```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CritterBase : MonoBehaviour {
    //Ref
    public GeneManager geneManagerRef;
    public NavigationControllerBase navigationControllerRef;
    public CritterBehaviourController behaviourControllerRef;
    public TileCheckerCritter tileCheckerRef;

    [Header("Info")]
    public string Species = "Critter";
    public Enums.CritterClass critterClass;
    int ID;
    public string identifier = "None";
    public int generationNr = 0;
    public Enums.Gender gender; //Sets gender through the matting Gene. If Na then Randomly Assign male or female Gender
    public Enums.MatingTypes matingType;

    [Header("Current Habitat")]
    public TileBase bornTile;
    public Enums.Habitat current_habitat;
    public Enums.SubHabitat current_subHabitat;

    //Habitat
    float habitat_Water_FluidAddPerc = 0.1f; //Amount of fluid is added if creature is in water
    float habitat_Water_FluidAdd;

    [Header("Base Values")]
    public float base_health = 100;
    public float base_energy = 100;
    public float base_fluid = 100;
    public float base_gas = 100;
    public float base_lifespan = 120;
    public float base_size = 1;
    public float base_strenght = 100;
    public float base_armour = 0;
```

```
    public float base_sight = 2;
    public float base_poop = 100;
    public Vector2 base_temperatureRange = new Vector2(-5, 30);


    [Header("Total Values")]
    public float total_health; //Decreases on attack, onBeingCold, onBeingHot, onNoEnergy,onNoFluid. Increases overtime if non
of the decrease is true
    public float total_energy;
    public float total_fluid;
    public float total_gas;
    public float total_lifespan;
    public float total_size;
    public float current_size;
    public float total_strenght;
    public float total_armour; //Reduces damage done on attack by this pergentage (AttackDamage -= (AttackDamage*armour)
    public float total_sight; //The distance a critter can see
    public float max_poop; //MaxPoop
    public Vector2 temperatureRange; //Min and Max temprature. Critter gets damage if below or above range


    //MinMaxValues
    Vector2 minMax_Size = new Vector2 (0.1f,4);
    float max_armour = 0.8f; //Safty

    //Lessen these numbers to lessen the effect Size has on the varriables (Or up them to have size more effect
    float sizeEffect_health = 1.5f;
    float sizeEffect_energy = 1f;
    float sizeEffect_fluid = 1;
    float sizeEffect_gas = 1;
    float sizeEffect_lifespan = 1;
    float sizeEffect_strenght = 1;
    float sizeEffect_poop = 1;

    [Header("Current Values")]
    public float currentHealth;
    public float currentEnergy;
    public float currentFluid;
```

```
public float currentGas;
public float currentPoop;

//Sets certain stats
[HideInInspector]
public float energyBeforeHunger = 0.51f; //The % needed in order to activate mating behaviour (Sets status_canMate)
[HideInInspector]
public float fluidBeforeThirsty = 0.51f; //(Sets status_canMate)
[HideInInspector]
public float gasBeforeWheeze = 0.7f; //(Sets status_canMate)

[Header("Add Remove Overtime")]
//Health Decrease/Increase
public float decrease_HP;
float HP_increaseOverTime = 0.01f;
public float addRemove_energy;
public float addRemove_fluids;
public float addRemove_gas;


[Header("Age")]
public float actualLifespan;
public float currentAge;
float juvenileSizeMulti = 0.5f;
public float growUpTimer;
public float currentGrowUpTimer;
float growUpOfAgePercentage = 0.1f;

[Header("Other Energy")]
public float current_growUpEnergy; //Used to grow up. Once grown up the energy is added to the totalEnergy
public float req_growUpEnergy = 0.25f; //Value given the the energy the critter needs of the juvinile total_Energy
float maxEatAmountPerc = 0.25f; //Sets maxEatAmount (total_energy * (maxEatAmountPerc * eatEffectiveness);)
public float eatEffectiveness = 1; //Dictates how effective eating is in one bite
public float maxEatAmount;
float eatAmountForOneSec = 25; //This is how much a creature needs to eat for it to take 1 sec of waiting time
public float eatTimer;
public float currentEatTimer;
```

```
    [Header("Other Fluid")]
    public float drinkEffectiveness = 1; //Dictates how effective drinking is while in water


    [Header("Mating Spore")]
    public bool mating_spore_canItSelfInseminate = true; //If it's own spore can create offspring
    public bool mating_canReleaseSpore; //After releasing spores the wait time before able to release again is longer then
just canMate time
    public float releaseSporeTimer;
    public float currentReleaseSporeTime;

    [Header("Mating Upclose")]
    public bool mating_upClose; //Can it be impregnated with mating

    [Header("Mating General")]
    public int totalChildrenInLifeTime; //Does not take into account for male if female dies before giving birth
    public int childrenToBeBirth; //How many children will be birth
    public float matingTimer;
    public float currentMatingTimer;
    public float matingOfAgePercentage = 0.1f; //After grown up
    public Vector2 minMaxChildAmount = new Vector2(1, 1);
    bool clonedSelf; //Needed to know if critter cloned itself or if gave birth
    List<PregnancyGenes> pregnancyGeneList; // used to save the genes of mother and father at mating but before birth
    public float isOldWhenReach; //This is calculated with: matingTimer + pregnancytimer + birthTimer. And does not allow
females to mate again

    //Mating costs (How much of a resource will a critter give upon mating (In %)) - VALUE SET IN MATINGGENE
    public float matingEnergyPercentage = 0.5f;
    public float matingFluidPercentage = 0.5f;
    public float matingGasPercentage = 0.5f;

    public float sporeEnergyPercentage = 0.25f;
    public float sporeFluidPercentage = 0.25f;
    public float sporeGasPercentage = 0.25f;

    //Is the cost of mating of both mother + fater / child amount
    float energyMatingCost;
    float fluidMatingCost;
```

```
float gasMatingCost;
float energyGivenToEachChild;
float fluidGivenToEachChild;
float gasGivenToEachChild;

[Header("Mating and Pregnancy timers")]
public float matingWaitTimePercentage; //The persentage the critter will have to wait of the lifetime of a char.
public float matingWaitTime; //How long mating takes (Cannot move)
public float currentMatingWaitTime;
public float pregnantTimePercentage; //The persentage the critter will will be pregnent before giving birth (Of Lifetime)
public float pregnantTime; //The actual PregnantTime
public float currentPregnantTime; //Current pregnant Time
public float birthWaitTimePercentage; //The persentage the critter will have to wait of the lifetime of a char.
