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


[System.Serializable]
public class GeneBase {
    //UPDATE COPY IF ADDING VALUES <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

    [Header("General Info")]
    public string name = "ERROR";
    [HideInInspector]
    public string motherOrFather; //Sets wetherever the gene is a father or mother gene. This is used to check where a evolve
or devolve needs to be added
    [HideInInspector]
    public string nameOnly; //Without the Gene Letter
    public string geneLetter = "?";
    public string parent = "None";
    public bool dominant = false;
    public bool essential = false; //If true then cannot be removed
    public bool updates = false;
    public bool ignoreBodyPartAmount = false; // if true then on evolve it will not check if the bodyPart amount is valid. It
checks the gene it evolves or devolves into (So not the gene that is evolving/devolving)

    [Header("Tier, Current and Max Genevalue")]
    public int tier = -1; //The first evolution is tier 0 any next is one tier up.
    public int geneValue = 1;
    public int maxGeneValue = 10;

    [Header("Habitat")]
    public Enums.Habitat habitat = Enums.Habitat.Na; //Part only activates if this is valid
    public List<Enums.SubHabitat> subHabitats = new List<Enums.SubHabitat>();


    [Header("BodyParts")]
    public Enums.BodyPart bodyPart = Enums.BodyPart.Na;
    public Vector2 minMaxBodyPartAmount = new Vector2(1, 1); //Only relevant if is a body part. The is the max amount of this
body part to exist
```

```
    public int bodyPartsIncrements = 1; //If more of the same body part is added it will be done in these increments
    public int bodyPartsAmount = 0;
    public Sprite[] bodyPartVisuals;
    public int bodyPartAddChance = 10;
    public int bodyPartRemoveChance = 10;

    [Header("BodyParts Effectiveness")]
    public float baseEffectiveness = 1;
    public float effectivenessForEachExtraLimp = 0.25f; //Every extra limp will add this amount to the overal minus and plus
value
    public float effectiveness;

    [Header("GeneralMutation")]
    public int mutatedGeneChangeChance = 20; //The chance this gene will be checked for adding a new gene, removing, evolving,
devolving or adding/Removing BodyPart Amount

    //Once the mutatedGeneChangeChance is passed it will pick any of the following
    [Header("Evolution/Devolution")]
    public int minEvolutionGeneValue = 5; //The value the gene needs to be in order to evolve (Or more)
    public int minDevolutionGeneValue = 5; //The value the gene needs to be in order to devolve (or less)

    public List<string> evolvesTo = new List<string>();
    public List<string> devolvesTo = new List<string>();
    public List<string> getGeneReq = new List<string>(); //All the following Genes need to be present in order to get this
gene
    public List<string> getGeneHasOneReq = new List<string>(); //Needs one of the following in order to evolve
    public List<string> getGeneCannot = new List<string>(); //All the following Genes cannot be present in order to get this
gene
    public int evolveChance = 20; //Chance that a body part evolves if has min evolve value and all other requirments
    public int devolveChance = 20;
    public Vector2 newlyEvolvedGeneValue = new Vector2(3, 4); //If the gene is newly evolved this will be the value
    public Vector2 newlyDevolvedGeneValue = new Vector2(6, 7);
    public Vector2 newlyAddedGeneValue = new Vector2(3, 4);
    public bool allowedToEvolve = true;
    public bool allowedToDevolve = true;


    [Header("Class")]
```

```
public Enums.CritterClass onlyForClass = Enums.CritterClass.Na; //Only this critter type can have this gene
public Enums.CritterClass cannotBeClass = Enums.CritterClass.Na; //Critter cannot be class
public Enums.CritterClass changeClassTo = Enums.CritterClass.Na; //Will set the class to the following if not Na


//Used for copying Gene
[HideInInspector]
public GeneBase geneCopy;

[Header("Either for Total_BaseValues or Percentage multipliers")]
public Vector4 zeroMinMidMaxValue;
public Vector2 minMaxValue;

[Header("Mutation")]
public int maxMutationValue = 3;
public int mutationChance = 40; //In Percentages
int dominanceChance = 50;

[Header("Remove Gene chance")]
public int removeGeneChance = 5;

//Reference
[HideInInspector]
public CritterBase critterBaseRef;


public virtual void Instantiate() {
    name = GetType().ToString();

