

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.

Script	Entity
Game Class	Knight
Weapon	None (Weapon)
Stats	
Health	0
Damage	0
Speed	0
Attack Speed	0
Critical Chance	0
Knockback Multiplier	1
Stat Effects	
Can Move	<input type="checkbox"/>
Knockbacked	<input type="checkbox"/>

DashRange indicates how long the dash distance is.

DashCD is the cooldown between dashes.

DashInvincibility is a short time in which the player is invincible during the dash.

Dashing is a bool indicating wether the Entity is currently dashing.

DashMask is a layer mask that the Entity can't dash through.

Dash	
Dash Range	8
Dash CD	1
Dash Invincibility	0
Dashing	<input type="checkbox"/>
Dash Mask	Solid, BossAttack

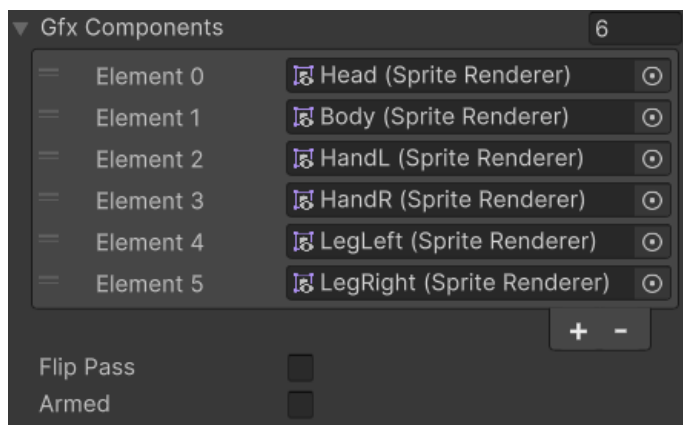
HandsManager has various Transform References of the entity that include it's **Back**, **FreeHand**, **OffHand** and **WeaponCenter**. These transforms are used for **weapon** wielding. When the entity is disarmed the weapon goes on it's **back**. When the entity is armed the **OffHand** holds the weapon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

Other		
GFX	 GFX	
Core Collider	 Player (Box Collider 2D)	
Trigger Collider	 Player (Box Collider 2D)	
Stun Effect	 None (Game Object)	
Xp Effect	 XPEffect (Particle System)	
Hit Effect	 Hit	
Crit Effect	 Crit	
Pivot Point	 WeaponCenter (Transform)	
Rb	 None (Rigidbody 2D)	
Animator	 None (Animator)	
Flipped	<input type="checkbox"/>	
Sword Invincible	<input type="checkbox"/>	

Player

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

















Shield is the gameObject Reference to the shield that comes along with **Sword Weapon**.

BlockCD is the cooldown before you can use the block ability.

SteelMat is the material that gets applied to the player when he uses the block ability.

SteelEffects is a list of Sprite Masks that get applied to the player's **GFX Components**. This way it makes the player's skin look as if made from steel.

SteelHands and **RegularHands** are Sprites instead of Sprite Masks and the hands' sprite gets changed instead of a mask applied.

Shield		
Shield	 None (Game Object)	
Block CD	3	
Steel Mat	 SteelSkin	
Steel Effects	4	
Element 0	 SteelEffect (Sprite Mask)	
Element 1	 SteelEffect (Sprite Mask)	
Element 2	 SteelEffect (Sprite Mask)	
Element 3	 SteelEffect (Sprite Mask)	
+ -		
Steel Hands	 SteelHands	
Regular Hands	 PlayerHands	

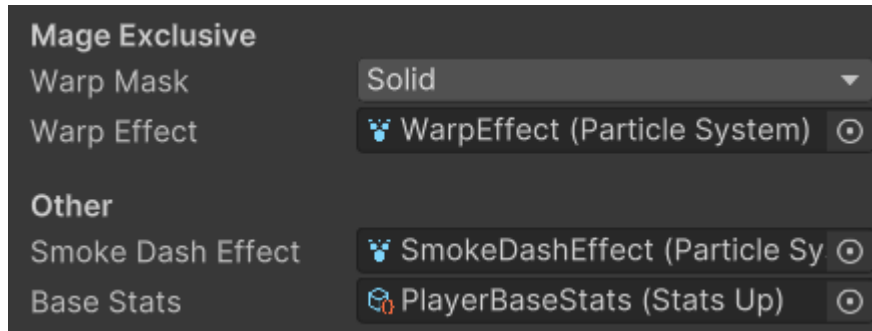
In case the player picked the **Mage Class** the **Dash** gets changed to a warp (small teleport).

WarpMask makes sure the player can't warp through certain layers.

WarpEffect is the effect that plays when the player warps.

SmokeDashEffect is the regular effect that plays when the player dashes.

BaseStats is a **StatsUp** reference that gets applied to the player at the start of the game.



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**.

The image shows a UI panel for configuring an Enemy. It is divided into two main sections: 'Possible Loot' and 'Behaviour ranges'.

Possible Loot: This section has a dropdown menu currently showing '3'. Below it is a list of three items, each with a small cube icon, a text label, and a circular button with a dot in the center.

Element	Item
Element 0	CoinSilver
Element 1	CoinGolden
Element 2	Diamond

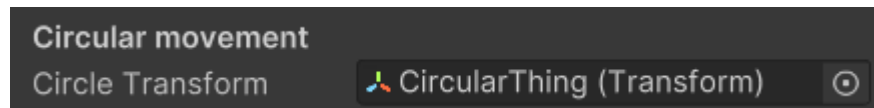
Below the list are two buttons: a plus sign (+) and a minus sign (-).

Behaviour ranges: This section contains a list of settings, each with a label and a corresponding input field.

Setting	Value
Aggro Range	6
Attack Range	2.5
Wander Range	3
Give Up Time	1
Wait Time	0
Spawn Point	X: 0, Y: 0, Z: 0
Target	X: 0, Y: 0, Z: 0
Distance	0

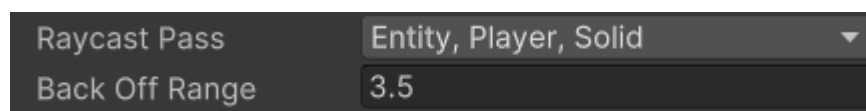
EnemyKnight

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 its 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**.

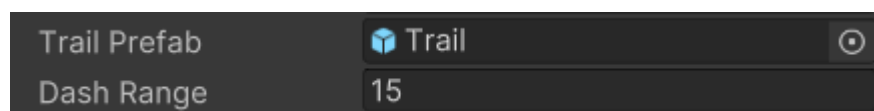


EnemyMage

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 its **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 its **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.



BossNecromancer

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

BossName is the name of the boss to be displayed in-game.

AttackCD is the main cooldown before the boss launches an attack. There are 3 types of attacks:

Summon - which summons 4 enemies to fight for the boss.

Slash is a melee attack that gets triggered only if the player is in melee range.

Circle is the spawn of **SatanicCircle** at the player's position. It is a **Hazard** that will damage the player if he collides with it.

When the **AttackCD** reaches 0 one of the attack types will be triggered based on these conditions:

Summon: Player isn't in melee range.

Slash: Player is in melee range.