public float birthWaitTime; //How long giving Birth takes (Cannot move)
public float currentBirthWaitTime;
public float eggHatchPercentage; //How long it takes to hatch a egg compaired to the lifespan
public float eggHatchTime;
public float currentEggHatchTime;


[Header("Death")]
public bool status_dying;
public bool status_dead;
public bool status_rotting;
bool dyingLogicExecuted;

//Dying Timer
public float dyingTimer = 1;
public float currentDyingTimer;

//Death Timer
public float releaseNutrientTimer = 1;
public float currentReleaseNutrientTimer;
public float rotTimer = 10;
public float currentRotTimer;

[Header("Resource Status")]
public bool status_hungry;
```

```
public bool status_thirsty;
public bool status_wheezing;

[Header("Other Status")]
public bool status_juvenile = true;
public bool status_canGrowUp; //If timer is done
public bool status_isOld;
public bool status_isEgg;
public bool status_smallAgain;
public bool status_canMate;
public bool status_lackOfGas;
public bool status_isChoking;
public bool status_isPregnant;
public bool status_isClone; //If it is preduced by cloning mating

[Header("Behaviour Status")]
public bool hasMovementControl = true; //This is set false when having no legs!
public bool status_canExecuteBehaviours = true;
public bool status_canMove = true;
public bool behaviour_isStationary; //Can never move such as rooted plants
public bool behaviour_isEating;
public bool behaviour_isDrinking;
public bool behaviour_isMating;
public bool behaviour_isGivingBirth;
public bool behaviour_attacking;
public bool behaviour_isBeingAttacked;

[Header("General Behaviour vars")]
public CritterBase target; //A target that can be set for various behaviours
public GameObject gameObjectTarget; //For any non critter targets


//Use this to easily create a autolist of statuses
[System.Serializable]
public struct Status{
    public string statusName;
    public bool statusBool;
    public float decreaseValue;
```

```
    public Enums.StatusEffectsOn statusEffectOn;
}

public Dictionary<string, Status> statusDic = new Dictionary<string, Status>();


//Variation (How much min and max % varriation is allowed
float lifeVariation = 0.05f;
float sizeVariation = 0.05f;
Vector3 startingSize;
float setSizeVariation = -10;


[Header("Timers")]
//Timer before health Regen
public float timerRegenHP = 5;
public float currentTimerRegenHP;

[Header("Traits")]
public bool trait_canEatNutrientObject = true;
public int trait_walk;
public int trait_swim;
public int trait_fly;

[Header("Corpse Values")]
public float corpseLeft; //Meat on a critter when it dies (= to size - 10%)
public float bonesLeft; //For scavigers (10% of total mass)


[Header("Developer")]
public bool isBorn = true;
public bool randomizeGenes = false;
public bool canMutateAtStart = true;


// Use this for initialization
void Start () {
    startingSize = transform.localScale;
```

```
//Ref
SetReference();

//Initialise before genes
geneManagerRef.Initialise();

//Identifier
ID = Global.ReturnNextID();
identifier = Species + "_" + critterClass + "_" + gender + "_" + ID;
gameObject.name = identifier;

//Dictionary
Global.staticGlobal.critterDic.Add(ID, this);

//Set base values
SetBaseValues();

//Execute GeneManager
geneManagerRef.ExecuteGeneSetUp();

//Create StatusList
CreateStatusList();

//Final Safty Checks
SaftyChecks();

//Set the actual current values
SetCurrentValues();

//Initialise after genes
navigationControllerRef.Initialize(tileCheckerRef);
behaviourControllerRef.Initiate(this, navigationControllerRef);
tileCheckerRef.InitialiseCritterTileChecker(this, bornTile);


if (!isBorn) {
```

```
        }

        //TEMP!
        DevLogicStart();
    }



    //Dev Logic
    void DevLogicStart() {

    }

    //============================SET BASE VALUES============================\\
    void SetBaseValues() {
        total_health = base_health;
        total_energy = base_energy;
        total_fluid = base_fluid;
        total_gas = base_gas;
        total_lifespan = base_lifespan;
        total_size = base_size;
        total_strenght= base_strenght;
        total_armour = base_armour;
        max_poop = base_poop;
        temperatureRange = base_temperatureRange;
    }


    //============================VALUES SAFTIES============================\\
    void SaftyChecks() {
        if (total_size < minMax_Size.x) {
            Debug.LogError("(" + gameObject.name + ") Creature has a lower size then the min! ('" + total_size + "' instead of
the min '" + minMax_Size.x + "') Size is set to min.");
            total_size = minMax_Size.x;
        }
        else if (total_size > minMax_Size.y) {
            Debug.LogError("(" + gameObject.name + ") Creature has a higher size then the max! ('" + total_size + "' instead
of the max '" + minMax_Size.y + "') Size is set to max.");
```

```
            total_size = minMax_Size.y;

        }


        if (total_armour > max_armour) {
            Debug.LogError("(" + gameObject.name + ") Creature has a higher armour then the max! ('" + total_armour + "'
instead of the max '" + max_armour + "') Armour is set to max.");
            total_armour = max_armour;
        }
    }

    //============================CALCULATE SIZE VALUES============================\\
    void CalculateSizeValues() {
        float oldMulti;

        if (status_juvenile) {
            total_health *= (current_size * sizeEffect_health);
            total_energy *= (current_size * sizeEffect_energy);
            total_fluid *= (current_size * sizeEffect_fluid);
            total_gas *= (current_size * sizeEffect_gas);
            total_strenght *= (current_size * sizeEffect_strenght);
            max_poop *= (current_size * sizeEffect_poop);

            //Energy required to grow up
            req_growUpEnergy = total_energy * req_growUpEnergy;
        }
        else {
            //Revert full grown multi
            if (status_smallAgain)
                oldMulti = total_size + setSizeVariation;
            else //Revert juvenile or small again multi
                oldMulti = (total_size + setSizeVariation) * juvenileSizeMulti; //Used to reverse the juvenile size calc and
set a new value

            //Revert to base values
            total_health /= (oldMulti * sizeEffect_health);
            total_energy /= (oldMulti * sizeEffect_energy);
```

```
        total_fluid /= (oldMulti * sizeEffect_fluid);
        total_gas /= (oldMulti * sizeEffect_gas);
        total_strenght /= (oldMulti * sizeEffect_strenght);
        max_poop /= (oldMulti * sizeEffect_poop);


        //Set small again values
        total_health *= (current_size * sizeEffect_health);
        total_energy *= (current_size * sizeEffect_energy);
        total_fluid *= (current_size * sizeEffect_fluid);
        total_gas *= (current_size * sizeEffect_gas);
        total_strenght *= (current_size * sizeEffect_strenght);
