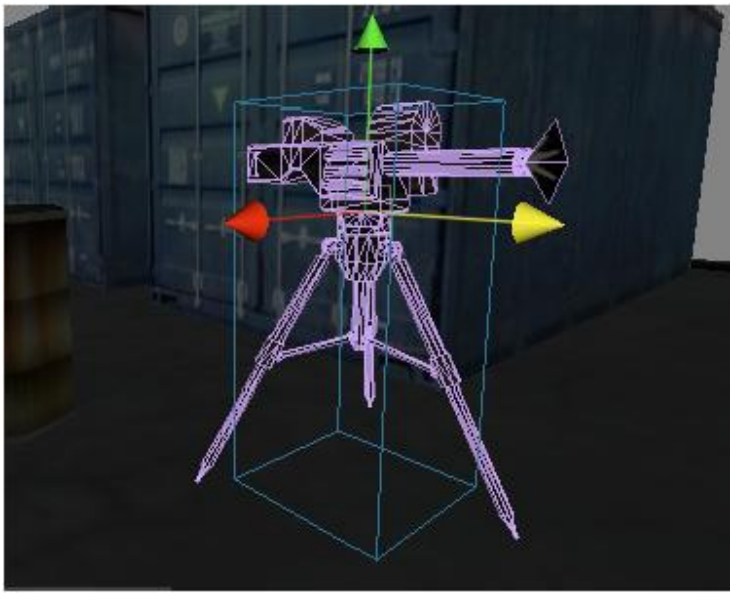
- سـپس کامپوننـت هـا ی Box Collider و Rigidbody را بـه Prefab ه Barrel-dead اضافه کنید (البته شاید به صورت پیش فرض داشته باشد. لطفا چک کنید.).

- Prefab ه Barrel-dead را داخـل متغیـر dead replacment در شـی Barrel درگ کنید.

- بازی را اجرا کنید. حال باید بشکه ها منفجر شده و بسوزند.

7. اسلحه ی نگهبان (Sentry Gun)

در انتها ما یک دشمن به بازی خود اضافه می کنیم که یک اسلحه ی نگهبان است. شی اسلحه ی نگهبان می تواند بازیکن را ببیند و به سمت او شلیک کند.



بیایید با وارد کردن اسلحه ی نگهبان کار را شروع کنیم:

- sentryGun را از داخل پوشه ی Objects/Weopens از قسمت Project View به داخل Scene View درگ کنید.
- کامپوننت های Box Collider و Rigidbody را به آن اضافه کنید.
- اندازه ی Box Collider را تنظیم کنید. باید لاغر و بلند باشد. مقدار Center of Gravity را نیز عدد بالا وارد کنید تا با یک شلیک ساده بیافتد.
- اسکریپت DamageReseiver.js را به SentryGun اضافه کنید.متغیر Explosion آن را نیز مشخص کنید.

حال ما آماده ایم تا کد اسلحه ی نگهبان را بررسی کنیم. کد کامل برای SentryGun.js به شرح زیر است:

```
var attackRange = 20.0;
var target : Transform;

function Start () {
     if (target == null && GameObject.FindWithTag("Player"))
          target = GameObject.FindWithTag("Player").transform;
}

function Update () {
     if (target == null)
          return;
     if (!CanSeeTarget ())
          return;
     // Rotate towards target
     var targetPoint = target.position;
     var targetRotation = Quaternion.LookRotation (targetPoint - transform.position,
          Vector3.up);

     transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation,
          Time.deltaTime * 2.0);


     // If we are almost rotated towards target - fire one clip of ammo
     var forward = transform.TransformDirection(Vector3.forward);
     var targetDir = target.position - transform.position;
     if (Vector3.AngleBetween(forward, targetDir) * Mathf.Rad2Deg < 10.0)
          SendMessage("Fire");
}

function CanSeeTarget () : boolean
{
     if (Vector3.Distance(transform.position, target.position) > attackRange)
          return false;
     var hit : RaycastHit;
     if (Physics.Linecast (transform.position, target.position, hit))
          return hit.transform == target;

     return false;
}
```

1. تابع Start چک می کند تا ببیند متغیر Target مشخص شده است یا خیر. اگر مشخص نشده است، خود آن را مشخص می کند.

2. اگر بازیکن در محدوده قرار دارد و اسلحه ی نگهبان می تواند آن را ببیند، مسلسل اسلحه ی نگهبان به سمت بازیکن می چرخد.

3. اگر درجه ی چرخش بین بازیکن و مکان اسلحه ی نگهبان کمتر از 10 درجه است، اسلحه ی نگهبان شروع به شلیک می کند.

4. تابع CanSeeTrget محاسبه می کند که آیا اسلحه ی نگهبان بازیکن را می بیند یا خیر.

بیایید تنظیمات اسلحه ی نگهبان را تمام کنیم:

- هدف را برای اسلحه ی نگهبان مشخص کنید. برای این کار شی FPS Controller را انتخاب کنید و در قسمت بالا ی Inspector View از قسمت Tag، Player را انتخاب کنید.

- اسکریپت SentryGun را به فرزند sentryGunRotateY ه شی sentryGun اضافه

کنید. این باعث می شود که تنها قسمت بالای sentryGun بچرخد و بدنه ثابت بماند.

- Prefab ه explosion را بـــه متغیـــر Explosion در اســـکریپت مربـــوط بـــه DamageReseiver در کامپوننت ها ی sentryGun اضافه کنید.

