Unity Game Engine

Introduction to Unity – Adding enemies and projectile (3)

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.
3. Make the targets wander around.
4. Spawn the wandering targets automatically.
5. Enable the targets/enemies to shoot fireballs at the player.

Spawning Enemies

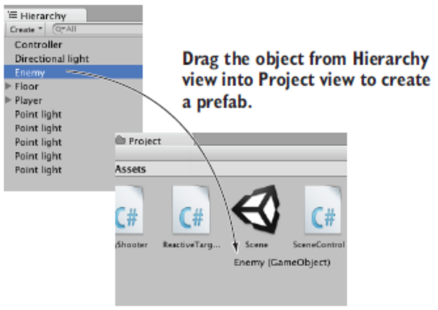
- At the moment if there's just one enemy in the scene, and when it dies, the scene is empty.
- Let's make the game spawn enemies in the scene so that whenever the enemy dies, a new one appears using *prefabs*.
  - When you have an object like an enemy that is reused in the scene several times, you want to create an instance of a particular object to have the same properties.

Prefab

- Prefab exists as an asset. It doesn't exist in any specific scene.
- The prefab acts as a template from which you can create new object instances in the scene.
- Once you generate a GameObject completed with components and properties, you can store it into a Prefab.
- Any edits made to a prefab asset are immediately reflected in all instances produced from it but you can also *override* components and settings for each instance individually.

Create a Prefab

- Steps to create a prefab:
  1. First create an object (e.g. Enemy) in the scene.
  2. Drag the object down from the Hierarchy view and drop it in the Project view; this will automatically save the object as a prefab.



Instantiate a Prefab

- Dragging the prefab asset from the project view to the scene view will create instances of the prefab.
- Objects created as prefab instances will be shown in the hierarchy view in blue text. Normal objects are shown in black text.

Spawning Enemies - Scene Control

- Spawning enemies should be part of the scene control.
- In order to control the scene, create an empty game object named Controller (Navigation: GameObject > Create Empty). The object won’t be visible in the scene.
- Create a script called `SceneController.cs`.
- Attach this script to the Controller object.

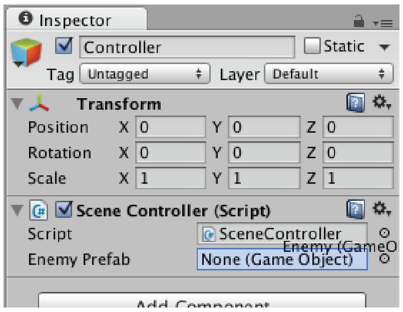
SceneController.cs Script

```
using UnityEngine;
using System.Collections;
public class SceneController : MonoBehaviour {
    [SerializeField] private GameObject enemyPrefab;
    private GameObject _enemy;
    void Update() {
    if (_enemy == null) {
      _enemy = Instantiate(enemyPrefab) as GameObject;
      _enemy.transform.position = new Vector3(0, 1, 0);
      float angle = Random.Range(0, 360);
      _enemy.transform.Rotate(0, angle, 0);
    }
  }
}
```

Instantiate a Prefab in the code.

SceneController.cs Explained

- `private` variables with `[SerializeField]` can be edited in Unity's editor but not by other scripts.
- In the Inspector you'll see a variable (`enemyPrefab`) slot for the enemy prefab.

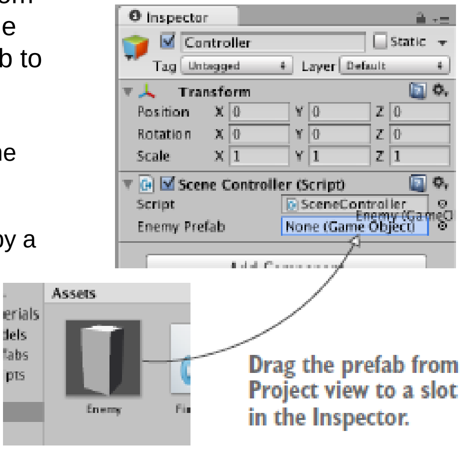

Inspector
Controller
Static
Tag Untagged
Layer Default
Transform

Position	X	0	Y	0	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Scene Controller (Script)
Script SceneController
Enemy Prefab None (Game Object)
Enemy (GameO
Add Component

SceneController.cs Explained

- Drag up the prefab asset from Project to the empty variable slot to link the enemy prefab to the SceneController script.
- Play the scene
  - An enemy will appear in the middle of the room
  - But now if you shoot the enemy it will be replaced by a new enemy.