        max_poop *= (current_size * sizeEffect_poop);
    }
}

//============================SET CURRENT VALUES============================\\
void SetCurrentValues() {
    float sizeMulti = 1;
    float newHealthMulti = 1;
    //float newEnergyMulti = 1;
    float newFluidMulti = 1;
    float newGasMulti = 1;
    float newPoopMulti = 1;


    if (status_juvenile || status_smallAgain) {
        sizeMulti = juvenileSizeMulti;

        //Get the current % of the value compaired to the total
        if (status_smallAgain) {
            newHealthMulti = currentHealth / total_health;
            //newEnergyMulti = currentEnergy / total_energy;
            newFluidMulti = currentFluid / total_fluid;
            newGasMulti = currentGas / total_gas;
            newPoopMulti = currentPoop / max_poop;
        }
    }
```

```
        if (status_smallAgain || !status_juvenile) {
            //For grow up and become small again values
            newHealthMulti = currentHealth / total_health;
            //newEnergyMulti = currentEnergy / total_energy;
            newFluidMulti = currentFluid / total_fluid;
            newGasMulti = currentGas / total_gas;
            newPoopMulti = currentPoop / max_poop;
        }


        //Size
        if (status_juvenile) {
            setSizeVariation = total_size * Random.Range(-sizeVariation, sizeVariation);
        }

        current_size = (total_size + setSizeVariation) * sizeMulti;
        transform.localScale = new Vector3(startingSize.x * current_size, startingSize.y * current_size, startingSize.z *
current_size);
        CalculateSizeValues();

        //Set current Values if is Juvenile
        if (status_juvenile) {
            gameObject.tag = critterClass.ToString();

            //Clone gets the same variable amounts as parent
            if (!status_isClone) {
                currentHealth = total_health;

                if (!isBorn) {
                    currentEnergy = total_energy;
                    currentFluid = total_fluid;
                    currentGas = total_gas;
                }
            }
            currentAge = 0;
            currentPoop = 0;

            //lifeSpan
```

```
            total_lifespan *= (total_size + setSizeVariation) * sizeEffect_lifespan;
            actualLifespan = total_lifespan + (total_lifespan * Random.Range(-lifeVariation, lifeVariation));

            //Grow up and mating
            currentGrowUpTimer = 0; //Is reset in mating for cloning
            growUpTimer = actualLifespan * growUpOfAgePercentage;
            matingTimer = actualLifespan * matingOfAgePercentage;

            //Spore release time is longer then mating time. This is to make sure that the spores are also picked up
            if (matingType == Enums.MatingTypes.SporeEggs || matingType == Enums.MatingTypes.SporeLifeYoung) {
                releaseSporeTimer = matingTimer + (matingTimer / 2);
                currentReleaseSporeTime = releaseSporeTimer / 2;
            }
            currentMatingTimer = matingTimer / 3; //Makes sure the critter can mate quite quickly from

            //Mating and Birth Timers
            matingWaitTime = actualLifespan * matingWaitTimePercentage;
            currentMatingWaitTime = 0;
            pregnantTime = actualLifespan * pregnantTimePercentage;
            currentPregnantTime = 0;
            birthWaitTime = actualLifespan * birthWaitTimePercentage;
            currentBirthWaitTime = 0;

            isOldWhenReach = actualLifespan - (matingWaitTime + pregnantTime + birthWaitTime);

            if (status_isEgg) {
                eggHatchTime = actualLifespan * eggHatchPercentage;
                currentEggHatchTime = 0;
            }

            //Sight temp (Eye Gene needs to affect this)
            total_sight = base_sight;

        }
        else {
            //Makes sure current values are the % of what it was when critter grew up. Aka if health was 25 of the 50 when
grew up and new grown up total health is 100 then new current health is 50
            currentHealth = total_health * newHealthMulti;
```

```
            //currentEnergy = total_energy * newEnergyMulti; //This is disabled so no new energy is ever created by ways of
growing up
            currentFluid = total_fluid * newFluidMulti;
            currentGas = total_gas * newGasMulti;
            currentPoop = max_poop * newPoopMulti;
        }

        //If already in habitat and some logic is linked to it
        if (habitat_Water_FluidAdd != 0) {
            //Habitat Add (First remove what was added, then recalculate and add again)
            addRemove_fluids -= habitat_Water_FluidAdd;
            habitat_Water_FluidAdd = total_fluid * (habitat_Water_FluidAddPerc * drinkEffectiveness);
            addRemove_fluids += habitat_Water_FluidAdd;
        }


        //-----------------------------Call both when adult as well as when juvinile and small-----------------------------\\
        //Action effectiveness
        maxEatAmount = total_energy * (maxEatAmountPerc * eatEffectiveness);

        //Navigation wander size update
        navigationControllerRef.UpdateActualWanderAmount();
    }


    //============================CRITTER STATUSES============================\\
    void CreateStatusList() {
        //Do not change this order!
        AddNewStatus("empty");
        AddNewStatus("status_noEnergy", 0.01f, Enums.StatusEffectsOn.Health);
        AddNewStatus("status_noFluids", 0.02f, Enums.StatusEffectsOn.Health);
        //Gas
        AddNewStatus("status_noGas", 0.10f, Enums.StatusEffectsOn.Health);
        //AddNewStatus("status_lackOfGas"); //Not enough gas in the air so effectiveness goes down
        //AddNewStatus("status_Choking"); //Is being chocked (Also sets noGas value)

        AddNewStatus("status_tooCold", 0.01f, Enums.StatusEffectsOn.Health);
        AddNewStatus("status_tooHot", 0.005f, Enums.StatusEffectsOn.Health);
```

```
    }

    void AddNewStatus(string statusName, float decreaseValue = 0, Enums.StatusEffectsOn statusEffectOn =
Enums.StatusEffectsOn.Na, bool statusBool = false) {
        Status newStatus = new Status();
        newStatus.statusName = statusName;
        newStatus.statusBool = statusBool;
        newStatus.decreaseValue = decreaseValue;
        newStatus.statusEffectOn = statusEffectOn;

        statusDic.Add(statusName, newStatus);
    }



    // Update is called once per frame
    void Update () {
        if (!status_dying) {
            //Cannot Move
            UpdateCannotMove();
            UpdateLifeStateMatingAndPregnancy();
            UpdateIncreaseDecrease();
            OtherStatusesAndBehaviours();
        }
        else if (!status_dead) {
            Dying();
        }
        else {
            Dead();
        }
    }

    void UpdateCannotMove() {
        //Cannot Move
