6CCGD003W

Network Games and Security

Tutorial 02

Starting our Tanks Game

Paul Rosson

UNIVERSITY OF WESTMINSTER

Tutorial Objectives

1. To create some of the basic moving and shooting mechanics for our tank.

2. To design a level for our tanks game.

3. To create a pooling system to handle projectiles, particles and sound effects.

Quick Note

- As we go through this module it is important to understand what each piece of code is doing. You can use the Unity documentation or ask me.

- Retyping the code will aid understanding, try to fully understand each section before moving on.

- Anything after '//' is a comment and does not need to be copied. You may want to write your own comments to aid your understanding.

Tutorial Package

1. Create a new Unity project (3D).

2. Download ‘Tutorial 02 Package’ from Blackboard and import this package into your Unity project.

3. Create a new scene, and save it as ‘Game’.

4. Delete the ‘MainCamera’ from the scene (leaving only Directional Light) and add one each of ‘MainCamera’ and ‘Player’ prefabs from Prefabs folder.

Camera Follow

1. It's worth noting the structure of the 'Player' object right now, including how the mesh is divided into different subsections and also the physics components.

2. Before we start working on our player class we want to ensure it can always be seen. We can add some code to our camera to ensure it always follows the player.

3. Add the following functions to 'FollowTarget.cs' in the Scripts folder (below the variables):

```
//initialize variables
void Start ()
{
    cam = GetComponent<Camera> ();
    camTransform = transform;

    //the AudioListener for this scene is not attached directly to this camera,
    //but to a separate gameobject parented to the camera. This is because the
    //camera is usually positioned above the player, however the AudioListener
    //should consider audio clips from the position of the player in 3D space.
    //so here we position the AudioListener child object at the target position.
    Transform listener = GetComponentInChildren<AudioListener> ().transform;
    listener.position = transform.position + transform.forward * distance;
}


//position the camera in every frame
void LateUpdate ()
{
    //cancel if we don't have a target
    if (!target)
        return;

    //convert the camera's transform angle into a rotation
    Quaternion currentRotation = Quaternion.Euler (0, transform.eulerAngles.y, 0);

    //set the position of the camera on the x-z plane to:
    //distance units behind the target, height units above the target
    Vector3 pos = target.position;
    pos -= currentRotation * Vector3.forward * Mathf.Abs (distance);
    pos.y = target.position.y + Mathf.Abs (height);
    transform.position = pos;

    //look at the target
    transform.LookAt (target);

    //clamp distance
    transform.position = target.position - (transform.forward * Mathf.Abs (distance));
}
```

Camera Follow

1. And also add the following two lines to the end of the ‘Awake’ function in ‘Player.cs’:

```
camFollow = Camera.main.GetComponent<FollowTarget> ();
camFollow.target = turret;
```

2. Now if we press play we should notice that the camera snaps to our tank. Once we’re moving it should follow us around.

3. Don’t forget to press play again to stop the game before making further changes. Changes made in play mode will not persist!

Player Movement

1. Now we want our tank to be able to move around using the keyboard.

2. Add the following functions to ‘Player.cs’ (below ‘Awake’) which will detect and implement movement using the keyboard:

```
void FixedUpdate ()
{

    //check for frozen Y position, regardless of other position constraints
    if ((rb.constraints & RigidbodyConstraints.FreezePositionY) != RigidbodyConstraints.FreezePositionY)
    {
        //Y position is not locked and the player is above normal height, apply additional gravity
        if (transform.position.y > 0)
            rb.AddForce (Physics.gravity * 2f, ForceMode.Acceleration);
    }

    //movement variables
    Vector2 moveDir;
    Vector2 turnDir;

    //reset moving input when no arrow keys are pressed down
    if (Input.GetAxisRaw ("Horizontal") == 0 && Input.GetAxisRaw ("Vertical") == 0)
    {
        moveDir.x = 0;
        moveDir.y = 0;
    } else
    {
        //read out moving directions and calculate force
        moveDir.x = Input.GetAxis ("Horizontal");
        moveDir.y = Input.GetAxis ("Vertical");
        Move (moveDir);
    }
}

//moves rigidbody in the direction passed in
void Move (Vector2 direction = default(Vector2))
{
    //if direction is not zero, rotate player in the moving direction relative to camera
    if (direction != Vector2.zero)
        transform.rotation = Quaternion.LookRotation (new Vector3 (direction.x, 0, direction.y))
        * Quaternion.Euler (0, camFollow.camTransform.eulerAngles.y, 0);

    //create movement vector based on current rotation and speed
    Vector3 movementDir = transform.forward * moveSpeed * Time.deltaTime;
    //apply vector to rigidbody position
    rb.MovePosition (rb.position + movementDir);
}
```

