Environ Brief/Proposal

PURPOSE

1. To simulate environmental effects that can impact the world around them in a meaningful way.
2. To be a simple, easy way to control how objects interact with each other as part of an environment.

GOALS

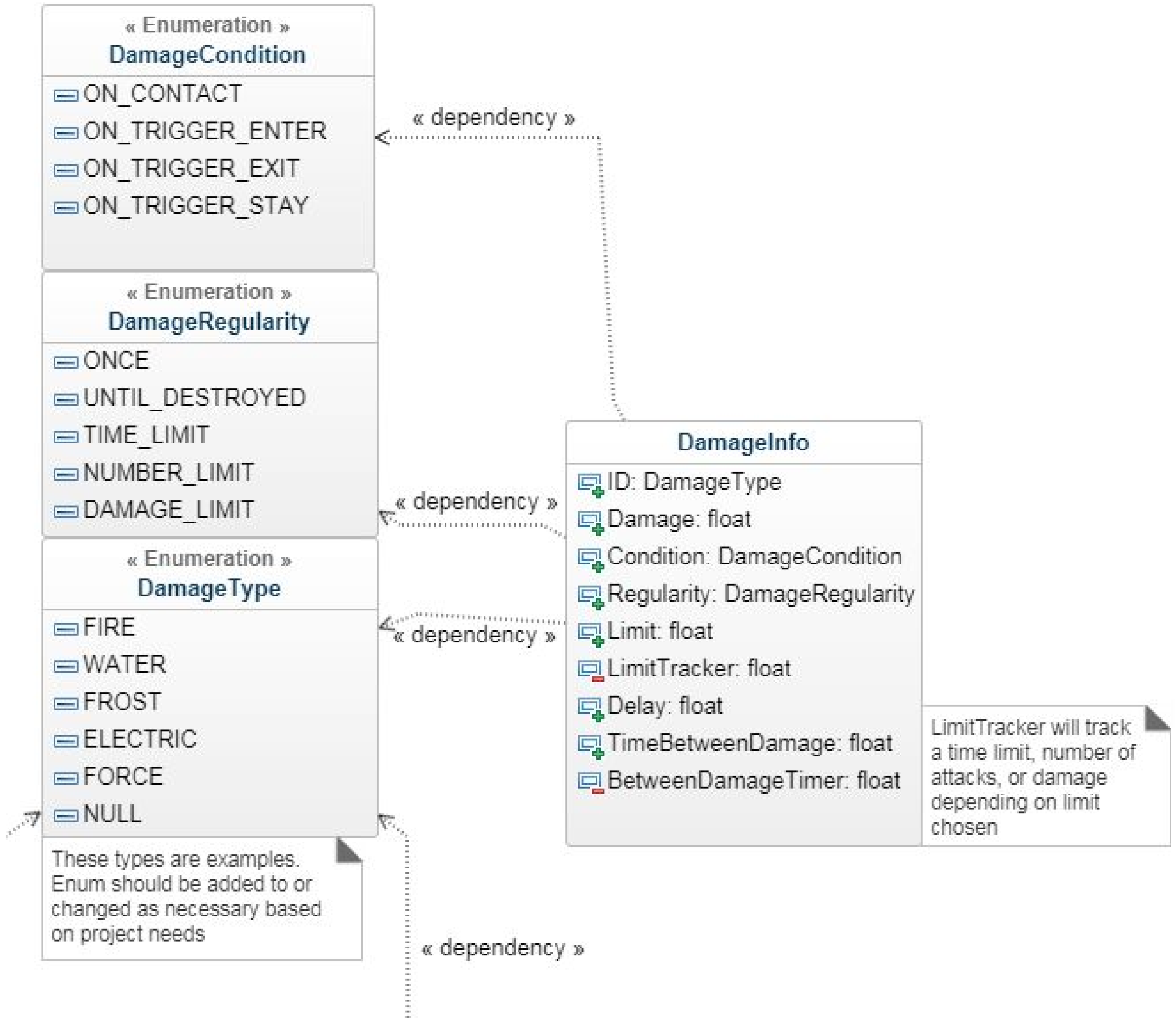
1. Allow creation of properties (such as damage information, appearance information, etc) as Scriptable Objects for ease of use and reusability.
2. Have an “Input” and an “Output”: a way to effect the environment around the object and have the object be affected respectively. Inputs should take information from Outputs.
3. Allow control of damage given and taken and destruction of the object and surrounding objects, texture of the object and surrounding objects at minimum.

