Full Metal AI

Technical Design Document

Naming Convention

In code we use **lowerCamelCase** for Assets **UpperCamelCase / PascalCase.**

lowerCamelCase - That means all words written together, first word starts with a not capitalized letter, all other words starts with a capitalized letter.

UpperCamelCase / PascalCase - That means all words written together and the first letter of a word is always capitalized .

This means to be descriptive and re-used. All names are written in English.

To specify an Asset use underscore(_).

Good Example	Bad Examples
Assets	
PlayerRocketLauncher_v1	RocketLauncher01_Blockout
BuildingTriangleUpPart	PartBuildingTriangleUp
ExplosionEnemyDeath	Explosion01
PickUpEffectHeal	HealPickUpEffect

Rule:
→ DescriptiveClassVariation
→ DescriptiveClassSubClassVariation

Good Example	Bad Examples
Scripts	
addDamage	adding_damage_to_player
postQuery	SendSomething
movement	Movement_for_Player
input	Input_for_player
setFocus	give_focus

Version control

General

Version control will be done with Mercurial and TortoiseHg as Graphical interface over the platform BitBucket.

The Repository can be used to push all directly related Project things. A good Structured Repository will improve the teams efficiency.

Commits

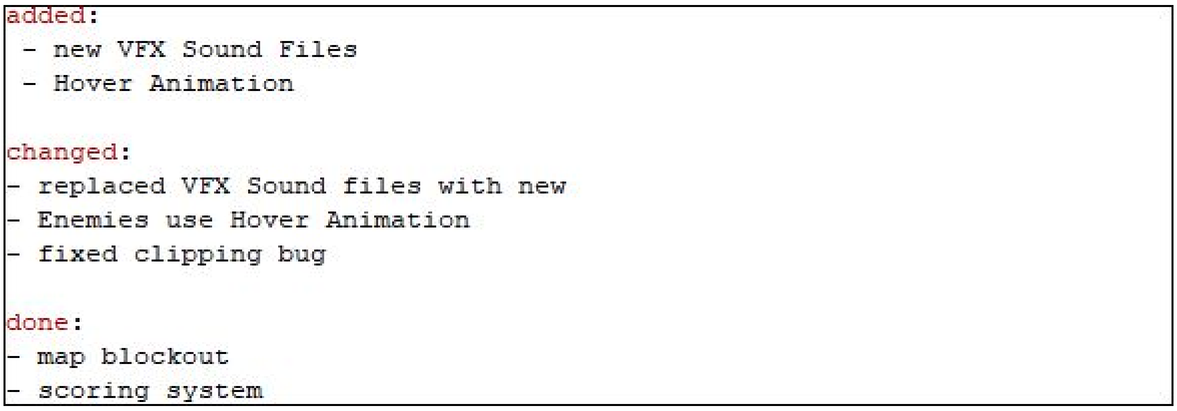
There are certain things you have to follow before you push a Commit !

- **no unnecessary file changes** e.g personal settings , changes to nodes you don't need for this commit,
- consider the correct **Naming Convention**
- use the **correct branch**
- encountering a **merge conflict**, talk to the person / department that is responsible for it

Commit Messages

There are 3 Headlines **added**, **changed** and **done**.

added
Is for things you just put in the Project, things you just started to working on.
changed
For things that maybe changed Frequently, restructuring Folder, renamed Files, Script changes.
done
Everything that you consider as done or completed.



```
added:
 - new VFX Sound Files
 - Hover Animation

changed:
- replaced VFX Sound files with new
- Enemies use Hover Animation
- fixed clipping bug

done:
- map blockout
- scoring system
```

Branches

Branches should be named accordingly to the the feature or department you are working on.
PascalCase is our naming convention.
Example :
VFX
VFXAudio
MapBlockout
ScoreMechanic
Experimental

Branches can be merged if a feature is done and complete, consider to ask for a second opinion to check if your branch is cleaned up before you **merge with the default branch.**