Circle: Summon is on cooldown and player isn't in melee range.

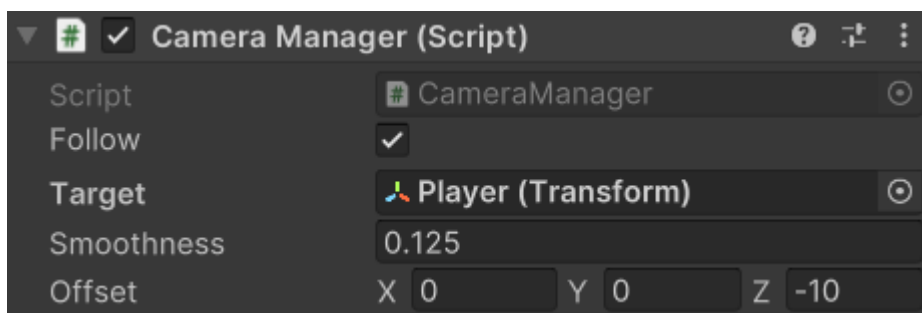
Boss Name	Necromancer
Cooldowns	
Attack CD	3
Summon CD	15
Slash CD	3
Circle CD	2

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;
}
```

Adjust Boundaries: AdjustBoundaries() is called every time a scene is changed. If the level is changed to a dungeon **Room**, you get the boundaries of that room (minMaxX and minMaxY) and apply them to the camera. This will ensure the camera can NOT leave the room. In case there is no **Room** in the level, the camera's boundaries get unlocked.

1 reference

```
private void AdjustBounds(Scene current, Scene next)
{
    target = Player.instance.transform;