        if (behaviour_isMating || behaviour_isGivingBirth || status_isEgg || behaviour_isEating) {
            StopBehaviours();
        }
        else if (!status_canMove || !status_canExecuteBehaviours) {
```

```
            status_canMove = true;
            status_canExecuteBehaviours = true;
        }
    }

    void HatchingEgg() {

    }

    void OtherStatusesAndBehaviours() {
        //Might want to globalize the stop movement logic with a global timer = Just add to GlobalWaitTIme and if not 0 then
if < 0 set 0 else -= customGameTime. Once done enable movement? (Probably have a status_WaitBehaviourTimer var)
        if (behaviour_isEating) {
            currentEatTimer += Global.customGameTime;

            if (currentEatTimer >= eatTimer)
                behaviour_isEating = false;
        }

    }

    //============================MATING AND PREGNANCY TIMERS============================\\
    void UpdateLifeStateMatingAndPregnancy(){
        if (currentAge >= isOldWhenReach && !status_isOld)
            status_isOld = true;

        //----------------------------Egg----------------------------\\
            if (status_isEgg) {
            if (currentEggHatchTime < eggHatchTime)
                currentEggHatchTime += Global.customGameTime;
            else {
                Hatch();
            }

        }
        else {
            //----------------------------Age----------------------------\\
            currentAge += Global.customGameTime;
```

```
            if (currentAge > actualLifespan) {
                KillCritterByStatus();
                return;
            }

            //----------------------------Growing Up, Mating and Pregnancy----------------------------\\
            //If Critter is juvenile or small again
            if (status_juvenile || status_smallAgain) {
                currentGrowUpTimer += Global.customGameTime;

                if (!status_canGrowUp && currentGrowUpTimer >= growUpTimer) {
                    status_canGrowUp = true;
                }
            }
            else {
                //If Critter has canMate on false
                if (!status_canMate) {
                    if (currentMatingTimer < matingTimer)
                        currentMatingTimer += Global.customGameTime;
                    else {
                        if (!behaviour_isMating && !status_isPregnant && !status_hungry && !status_thirsty &&
!status_wheezing) {
                            status_canMate = true;
                        }
                    }
                }
                //If Critter has canMate on true
                else {
                    if (behaviour_isMating || status_hungry || status_thirsty || status_wheezing) {
                        status_canMate = false;
                    }
                }

                //If Critter is mating
                if (behaviour_isMating) { //If critter is mating have a wait time so it doesn't move
                    if (currentMatingWaitTime < matingWaitTime)
                        currentMatingWaitTime += Global.customGameTime;
```

```
                    else
                        behaviour_isMating = false;
                }
                //If Critter is pregnant
                else if (status_isPregnant) { //If Critter is pregnant check when should give birth
                    if (currentPregnantTime < pregnantTime)
                        currentPregnantTime += Global.customGameTime;
                    else {
                        behaviour_isGivingBirth = true; //Pregnant timer done now give birth

                        if (currentBirthWaitTime < birthWaitTime) //Wait time before the critter actually gives birth (Cannot
move)
                            currentBirthWaitTime += Global.customGameTime;
                        else
                            GiveBirth(); //Wait time over now give birth
                    }
                }

            }
        }
    }


    //============================INCREASE/DECREASE============================\\
    void UpdateIncreaseDecrease() {
        if (!status_isEgg) { //As an egg it does not take any energy, fluid or gas
            //----------------------------Energy----------------------------\\
            if (currentEnergy > 0) {
                if (GetStatus("status_noEnergy").statusBool) {
                    DisableStatusEffect("status_noEnergy");
                }

                //Update Energy
                currentEnergy = currentEnergy + ((total_energy * addRemove_energy) * Global.customGameTime);

                if (!status_hungry && currentEnergy < total_energy * energyBeforeHunger) {
                    status_hungry = true;
                }
```

```
            else if (status_hungry && currentEnergy >= total_energy * energyBeforeHunger) {
                status_hungry = false;
            }

            //Cap Energy (Remove cap for now
            if (currentEnergy > total_energy) {
                Debug.LogWarning("(" + name + ") Has more energy then the cap. (Energy overcap: " + (currentEnergy -
total_energy) + ") This energy is converted to Nutrients instead.");
                CreateNutrient(currentEnergy - total_energy,false);
                currentEnergy = total_energy;
            }
        }
        else {
            if (!GetStatus("status_noEnergy").statusBool) {
                EnableStatusEffect("status_noEnergy");
            }
        }

        //-----------------------------Poop-----------------------------\\
        //Any energy that is removed is added again to poop so never destroyed
        if (addRemove_energy < 0)
            currentPoop += (total_energy * addRemove_energy) *-1;

        if (currentPoop >= max_poop)
            Poop();


        //-----------------------------Fluids-----------------------------\\

        if (currentFluid > 0) {
            if (GetStatus("status_noFluids").statusBool) {
                DisableStatusEffect("status_noFluids");
            }

            //Update Fluids
            currentFluid = currentFluid + ((total_fluid * addRemove_fluids) * Global.customGameTime);

            if (!status_thirsty && currentFluid < total_fluid * fluidBeforeThirsty) {
```

```
        status_thirsty = true;
    }
    else if (status_thirsty && currentFluid >= total_fluid * fluidBeforeThirsty) {
        status_thirsty = false;
    }

    //Cap Fluids
    if (currentFluid > total_fluid)
        currentFluid = total_fluid;

}
else {
    if (!GetStatus("status_noFluids").statusBool) {
        EnableStatusEffect("status_noFluids");
    }
}

//-----------------------------Gasses-----------------------------\\
//Update Gas
currentGas = currentGas + ((total_gas * addRemove_gas) * Global.customGameTime);

if (currentGas > 0) {
    if (GetStatus("status_noGas").statusBool) {
        DisableStatusEffect("status_noGas");
    }