Code Structure

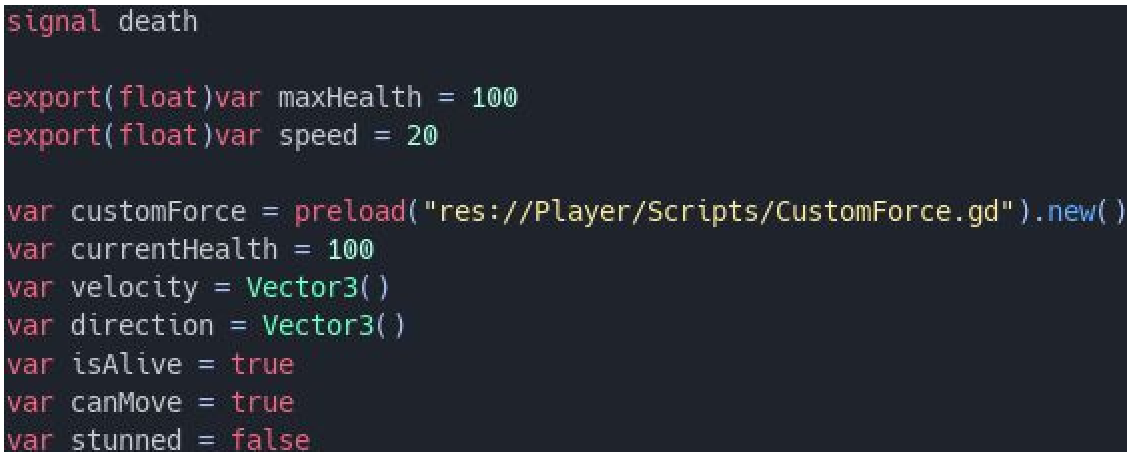
We work with Godot Game Engine Version 3.0.6 and use the built in Scripting Language GDScript which is similar to Python.

Structure

At the beginning of a new Script should always placed the Variables first and then the Functions.

Variables

- Variables should be written in lowerCamelCase
- Variables should be Initialized
- Variables in the following Order
  - signals
  - const variables
  - export variables
  - local variables



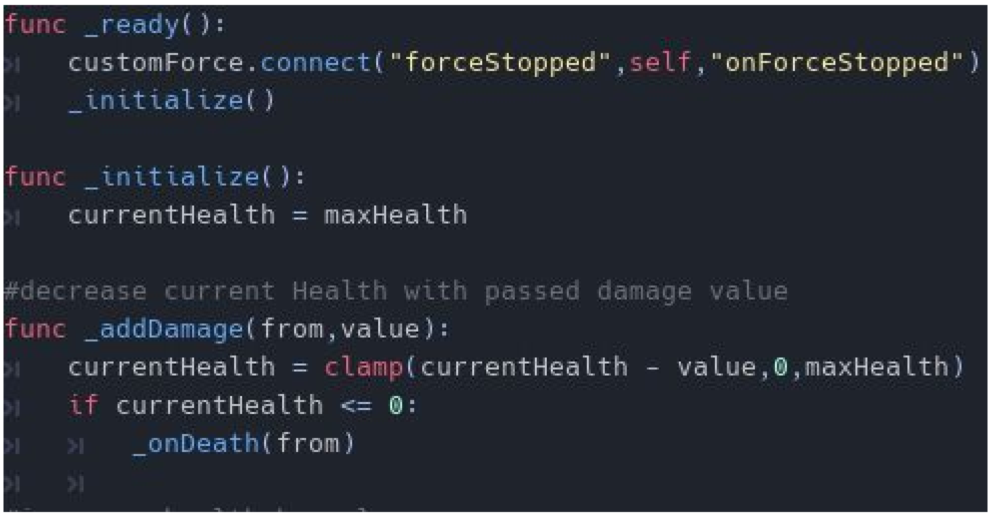
```
signal death

export(float)var maxHealth = 100
export(float)var speed = 20

var customForce = preload("res://Player/Scripts/CustomForce.gd").new()
var currentHealth = 100
var velocity = Vector3()
var direction = Vector3()
var isAlive = true
var canMove = true
var stunned = false
```

Functions

- Functions are written in lowerCamelCase
- Functions which can be overridden starts with a underscore(_)
- Functions don't necessary needs a comment, only if the body do more complex stuff which has to be explained in some way