    //Gene Letters
    if (bodyPart == Enums.BodyPart.Head)
        geneLetter = "V"; //
    else if (bodyPart == Enums.BodyPart.Torso)
        geneLetter = "T";
    else if (bodyPart == Enums.BodyPart.Limps)
        geneLetter = "L";
    else if (bodyPart == Enums.BodyPart.Skeleton)
        geneLetter = "K";
```

```
else if (bodyPart == Enums.BodyPart.Skin)
    geneLetter = "Z";
else if (bodyPart == Enums.BodyPart.Tail)
    geneLetter = "I";
else if (bodyPart == Enums.BodyPart.Eyes)
    geneLetter = "Y";
else if (bodyPart == Enums.BodyPart.Anteny)
    geneLetter = "G";
else if (bodyPart == Enums.BodyPart.Wings)
    geneLetter = "W";
else if (bodyPart == Enums.BodyPart.Horn)
    geneLetter = "R";
else if (bodyPart == Enums.BodyPart.Defensive)
    geneLetter = "D";
else if (bodyPart == Enums.BodyPart.Offensive)
    geneLetter = "O";
else if (bodyPart == Enums.BodyPart.Lungs)
    geneLetter = "U";
else if (bodyPart == Enums.BodyPart.EnergyProduction)
    geneLetter = "P";
else if (bodyPart == Enums.BodyPart.Color)
    geneLetter = "C";
else if (bodyPart == Enums.BodyPart.Brain)
    geneLetter = "B";
else if (bodyPart == Enums.BodyPart.Digestion)
    geneLetter = "N";
else if (bodyPart == Enums.BodyPart.Mating)
    geneLetter = "M";
else if (bodyPart == Enums.BodyPart.Social)
    geneLetter = "A";
else
    geneLetter = "'MISSING:" + name + "'";


//NA BodyPart Letters (Genes no longer used)
if (name == "GeneHealth")
    geneLetter = "H";
else if (name == "GeneEnergy")
```

```
            geneLetter = "E";
        else if (name == "GeneFluid")
            geneLetter = "F";
        if (name == "GeneSize")
            geneLetter = "S";
        else if (name == "GeneAge")
            geneLetter = "A";

        nameOnly = name;
        name = name + "(" + geneLetter + ")";

        //Min BodyPart
        bodyPartsAmount = (int)minMaxBodyPartAmount.x;

        //Calc Effectiveness
        CalcBodyEffectiveness();
    }

    public void SetUpDevolution() {

        for (int i = 0; i < evolvesTo.Count; i++) {

            for (int e = 0; e < GenePool.staticGenePool.essentialGenes.Count; e++) {
                if (evolvesTo[i] == GenePool.staticGenePool.essentialGenes[e].nameOnly) {
                    GenePool.staticGenePool.essentialGenes[e].devolvesTo.Add(nameOnly);
                    break;
                }
            }

            //If gene is not essential
            for (int p = 0; p < GenePool.staticGenePool.primitiveGenes.Count; p++) {
                if (evolvesTo[i] == GenePool.staticGenePool.primitiveGenes[p].nameOnly) {
                    GenePool.staticGenePool.primitiveGenes[p].devolvesTo.Add(nameOnly);
                    break;
                }
            }

            //If gene is not essential
```

```
        for (int o = 0; o < GenePool.staticGenePool.allOtherGenes.Count; o++) {
            if (evolvesTo[i] == GenePool.staticGenePool.allOtherGenes[o].nameOnly) {
                GenePool.staticGenePool.allOtherGenes[o].devolvesTo.Add(nameOnly);
                break;
            }
        }

    }
}

public void InstantiateLocalGene(CritterBase critterBase, string motherOrFatherGene) {
    //Ref
    critterBaseRef = critterBase;

    motherOrFather = motherOrFatherGene;


}

public virtual void GeneUpdate() {

}


//Mutate Gene
public void Mutate() {
    //For dev print
    int prevGeneValue = geneValue;
    bool prevDominant = dominant;

    int mutationValue = Random.Range(-maxMutationValue, maxMutationValue + 1);

    geneValue += mutationValue;

    if (geneValue < 0)
        geneValue = 0;
    else if (geneValue > maxGeneValue)
        geneValue = maxGeneValue;
```

```
        //new Dominant
        int dominantRandomise = Random.Range(0, 101);
        if (dominantRandomise <= dominanceChance)
            dominant = true;
        else
            dominant = false;

        if (critterBaseRef == null)
            Debug.LogError("CritterBase is null for '" + name + "'! (The gene is probably added newly somewhere after the
original localInstantiate. Make sure it Instantiate the newly added gene)");
        else
            parent = critterBaseRef.identifier;