    if (!status_wheezing && currentGas < total_gas * gasBeforeWheeze) {
        status_wheezing = true;
    }
    else if (status_wheezing && currentGas >= total_gas * gasBeforeWheeze) {
        status_wheezing = false;
    }

    //Cap Gas
    if (currentGas > total_gas)
        currentGas = total_gas;

}
```

```
    else {
        if (currentGas < 0)
            currentGas = 0;

        if (!GetStatus("status_noGas").statusBool) {
            EnableStatusEffect("status_noGas");
        }
    }
}
//----------------------------Temperature----------------------------\\
if (Global.currentTemperature < temperatureRange.x && !GetStatus("status_tooCold").statusBool) {
    EnableStatusEffect("status_tooCold");
}
else if (Global.currentTemperature > temperatureRange.x && GetStatus("status_tooCold").statusBool) {
    DisableStatusEffect("status_tooCold");
}
else if (Global.currentTemperature > temperatureRange.y && !GetStatus("status_tooHot").statusBool) {
    EnableStatusEffect("status_tooHot");
}
else if (Global.currentTemperature < temperatureRange.y && GetStatus("status_tooHot").statusBool) {
    DisableStatusEffect("status_tooHot");
}


//----------------------------Health----------------------------\\
if (currentHealth > 0) {
    //Decrease Health
    if (decrease_HP < 0) {
        currentHealth = currentHealth + ((total_health * decrease_HP) * Global.customGameTime);
        currentTimerRegenHP = 0;
    }

    //Increase health over time
    else if (currentHealth < total_health && decrease_HP == 0) {

        //Check if can regen health
        if (currentTimerRegenHP < timerRegenHP)
            currentTimerRegenHP += Global.customGameTime;
```

```
            else {
                currentHealth = currentHealth + ((total_health * HP_increaseOverTime) * Global.customGameTime);
            }

        }
        //Cap Health
        if (currentHealth > total_health)
            currentHealth = total_health;


        //Reset regen timer
        if (currentTimerRegenHP != 0 && currentHealth == total_health)
            currentTimerRegenHP = 0;

    }
    else {
        KillCritterByStatus();
        return;
    }
}


//============================NEW DECREASE HEALTH============================\\
void DecreaseHP(float decreaseHealthAmount) {
    decrease_HP -= decreaseHealthAmount;
    currentTimerRegenHP = 0;
}

void NoMoreDecreaseHP(float removeDecreaseAmount) {
    decrease_HP += removeDecreaseAmount;
}

//============================SPECIFIC STATUS DECREASE VALUES============================\\
void EnableStatusEffect(string statusKey) {
    Status foundStatus = GetStatus(statusKey);
    foundStatus.statusBool = true;

    if (foundStatus.statusEffectOn == Enums.StatusEffectsOn.Health) {
```

```
            decrease_HP -= foundStatus.decreaseValue;
            currentTimerRegenHP = 0;
        }

        statusDic.Remove(statusKey);
        statusDic.Add(statusKey, foundStatus);
    }

    void DisableStatusEffect(string statusKey) {
        Status foundStatus = GetStatus(statusKey);
        foundStatus.statusBool = false;


        if (foundStatus.statusEffectOn == Enums.StatusEffectsOn.Health) {
            decrease_HP += foundStatus.decreaseValue;
        }

        statusDic.Remove(statusKey);
        statusDic.Add(statusKey, foundStatus);
    }

    //============================RETURN STATUS============================\\
    Status GetStatus(string key) {
        if (statusDic.ContainsKey(key)) {
            return statusDic[key];
        }
        else {
            Debug.LogError("Trying to find: '" + key + "' key in statusDic, but it does not exists. An empty is given now but
this will brake things.");
            return statusDic["empty"];
        }

    }



    //============================DEATH LOGICS============================\\
    public void KillCritterByStatus() {
```

```
    SetDeathValues();

    string diedOf = "'" + gameObject.name + "' died of:";

    //Report type of overtime death
    if (currentAge > actualLifespan) {
        diedOf += " |Old Age|";
    }
    else {
        if (GetStatus("status_noEnergy").statusBool)
            diedOf += " |Hunger|";

        if (GetStatus("status_noFluids").statusBool)
            diedOf += " |Thirst|";
        if (GetStatus("status_noGas").statusBool)
            diedOf += " |Lack of Air|";


    }

    //Death Report
    print(diedOf);
}

public void KillCritterByAttack() {
    SetDeathValues();

    //DeathReport
    print("'" + gameObject.name + "' died of: an Attack");

}

void Dying() {
    if (!dyingLogicExecuted) {
        StopBehaviours();

        //Remove From alive dic and add to dead dic
        Global.staticGlobal.critterDic.Remove(ID);
```

```
        Global.staticGlobal.deadCritterDic.Add(ID,this);

        dyingLogicExecuted = true;




        if (status_juvenile || status_smallAgain) {
            //Add current growup energy to corpse
            currentEnergy += current_growUpEnergy;
        }
        else {
            //Add children energy to corpse if pregnant
            currentEnergy += ReturnPregnancyEnergy();
        }

        corpseLeft = currentEnergy * 0.9f;
        bonesLeft = currentEnergy * 0.1f;
    }

    if (dyingTimer > currentDyingTimer)
        currentDyingTimer += Global.customGameTime;
    else
        status_dead = true;
}

void Dead() {
    //If dies then release all poop
    if (currentPoop > 0) {
        Poop(true);
    }

    if (!status_rotting && currentRotTimer < rotTimer) {
        currentRotTimer += Global.customGameTime;

        if (currentRotTimer >= rotTimer) {
            status_rotting = true;
            print("'" + gameObject.name + "' is rotting.");
```

```
        }
    }
    else if (status_rotting) {
        currentReleaseNutrientTimer += Global.customGameTime;

        if (currentReleaseNutrientTimer >= releaseNutrientTimer) {

            CreateNutrient(GetNutrianValueFromCorpse(NutrientObject.maxNutritionValue), true);
            currentReleaseNutrientTimer = 0;
        }
    }
}

float GetNutrianValueFromCorpse(float maxValue) {
    float newNutrianValue = 0;

    if (corpseLeft != 0) {
        if (maxValue >= corpseLeft) {
            newNutrianValue += corpseLeft;
            corpseLeft = 0; ;
        }
        else {
            corpseLeft -= maxValue;
            newNutrianValue = maxValue;
        }
    }

    if (newNutrianValue < maxValue) {
        if ((maxValue - newNutrianValue) > bonesLeft) {
            newNutrianValue += bonesLeft;
            bonesLeft = 0;
            DisapearBody();
        }
        else {
            bonesLeft -= (maxValue - newNutrianValue);
            newNutrianValue = maxValue;
        }
    }
```

```
        return newNutrianValue;
    }

    //Creates a NutrianObject (NEEDS TO ADD HALF THE NUTRIANS TO TILE. Maybe First add to tile and if tile is full then add an
Nutrian object instead!)