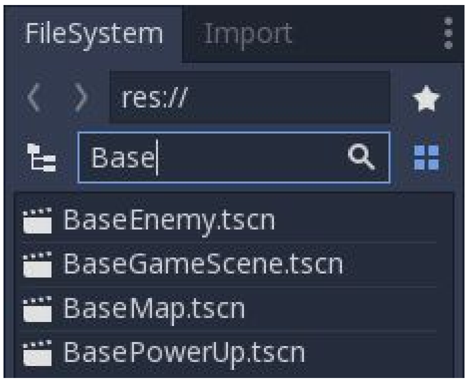


```
func _ready():
	customForce.connect("forceStopped",self,"onForceStopped")
	_initialize()

func _initialize():
	currentHealth = maxHealth

#decrease current Health with passed damage value
func _addDamage(from,value):
	currentHealth = clamp(currentHealth - value,0,maxHealth)
	if currentHealth <= 0:
		_onDeath(from)
```

Scene Structure



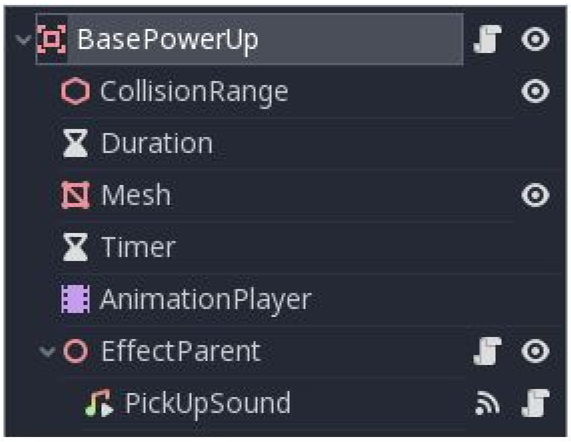
```
FileSystem   Import
res://
Base
BaseEnemy.tscn
BaseGameScene.tscn
BaseMap.tscn
BasePowerUp.tscn
```

We using Inherited Scenes for Game Scenes, Maps, Enemies and Power Ups.

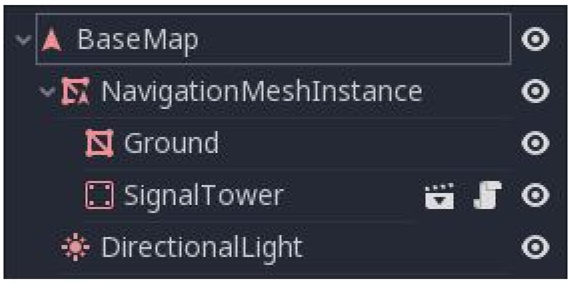
Each of these has a Base Scene, e.g a new Map has to inherit from the BaseMap.

This way we can easily make major changes to one of the Base Scenes and it will affect all inherited Scenes of this kind.