- sentryGun را به متغیر dead replcement ه مربوط بـه Damage Reseiver اضـافه کنید. (اگر می خواهید یکی جدیدش رو بسازید!!).

- اسـکریپت MachineGun را بـه شـی SentryGunRotateY در زیـر مجموعـه ی شـی sentryGun اضافه کنید.

- متغیـــر Muzzle Flash موجـــود در اســـکریپت MachineGun را بـــا محتـــوای muzzleFlash در زیر مجموعه ی sentryGunTop پر کنید.

- بـر روی muzzleFlash در قـسمت Hierarchy View کلیـک کـرده و شـیدر آن را در قسمت Inspector View به Particle/Addaptive تغییر دهید.

- بازی را اجرا کنید.حال شما می توانید به بشکه ها و اسلحه ی نگهبان شلیک کنید.

وآخر..

مکعب آسما (SkyBox)

بیایید یک آسمان به محیطمان اضافه کنیم:

- متریال SkyBoxTest.mat را از پوشه ی Assets/Materials وارد کنید.

- فایل های R,L,F,D,B و U.tif را از پوشه ی Assets/SkyBox وارد کنید. راحت ترین کار کپی کردن این فایل ها و Paste کردن آنها در پوشه ی محتویات پروژه است.

- SkyBoxTest را از قسمت Project View انتخاب کرده و تکسچر ها ی F برای Front، B برای Back و... را مشخص کنید.

- از منوی اصلی گزینه ی Edit/Render Setting را انتخاب کنید. SkyBoxTest را بر روی متغیر SkyBox در قسمت Inspector View درگ کنید. حال شما یک آسمان دارید.

سرانجام

حال شما یک بازی اول شخص با قابلیت های زیر دارید:

- چندین اسلحه
- یک اسلحه ی نگهبان با هوش مصنوعی (AI) ساده.
- بشکه هایی که فیزیک دارند و منفجر می شوند.
- اشیایی که می توانند مدل ها ی مختلفی را هنگام نابود شدن داشته باشند. (سیستم تخریب پذیریه پایه ی بیشتر بازی ها).
- کنترل صدمه دیدن (hitPoints)

تلاش کنید تا محتویات دیگری از پوشه ی Assets به پروژه و صحنه ی خود اضافه کنید و Enjoy!!!!

در قسمت بعدی آموزش از سری FPS، در مورد ویژگی ها ی پیشرفته ای مانند Character ragdoll، AI پیشرفته و اضافه کردن HUD به بازی آموزش می بینید.

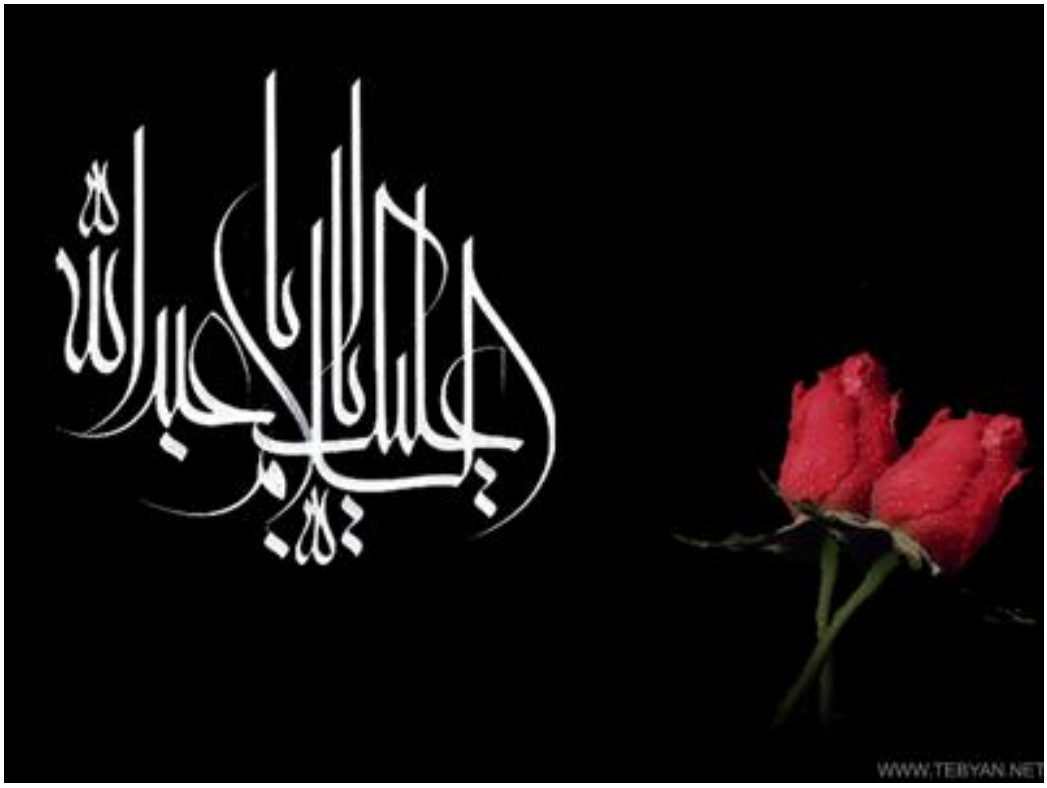
سپاسگزاری:

تشکر فراوان از خودم، خودت، خودش و همه!!!!


fekrejadid.myfablog.ir


www.tebyan.net


یا اباعبدالله الحسین


السلام علیک یا اباعبدالله
WWW.TEBYAN.NET