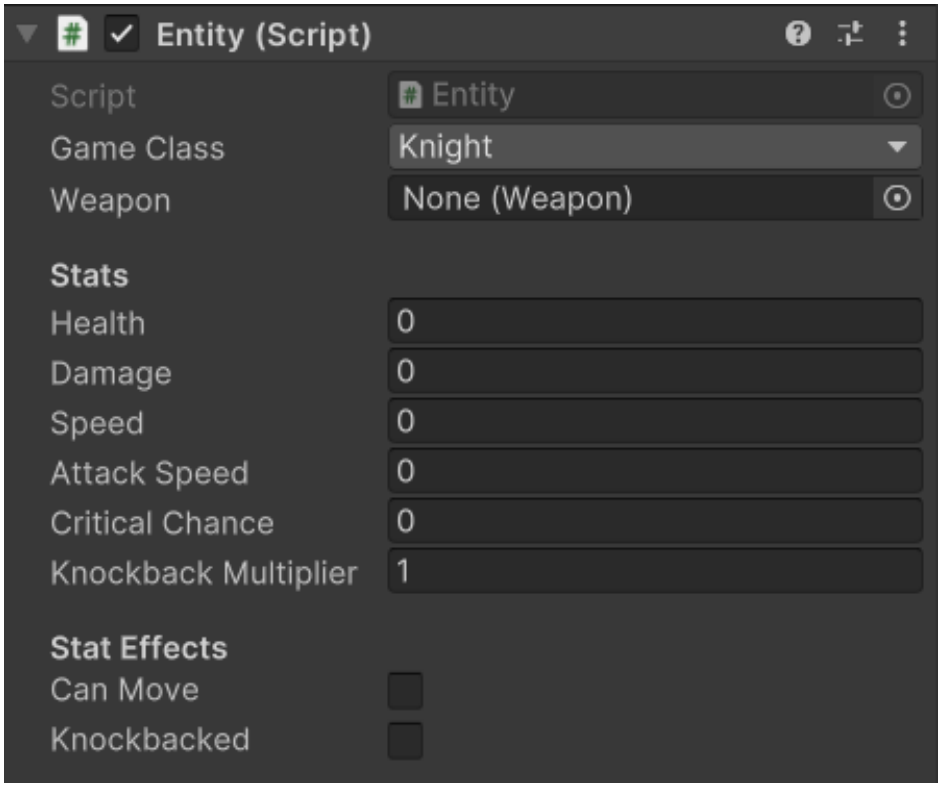
Entity

Entity is the base class for every entity in this pack. It stores base variables that every entity shares and uses. Starting with:

GameClass is an **enum** that stores the classes available in the pack. They are Knight, Archer, Mage and Monster. The entity must have a weapon according to it's class (Sword, Bow, Staff, None)

Weapon is the weapon reference that the entity wields.

Stats and StatEffects play role in the entity's character.

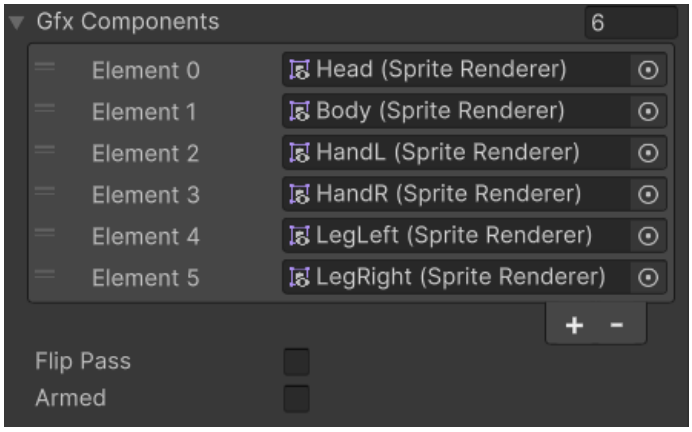


HandsManager has various Transform References of the entity that include it's **Back**, **FreeHand**, **OffHand** and **WeaponCenter**. These transforms are used for **weapon** wielding. When the entity is disarmed the weapon goes on it's **back**. When the entity is armed the **OffHand** holds the weapon and **FreeHand** is ready to hold the weapon when entity flips look direction.