    void CreateNutrient(float newNutrientValue, bool canBeEatenFromStart) {
        print("Create Nutrient should only be called in SpawnManager");

        NutrientObject newNutrientObject;

        //Spawn position
        float maxValueForEach = NutrientObject.maxNutritionValue;

        float leftOverNutrientValue = newNutrientValue;

        //Create new nutrient objects for leftOverNutrientValue
        while (leftOverNutrientValue > 0) {
            //Spawn new nutrient
            newNutrientObject = Instantiate(SpawnManager.staticBirthManager.nutrientObjectPrefab, GetSpawnPostion(),
Quaternion.identity);
            newNutrientObject.canBeEaten = canBeEatenFromStart;

            //Last Bit is given
            if (leftOverNutrientValue <= maxValueForEach) {
                newNutrientObject.SetNutritionValue(leftOverNutrientValue);
                leftOverNutrientValue = 0;
            }
            else {
                //Give max nutrient value
                leftOverNutrientValue -= maxValueForEach;
                newNutrientObject.SetNutritionValue(maxValueForEach);
            }
        }
    }

    void SetDeathValues() {
        status_dying = true;
    }
```

```
void DisapearBody() {
    Global.staticGlobal.deadCritterDic.Remove(ID);
    print("'" + gameObject.name + "' body disapeared.");
    Destroy(gameObject);
}


//==============================ACTIONS==============================\\
public void Eat(float eatAmount) {
    //print("(" + name + ") Eat amount: " + eatAmount);

    if (status_juvenile || status_smallAgain) {
        //If small and hungry then give critter half the energy to grow up and half to energy
        if (status_hungry) {
            currentEnergy += eatAmount / 2;
            current_growUpEnergy += eatAmount / 2;
        }
        //Else add all food value to growUp
        else {
            current_growUpEnergy += eatAmount;
        }

        if (current_growUpEnergy >= req_growUpEnergy && status_canGrowUp) {
            GrowUp();
        }
    }
    else {
        //If adult
        currentEnergy += eatAmount;
    }
}

float GetEatAmount() {
    float eatAmount;

    //If small critter then add growUpEnergy
```

```
    if (status_juvenile || status_smallAgain)
        eatAmount = (total_energy - currentEnergy) + (req_growUpEnergy - current_growUpEnergy);

    //Else just eat amount
    else
        eatAmount = total_energy - currentEnergy;

    //Make sure the bite is not bigger then max eat amount
    if (eatAmount > maxEatAmount)
        eatAmount = maxEatAmountPerc;

    if (eatAmount > 0) {
        behaviour_isEating = true;
        eatTimer = eatAmount / eatAmountForOneSec;
        currentEatTimer = 0;
    }

    return eatAmount;
}

public void Drink(float drinkAmount) {
    currentFluid += drinkAmount;
}

public void Poop(bool releaseBowels = false) {

    //HAVE POOP ONLY ADD TO SOIL SO ONLY PLANTS CAN ACCESS IT AS POOP CREATES MORE ENERGY

    //If creature dies then release all poop that it has
    if (releaseBowels) {
        CreateNutrient(currentPoop, true);
        currentPoop = 0;
        return;
    }
    float poopLeft = currentPoop - max_poop;

    if (poopLeft >= max_poop) {
```

```
            Debug.LogError("(" + gameObject.name + ")  warning poop for some reason is still higher or equal to max poop after pooping. It either ate something really big or something is wrong");
            currentPoop = 0;
        }
        else
            currentPoop = poopLeft;

        //Creates tile or Object Nutrians
        CreateNutrient(max_poop, false);
    }

    void GrowUp() {
        status_juvenile = false;
        status_smallAgain = false;
        status_canGrowUp = false;
        SetCurrentValues();

        currentEnergy += current_growUpEnergy;
        current_growUpEnergy = 0;

        if (currentEnergy > total_energy) {
            Debug.LogError("(" + name + ") currentEnergy is greater then total_Energy after small critter grew up. The energy is automaticly dropped");
        }
    }

    //Used for cloning
    public void BecomeSmall() {
        status_canGrowUp = false;
        currentGrowUpTimer = 0;
        status_smallAgain = true;
        SetCurrentValues();
    }

    //============================EAT FROM CRITTER============================\\
    //Another creature eats the corpse. it returns a food value
    public float EatCorpse(float eatAmount) {
        if (!status_dying) {
```

```
            Debug.LogError("(" + gameObject.name + ") another creature is trying to eat the corpse but the creature is not
dead or dying!");
            return 0;
        }

        float returnAmount = 0;

        if (eatAmount > corpseLeft) {
            returnAmount = corpseLeft;
            corpseLeft = 0; ;
            return returnAmount;
        }
        else {
            corpseLeft -= eatAmount;
            return eatAmount;
        }
    }

    //----------------------------Scavenger Only----------------------------\\
    //If corpse is rotting or only bones then scavengers will try and eat it
    public float EatBones(float eatAmount) {
        if (!status_dying) {
            Debug.LogError("(" + gameObject.name + ") another creature is trying to eat the bones but the creature is not dead
or dying!");
            return 0;
        }

        float returnAmount = 0;

        //If there is still a corpse left then eat part or all of the corpse
        if (corpseLeft != 0) {
            if (eatAmount > corpseLeft) {
                returnAmount += corpseLeft;
                corpseLeft = 0; ;
            }
            else {
                corpseLeft -= eatAmount;
                return eatAmount;
```

```
            }
        }

        //If eat amount not yet full enough then also eat all or part of the bones
        if ((eatAmount - returnAmount) > bonesLeft) {
            returnAmount += bonesLeft;
            bonesLeft = 0;
            DisapearBody();

            return returnAmount;
        }
        else {
            bonesLeft -= (eatAmount - returnAmount);
            return eatAmount;
        }
    }

    //-----------------------------Eat From Plant (Can be alive)-----------------------------\\
    public float EatPlant(float eatAmount) {
        if (critterClass != Enums.CritterClass.Plant) {
            Debug.LogError("(" + gameObject.name + ") a critter tries to eatPlant() but the critter is not a plant!");
            return 0;
        }

        float returnAmount = 0;

        //Kill the plant while eating
        if (eatAmount > currentHealth) {
            returnAmount += currentHealth;
            GainDamage(eatAmount);
            returnAmount += EatCorpse(eatAmount - returnAmount); //If it kills the plant while eating then still get the food
from the corpse
            return returnAmount;
        }
        //Take away some health while eating
        else {
            GainDamage(eatAmount);
            return eatAmount;
```

```
        }
    }
    //============================HATCH EGG============================\\
    void Hatch() {
        status_isEgg = false;

    }


    //============================ON BEING ATTACKED============================\\
    public void GainDamage(float damageAmount) {
        Debug.Log("(" + gameObject.name + ") Has gained '" + (damageAmount - (damageAmount * total_armour)) + "' Dmg (Dmg:'" +
damageAmount + "' armor:'" + total_armour + "')");
        currentHealth -= damageAmount - (damageAmount * total_armour);
        currentTimerRegenHP = 0;

        if (currentHealth <= 0) {
            currentHealth = 0;
            KillCritterByAttack();
        }
    }

    public void ChangeCritterClass(Enums.CritterClass newClass, string changedBy) {