The resulting project should allow a high degree of user customization.

STRUCTURE

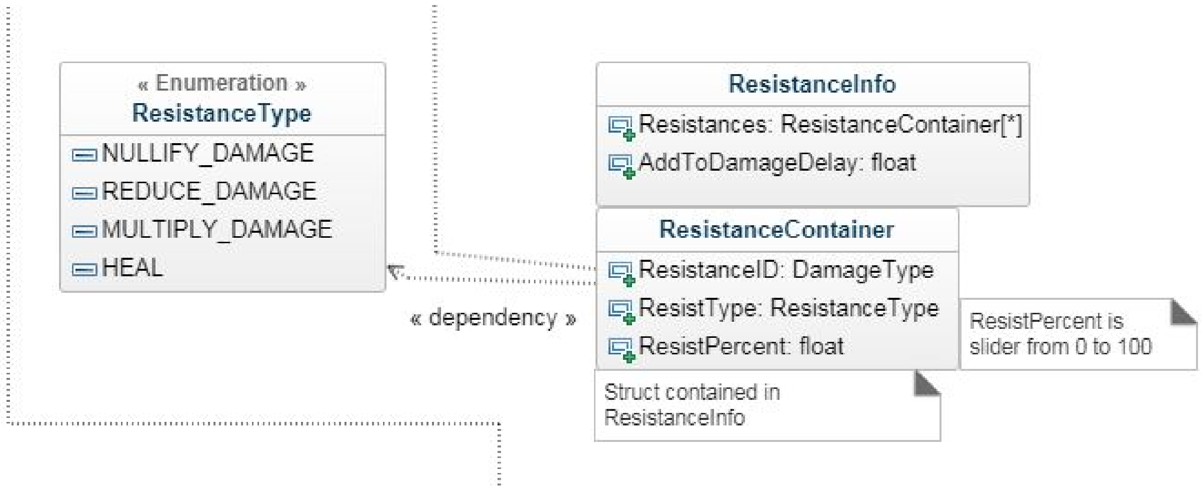
The following UML diagrams are a rough prototype meant to serve as an example and is subject to change, though the general function of each element will still adhere to the stated goals.

DamageInfo:



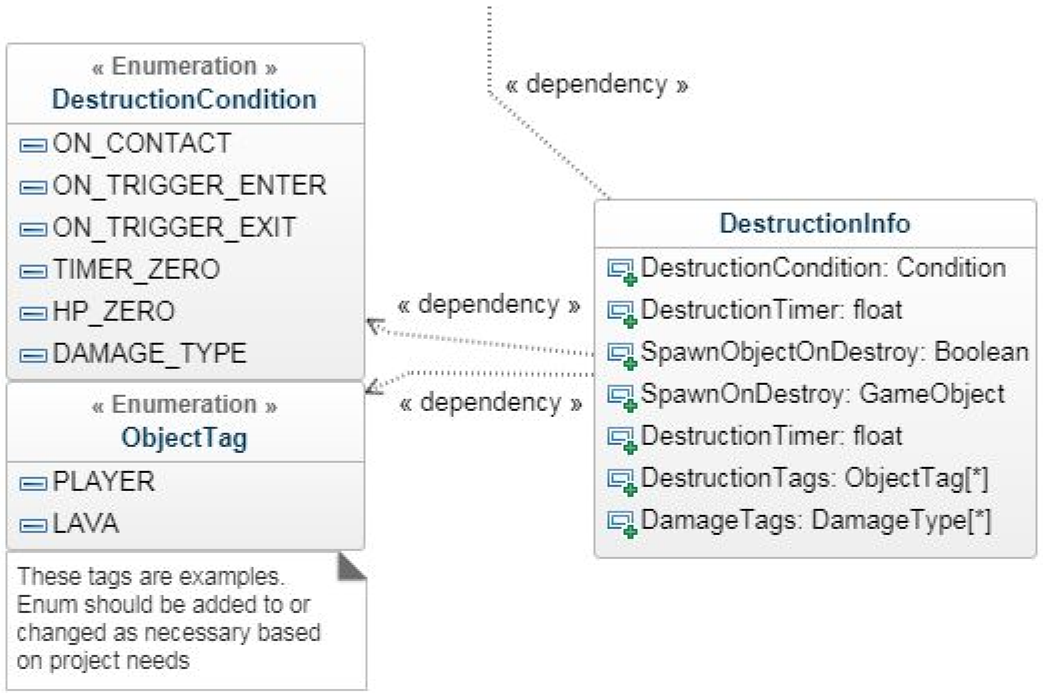
DamageInfo should contain a damage value, an ID for the type of damage it produces, an option to set whether colliders or triggers are used to detect other Environ Objects, and configurable regularity for the damage (as we don't want to apply damage on every update, usually). This regularity must include, at minimum, apply once, apply until object is destroyed, and apply for a user defined period of time. Additionally, a delay timer to delay when damage can be applied would be useful for customization, especially in the case of players, animals, etc drowning in a body of water or losing oxygen in some form.