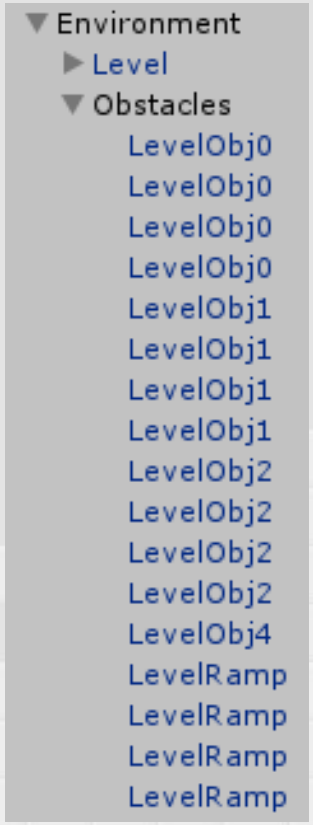
Building an Environment

1. If we press play now we should be able to move our tank around in the x-z plane. However, since there is no environment the y position of the tank is continuously dropping.

2. Create an empty GameObject (by right clicking in the hierarchy) and call it ‘Environment’, then drag a ‘Level’ prefab onto this from ‘Prefabs -> Level’. If the player is positioned correctly they should be able to drive around this empty terrain.

Adding Obstacles

1. If you look in the ‘Prefabs -> Level’ folder you’ll notice six obstacle prefabs to work with. Use as many as you like to create your own level, but try to keep the blank squares empty as they will be our spawn areas in a future tutorial.



2. Make sure these obstacles are organised appropriately in the hierarchy, and don’t forget to test your level regularly!

Adding Aiming

1. Now we're going to add an aiming mechanic, based on the current position of the mouse. Add the following lines to the bottom of 'FixedUpdate' in 'Player.cs':

```
//cast a ray on a plane at the mouse position for detecting where to shoot
Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
Plane plane = new Plane (Vector3.up, Vector3.up);
float distance = 0f;
Vector3 hitPos = Vector3.zero;
//the hit position determines the mouse position in the scene
if (plane.Raycast (ray, out distance))
{
    hitPos = ray.GetPoint (distance) - transform.position;
}

//we've converted the mouse position to a direction
turnDir = new Vector2 (hitPos.x, hitPos.z);

//rotate turret to look at the mouse direction
RotateTurret (new Vector2 (hitPos.x, hitPos.z));
```

Adding Aiming

1. And also add the following function to the bottom of ‘Player.cs’ after ‘Move’:

```
//rotates turret to the direction passed in
void RotateTurret (Vector2 direction = default(Vector2))
{
    //don't rotate without values
    if (direction == Vector2.zero)
        return;

    //get rotation value as angle out of the direction we received
    int newRotation = (int)(Quaternion.LookRotation (new Vector3 (direction.x, 0, direction.y)).eulerAngles.y
                                                                    + camFollow.camTransform.eulerAngles.y);

    turretRotation = newRotation;

    turret.rotation = Quaternion.Euler (0, newRotation, 0);
}
```

2. Now hit play and you should see the turret following the mouse position.

Default Cursor

1. One extra thing we can do is change the default cursor to a crosshair by going to ‘Edit -> Project Settings -> Player’ and setting ‘Default Cursor’ to ‘CrosshairMouse’. We also want the ‘Cursor Hotspot’ to be in the centre of our crosshair.