Inspector
Assets
Enemy
Drag the prefab from Project view to a slot in the Inspector.

Controller | 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

Scene Controller (Script)
Script SceneController
Enemy Prefab None (Game Object)

SceneController.cs Explained

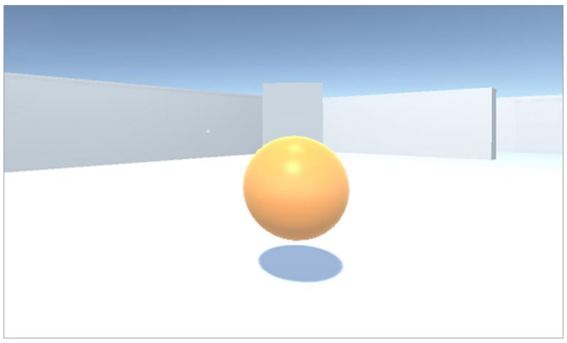
- `Instantiate()` is called only when `_enemy` is empty (or null.)
- `Instantiate()` method instantiate the prefab, that creates a copy in the scene.
- By default, `Instantiate()` returns the new object as a generic `Object` type, but we need to handle it as a `GameObject`.
  - In C#, use the `as` keyword for typecasting to convert from one type of code object into another type (written with the syntax original-object as new-type).

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.

Enemy who Shoots

- Let's make the enemy shoot a fireball that will involve a projectile.



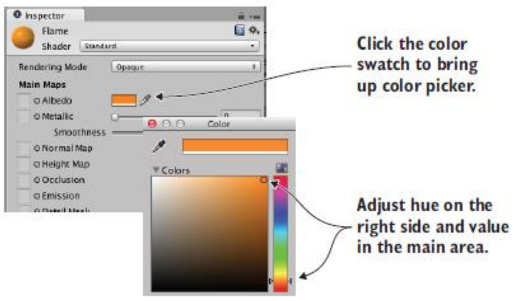
Fireball Object

- Create a fireball object in a bright orange color.
  - Navigation: GameObject > 3D Object > Sphere.

- Name it Fireball.

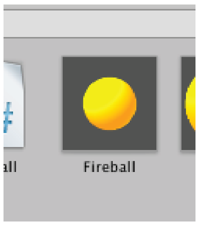
Fireball Object and Material

- Right click on the Fireball object and choose Create a new material. Name the new material Flame.
  - Navigation: Assets > Create > Material.
  - Select the material in the Project view in order to see the material's properties in the Inspector. Click the color swatch labeled Albedo and Emission Value.


Inspector
Flame
Shader: Standard
Rendering Mode: Opaque
Main Maps
Albedo
Metallic
Smoothness
Normal Map
Height Map
Occlusion
Emission
Color
Colors
Click the color swatch to bring up color picker.
Adjust hue on the right side and value in the main area.

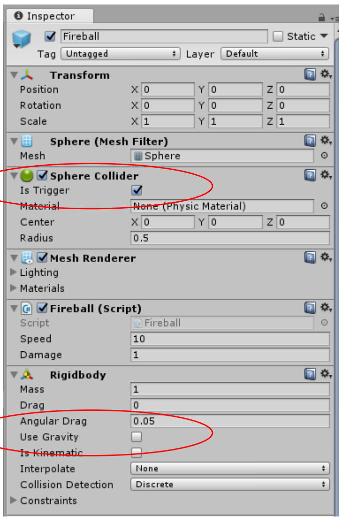
Fireball Prefab

- Turn the fireball object into a prefab by dragging the object down from Hierarchy into Project.



Fireball Collider as Trigger

- Make the fireball as a trigger object that will sends a message to Unity when a collision occurs.
  - To do so, click the "`Is Trigger`" check box in the Sphere Collider component.
  - In the Inspector, click Add Component > Physics > Rigidbody.
    - De-select "`Use Gravity`" so that the fireball won't be pulled down due to gravity.



```
Inspector
[✓] Fireball                                  [ ] 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
Sphere (Mesh Filter)
  Mesh                Sphere
[✓] Sphere Collider
  Is Trigger          [✓]
  Material            None (Physic Material)
  Center              X 0      Y 0      Z 0
  Radius              0.5
[✓] Mesh Renderer
  Lighting
  Materials
[✓] Fireball (Script)
  Script              Fireball
  Speed               10
  Damage              1
Rigidbody
  Mass                1
  Drag                0
  Angular Drag        0.05
  Use Gravity         [ ]
  Is Kinematic        [ ]
  Interpolate         None
  Collision Detection Discrete
  Constraints
```

Fireball.cs Script

- Create a new script named Fireball.cs
- Attach it to Fireball object.