ResistanceInfo:



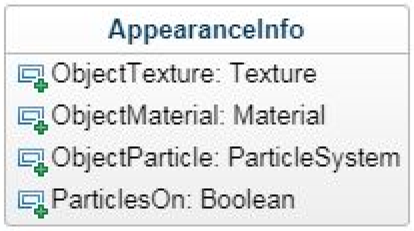
ResistanceInfo should allow the player to set resistance levels for as many damage IDs as they wish. No resistance values for a damage ID should result in DamageInfo being used unmodified. Resistance should contain types, at minimum for damage reduction and damage nullification, and have some way to alter the degree of resistance. Providing a float to add to Damage Delay as part of resistance would also be useful.

DestructionInfo:



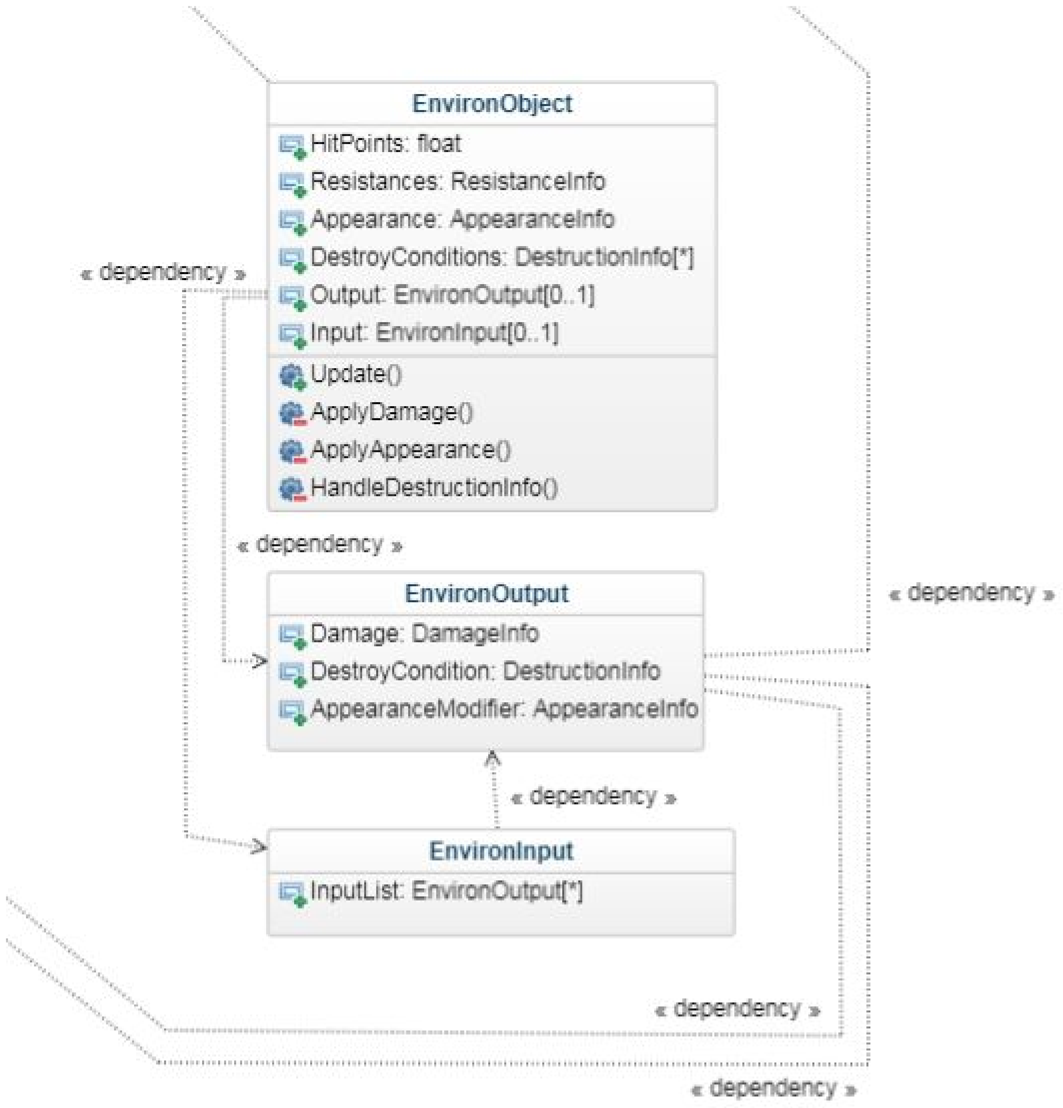
DestructionInfo should contain a way for the user to control if, when and why an Environ Object is destroyed, as well as an option for a GameObject or prefab to spawn before destruction. Conditions for destruction should at minimum be in response to a damage ID, and on HitPoints reaching 0.