Pooling and Audio

1. Before we can fire bullets we need to add some utility classes to our scene. Create an empty object called ‘Scripts’ which we’re going to use to hold our manager classes.

2. Create two more empty objects as children of ‘Scripts’ and call them ‘PoolManager’ and ‘AudioManager’. Add a ‘PoolManager’ component to ‘PoolManager’ and an ‘AudioManager’ component to ‘AudioManager’. Under ‘One Shot Prefab’ on ‘AudioManager’ add the ‘OneShotAudio’ prefab.

Pooling and Audio

1. Both of these classes are extremely useful in most Unity projects and it is worth fully understanding how they work. Object pooling maintains a pool of objects which would otherwise be created and destroyed regularly, which impacts performance.

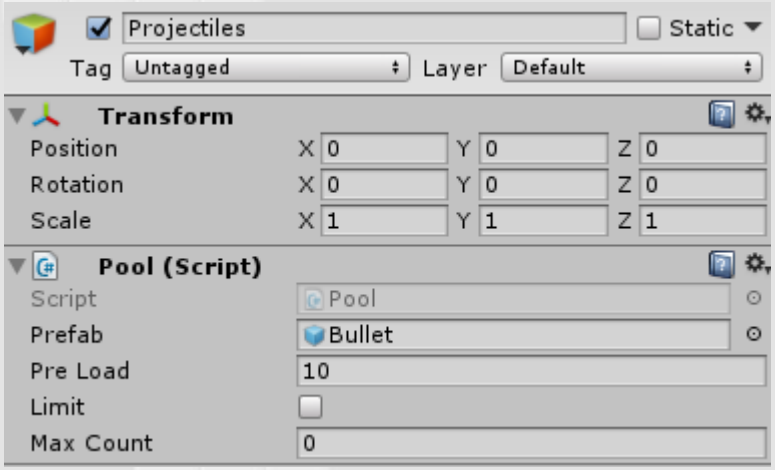
2. There are three types of objects we are going to pool (for now), ‘Projectiles’, ‘Particles’, and ‘Audio’ so create an empty child object of ‘PoolManager’ for each.