        Debug.Log("(" + critterBaseRef.name + ") Gene: '" + name + "' has mutated its gene value. (mutationChance: '" +
(mutationChance * critterBaseRef.geneManagerRef.mutationMulti) * Global.globalMutationMulti + "' | prevValue: '" +
prevGeneValue + "', newValue: '" + geneValue + "' | prevDomi: '" + prevDominant + "' newDomi: '" + dominant + "')");
    }



    public void Randomize() {
        if (Global.staticGlobal.devMode) {
            geneValue = Random.Range(0, (int)maxGeneValue + 1);

            //new Dominant
            int dominantRandomise = Random.Range(0, 101);
            if (dominantRandomise <= 50)
                dominant = true;
            else
                dominant = false;

        }

        parent = critterBaseRef.identifier;
    }
```

```
    //Express Gene
    public virtual void GeneExpression() {
        if (changeClassTo != Enums.CritterClass.Na) {
            critterBaseRef.ChangeCritterClass(changeClassTo, name);
        }
    }


    //============================FUNCTIONS TO CALCULATE TOTAL BASE VALUES SUCH AS HEALTH============================\\
    //What I know of it is that: It first calculate the increments it needs (Using CalculateIncrements), then calculates the
value the base(HalfValue) will be added to or removed (Using PositiveNegativeCalc). and finialy in the gene itself it will add
the number to the base value (eg total_health)
    public float CalculateTotalValueAddRemove(float halfValue) {
        if (halfValue < minMaxValue.x)
            Debug.LogError("'" + name + "': WARNING! the base_Value given is lower than low Value. This should never be the
case!");
        if (halfValue > minMaxValue.y)
            Debug.LogError("'" + name + "': WARNING! the base_Value given is higher than high Value. This should never be the
case!");

        return PositiveNegativeCalc(CalculateIncrements(halfValue, minMaxValue.x * effectiveness),
CalculateIncrements(halfValue, minMaxValue.y * effectiveness));
    }

    public float CustomCalculateTotalValueAddRemove(float lowValue, float halfValue, float highValue) {
        return PositiveNegativeCalc(CalculateIncrements(halfValue, lowValue * effectiveness), CalculateIncrements(halfValue,
highValue * effectiveness));
    }

    //This calculates the increments of each step for total values
    public float CalculateIncrements(float halfValue, float inputValue) {
        float returnValue = 0;

        if (inputValue > halfValue) {
            returnValue = (inputValue - halfValue) / (maxGeneValue / 2);
        }
        else if (inputValue < halfValue) {
            returnValue = (halfValue - inputValue) / (maxGeneValue / 2);
```

```
    }

    return returnValue;
}


//This will keep the current value if gene is half
public float PositiveNegativeCalc(float negative, float positive) {
    //If geneValue is half of maxGeneValue then nothing is substracted or added
    //If less then half then negative float is used to calculate a value that is substracted of the current amount
    //If more then half then positive float is used to calculate a value that is added to the current amount


    float multi;
    float returnValue = 0;

    if (geneValue > (maxGeneValue / 2)) {
        multi = geneValue - (maxGeneValue / 2);
        returnValue = positive * multi;
    }
    //Negative
    else if (geneValue < (maxGeneValue / 2)) {
        multi = (maxGeneValue / 2) - geneValue;
        returnValue = (negative * multi) * -1;
    }

    return returnValue;
}


//============================FUNCTION TO CALCULATE PERCENTAGE MULTIPLIERS============================\\
//ZERO will DISABLE the gene
//This calcualtion returns the proper value if giving a min and max value
public float AddRemoveCalcZeroDisable(float oneValue, float halfValue, float maxValue, float zeroValue) {
    float returnValue = 0;

    if (geneValue == 0)
        return (zeroValue * effectiveness);
```

```
    else if (geneValue == maxGeneValue / 2)
        return (halfValue * effectiveness);
    else if (geneValue < maxGeneValue / 2) {
        float increment = (halfValue - oneValue) / ((maxGeneValue / 2) -1);
        returnValue = (halfValue - (increment * ((maxGeneValue / 2) - geneValue))) * effectiveness;