    // Find the dungeon Room of the level and set the camera boundaries to it's boundaries
    Room room = FindObjectOfType<Room>();
    if (room != null)
    {
        minMaxX = room.minMaxX;
        minMaxY = room.minMaxY;
    }
    else // If not in a dungeon room - don't limit the camera
    {
        minMaxX = new Vector2(-100, 100);
        minMaxY = new Vector2(-100, 100);
    }
}
```

Collectable

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

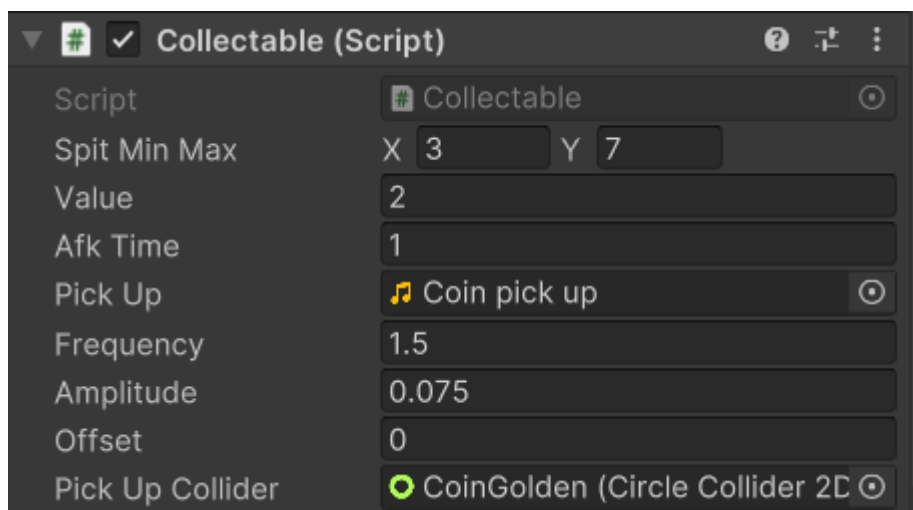
SpitMinMax is a Vector2 that holds the minimum and maximum force that will be applied to the collectable once spawned.

Value is the monetary value of the collectable.

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

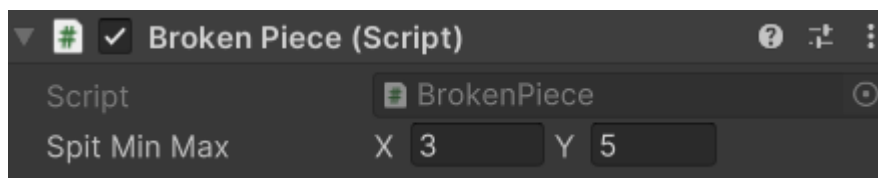
Frequency and **Amplitude** influence the speed and range with which the Collectable floats up and down.

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



BrokenPiece

BrokenPiece.cs is like **Collectable.cs**. When a Vase is broken it will spawn broken pieces, which will get scattered in different directions and will disappear shortly.



TreasureChest

The TreasureChest script is responsible for dropping loot when hit once and then disappearing.

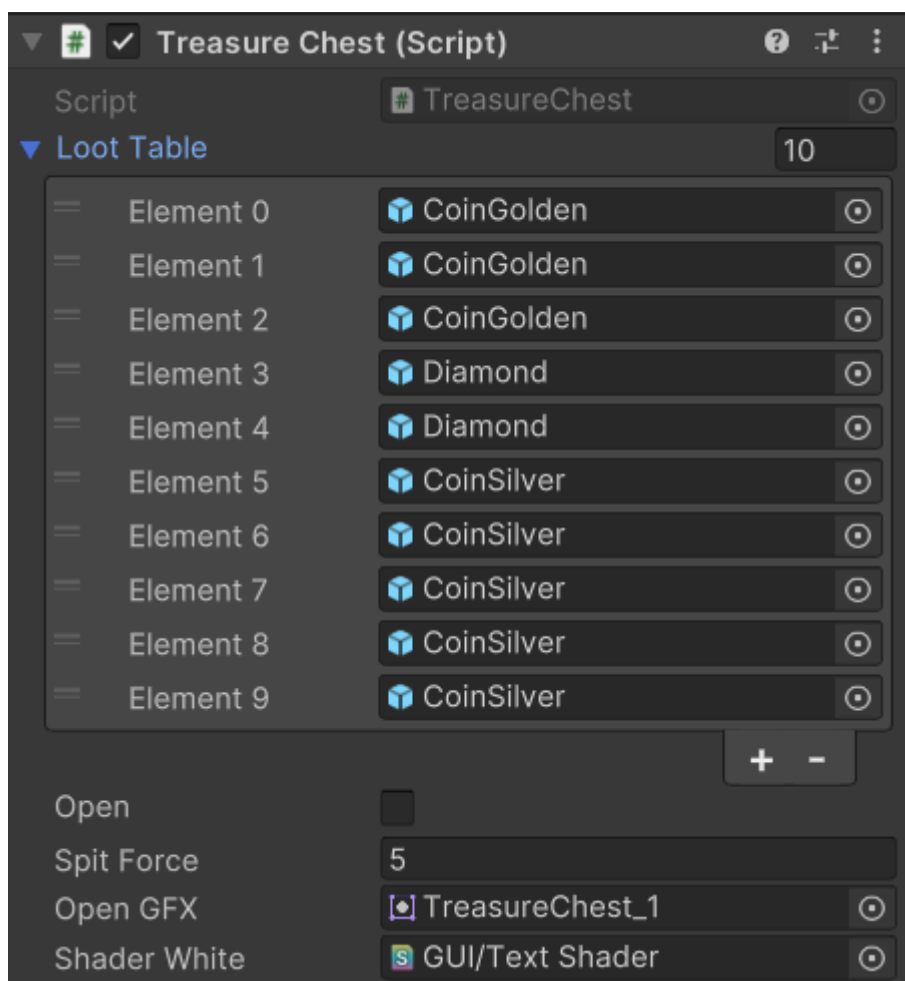