        if (newClass != critterClass) {
            if (isBorn)
                Debug.Log("CritterClass changed from: '" + critterClass + "' to '" + newClass + "' by '" + changedBy + "'");

            critterClass = newClass;

            string oldIdentifier = identifier;

            //Update identifiers
            identifier = Species + "_" + critterClass + "_" + ID;
            gameObject.name = identifier;

            geneManagerRef.UpdateParentToNew(oldIdentifier, identifier);
        }
    }
```

```
void StopBehaviours(bool includeBehaviour = true) {
    if (includeBehaviour)
        status_canExecuteBehaviours = false;

    status_canMove = false;
    navigationControllerRef.ClearMovement();
}


//============================MATTING LOGIC============================\\
//----------------------------Male Targeting Female----------------------------\\
void SetMaleMatingVariables(bool createSpore = false) {
    GeneralMatingLogics(createSpore);
    childrenToBeBirth = Random.Range((int)minMaxChildAmount.x, (int)minMaxChildAmount.y + 1);

    //Set is mating (Disables movement)
    behaviour_isMating = true;

    totalChildrenInLifeTime += childrenToBeBirth;
}

public void MateWithTarget() {
    SetMaleMatingVariables();

    //Child amount
    int childrenSet = 0;

    //General Mating logics
    target.SetFemaleMatingVariables(childrenToBeBirth, energyMatingCost, fluidMatingCost, gasMatingCost);
    clonedSelf = false;


    while (childrenSet < childrenToBeBirth) {
        //Create mating genes
        geneManagerRef.CreateMatingGene();
```

```
            target.GetMatedWith(geneManagerRef.mattingGene, childrenToBeBirth);
            childrenSet++;
        }
    }

    //----------------------------CreateSpore----------------------------\\
    public void CreateMatingSpore() {
        currentReleaseSporeTime = 0;
        mating_canReleaseSpore = false;

        SetMaleMatingVariables(true);

        //Child amount
        int childrenSet = 0;

        //Spawn Spore
        SporeBase spore = Instantiate(SpawnManager.staticBirthManager.sporePrefab, GetSpawnPostion(), Quaternion.identity);

        spore.SetSporeVariables(this, energyMatingCost, fluidMatingCost, gasMatingCost);

        //Add mating genes to spore
        while (childrenSet < childrenToBeBirth) {
            //Create mating genes
            geneManagerRef.CreateMatingGene();

            spore.AddMatingGenes(geneManagerRef.mattingGene);
            childrenSet++;
        }
        spore.GetComponent<TileCheckerBase>().Initialise(tileCheckerRef.currentTile);
    }

    //Called before the actual children are created
    //----------------------------Female Mating Varialbes----------------------------\\
    public void SetFemaleMatingVariables(int childAmount, float fatherEnergyMatingCost, float fatherFluidMatingCost, float
fatherGasMatingCost) {
        behaviour_isMating = true;
        status_isPregnant = true;
        GeneralMatingLogics();
```

```
    //The resources that the father and mother put in devided by the childAmount
    energyGivenToEachChild = (energyMatingCost + fatherEnergyMatingCost) / childAmount;
    fluidGivenToEachChild = (fluidMatingCost + fatherFluidMatingCost) / childAmount;
    gasGivenToEachChild = (gasMatingCost + fatherGasMatingCost) / childAmount;
}

//----------------------------Female Mating (Called by Male)----------------------------\\
public void GetMatedWith(GeneBase[] FatherMatingGenes, int childrenAmount) {
    childrenToBeBirth++;
    totalChildrenInLifeTime++;

    //Create mating genes
    geneManagerRef.CreateMatingGene();

    //The actual birth logics
    CreateChildrenGeneList(geneManagerRef.mattingGene, FatherMatingGenes);

    clonedSelf = false;
}

public void CloneSelf() {
    status_isPregnant = true;
    GeneralMatingLogics();
    clonedSelf = true;

    totalChildrenInLifeTime ++;
    childrenToBeBirth ++;

    energyGivenToEachChild = energyMatingCost;
    fluidGivenToEachChild = fluidMatingCost;
    gasGivenToEachChild = gasMatingCost;

    CreateChildrenGeneList(geneManagerRef.motherGeneList.ToArray(), geneManagerRef.fatherGeneList.ToArray());
}

//============================OTHER MATTING LOGICS============================\\
//----------------------------General Mating logic for male and female and Cloning----------------------------\\
```

```
void GeneralMatingLogics(bool createSpore = false) {
    childrenToBeBirth = 0;

    //Not set when creating spores
    if (!createSpore) {
        pregnancyGeneList = new List<PregnancyGenes>();

        //Reset Mating timer
        currentMatingTimer = 0;
        currentMatingWaitTime = 0;
        currentPregnantTime = 0;
        currentBirthWaitTime = 0;
        status_canMate = false;

        energyMatingCost = total_energy * matingEnergyPercentage;
        fluidMatingCost = total_fluid * matingFluidPercentage;
        gasMatingCost = total_gas * matingGasPercentage;
    }
    else {
        mating_canReleaseSpore = false;
        currentReleaseSporeTime = 0;
        energyMatingCost = total_energy * sporeEnergyPercentage;
        fluidMatingCost = total_fluid * sporeFluidPercentage;
        gasMatingCost = total_gas * sporeGasPercentage;
    }

    currentEnergy -= energyMatingCost;
    currentFluid -= fluidMatingCost;
    currentGas -= gasMatingCost;
}

struct PregnancyGenes {
    public GeneBase[] motherMatingGenes;
    public GeneBase[] FatherMatingGenes;
}

void CreateChildrenGeneList(GeneBase[] motherMatingGenes = null, GeneBase[] FatherMatingGenes = null) {
    List<GeneBase> tempMotherList = new List<GeneBase>();
```

```
        List<GeneBase> tempFatherList = new List<GeneBase>();
        PregnancyGenes pregnancyGenes = new PregnancyGenes();

        //This means cloning
        if (motherMatingGenes == null || motherMatingGenes == null) {
            if (!clonedSelf)
                Debug.LogError("(" + name + ") Trying to Create Children GeneList with Mating Genes null while it does not
have mating Type Clone");

            for (int m = 0; m < geneManagerRef.motherGeneList.Count; m++) {
                tempMotherList.Add(geneManagerRef.motherGeneList[m].copyGene());
            }
            for (int f = 0; f < geneManagerRef.fatherGeneList.Count; f++) {
                tempFatherList.Add(geneManagerRef.fatherGeneList[f].copyGene());
            }
        }
        else {
            for (int m = 0; m < motherMatingGenes.Length; m++) {
                tempMotherList.Add(motherMatingGenes[m].copyGene());
            }
            for (int f = 0; f < FatherMatingGenes.Length; f++) {
                tempFatherList.Add(FatherMatingGenes[f].copyGene());
            }
        }

        pregnancyGenes.motherMatingGenes = tempMotherList.ToArray();
        pregnancyGenes.FatherMatingGenes = tempFatherList.ToArray();

        pregnancyGeneList.Add(pregnancyGenes);
    }

    //----------------------------If Mother dies-----------------------------\\
    //If creature is pregnent and dies then add the energy back into the world
    public float ReturnPregnancyEnergy() {
        if (status_isPregnant) {
            return energyGivenToEachChild * childrenToBeBirth;
        }
        else
```

```
        return 0;
}


//-----------------------------Birth the actual child-----------------------------\\
public void GiveBirth() {
    status_isPregnant = false;
    behaviour_isGivingBirth = false;

    if (pregnancyGeneList.Count <= 0) {
        Debug.LogError("(" + name + ") Trying to give birth  but no pregnancyGeneList Created!");
    }
    else {
        for (int i = 0; i < pregnancyGeneList.Count; i++) {
            GeneralBirthLogics(pregnancyGeneList[i].motherMatingGenes, pregnancyGeneList[i].FatherMatingGenes);