    }
    else if (geneValue > maxGeneValue / 2) {
        float increment = (maxValue - halfValue) / (maxGeneValue / 2);
        returnValue = (halfValue + (increment * (geneValue - (maxGeneValue / 2)))) * effectiveness;
    }
    else
        Debug.LogError("Something went wrong with '" + name + "' as the addRemove Calculation is never touched");


    return returnValue;
}

//zero will not disable the gene
//This calcualtion returns the proper value if giving a min and max value
public float AddRemoveCalc(float zeroValue, float halfValue, float maxValue) {
    float returnValue = 0;

    if (geneValue == maxGeneValue / 2)
        return (halfValue * effectiveness);
    else if (geneValue < maxGeneValue / 2) {
        float increment = (halfValue - zeroValue) / (maxGeneValue / 2);
        returnValue = (halfValue - (increment * ((maxGeneValue / 2) - geneValue))) * effectiveness;

    }
    else if (geneValue > maxGeneValue / 2) {
        float increment = (maxValue - halfValue) / (maxGeneValue / 2);
        returnValue = (halfValue + (increment * (geneValue - (maxGeneValue / 2)))) * effectiveness;
    }
    else
        Debug.LogError("Something went wrong with '" + name + "' as the addRemove Calculation is never touched");
```

```
        return returnValue;
    }


    //============================COPY GENE============================\\
    public virtual GeneBase copyGene() {

        if (geneCopy == null) {
            Debug.LogError("Tried to copy gene: '"+ name + "' but is null! Thus created an empty geneBase to prevent crashing.
GENE WILL NOT WORK PROPERLY!");
            //Make sure the copyGene script has the following:
            //geneCopy = new GeneSomething();
            //return base.copyGene();
            //If it is using an other GeneAsSub base then add the following to the base: geneCopy = copyGeneSubBase. In the
child add: copyGeneSubBase = new GeneSomething();

            geneCopy = new GeneBase();
        }

        //Gene Info
        geneCopy.name = name;
        geneCopy.nameOnly = nameOnly;
        geneCopy.geneLetter = geneLetter;
        geneCopy.parent = parent;
        geneCopy.dominant = dominant;
        geneCopy.essential = essential; //If true then cannot be removed
        geneCopy.updates = updates;

        //Body Part
        geneCopy.bodyPart = bodyPart;
        geneCopy.minMaxBodyPartAmount = minMaxBodyPartAmount; //Only relevant if is a body part. The is the max amount of this
body part to exist
        geneCopy.bodyPartsIncrements = bodyPartsIncrements; //If more of the same body part is added it will be done in these
increments
        geneCopy.bodyPartVisuals = bodyPartVisuals;
        geneCopy.bodyPartsAmount = bodyPartsAmount;
        geneCopy.effectivenessForEachExtraLimp = effectivenessForEachExtraLimp;
        geneCopy.baseEffectiveness = baseEffectiveness;
```

```
        geneCopy.effectiveness = effectiveness;
        geneCopy.bodyPartAddChance = bodyPartAddChance;
        geneCopy.bodyPartRemoveChance = bodyPartRemoveChance;

        //General Gene
        geneCopy.tier = tier;
        geneCopy.removeGeneChance = removeGeneChance;

        //General Mutation
        geneCopy.mutatedGeneChangeChance = mutatedGeneChangeChance;

        //Evolution
        geneCopy.evolvesTo = evolvesTo;
        geneCopy.devolvesTo = devolvesTo;
        geneCopy.getGeneReq = getGeneReq;
        geneCopy.getGeneHasOneReq = getGeneHasOneReq;
        geneCopy.getGeneCannot = getGeneCannot;
        geneCopy.evolveChance = evolveChance;
        geneCopy.devolveChance = devolveChance;
        geneCopy.allowedToEvolve = allowedToEvolve;
        geneCopy.allowedToDevolve = allowedToDevolve;

    //New gene values
    geneCopy.newlyEvolvedGeneValue = newlyEvolvedGeneValue;
        geneCopy.newlyDevolvedGeneValue = newlyDevolvedGeneValue;
        geneCopy.newlyAddedGeneValue = newlyAddedGeneValue;

        //Class
        geneCopy.onlyForClass = onlyForClass;
        geneCopy.cannotBeClass = cannotBeClass;
        geneCopy.changeClassTo = changeClassTo;


        //habitat
        geneCopy.habitat = habitat; //Part only activates if this is valid
        geneCopy.subHabitats = subHabitats;

        geneCopy.geneValue = geneValue;
```

```
        geneCopy.maxGeneValue = maxGeneValue;