LootTable is a list of GameObjects that will spawn when the chest is hit/opened. Populate it with the desired loot.

Open is a bool indicating whether the chest is open or not.

SpitForce is the force which will be applied to the loot when it's spawned.

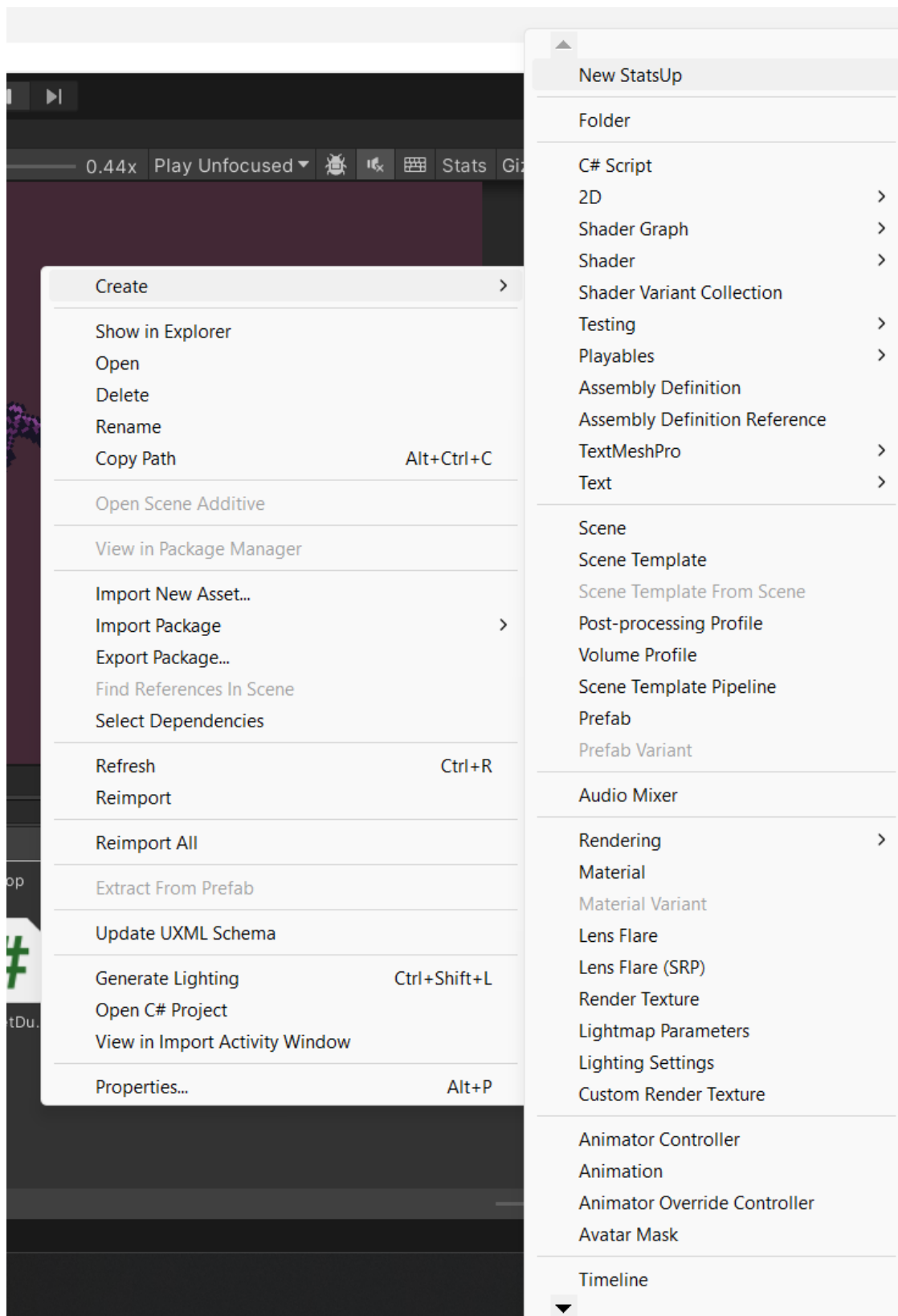
OpenGFX is a sprite of the open chest. When it's opened the closed sprite will be switched for the opened one.

ShaderWhite is a shader that will be applied when the chest is hit, in this case it turns the entire chest white.



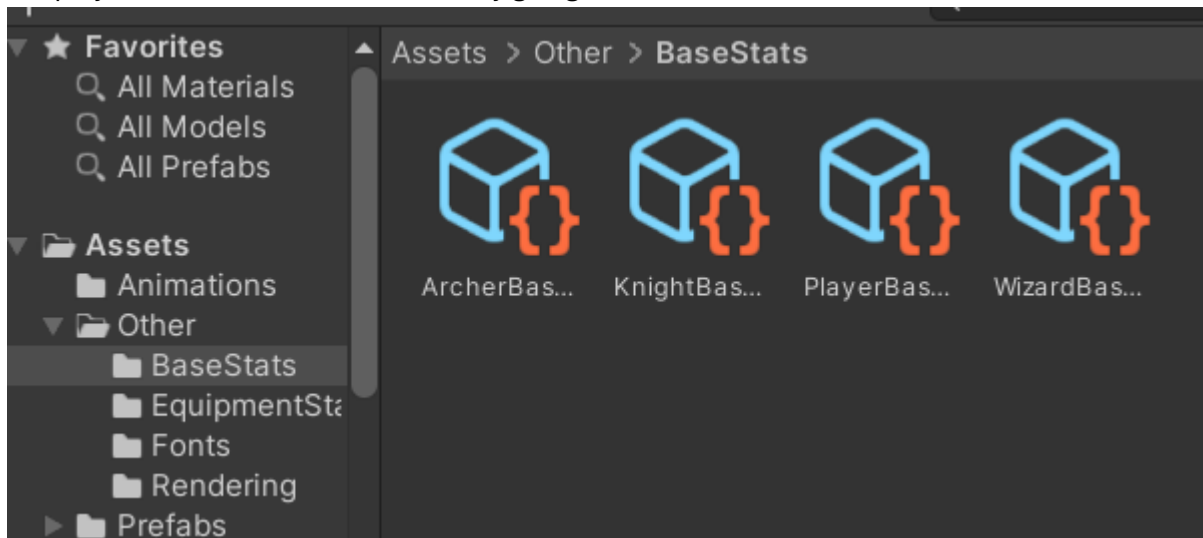
StatsUp

StatsUp script is a **ScriptableObject**. To create it right click in Project tab, select Create and click “New StatsUp”

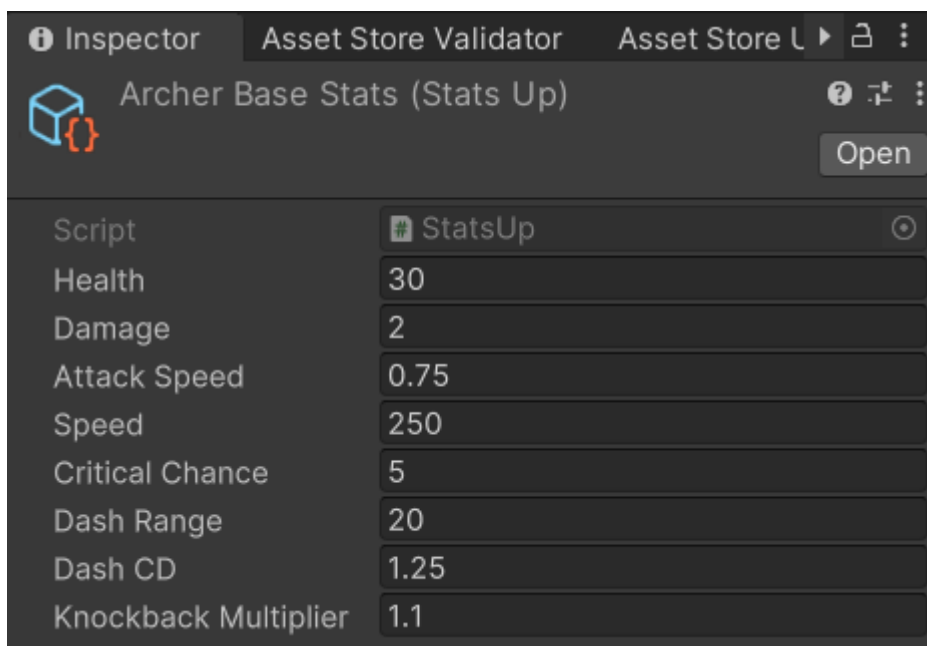


StatsUp is used when selecting a class and when leveling up:

When you select a class at the start of the game one of the premade **StatsUps** gets applied to the player stats. You can find them at by going to **Assets > Other > BaseStats**:



And edit them by clicking on them and adjusting the values in the inspector:

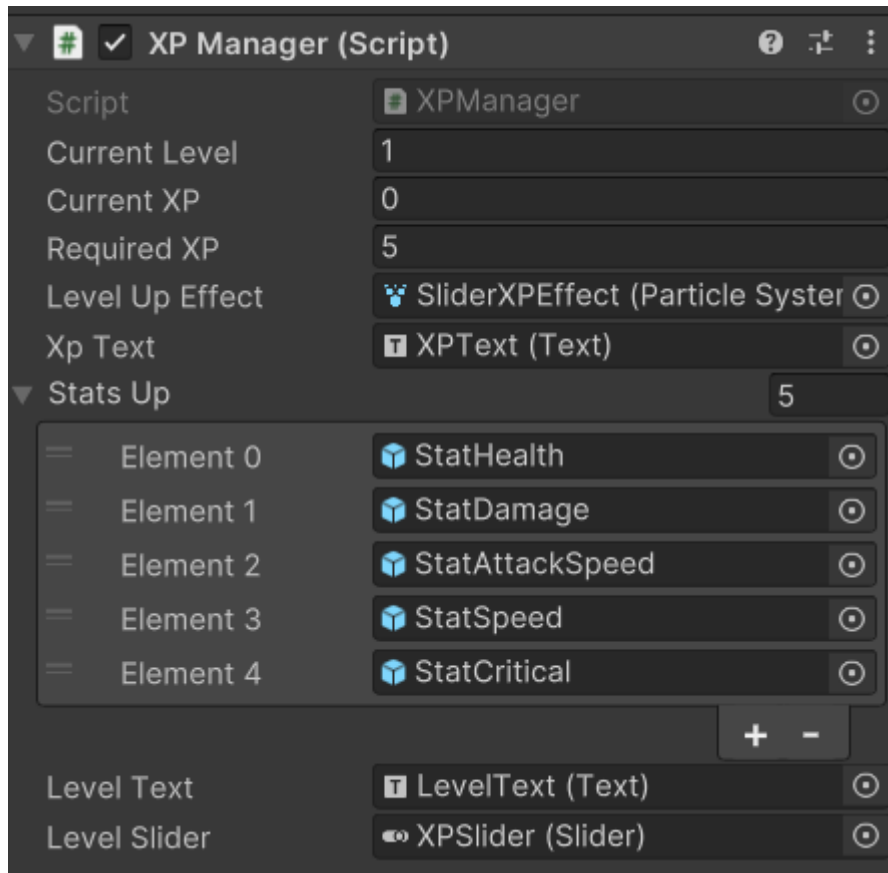


The other usage of Stats up is when the player levels up. When **LevelUp()** of **XPManager.cs** gets called a **StatsUp** ScriptableObject gets created in code and then applied to the **Player** like so:

```
48 public void LevelUp()  
49 {  
50     // Next level  
51     audioSource.Play();  
52     currentLevel++;  
53     levelText.text = currentLevel.ToString();  
54     requiredXP += 3;  
55     xpText.text = currentXP + "/" + requiredXP;  
56     levelUpEffect.Play();  
57  
58     // Choose on stats preset depending on current level  
59     StatsUp statsToAdd = ScriptableObject.CreateInstance<StatsUp>();  
60  
61     if (currentLevel %3 == 0) // Every third level do a medium boost  
62     {  
63         statsToAdd.SetStats(2, 2, 0, 0, 0, 0);  
64     }  
65     else if (currentLevel %5 == 0) // Every fifth level do a large boost  
66     {  
67         statsToAdd.SetStats(3, 2, 1, 20, 5, 0);  
68     }  
69     else // Every other level do a small boost  
70     {  
71         statsToAdd.SetStats(1, 1, 0, 10, 0, 0);  
72     }
```

XPManager

XPManager is responsible for the entire XP and leveling up system. It holds values such as CurrentLevel, CurrentXP and RequiredXP.



StatsUp are not to be confused with the **StatsUp.cs ScriptableObject**. They are UI elements that get displayed when the player levels up. You can find more about leveling up by taking a look at the code.



XpText and **LevelText** are the texts on the **player UI** (XP 1/10 and LVL 1). They get updated when the game starts. The **LevelSlider** is the orange slider.



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.



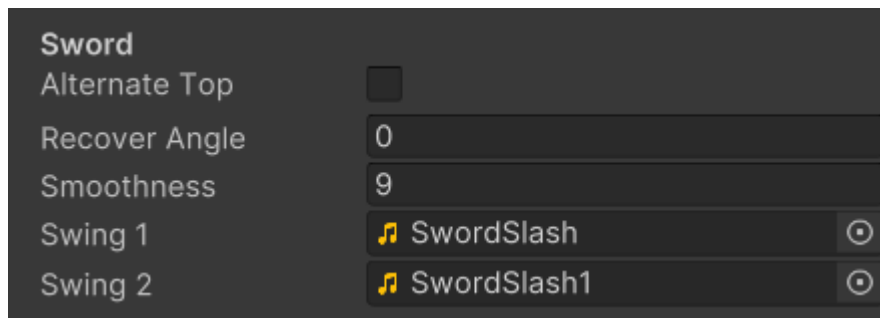
Sword:

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.

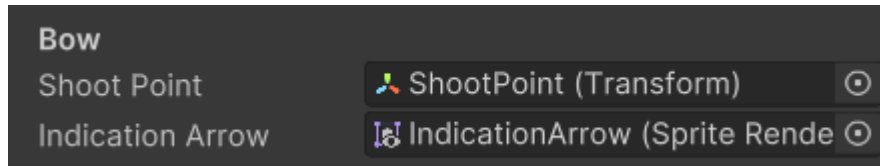
Swing1 and **Swing2** are sound effects that play when the sword is swung.



Bow:

ShootPoint is a transform from which the arrows are shot

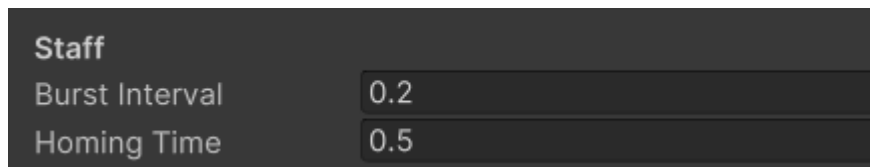
IndicationArrow is a dummy arrow that isn't shot but used to indicate the reloading of the bow. When you look at the animation you can notice an arrow is being inserted in the bow, but that's a different arrow from the one actually shot.



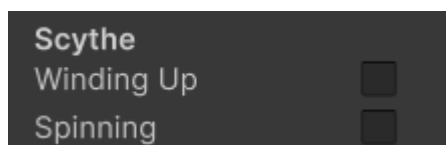
Staff shoots multiple homing projectiles at once.

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.

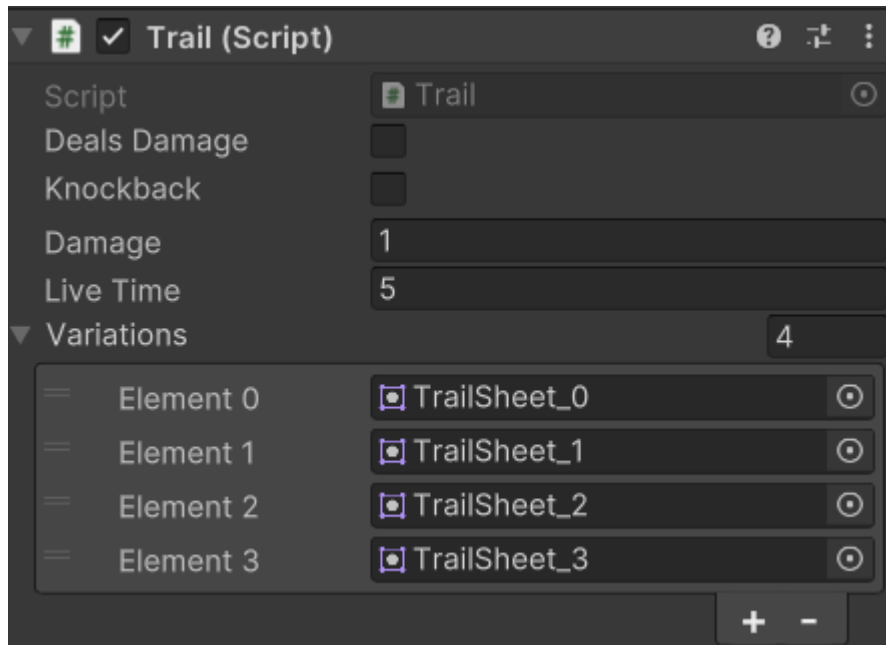


Scythe **WindingUp** and **Spinning** indicate whether the scythe is in the midst of an attack.



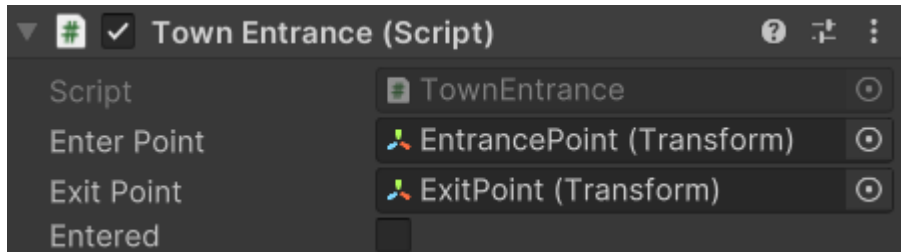
Trail

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**



TownEntrance

TownEntrance.cs is a custom and similar version of **DungeonEntrance.cs**. The only difference is that it is the first entrance to the dungeon and the only connection between the Village and Dungeon. When the player enters it will make the character smaller and invisible to simulate him going down the stairs. For more information check out the code itself.



TargetDummy

TargetDummy.cs is an **Entity** subclass where the TakeDamage() function has been modified to provide the feedback of getting hit (shaking, sound, visual effects) but not actually take any hitpoints or knockback.

Room

Room.cs has the logic for Type, Boundaries, Time, Level and Wave Spawning.