GFX Components are a list of sprites that are the structure of the entity. The entity is a sprites stitch of it's Head, Body, Left Hand, Right Hand, Left Leg and Right Leg **in this order**. If the entity has a shield as part of it's weapon, the shild will be added as the 6th element of this List so visual effects like taking damage apply to the shield as well.

FlipPass is a bool that enables FlipSprite() to be called at any point in time. Normally FlipSprite() doesn't run in Update() and it gets called only when the sprite flips direction based on input, movement or facing the mouse for optimization purposes.

Armed indicates wether the entity has it's weapon ready for combat.



GFX is the parent of all the sprites from **GFX Components**

CoreCollider is the collision collider the Entity uses to collide with world objects.

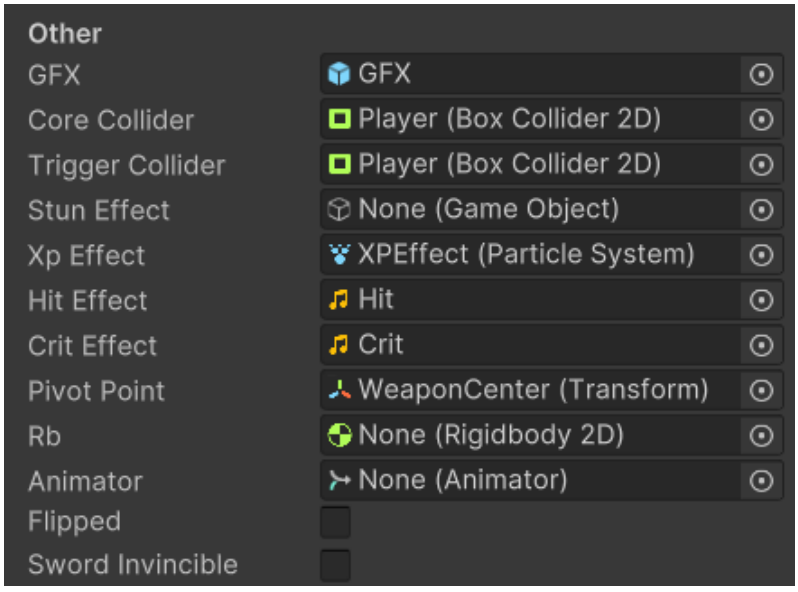
TriggerCollider is the trigger collider the Entity uses to handle game logic such as pick up and take damage.

XPEffect is the effect that plays when the entity dies.

PivotPoint is the pivot point around which the weapon rotates when aimed at an enemy.

Flipped indicates wether the Entity is flipped left or right.

SwordInvincible is a bool that ensures the Entity will get hit only once from a sword swing.



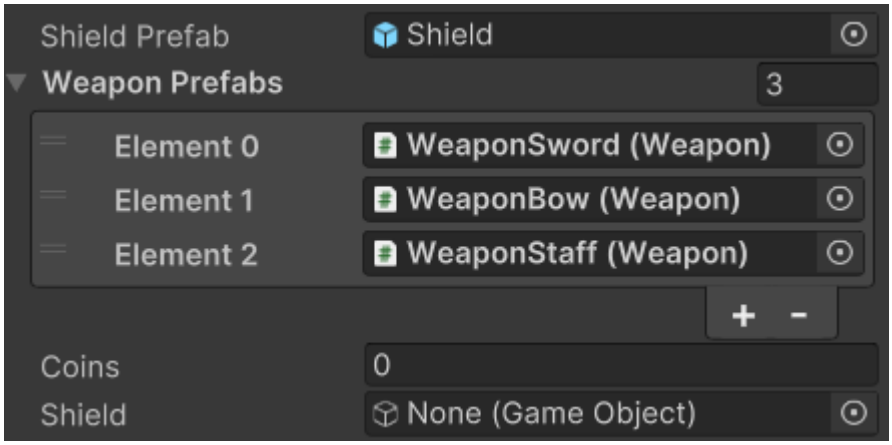
Player

Player is a direct child class of **Entity**. It has all the base variables and values as Entity with the only addition being:

WeaponPrefabs is an array of the 3 available weapons. Player class gets assigned by pressing 1, 2 or 3 on the keyboard once the game starts. 1 is a Knight, 2 is an Arhcer, 3 is Mage.