        }
    }

}

//Set genes, Class, Spawn location and any aditional logics depending on mating type
public void GeneralBirthLogics(GeneBase[] motherMatingGenes, GeneBase[] FatherMatingGenes) {

    CritterBase child;

    //Spawn Child
    child = Instantiate(SpawnManager.staticBirthManager.critterPrefab, GetSpawnPostion(), Quaternion.identity);

    //Set Child Class
    child.critterClass = critterClass;

    //Set isBorn and Juvenile status just to be sure
    child.isBorn = true;
    child.status_juvenile = true;

    //Generation Number
    child.generationNr = generationNr + 1;
```

```
        //Born Tile
        child.bornTile = tileCheckerRef.currentTile;


        //Critter Genes (Mother)
        foreach (GeneBase motherGene in motherMatingGenes) {
            //GeneBase newChildGene = motherGene.Value.copyGene();
            GeneBase newChildGene = motherGene.copyGene();
            newChildGene.InstantiateLocalGene(child, "Mother");

            child.geneManagerRef.motherGeneList.Add(newChildGene);
            child.geneManagerRef.motherGenesDic.Add(newChildGene.name, newChildGene);

        }
        //Critter Genes (Father)
        foreach (GeneBase fatherGene in FatherMatingGenes) {
            GeneBase newChildGene = fatherGene.copyGene();
            newChildGene.InstantiateLocalGene(child, "Father");

            child.geneManagerRef.fatherGeneList.Add(newChildGene);
            child.geneManagerRef.fatherGenesDic.Add(newChildGene.name, newChildGene);
        }

        //Set the current Resource values
        child.currentEnergy = energyGivenToEachChild;
        child.currentFluid = fluidGivenToEachChild;
        child.currentGas = gasGivenToEachChild;

        //Aditional Logics
        if (clonedSelf)
            CloneCritter(child);
        else if (matingType == Enums.MatingTypes.Eggs || matingType == Enums.MatingTypes.SporeEggs || matingType ==
Enums.MatingTypes.CloneAndSporeEggs)
            EggBirthCritter(child);
        else if (matingType == Enums.MatingTypes.LifeYoung || matingType == Enums.MatingTypes.SporeLifeYoung)
            LifeBirthCritter(child);
    }
```

```
    //----------------------------Life or Eggs Critter----------------------------\\
    void LifeBirthCritter(CritterBase child) {
        print("Birth Life Young or Spore Life Young");

    }

    void EggBirthCritter(CritterBase child) {
        print("Birth Eggs or Spore Eggs");
        child.status_isEgg = true;
    }


    //----------------------------Clone Critter----------------------------\\
    void CloneCritter(CritterBase child) {
        BecomeSmall();

        //Give Clone child Same Resources
        currentHealth /= 2;
        child.currentHealth = currentHealth;
        child.status_isClone = true;
    }


    //============================TRIGGERS============================\\
    void OnTriggerEnter(Collider trigger) {
        if (status_canExecuteBehaviours) {

            //Eating
            if (trait_canEatNutrientObject && trigger.tag == "NutrientObject") {
                Eat(trigger.GetComponent<NutrientObject>().EatNutrientObject(GetEatAmount()));
            }
        }

        //Mating spore
        if ((matingType == Enums.MatingTypes.SporeEggs || matingType == Enums.MatingTypes.SporeLifeYoung) && status_canMate &&
trigger.tag == "Spore") {
```

```
        //Check if correct Gender
        if (gender == Enums.Gender.Female || gender == Enums.Gender.Hermaphrodite) {

            SporeBase spore = trigger.GetComponent<SporeBase>();

            //Check if it is it's own spore and if it can selfInseminate
            if (spore.originParent == this && !mating_spore_canItSelfInseminate) {
                Debug.Log("(" + name + ") cannot self inseminate with spore.");
            }
            else {
                spore.Impregnate(this);
            }
        }
    }
}

//============================Habitat============================\\
public void EnterHabitat(Enums.Habitat newHabitat, Enums.SubHabitat newSubHabitat = Enums.SubHabitat.Na) {
    if (newHabitat != current_habitat || newSubHabitat != current_subHabitat) {
        Enums.Habitat oldHabitat = current_habitat;
        Enums.SubHabitat oldSubHabitat = current_subHabitat;

        current_habitat = newHabitat;
        current_subHabitat = newSubHabitat;

        if (current_habitat == Enums.Habitat.Water && oldHabitat != Enums.Habitat.Water) {
            habitat_Water_FluidAdd = total_fluid * (habitat_Water_FluidAddPerc * drinkEffectiveness);
            addRemove_fluids += habitat_Water_FluidAdd;
        }
        else {
            if (oldHabitat == Enums.Habitat.Water) {
                addRemove_fluids -= habitat_Water_FluidAdd;
                habitat_Water_FluidAdd = 0;
            }

        }
    }
}
```

```
//============================OTHERS============================\\
//----------------------------Destroy Behaviour Component and set new----------------------------\\
/*
public void ReplaceController(CritterControllerBase newController) {
    CritterControllerBase controller = GetComponent<CritterControllerBase>();

    if (controller != null)
        Destroy(controller);

    if (newController.GetType().ToString() == "AmoebaController")
        controller = gameObject.AddComponent<AmoebaController>();
    else if (newController.GetType().ToString() == "CreatureController")
        controller = gameObject.AddComponent<CreatureController>();
    else if (newController.GetType().ToString() == "PlantController")
        controller = gameObject.AddComponent<AmoebaController>();
    else
        Debug.LogError("(" + name + ") Given Controler does not yet exist");

    controller.Initiate();

}
*/

Vector3 GetSpawnPostion() {
    return navigationControllerRef.GetValidNavmeshLocation(transform.position, transform.localScale * 0.5f);
}

//----------------------------Return 1 or -1----------------------------\\
int PositiveOrNegative() {
    int postiveOrNegative = Random.Range(0, 2);

    if (postiveOrNegative == 0)
        return 1;
    else
        return -1;
```

```
    }

    //----------------------------Reference----------------------------\\
    void SetReference() {
        if (geneManagerRef == null)
            geneManagerRef = GetComponent<GeneManager>();
        if (behaviourControllerRef == null)
            behaviourControllerRef = GetComponent<CritterBehaviourController>();
        if (navigationControllerRef == null)
            navigationControllerRef = GetComponent<NavigationControllerBase>();
        if (tileCheckerRef == null)
            tileCheckerRef = GetComponent<TileCheckerCritter>();

        if (geneManagerRef == null) {
            geneManagerRef = gameObject.AddComponent<GeneManager>();
            Debug.LogError("(" + name + ") GeneManager was null and added to Critter (Some settings might not be set
proper!)");
        }

        if (behaviourControllerRef == null) {
            behaviourControllerRef = gameObject.AddComponent<CritterBehaviourController>();
            Debug.LogError("(" + name + ") behaviourController was null and added to Critter (Some settings might not be set
proper!)");
        }

        if (navigationControllerRef == null) {
            behaviourControllerRef = gameObject.AddComponent<CritterBehaviourController>();
            Debug.LogError("(" + name + ") behaviour Controller was null and added to Critter (Some settings might not be set
proper!)");
        }

        if (tileCheckerRef == null) {
            tileCheckerRef = gameObject.AddComponent<TileCheckerCritter>();
            Debug.LogError("(" + name + ") TileChecker was null and added to Critter.");
        }
    }
}
```