AppearanceInfo:



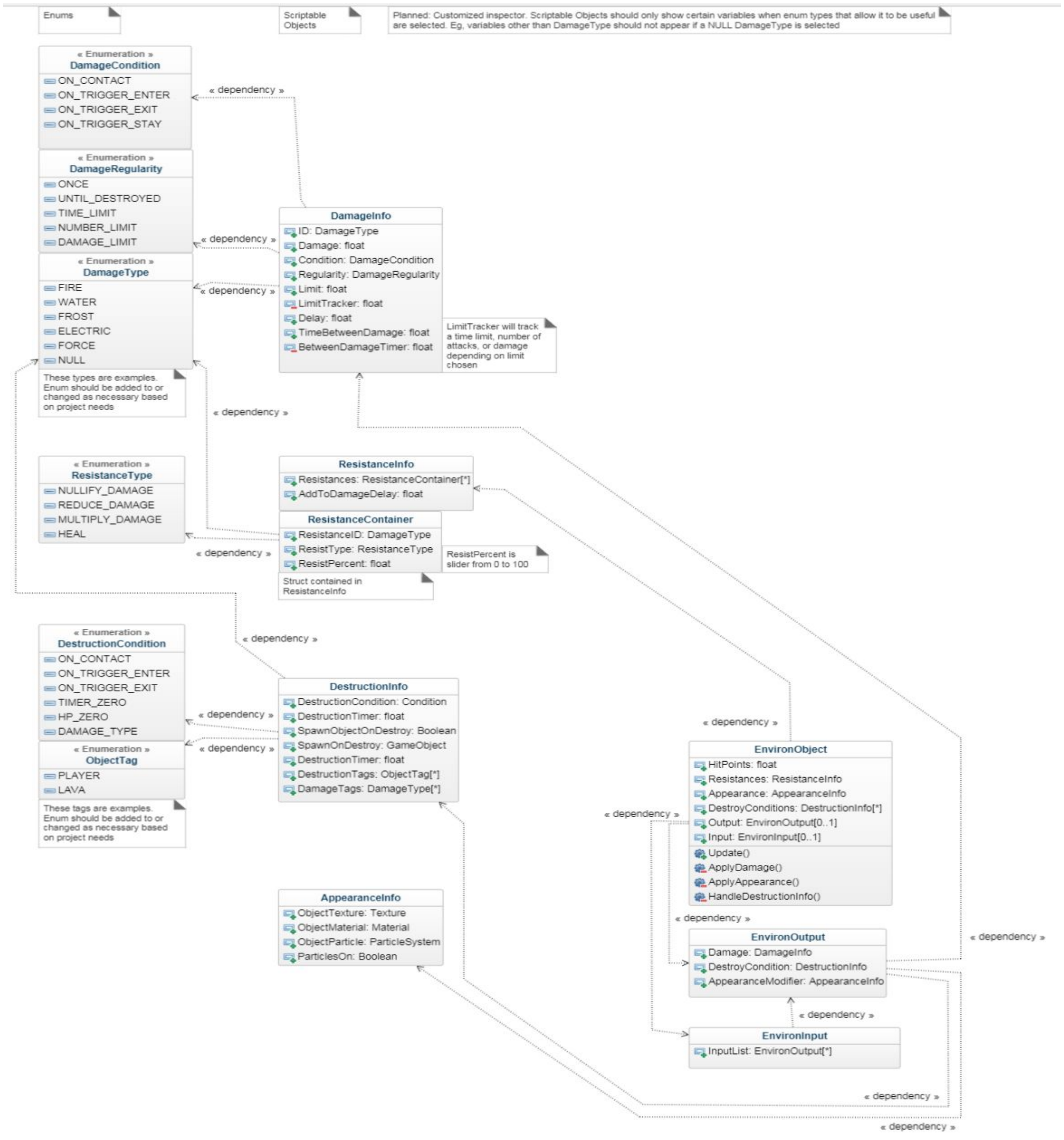
AppearanceInfo should be quite simple- it should contain any textures, normals, materials, particles, and any other variable that would influence how an Environ Object looks. Two should be used in the main Environ Object class- one as an output, to be applied to another Environ Object, and another as a personal version that defines what the Environ Object the script is on will look like. Any particle effects should have a way to toggle them on or off, to allow scripts outside Environ to alter appearance to a greater extent.