Coins is the value of coins the player posseses.

Shield is a reference to the shield the player gets if he selects a knight class.



Enemy

Enemy is a direct child class of **Entity**. It has all the base variables and values as Entity with a couple additions:

PossibleLoot is an array of items that the Enemy will drop once defeated.

Behaviour Ranges has all the values that control the enemy's behaviour.

AggroRange is a circular range in which the enemy will actively **follow** the player. It is displayed by a blue wire sphere gizmo in scene view.

AttackRange is a circular range in which the enemy will **attack** the player. It is displayed by a red wire sphere gizmo in scene view.

WanderRange is a circular range in which the enemy will wander when not aggroed by the Player. This range is based around the enemy's spawn position.

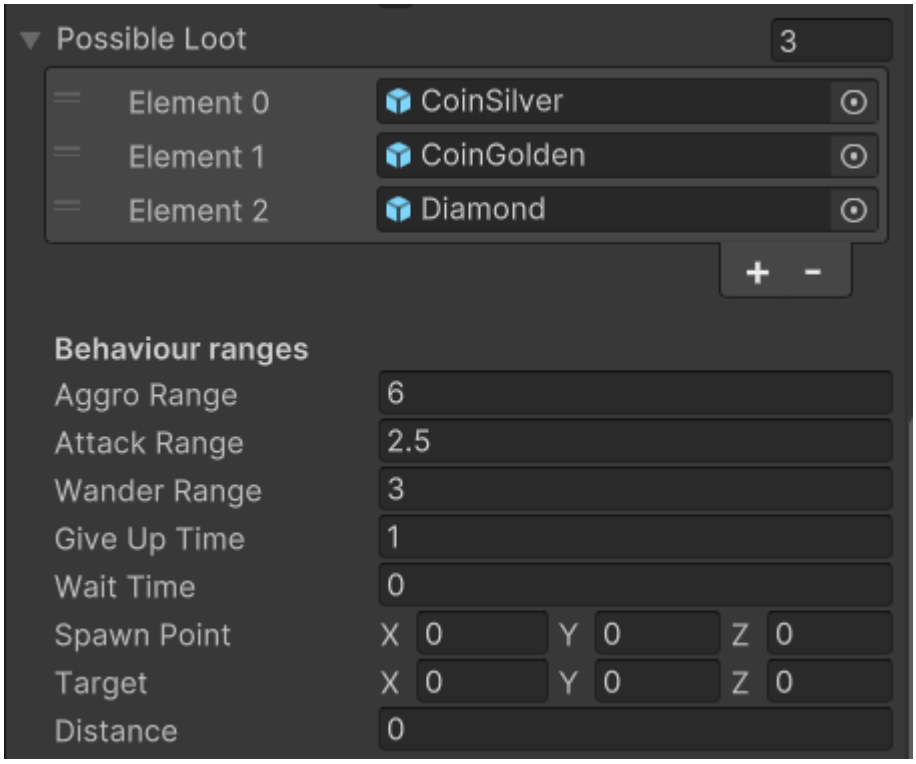
GiveUpTime is the time it takes for the enemy to give up on chasing the player once he exits the **AggroRange**.

WaitTime is a random time interval that the enemy will wait before wandering withing the **WanderRange**.

SpawnPoint is the spawn position of the Enemy. It gets calculated when the game starts.

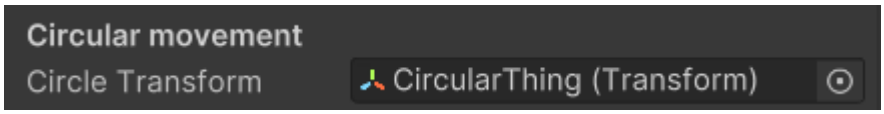
Target is the Vector3 position of the desired destination. The Enemy will actively go in the direction of that point. This point could be the player or a wander point within **WanderRange**.

Distance is the distance between the **Enemy** and the **Target**.