Fireball.cs Script

```
using UnityEngine;
using System.Collections;
public class Fireball : MonoBehaviour {
  public float speed = 10.0f;
  public int damage = 1;
  void Update() {
    transform.Translate(0, 0, speed * Time.deltaTime);
  }
  void OnTriggerEnter(Collider other) {
    PlayerCharacter player =
      other.GetComponent<PlayerCharacter>();
    if (player != null) {
      Debug.Log("Player hit");
    }
    Destroy(this.gameObject); // Do Not Use Destory(this);
  }
}
```

Fireball.cs Explained

- `OnTriggerEnter(other)` method is called automatically when the object has a collision, such as colliding with the walls or with the player.
  - **It is executed before `Update()`.**
  - `other –` the other Collider involved in this collision.
  - `OnTriggerEnter` is not technically part of Collision. It is a MonoBehaviour function.

Destroy and Memory Management

- When an object destroys itself, existing references become null.
- In a memory-managed programming language like C#, normally you aren't able to directly destroy objects; you can only dereference them so that they can be destroyed automatically.
- This is still true within Unity, but the way GameObjects are handled behind the scenes makes it look like they were destroyed directly.

Enemy Shooting Fireballs

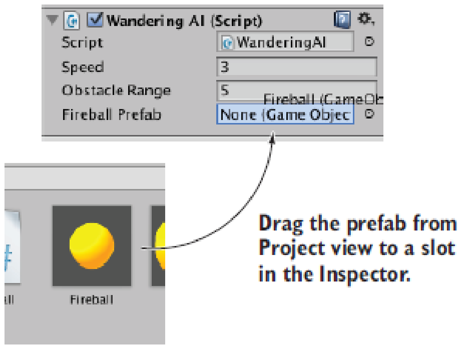
- Step 1 – Modify WanderingAI.cs script as shown in the next slide.
- Step 2 -
  - Due to the change in the step 1, a new Fireball Prefab slot will appear when you view the WanderingAI script component in the Inspector.
  - Drag up the fireball prefab from Project onto the Fireball Prefab slot in the Inspector.

Modify WanderingAI.cs

```
...
[SerializeField] private GameObject fireballPrefab;
private GameObject _fireball;
...
if (Physics.SphereCast(ray, 0.75f, out hit)) {
  GameObject hitObject = hit.transform.gameObject;
  if (hitObject.GetComponent<PlayerCharacter>()) {
    if (_fireball == null) {
      _fireball = Instantiate(fireballPrefab) as GameObject;
      _fireball.transform.position =
          transform.TransformPoint(Vector3.forward * 1.5f);
      _fireball.transform.rotation = transform.rotation;
    }
  }
  else if (hit.distance < obstacleRange) {
    float angle = Random.Range(-110, 110);
    transform.Rotate(0, angle, 0);
  }
}
...
```

WanderingAI additions for emitting fireballs

Modify WanderingAI.cs



Damaging the Player

- Create a new script named PlayerCharacter.cs
- Attach it to the player object.
  - The script will save the player's health - `_health` variable.
  - The script will decrement the health - `Hurt(damage)` function.
- Modify Fireball.cs to invoke Hurt() function when a fireball hits the player.

PlayerCharacter.cs

```
using UnityEngine;
using System.Collections;
public class PlayerCharacter : MonoBehaviour {
  private int _health;
  void Start() {
    _health = 5;
  }
  public void Hurt(int damage) {
    _health -= damage;
    Debug.Log("Health: " + _health);
  }
}
```

Modify Fireball.cs

```
using UnityEngine;
using System.Collections;
public class Fireball : MonoBehaviour {
  public float speed = 10.0f;
  public int damage = 1;
  void Update() {
    transform.Translate(0, 0, speed * Time.deltaTime);
  }
  void OnTriggerEnter(Collider other) {
    PlayerCharacter player =
      other.GetComponent<PlayerCharacter>();
    if (player != null) {
      // Debug.Log("Player hit");
      player.Hurt(damage);
    }
    Destroy(this.gameObject);
  }
}
```

Resources

- Fireball.cs
- PlayerCharacter.cs
- RayShooter.cs
- ReactiveTarget.cs
- SceneController.cs
- WanderingAI.cs

Summary – Objects and Attached Scripts

- Player – MouseLook, FPSInput, PlayerCharacter
- Camera – MouseLook, RayShooter
- Enemy – ReactiveTarget, WanterdingAI
- Fireball - Fireball
- Controller - SceneController