Type is the room type, it can be Fight, Survive, Shop, Boss. **Fight** room is a regular room where waves of enemies will spawn. **Survive** room is a room where enemies will continuously spawn until a cap is reached. The goal of the player is to survive. **Killing these enemies doesn't award XP!** In a **Shop** room no enemies will be spawned. **Boss** room no enemies will be spawned except one Boss.

ChestPos is the desired spawn position for a **Treasure Chest** when the room has been cleared.

Doors is an array of **DungeonEntrance.cs** the room has. All the rooms have just one door, but in case of more doors, they can easily be added.

Exit is the door by which the player enters the room. After entering the room, it gets permanently closed. It is also the door that decides from which direction the player will enter.

Boundaries are two Vector2 vectors that store the minimum and maximum for both X and Y. When the player enters the room, these values get applied to the **CameraManager.cs** boundaries.



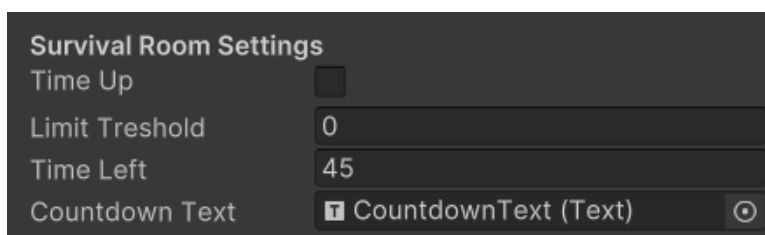
Survival Room Settings include:

TimeUp is a boolean. When the time is up all of the enemies will be destroyed and the room will be considered complete.

LimitTreshold is the limit of enemies that will get spawned within the room.

TimeLeft displays how much time is left before all the enemies die and the room is complete.

CountdownText is a reference to a UI text that displays **TimeLeft**.



Wave Management contains everything about spawning waves.

SpawnIndicatorPrefab is the indicator that gets spawned before the actual **Enemy**.

Waves is a list of integers. The count of the list is the number of waves, and the elements are the number of enemies per wave. In this case: 2 total waves will be spawned. One of 4 enemies and one of 3 enemies.

EnemiesToSpawn are all the enemies to spawn in the waves. The first wave will spawn 4 enemies which will be element 0 to 3 included, the second wave will spawn 3 enemies which will be element 4 to 6 included. Wave 1 (Mage, Knight, Archer, Knight), Wave 2 (Arhcer, Slime, Archer)

SpawnPositions are the spawn positions of the enemies. **EnemiesToSpawn** Element 0 (EnemyMage) will get spawned at **SpawnPositions** Element 0 (SpawnPoint) and so on.

The image shows a 'Wave Management' configuration panel with three main sections: 'Waves', 'Enemies To Spawn', and 'Spawn Positions'. Each section has a list of elements and a count of items.

- Waves:** Shows 2 waves. Wave 0 has 4 enemies (Element 0 to 3), and Wave 1 has 3 enemies (Element 4 to 6).
- Enemies To Spawn:** Shows 7 enemies. Element 0 is EnemyMage, Element 1 is EnemyKnight, Element 2 is EnemyArcher, Element 3 is EnemyKnight, Element 4 is EnemyArcher, Element 5 is EnemySlime, and Element 6 is EnemyArcher.
- Spawn Positions:** Shows 7 spawn positions. Element 0 to 3 are SpawnPoint (Transform), Element 4 to 6 are SpawnPoint2 (Transform).

Section	Element	Value / Enemy Type
Waves	Wave 0	4
	Wave 1	3
Enemies To Spawn	Element 0	EnemyMage
	Element 1	EnemyKnight
	Element 2	EnemyArcher
	Element 3	EnemyKnight
	Element 4	EnemyArcher
	Element 5	EnemySlime
	Element 6	EnemyArcher
Spawn Positions	Element 0	SpawnPoint (Transform)
	Element 1	SpawnPoint (Transform)
	Element 2	SpawnPoint (Transform)