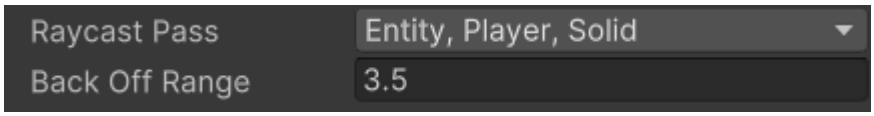
EnemyKnight

EnemyKnight is a direct child class of **Enemy** and indirect child class of **Entity**. It has all the base variables and values as **Entity** along with **Enemy** subclass additions. The final addition is the **CircleTransform**. It is used for the Knight's circular movement around the **Player**. You can find more about that in EnemyKnight.cs.



EnemyArcher

EnemyArcher is a direct child class of **Enemy** and indirect child class of **Entity**. It has all the base variables and values as **Entity** along with **Enemy** subclass additions. The final addition are the **RaycastPass** and **BackOffRange**. RaycastPass is used when the enemy aims it's bow. It determines which layers the bow raycast should hit. The raycast is a straight line from the **ShootPoint** forward. When a gameObject with any of the following tags gets hit, the EnemyArcher will fire an arrow. **BackOffRange** is the minimum distance between the enemy and the player in which the enemy is comfortable. If the player is less than 3.5 units away from the **EnemyArcher**, it will start backing off further away from the **Player**.

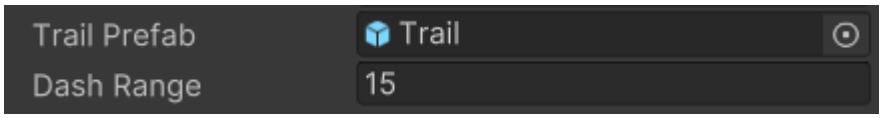


EnemyMage

EnemyMage is a direct child class of **Enemy** and indirect child class of **Entity**. It has all the base variables and values as **Entity** along with **Enemy** subclass additions. Instead of following the Player when in **AggroRange**, the EnemyMage just wanders around it's **SpawnPoint**. When the player gets in **AttackRange**, the staff fires homing projectiles that strive to hit the **Player**.

EnemySlime

EnemySlime is a direct child class of **Enemy** and indirect child class of **Entity**. It has all the base variables and values as **Entity** along with **Enemy** subclass additions. The final addition are the **TrailPrefab** and **DashRange**. **EnemySlime** moves in bursts of movement. Each time it leaps towards it's **Target** it leaves a **Trail** behind. The **Trail** is a subclass of the **Hazard** class. When the Player collides with it, he will take damage. **DashRange** is the distance the **EnemySlime** will move by when leaping.



CameraManager

The CameraManager script is responsible for making the camera follow the player, shaking and setting boundaries.

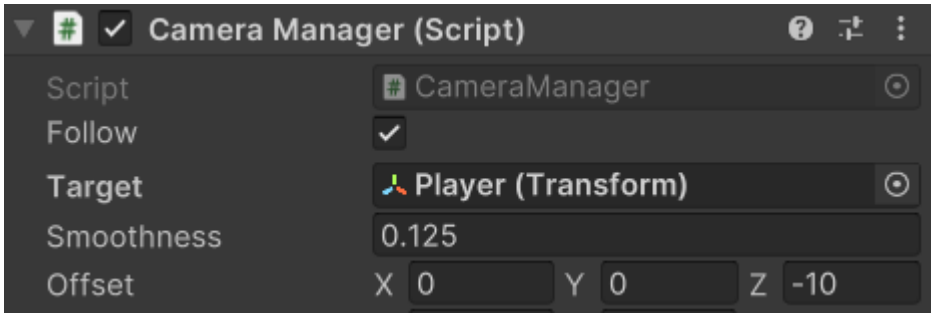
How to setup camera **Follow**:

-Set follow to **true**

-Assign **Target**

-Adjust **Smoothness**

- Set **Offset** of camera



Camera Shaking: Shaking the camera is done through a coroutine. You can call the function like so: **StartCoroutine(Shake(.2f, .1f));**

```
4 references
public IEnumerator Shake(float duration, float force)
{
    YieldInstruction waitForFixedUpdate = new WaitForFixedUpdate();

    Vector3 originalPos = transform.position;
    Vector3 shakePos;

    while (duration > 0)
    {
        duration -= Time.deltaTime;
        float randomX = Random.Range(originalPos.x - 1 * force, originalPos.x + 1 * force);
        float randomY = Random.Range(originalPos.y - 1 * force, originalPos.y + 1 * force);
        shakePos = new Vector3(randomX, randomY, transform.position.z);
        transform.position = shakePos;
        yield return waitForFixedUpdate;
    }

    transform.position = originalPos;
}
```

Collectable

The Collectable script is responsible for picking up coins and items.

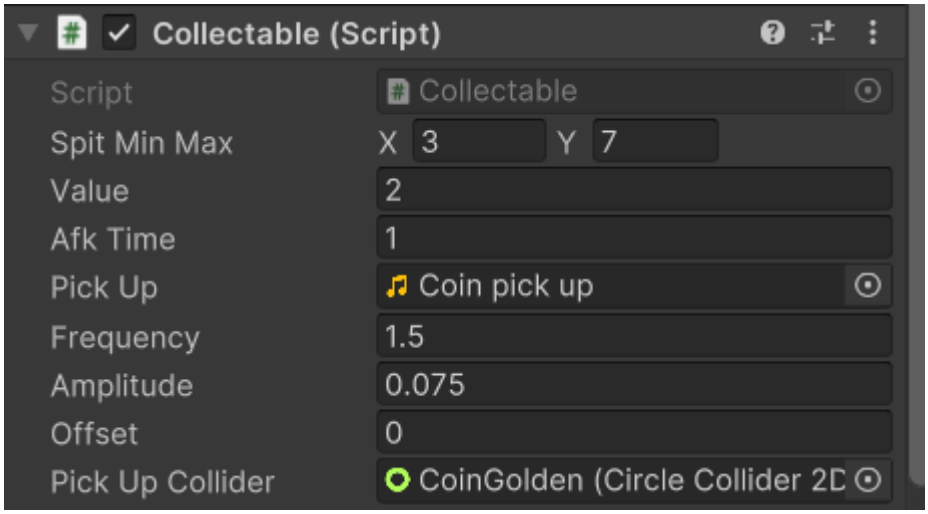
SpitMinMax is a Vector2 that holds the minimum and maximum force that will be applied to the collectable once spawned.

Value is the monetary value of the collectable.

AfkTime is the time before the loot can be picked up by the player.

Frequency and **Amplitude** influence the speed and range with which the Collectable floats up and down.

The **Offset** is the starting (float up and down) position of the collectable. It is randomized between -0.2f and 0.2f when spawned.



Weapon

Weapon.cs has all the logic for weapons in the game.

Type is an enum that indicates the weapon type (sword, bow, staff, scythe)

Wielder is an **Entity** reference that indicates which entity owns the weapon

Qattack indicates whether an attack is queued to play after the previous attack

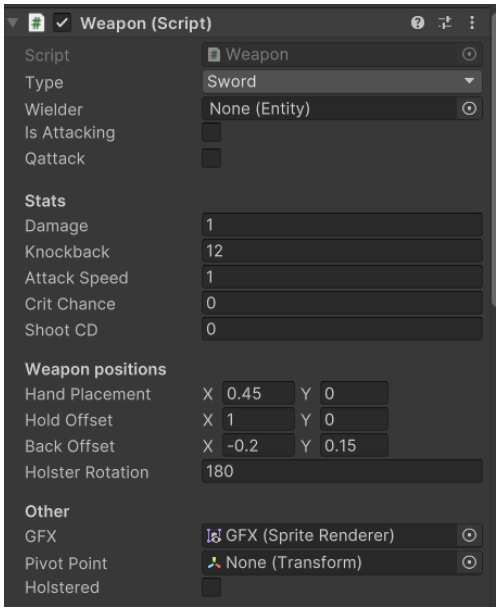
Stats contain various statistics about the weapon. Keep in mind these get added on top of player stats (For example if the player deals one damage and the weapon deals 1 damage, it equals 2 damage in total)

WeaponPositions contains all the positions for the weapon. If you must change it, adjust it in play mode and write down the values to save time.

