Unity Game Engine

Introduction to Unity – Adding enemies and projectile (2)

Unity Manual: http://docs.unity3d.com/Manual/index.html
Unity Script References: http://docs.unity3d.com/ScriptReference/index.html

Detail steps to take

1. Enable the player to shoot into the scene.
2. Create static targets that react to being hit.
3. Make the targets wander around.
4. Spawn the wandering targets automatically.
5. Enable the targets/enemies to shoot fireballs at the player.

Detail steps to take

1. Enable the player to shoot into the scene.
2. Create static targets that react to being hit. (review)
3. Make the targets wander around.
4. Spawn the wandering targets automatically.
5. Enable the targets/enemies to shoot fireballs at the player.

Create a Target Enemy (Review)

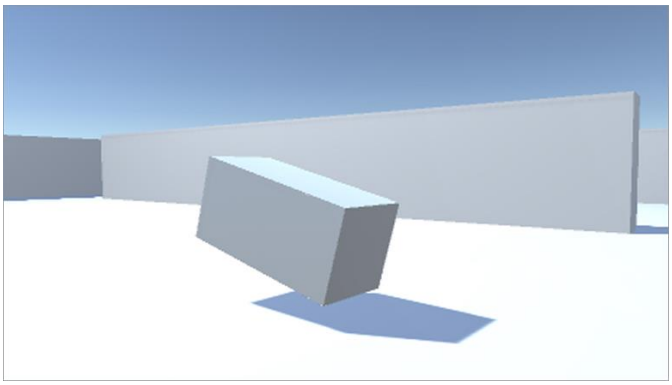
- Create an enemy
  - Create a new cube object (Navigation: GameObject > 3D Object > Cube)
  - Scale as you want.
  - Name the object Enemy.
- Create a new script called **ReactiveTarget.cs** and attach it to the Enemy.

ReactiveTarget.cs Script (Review)

```
using UnityEngine;
using System.Collections;
public class ReactiveTarget : MonoBehaviour {
   public void ReactToHit() {
     StartCoroutine(Die());
   }
   private IEnumerator Die() {
     this.transform.Rotate(-75, 0, 0);
     yield return new WaitForSeconds(1.5f);
     Destroy(this.gameObject);
   }
}
```

The target object will fall over and disappear when you shoot it.

ReactiveTarget.cs Script (Review)



The target object will fall over and disappear when you shoot it.

Modify Rayshooter.cs Script (Review)

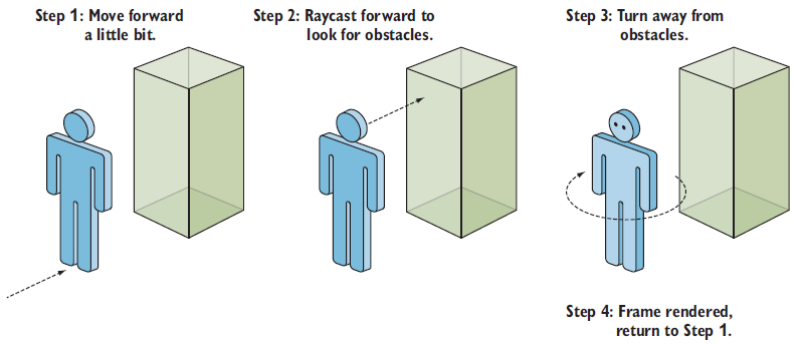
```
…
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit)) {
         GameObject hitObject = hit.transform.gameObject;
         ReactiveTarget target =
             hitObject.GetComponent<ReactiveTarget>();
         if (target != null) {
             //Debug.Log("Target hit");
             target.ReactToHit();
         } else {
           StartCoroutine(SphereIndicator(hit.point));
         }
…
```


Detail steps to take

1. Enable the player to shoot into the scene.
2. Create static targets that react to being hit.
3. Make the targets wander around.
4. Spawn the wandering targets automatically.
5. Enable the targets/enemies to shoot fireballs at the player.

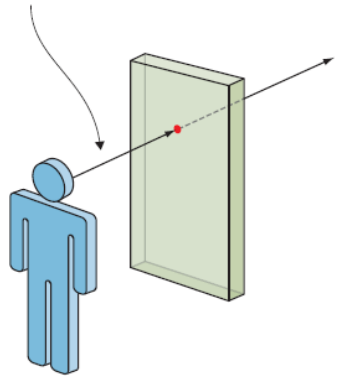
Moving Enemy (Target)

- Simple AI – the enemy will ping-pong around the room, always moving forward and turning to avoid walls.



"Seeing" obstacles with a raycast

For every frame the AI character projects a ray in front of it in order to detect obstacles. Here the character is facing a wall, so the raycast will detect a close obstacle.