	Element 3	SpawnPoint (Transform)
	Element 4	SpawnPoint2 (Transform)
	Element 5	SpawnPoint2 (Transform)
	Element 6	SpawnPoint2 (Transform)

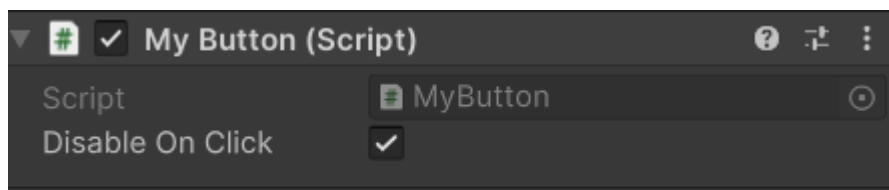
Prop

Prop.cs is a subclass of **Entity.cs** and its TakeDamage() function has been modified to call the DestroyProp() function, which drops loot and breaks the prop into random pieces, that later disappear. For more information check out the code.

MyButton

My button is a fancy button that gets bigger when you hover it and has a click effect.

DisableOnClick should be true if the button gets disabled when clicked. Otherwise, Unity will throw an error saying the Coroutine can't be run on an inactive object.



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.

NPC

NPC script is responsible for NPC movement and dialogue.

NpcName is the name of the NPC

DelegatesName is a list of strings containing the name of delegate voids that will get called during or at the end of the conversation. You can find more information in **DialogueManager.cs**.

Phrases is a list of strings that the Npc will say during the conversation

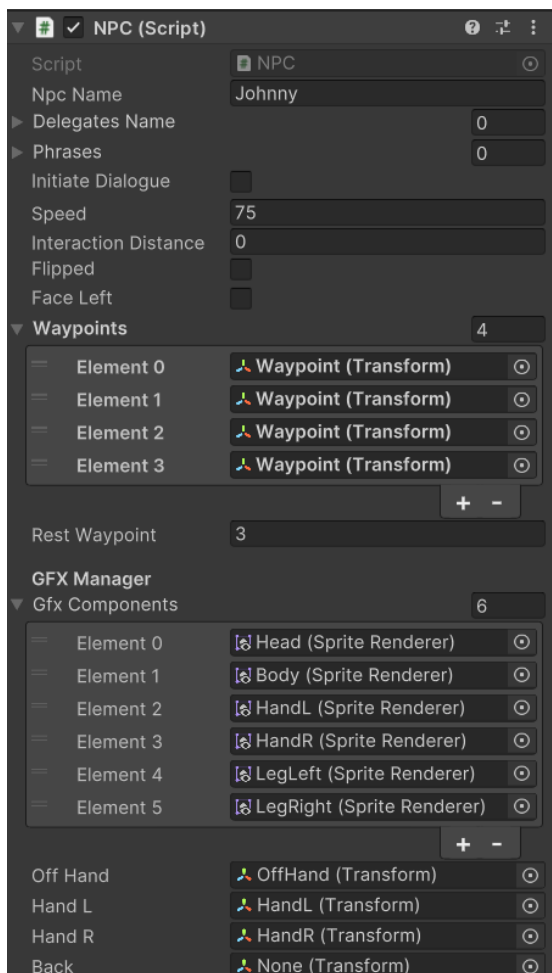
InitiateDialogue is a bool that if set to true, the Npc will automatically start a conversation with the player when he is in range (Determined by **Interaction Distance**).

Flipped and **FaceLeft** are bools that control the face direction of the NPC.

Waypoints is a list of Transforms that the NPC will follow in a loop.

RestWaypoint is a waypoint at which the NPC will stop for a certain time and take a rest, before continuing to follow the waypoints.

GfxComponents is a list of SpriteRenderers containing the Head, Body, HandL, HandR, LegL, LegR **in this order**. It is used for animation and sprite flipping. Same goes for **OffHand, HandL, HandR** and **Back**. You can find more on that in **Entity.cs**.



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?

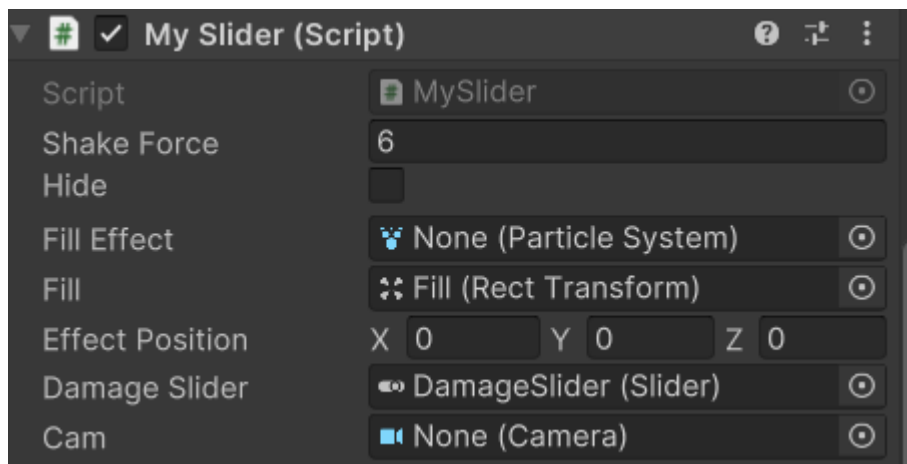
FillEffect is the particle system that gets played when value is being changed.

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**.



LevelManager

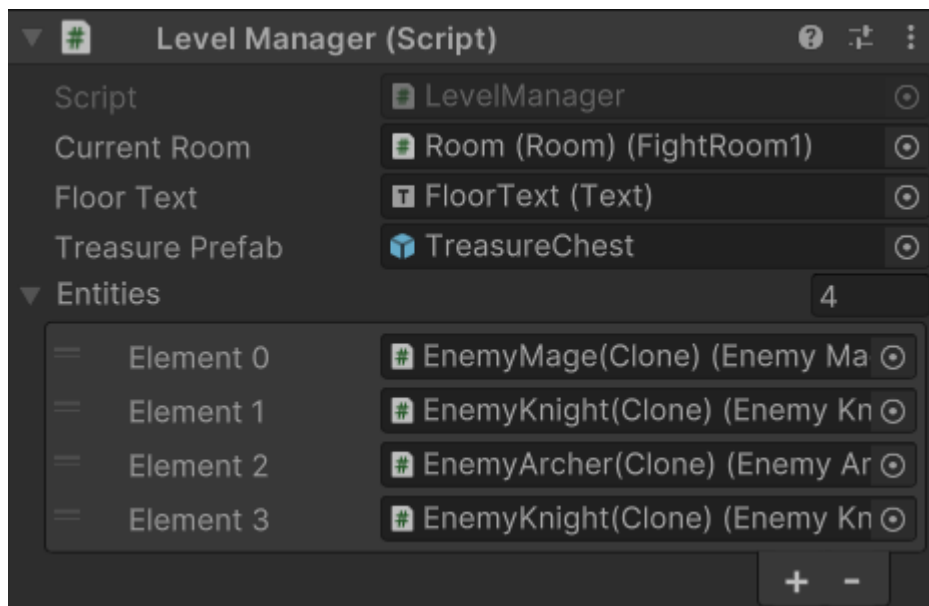
LevelManager.cs is responsible for controlling the current room in the dungeon.

CurrentRoom is automatically found when the scene changes and it's values such as boundaries, room type and chest spawn position are used in LevelManager.

FloorText is an UI Element that displays the **CurrentRoom**'s floor.

TreasurePrefab is a prefab of the treasure chest that gets spawned when the floor is cleared.

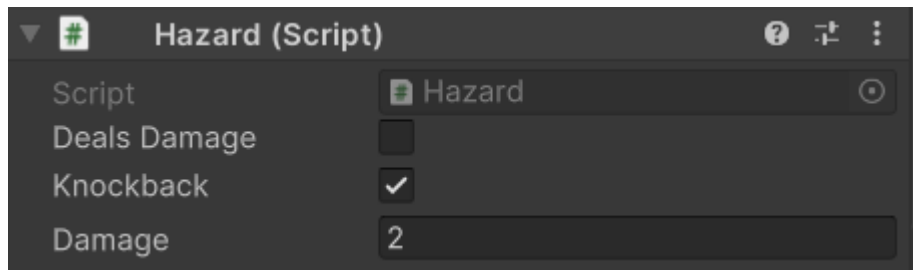
Entities is a list of Enemy classes. When the list is empty and no more waves will be spawned the room is considered cleared.



Hazard

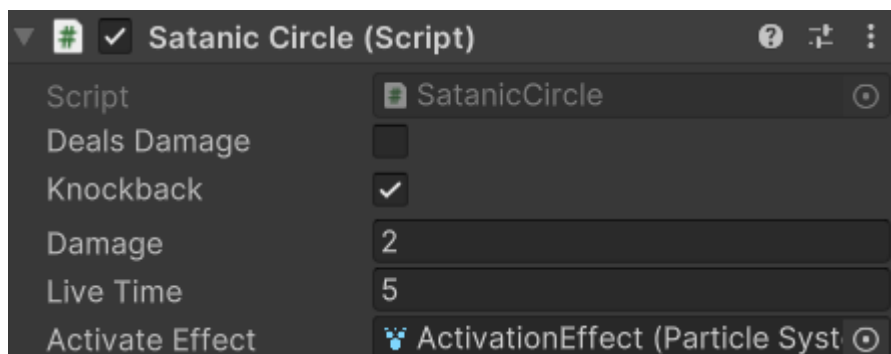
Hazard is a base class which makes an object deal damage upon collision. It requires a Rigidbody2D and a Collider2D to work.

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



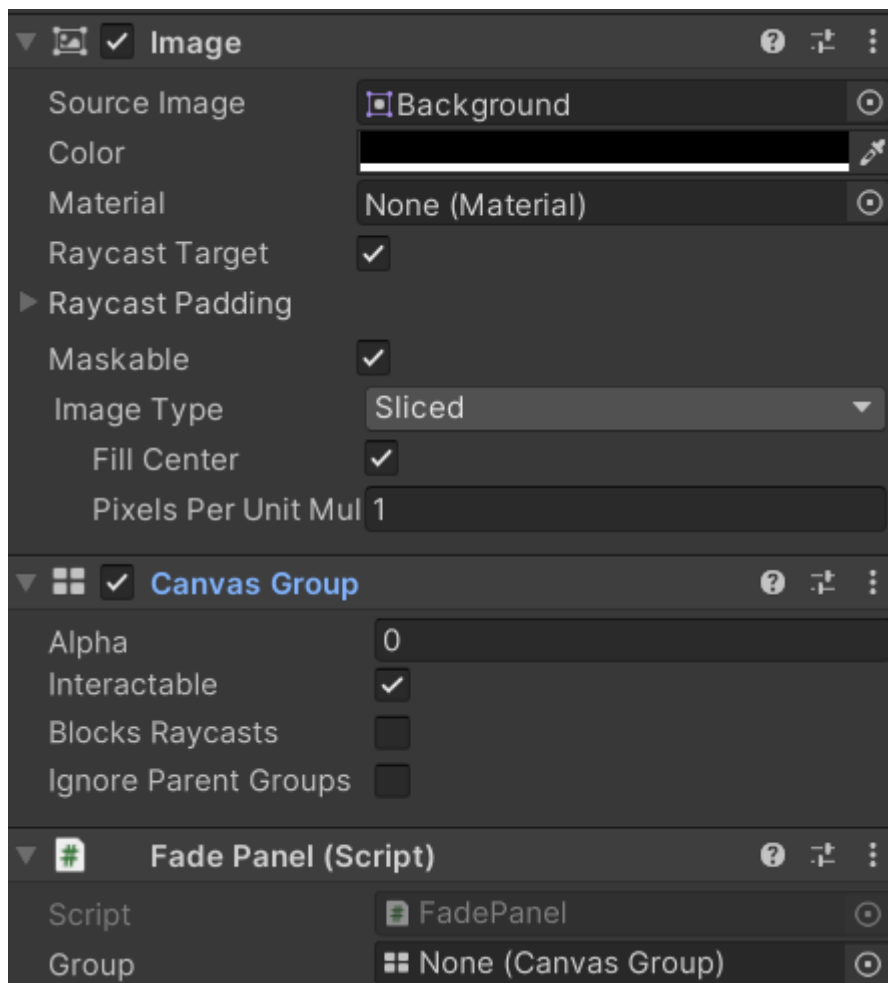
SatanicCircle

SatanicCircle is a subclass of Hazard. The difference is that **DealsDamage** gets set to true after 1 second and an **ActivateEffect** gets played. This gives time for the player to move out of the way. **SatanicCircle** also has a **LiveTime** set to 5, after that it gets destroyed.