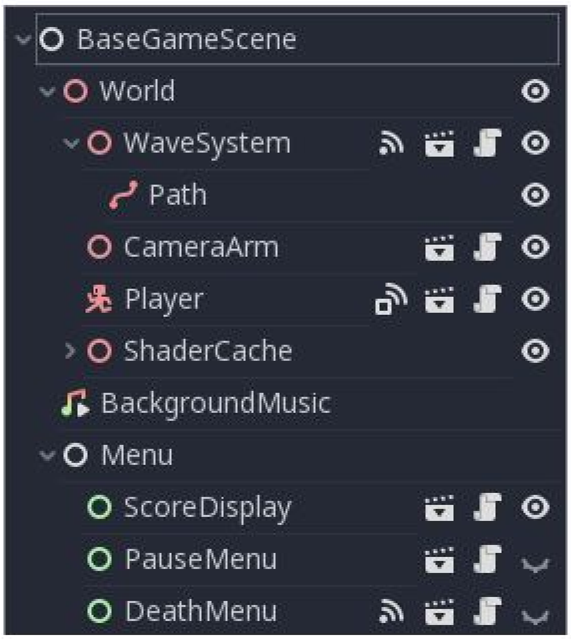
Base Scenes



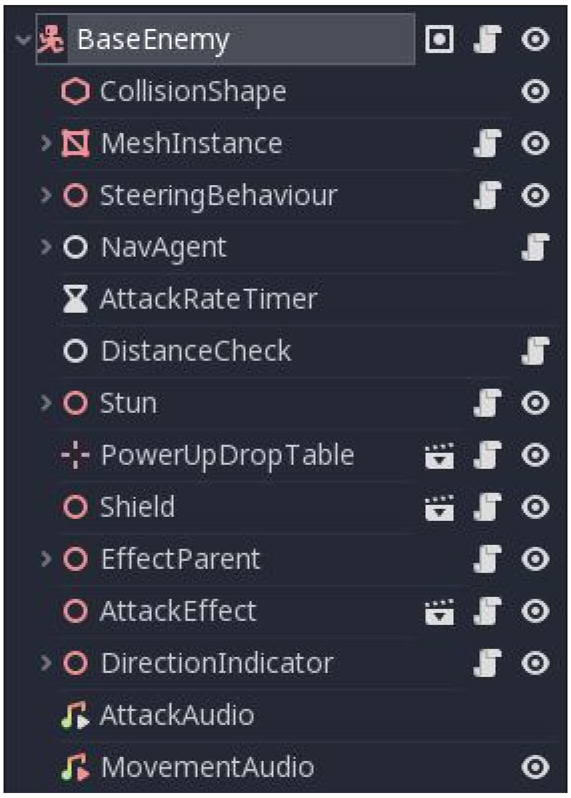
```
BasePowerUp
  CollisionRange
  Duration
  Mesh
  Timer
  AnimationPlayer
  EffectParent
    PickUpSound
```



```
BaseMap
  NavigationMeshInstance
    Ground
    SignalTower
  DirectionalLight
```



```
BaseGameScene
  World
    WaveSystem
      Path
    CameraArm
    Player
    ShaderCache
  BackgroundMusic
  Menu
    ScoreDisplay
    PauseMenu
    DeathMenu
```



```
BaseEnemy
  CollisionShape
  MeshInstance
  SteeringBehaviour
  NavAgent
  AttackRateTimer
  DistanceCheck
  Stun
  PowerUpDropTable
  Shield
  EffectParent
  AttackEffect
  DirectionIndicator
  AttackAudio
  MovementAudio
```

LazyLink

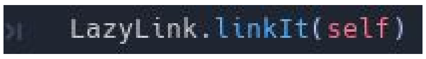
Is a custom *Exporter Script* which saves Node Names, Variable Names and Values in a csv like Format in a Text File. With the *Lazy Link too*l you can import, change and save the File.

This improve the Workflow of Game Balancing and creates new Ways of changing our Game, without working in Godot.

With the created File it is possible to change all Variable Values.

Link a Node

To Link a Node only one Line is Necessarily.

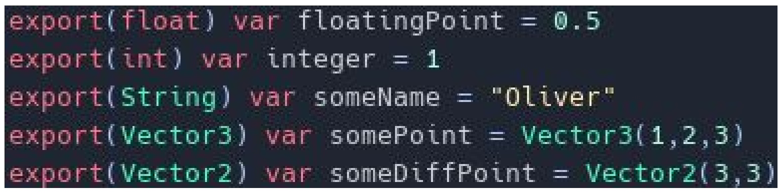


```
LazyLink.linkIt(self)
```

Lazy Link has to be initialized at First.
Add this Line at first in the _ready() Function.
This makes sure that LazyLink can read and write to the Node/Script.

Variables

Lazy Link can only read Variables which match its Conditions.
Only variables marked with **export** with a type of **float** ,**int** ,**String** ,**Vector3** ,**Vector2**.



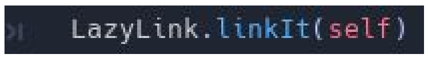
```
export(float) var floatingPoint = 0.5
export(int) var integer = 1
export(String) var someName = "Oliver"
export(Vector3) var somePoint = Vector3(1,2,3)
export(Vector2) var someDiffPoint = Vector2(3,3)
```

Commands

The Commands you have to tell the LazyLink Export Script what to do are fairly simple.
The first Line always should contain the Command.

0 Command

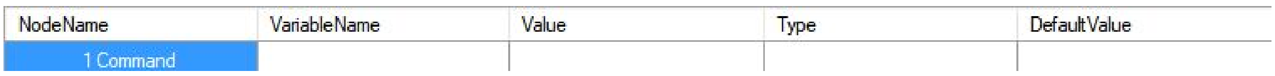
LazyLink saves all Variables from the Node / Script which called this Line.



```
LazyLink.linkIt(self)
```