EnvironObject, Input, Output:


EnvironObject
HitPoints: float
Resistances: ResistanceInfo
Appearance: AppearanceInfo
DestroyConditions: DestructionInfo[*]
Output: EnvironOutput[0..1]
Input: EnvironInput[0..1]
Update()
ApplyDamage()
ApplyAppearance()
HandleDestructionInfo()
EnvironOutput
Damage: DamageInfo
DestroyCondition: DestructionInfo
AppearanceModifier: AppearanceInfo
EnvironInput
InputList: EnvironOutput[*]
« dependency »
« dependency »
« dependency »
« dependency »
« dependency »
« dependency »

EnvironObject should contain all the previous features, alongside a float for HitPoints (which could be improved with a “Max HitPoints” public variable for the inspector, and another for the current HitPoints, given a Heal Resistance Type is planned). As ResistanceInfo can already contain an array of Resistances, we will only need one. We should also have only one personal current appearance. To allow multiple conditions for destroying the Environ Object, DestructionInfo should be an array. This class should contain functions that update damage, appearance and destruction from both personal variables and Inputs. Inputs should simply be an array (or for ease of use, a List) of Outputs. Output should contain only the information necessary to change and affect other objects, in this case DamageInfo, DestructionInfo and AppearanceInfo for the other Environ Object.

Full UML:


Enums
Scriptable Objects
Planned: Customized inspector. Scriptable Objects should only show certain variables when enum types that allow it to be useful are selected. Eg, variables other than DamageType should not appear if a NULL DamageType is selected
« Enumeration »
DamageCondition
ON_CONTACT
ON_TRIGGER_ENTER
ON_TRIGGER_EXIT
ON_TRIGGER_STAY
« Enumeration »
DamageRegularity
ONCE
UNTIL_DESTROYED
TIME_LIMIT
NUMBER_LIMIT
DAMAGE_LIMIT
« Enumeration »
DamageType
FIRE
WATER
FROST
ELECTRIC
FORCE
NULL
These types are examples. Enum should be added to or changed as necessary based on project needs
DamageInfo
ID: DamageType
Damage: float
Condition: DamageCondition
Regularity: DamageRegularity
Limit: float
LimitTracker: float
Delay: float
TimeBetweenDamage: float
BetweenDamageTimer: float
LimitTracker will track a time limit, number of attacks, or damage depending on limit chosen
« dependency »
« Enumeration »
ResistanceType
NULLIFY_DAMAGE
REDUCE_DAMAGE
MULTIPLY_DAMAGE
HEAL
ResistanceInfo
Resistances: ResistanceContainer[*]
AddToDamageDelay: float
ResistanceContainer
ResistanceID: DamageType
ResistType: ResistanceType
ResistPercent: float
ResistPercent is slider from 0 to 100
Struct contained in ResistanceInfo
« Enumeration »
DestructionCondition
ON_CONTACT
ON_TRIGGER_ENTER
ON_TRIGGER_EXIT
TIMER_ZERO
HP_ZERO
DAMAGE_TYPE
« Enumeration »
ObjectTag
PLAYER
LAVA
These tags are examples. Enum should be added to or changed as necessary based on project needs
DestructionInfo
DestructionCondition: Condition
DestructionTimer: float
SpawnObjectOnDestroy: Boolean
SpawnOnDestroy: GameObject
DestructionTimer: float
DestructionTags: ObjectTag[*]
DamageTags: DamageType[*]
AppearanceInfo
ObjectTexture: Texture
ObjectMaterial: Material
ObjectParticle: ParticleSystem
ParticlesOn: Boolean
EnvironObject
HitPoints: float
Resistances: ResistanceInfo
Appearance: AppearanceInfo
DestroyConditions: DestructionInfo[*]
Output: EnvironOutput[0..1]
Input: EnvironInput[0..1]
Update()
ApplyDamage()
ApplyAppearance()
HandleDestructionInfo()
EnvironOutput
Damage: DamageInfo
DestroyCondition: DestructionInfo
AppearanceModifier: AppearanceInfo
EnvironInput
InputList: EnvironOutput[*]

Notes: Having multiple Input AppearanceInfo could be handled a number of ways, such as only copying the first appearance in the list, allowing a priority variable for each appearance, or using all particle effects together without changing material, texture, etc. This is something that will be looked into further during the project, possibly left up to the user to decide what should be done altogether.

LIBRARIES

This project will be made using Unity and their provided libraries, as well as some of their downloadable assets.

MODULARITY

To use Environ, the end user can configure individual Scriptable Object files that can interact together or separately to create a “material” that can work with or against other Environ Objects using the EnvironObject script. Environ is responsible for handing these objects, if available, to define the interaction it has with the world. Related files can be found under the “EnvironFile” Unity Package.

The Test game will allow the user to pick up with and interact with blocks that contain an EnvironObject script, demonstrating how this system works

As EnvironObject is a collection of Scriptable Objects that define how it behaves, each Scriptable Object should be able to be used by itself on EnvironObject, with the intended function working as expected.

INTENDED USE EXAMPLES

Based on the configuration of an EnvironObject, users could recreate:
-An EnvironObject that simulates fire, by giving its Input to other Environ Objects on contact, damaging it by a base (no resistance configuration) of 1 point every 0.4 seconds, until the other Environ Object has the “fire” effect removed or the Object’s HitPoints reach 0, as well as adding a fire particle effect to the Object for visual representation.
-An EnvironObject that simulates a fire blanket, that cannot have a “fire” Input, and will extinguish the “fire” input on other objects that it collides with.
-An EnvironObject that simulates a body of water, working similarly to the “fire blanket”, and additionally damaging any other Environ Objects that allow it if they stay in the water for a certain period of time, simulating drowning.
-An EnvironObject that simulates a cactus, damaging any other Environ Objects that allow it once on contact, with a 0.5 second cooldown.
-An EnvironObject simulating rain, which would slowly extinguish any “fire” effect either through damage or a timed destruction configuration.

SIMILAR PROJECTS

Interactive Elements Script Solution C#, by Gregory W Lyons
Follows the same basic idea, though limited to a few predefined elements.