GFX is the child sprite renderer of the weapon.

PivotPoint is the point around which the weapon rotates. The **PivotPoint** is calculated in code.

Holstered indicates whether the weapon is in the players hands, ready to be used, or on his back, stashed for later use.



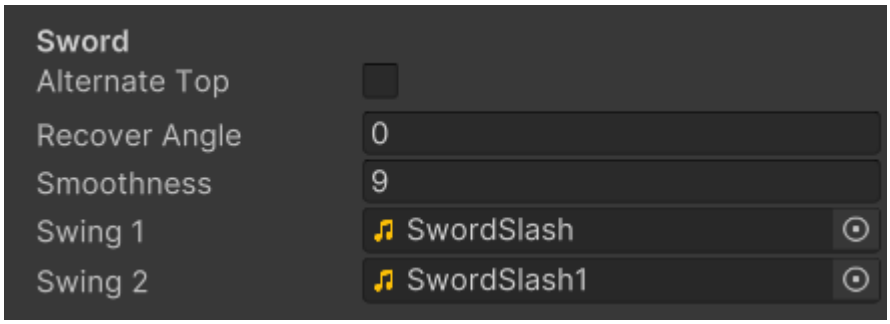
Sword:

AlternateTop is a boolean which indicates from which side the sword will swing. An attack will be either a top or bottom swing. It is set to alternate between the two.

RecoverAngle is the angle at which the sword can be used again. It is calculated in code.

Smoothness affects the sword speed.

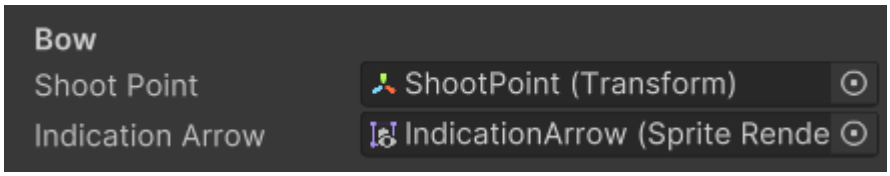
Swing1 and **Swing2** are sound effects that play when the sword is swung.



Bow:

ShootPoint is a transform from which the arrows are shot

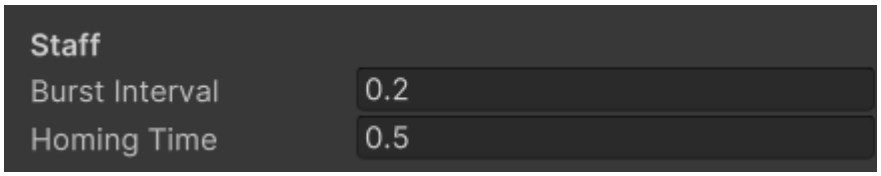
IndicationArrow is a dummy arrow that isn't shot but used to indicate the reloading of the bow. When you look at the animation you can notice an arrow is being inserted in the bow, but that's a different arrow from the one actually shot.



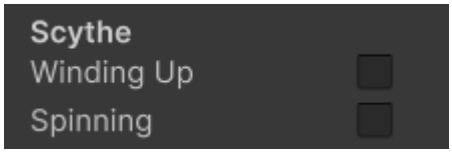
Staff shoots multiple homing projectiles at once.

BurstInterval indicates the interval between these projectiles.

HomingTime indicates for how long the projectiles will seek a target. After 0.5 seconds they will continue in the direction they were going and no longer seek a target.