1 Command

Lazy Link applies the Variables from the Files to the Nodes / Scripts



NodeName	VariableName	Value	Type	DefaultValue
1 Command				

Naming Convention

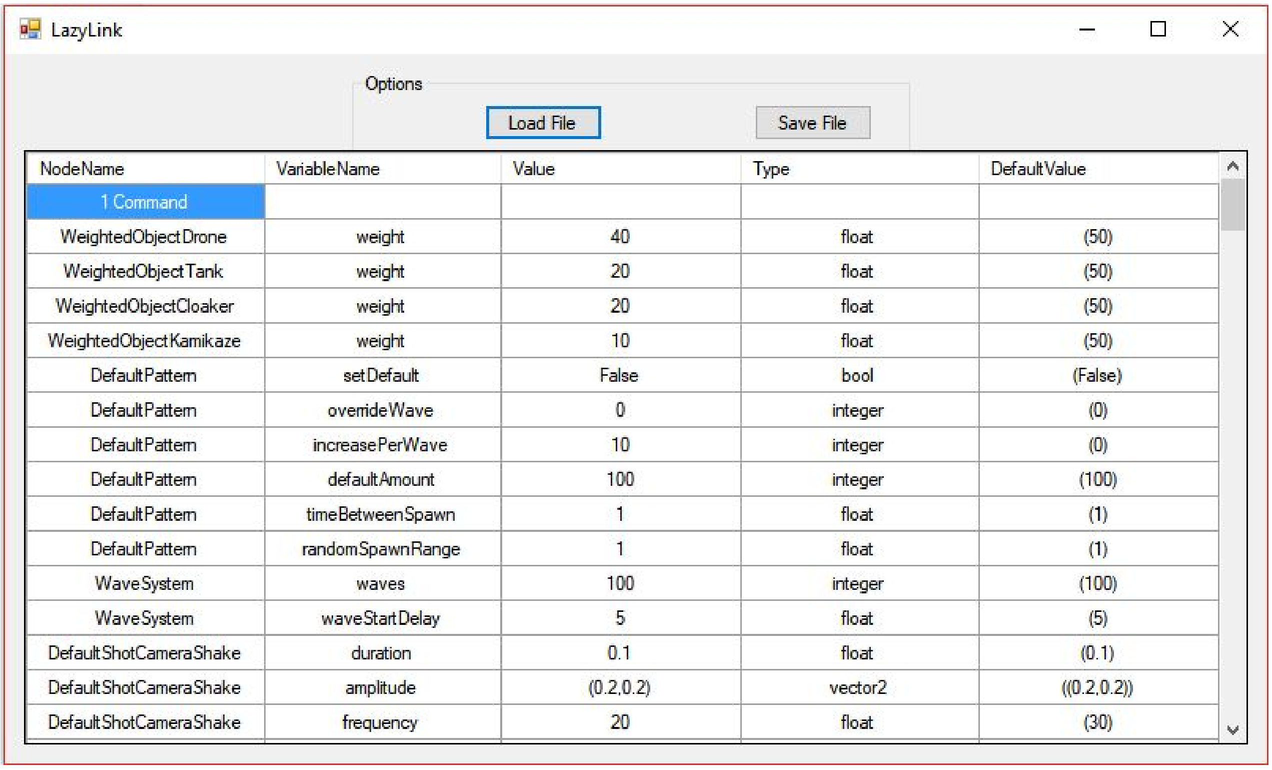
When it comes to Name duplication e.g a duplicated Node Lazy Link can only write / read duplications with match the Following Rules.

Godots runtime duplication Naming Convention and custom Convention for duplication in the Editor.

- @NodeName@123
- NodeName
- NodeName#1
- NodeName#123124

The result what LazyLink sees is NodeName.

Tool



NodeName	VariableName	Value	Type	DefaultValue
1 Command				
WeightedObjectDrone	weight	40	float	(50)
WeightedObjectTank	weight	20	float	(50)
WeightedObjectCloaker	weight	20	float	(50)
WeightedObjectKamikaze	weight	10	float	(50)
DefaultPattern	setDefault	False	bool	(False)
DefaultPattern	overrideWave	0	integer	(0)
DefaultPattern	increasePerWave	10	integer	(0)
DefaultPattern	defaultAmount	100	integer	(100)
DefaultPattern	timeBetweenSpawn	1	float	(1)
DefaultPattern	randomSpawnRange	1	float	(1)
WaveSystem	waves	100	integer	(100)
WaveSystem	waveStartDelay	5	float	(5)
DefaultShotCameraShake	duration	0.1	float	(0.1)
DefaultShotCameraShake	amplitude	(0.2,0.2)	vector2	((0.2,0.2))
DefaultShotCameraShake	frequency	20	float	(30)

Only the Command and the Values should be changed. As a Backup if something mess up you always have the Default Values.