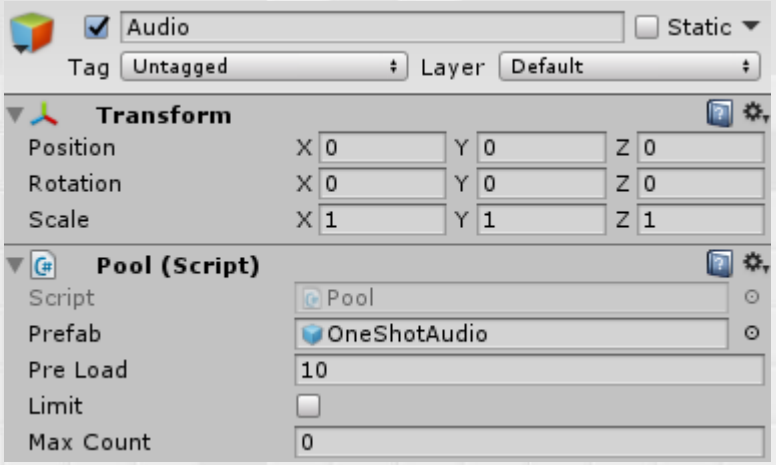
Pooling and Audio

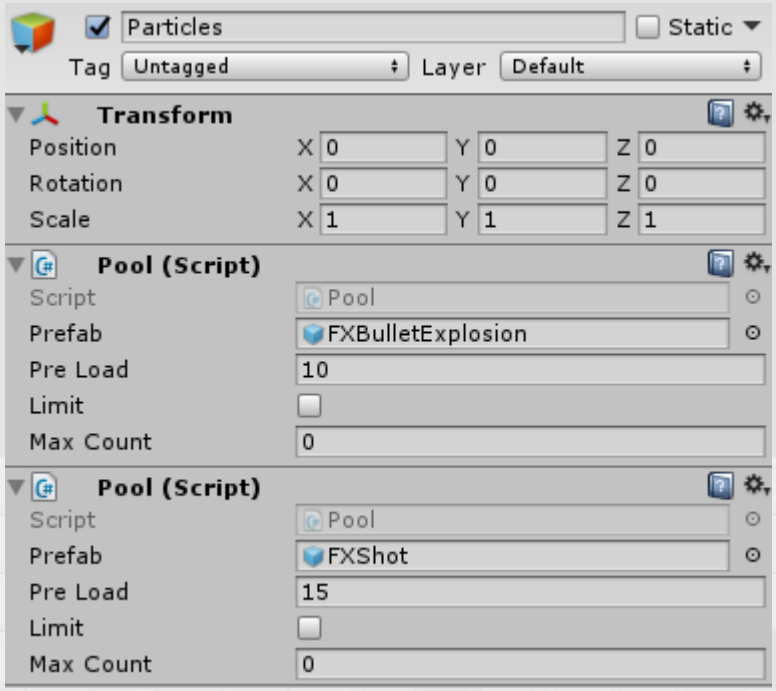
1. The scripts object should look like this in the hierarchy now:

2. Now we're going to add our pools. Add 'Pool' components to the appropriate objects and set the correct prefab for each pool. They should look like the following:

Pools


Projectiles
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Pool (Script)
Script Pool
Prefab Bullet
Pre Load 10
Limit
Max Count 0


Audio
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Pool (Script)
Script Pool
Prefab OneShotAudio
Pre Load 10
Limit
Max Count 0


Particles
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Pool (Script)
Script Pool
Prefab FXBulletExplosion
Pre Load 10
Limit
Max Count 0
Pool (Script)
Script Pool
Prefab FXShot
Pre Load 15
Limit
Max Count 0

Shooting

1. Now that our pooling is set up correctly we should be able to add shooting mechanics to our tank. Add the following lines to the bottom of ‘FixedUpdate’ in ‘Player.cs’:

```
//shoot bullet on left mouse click
if (Input.GetButton ("Fire1"))
    Shoot ();
```

Shooting

1. Add the following function to ‘Player.cs’ below ‘RotateTurret’:

```
//shoots a bullet in the direction passed in
void Shoot (Vector2 direction = default(Vector2))
{
    //if shot delay is over
    if (Time.time > nextFire)
    {
        //set next shot timestamp
        nextFire = Time.time + fireRate;

        //spawn bullet using pooling, locally
        GameObject obj = PoolManager.Spawn (bullet, shotPos.position, turret.rotation);
        Bullet blt = obj.GetComponent<Bullet> ();

        if (shotFX)
            PoolManager.Spawn (shotFX, shotPos.position, Quaternion.identity);
        if (shotClip)
            AudioManager.Play3D (shotClip, shotPos.position, 0.1f);
    }
}
```

Bullets

1. Finally add the following functions to ‘Bullet.cs’, below ‘Awake’:

```
//set initial travelling velocity
void OnSpawn ()
{
    myRigidbody.velocity = speed * transform.forward;
}

//check what was hit on collisions
void OnTriggerEnter (Collider col)
{
    //despawn gameobject
    PoolManager.Despawn (gameObject);
}


//set despawn effects and reset variables
void OnDespawn ()
{
    //create clips and particles on despawn
    if (explosionFX)
        PoolManager.Spawn (explosionFX, transform.position, transform.rotation);
    if (explosionClip)
        AudioManager.Play3D (explosionClip, transform.position);

    //reset modified variables to the initial state
    myRigidbody.velocity = Vector3.zero;
    myRigidbody.angularVelocity = Vector3.zero;
}
```

Mission Complete

1. Now press play and the tank should be able to move around your environment, rotate and fire bullets.

2. Feel free to make your environment more interesting or even create multiple levels to use in your final games.

3. And make sure you understand the code as thoroughly as you can! The better your understanding the easier you will find it to extend and improve.