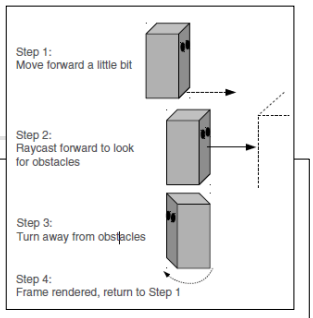
“Shooting” vs “Seeing”

- The ray is very thin and it is suitable for shooting. But for AI’s line of sight, we need a large cross-section. The reason for this difference is that bullets are tiny, whereas to check for obstacles in front of the character, we need to account for the view angle of the character.
- In terms of the code, we use `SphereCast()` method instead of `Raycast()`.

WanderingAI.cs

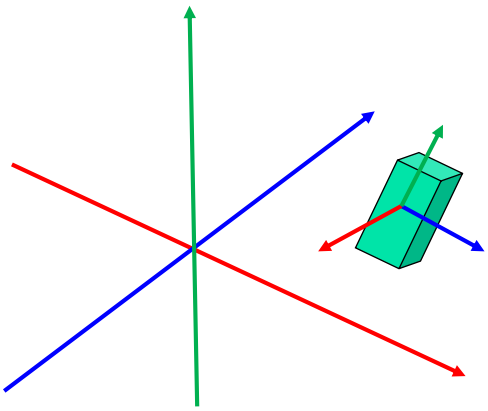
- Create a new script named **WanderingAI.cs**
- Attach that to the Enemy object (alongside the **ReactiveTarget.cs** script),
  - Create a ray that originates at the enemy.
  - Use the method `SphereCast()` instead of `Raycast()`
    - SphereCast() casts a sphere along a ray direction and returns true when the sphere sweep intersects any objects (collider), otherwise false. When it is true, it returns detailed information on what was hit.
    - Think of the sphere cast like a thick raycast.

WanderingAI.cs




```
using UnityEngine;
using System.Collections;
public class WanderingAI : MonoBehaviour {
  public float speed = 3.0f;
  public float obstacleRange = 5.0f;
  void Update() {
    //step 1
    transform.Translate(0, 0, speed * Time.deltaTime);
    //step 2
    Ray ray = new Ray(transform.position, transform.forward);
    RaycastHit hit;
    if (Physics.SphereCast(ray, 0.75f, out hit)) {
      if (hit.distance < obstacleRange) {
        // step 3
        float angle = Random.Range(-110, 110);
        transform.Rotate(0, angle, 0);
      }
    }
  }
}
```

Global and Local Axes



Local axes are aligned with the object's direction.

In Unity, z-axis is for the forward direction.

`RaycastHit` Data Structure

- Variables:

barycentricCoordinate	The barycentric coordinate of the triangle we hit.
collider	The Collider that was hit.
distance	The distance from the ray's origin to the impact point.
lightmapCoord	The uv lightmap coordinate at the impact point.
normal	The normal of the surface the ray hit.
point	The impact point in world space where the ray hit the collider.
rigidbody	The Rigidbody of the collider that was hit. If the collider is not attached to a rigidbody then it is null.
textureCoord	The uv texture coordinate at the collision location.
textureCoord2	The secondary uv texture coordinate at the impact point.
transform	The Transform of the rigidbody or collider that was hit.
triangleIndex	The index of the triangle that was hit.

WanderingAI.cs Explained

- Used `Translate()` method in order to move forward continuously.
- The ray is created using **the enemy's position** and **its forward direction**.
- The raycasting calculation was done using the method `Physics.SphereCast()` with a sphere radius parameter.
- `Random.Range(float min, float max)` returns a random value between constraints.

Add Character State

- The enemy keeps moving forward after falling over from being hit.
  - It is necessary to keep track of the current state of the object.
- Add alive state:

```
private bool _alive;
void Start() {
   _alive = true;
}
```


Modify WanderingAI.cs

```
// WanderingAI
..
private bool _alive;
..
void Start() {
  _alive = true;
}
..
void Update() {
  if (_alive) {
    transform.Translate(0, 0, speed * Time.deltaTime);
    ...
  }
}
public void SetAlive(bool alive) {
  _alive = alive;
}
...
```

Modify ReactiveTarget.cs

```
//ReactiveTarget.cs
...
public void ReactToHit() {
  ...
  // Check if this character has a WanderingAI script
  WanderingAI behavior = GetComponent<WanderingAI>();
  if (behavior != null) {
    behavior.SetAlive(false);
  }
  StartCoroutine(Die());
}
......
```


Resources

- RayShooter.cs
- ReactiveTarget.cs
- WanderingAI.cs