        //Positive negative
        //geneCopy.lowestValue = lowestValue;
        //geneCopy.highestValue = highestValue;
        geneCopy.zeroMinMidMaxValue = zeroMinMidMaxValue;
        geneCopy.minMaxValue = minMaxValue;

        geneCopy.mutationChance = mutationChance; //In Percentages
        geneCopy.dominanceChance = dominanceChance;

        return geneCopy;
    }


    //On Evolve
    public virtual void OnEvolveGene(string evolveInto, Dictionary<string, GeneBase> holderDic, List<GeneBase> holderList,
string motherOrFather) {
        //Debug.Log (this.GetType().ToString() + " evolved into " + evolveInto);
    }

    //On Devolve
    public virtual void OnDevolveGene(string devolveInto, Dictionary<string, GeneBase> holderDic, List<GeneBase> holderList,
string motherOrFather) {
        //Debug.Log(this.GetType().ToString() + " devolved into " + devolveInto);
    }

    //If Gene is removed
    public virtual void OnRemoveGene() {

    }


    public void SetVisual() {
        if (bodyPartVisuals.Length > 0) {
            //Take max value, devide by the visual lenght
            //if there are 5 visuals then; Visual 0 = 1&2, 1 = 3&4, 2 = 5&6, 3 = 7&8 and  4 = 9& 10
```

```
    }

}

public void CalcBodyEffectiveness() {
    float calc = 0;

    //Gets a value for each extra bodyPart
    if (bodyPartsAmount > 1)
        calc = (bodyPartsAmount -1) * effectivenessForEachExtraLimp;

    calc += baseEffectiveness;
    effectiveness = calc;
}

//============================ADD GENE FUNCTIONS============================\\
public void EvolveToF(GeneBase gene) {
    evolvesTo.Add(gene.GetType().ToString());
}

public void DevolveToF(GeneBase gene) {
    devolvesTo.Add(gene.GetType().ToString());
}

public void GetGeneReqF(GeneBase gene) {
    getGeneReq.Add(gene.GetType().ToString());
}

public void GetGeneHasOneReqF(GeneBase gene) {
    getGeneHasOneReq.Add(gene.GetType().ToString());

}

public void GetGeneCannotF(GeneBase gene) {
    getGeneCannot.Add(gene.GetType().ToString());
}

//============================REPLACE GENE FUNCTION============================\\
```

```
    public void ReplaceGene(string geneToReplace, string geneToAddInstead, Dictionary<string, GeneBase> holderDic,
List<GeneBase> holderList, string motherOrFather, bool copyOverGeneValue = true) {
        GeneBase geneInDic;
        GeneBase newGene;
        int index = -1;
        bool geneFoundInList = false;
        int newGeneValue = 0;



        newGene = GenePool.staticGenePool.GetGene(geneToAddInstead, true);

        if (newGene == null) {
            Debug.LogError("(Gene: '" + name + "') Could not find the gene called: '" + geneToAddInstead + "' in the genePool
to replace '" + geneToReplace + "' With!");
        }

        if (holderDic.TryGetValue(geneToReplace, out geneInDic)) {
            //If copy value is true then the value of the replaced gene is copied to the new gene
            if (copyOverGeneValue)
                newGeneValue = geneInDic.geneValue;



            for (int i = 0; i < holderList.Count; i++) {
                if (holderList[i].nameOnly == geneToReplace) {
                    holderList.RemoveAt(i);
                    holderList.Insert(i, newGene);
                    geneFoundInList = true;
                    index = i;
                    break;
                }
            }
            if (!geneFoundInList) {
                Debug.LogError("(Gene: '" + name + "') WARNING! Gene was removed from Dictionary but was not found in the
List! This is very wrong as the two should be identical!");
                return;
            }
```

```
            Debug.LogWarning("Found gene to replace: '" + geneToReplace + "' and replaced it with: '" + geneToAddInstead +
"'.");

            //If does not copy over gene value then set it to minDevolution
            if (!copyOverGeneValue)
                newGeneValue = (int)Random.Range((int)holderList[index].newlyAddedGeneValue.x,
(int)holderList[index].newlyAddedGeneValue.y + 1);

            //Set gene value
            holderList[index].geneValue = newGeneValue;

            //Remove Old and add new
            holderDic.Remove(geneToReplace);
            holderDic.Add(holderList[index].nameOnly, holderList[index]);

            //Instantiate gene
            holderList[index].InstantiateLocalGene(critterBaseRef, motherOrFather);

        }
        else
            Debug.LogWarning("Did not find gene to replace: '" + geneToReplace + "' so could not replaced it with: '" +
geneToAddInstead + "'.");


    }


}
```