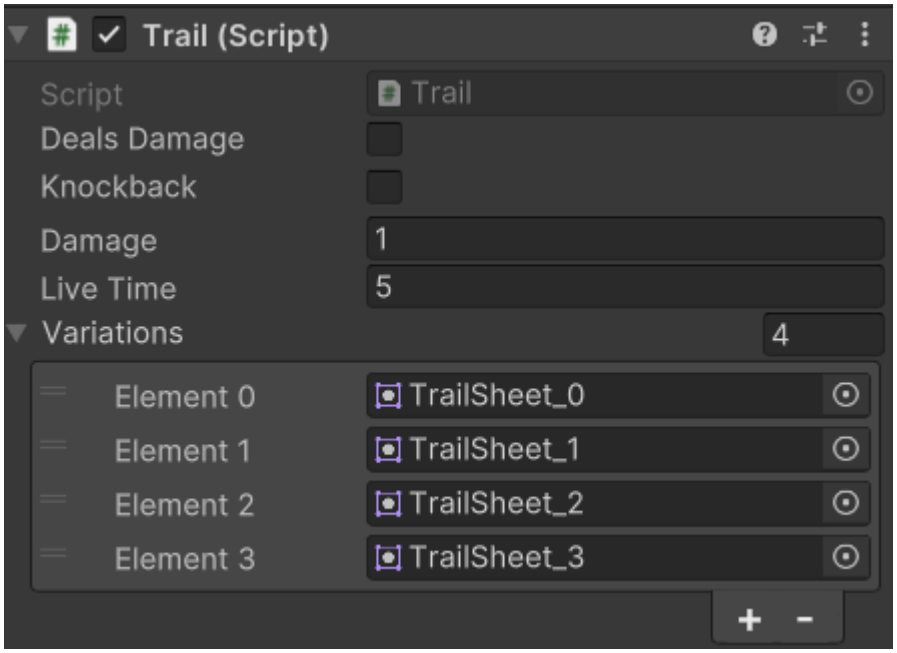


Scythe **WindingUp** and **Spinning** indicate whether the scythe is in the midst of an attack.



Trail

Trail.cs is responsible for the trail left after the **Slime** Entity. It is a subclass of **Hazard.cs**. The only difference is the **Variations** sprite array, that sets a random slime trail sprite for each object spawned. For more information check out **Hazard.cs**



HitNumber

HitNumber.cs is a world canvas that gets spawned at the hit position with a Text child. The Text displays the damage dealt. If it is critical damage, the color of the text will turn yellow. To find out more, read through the comments of HitNumber.cs and check out HitNumber prefab.

MySlider

MySlider.cs is an upgraded version of a regular Slider. It can shake, gradually deplete/increase and to display particle effects when the value is being changed.

ShakeForce is the force with which the slider is shaking when value is being changed.

Hide?

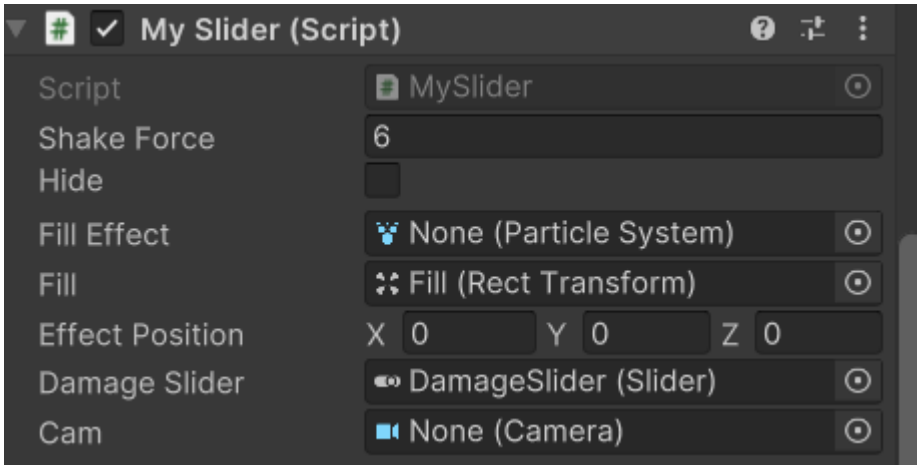
FillEffect is the particle system that gets played when value is being changed.

Fill is the UI element of the slider that displays the fill.

EffectPosition is the offset of the Particle effect that plays.

DamageSlider is used in the case of **HealthSlider** to recreate the effect of first taking damage and then depleting the slider.

Cam is the camera which is rendering the **FillEffect**.



Hazard

Hazard is a base class which makes an object deal damage upon collision. It requires a Rigidbody2D and a Collider2D to work.

You can adjust whether it **DealsDamage**, does **Knockback** and how much **Damage** it deals.