Bag of Rocks Weapon Prefab

What It Does

This weapon is a bag of rocks that points towards the mouse. On attack, a rock flies out of the bag, and goes in a straight line until it hits something. On hitting something, the rock shatters into a random number of fragments, which each fly in a random direction. The fragments are destroyed on hitting something. If the thing a fragment hits is an enemy, the enemy takes damage.

How To Use It

Install the prefab and all its needed scripts and sprites. It needs the scripts:

- AbstractWeapon.cs
- RangedWeapon.cs
- Projectile.cs
- ProjectileCluster.cs
- SimpleParticle.cs

And the sprites:

- openBag
- Rock_1
- rock_2

After this, drag it into a scene, and assign it to a player object. The player object's script should set the weapon's game object to active – to support creation of weapons before they are equipped, weapons start as inactive. The player object should also have an input mapped to call the attack() function on the weapon.

What It Expects

This weapon is designed for side-on, not top-down, 2D games.

By default, the weapon requires them to have the tag "Enemy" and a script Enemy.cs in order to damage them. The enemy script should have a function takeDamage(int) which handles the enemy-side logic of having its health reduced and eventually being defeated.

There should be a class AudioManager with a static AudioManager variable called instance and a function PlaySFX(string) which plays a sound. When the weapon attacks, it calls this function to play a specific sound effect, which is referred to by a string ID. By default, the bag of rocks uses the id rangedAttack for its sound effect. The AudioManager.instance variable can be null, in which case no sound will be played.

When the weapon is in usage, its transform's parent should be the player object. This player object should have the ability to face left and right, and should do so by rotating 180 degrees around the y axis. The weapon is constrained to facing in a 240-degree arc, centered around the direction the player is facing, so it will not be able to fire in all directions if the player cannot turn.

How To Modify It

The top-level object in the prefab can configure the damage dealt by fragments and the time between launching rocks. It also controls the ID of the sound played on attacks, and has traits for a Display Name and a Price to support features like shops selling weapons. The launch point field is a child transform where the projectile is spawned on attacking. The weapon can additionally be given a cooldown particle object with a SimpleParticle script – when the player tries to attack faster than the weapon allows, several instances of that particle will be spawned and shortly destroyed.

The projectile fired by the weapon can be customized in two stages. The initial rock which splits into fragments is of type ProjectileCluster, and has fields to control the speed it travels at and the number of fragments it splits into. The fragments are of type Projectile, and have a field to control their speed. The ProjectilePrototype field in the ProjectileCluster projectile can be changed to change what is spawned when the rock shatters. The ProjectilePrototype field in the top-level object can be changed to change the initial projectile that the weapon fires. If the top-level object's ProjectilePrototype is set to a Projectile and not a cluster, the initial projectile will not shatter, and will instead directly damage enemies and be destroyed.

All sprites can be freely changed. Sprites should not be attached to the game object with the script, but to its first child. By default, this prefab has each of the first children given the name Sprite, but any name can be used.

The script function RangedWeapon.FixedUpdate() controls the mouse-pointing behavior. This function can be modified to change this behavior.

To change or remove the integration with the AudioManager, edit the function AbstractWeapon.attack()

Enemy integration happens in two places – Projectile.OnCollisionEnter2D() checks that the object collided with is tagged as an enemy, and AbstractWeapon.processHit() fetches the enemy script and calls the take damage function.