FadePanel

FadePanel.cs is responsible for smooth fade to black transitions between scenes. It requires a **CanvasGroup** to work. If you want to learn more, check inside FadePanel.cs.



DungeonEntrance

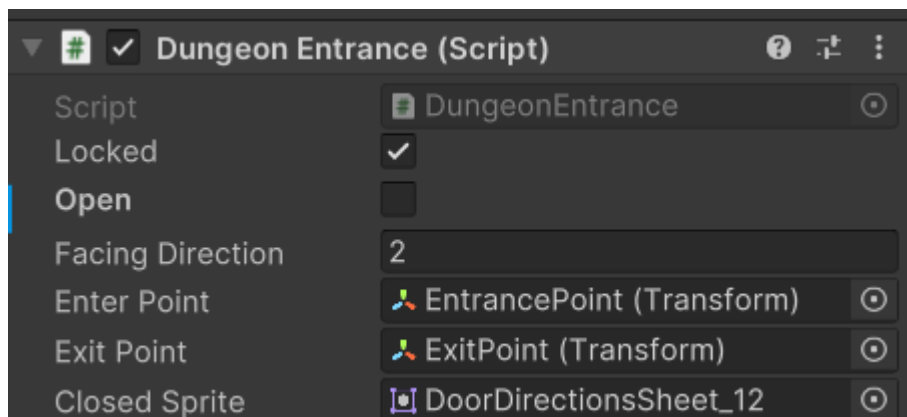
DungeonEntrance.cs is responsible about entering doors.

Locked/Open indicates whether the door is locked/opened.

FacingDirection indicates which direction the door is facing. 0 = down, 1 = left, 2 = up, 3 = right

EnterPoint and **ExitPoint** are used when the player enters/exits the door. When the player wants to enter, the character automatically walks towards the EnterPoint. When the scene changes the character automatically walks to the ExitPoint.

ClosedSprite is a sprite of the door when it's closed. It is used when the door gets closed.

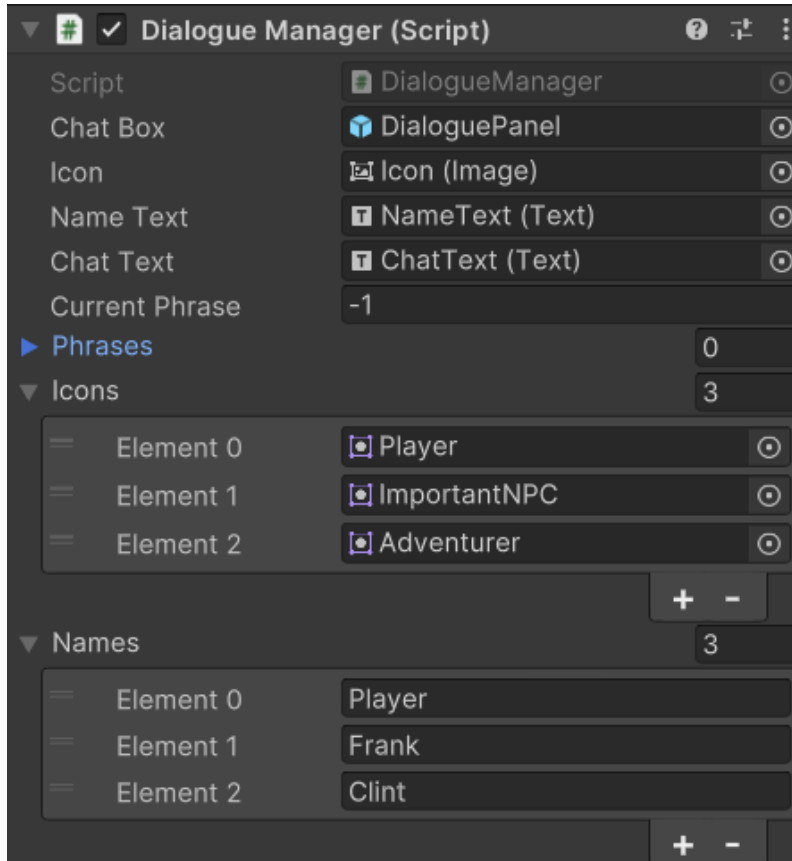


DialogueManager

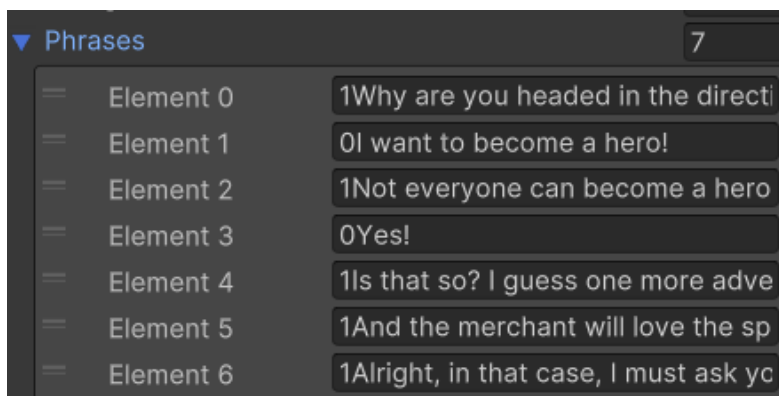
DialogueManager.cs has the entire logic for dialogues in the game.

ChatBox is a reference to the UI DialoguePanel gameObject, **Icon** is a reference to the UI Icon gameObject, and the same goes for **NameText** and **ChatText**.

CurrentPhrase is the index of the **Phrases** string list



Icons is a **Sprite** array of the icons of characters. Everytime a phrase is displayed, the icon updates based on which character said the line. To indicate which character is talking insert the **Index** of the icon before the phrase like so:



The **Index** will NOT be displayed in the phrase and is used only for differentiation between character icons and names. Make sure the **Icons** array matches the **Names** array in terms of characters.

ClassCard

ClassCard.cs is responsible for giving the player a class. There are 3 classes – Knight, Archer, Mage. You can click on the card to flip it and reveal more information about said class and click the button “choose” to select the class.

ClassIndex is the type of class (0 = Knight, 1 = Archer, 2 = Mage)

ClassBaseStats is a reference to the ScriptableObject **StatsUp**, which has the base stats of the class (such as health, damage, speed, etc.)

StatsTexts are UI Text elements that are at the back of the card.

Position is an index relative to the position of the card relative to the **CardNeighbours**. If the card is to the left of the screen the position = -1, to the right = 1, center = 0;

ChosenEffect is the effect that plays when the card is selected by the player.

WeaponPrefab is the weapon the player will receive when choosing that class. **ShieldPrefab** is only assigned for the Knight class, as he spawns with a sword and shield.

NeighbourCards is an array of cards that are next to the current card. In case the current card is selected, the **NeighbourCards** will be sent off screen.

Front and **Back** are parents of UI Elements that hold all the information for the front or back of the card. They get displayed based on whether the card is **flipped** or not.

Contents is a parent of everything visual about the card. It is used for flipping the whole card.

