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

public class GeneManager : MonoBehaviour {
    //Ref
    public CritterBase critterBaseRef;

    public string DNA;
    //public string expressedDNA;

    //Gene Lists
    //public List<GeneBase> motherGenes;
    //public List<GeneBase> fatherGenes;
    public Dictionary<string, GeneBase> motherGenesDic = new Dictionary<string, GeneBase>();
    public Dictionary<string, GeneBase> fatherGenesDic = new Dictionary<string, GeneBase>();
    public List<GeneBase> motherGeneList;
    public List<GeneBase> fatherGeneList;

    //Max Parent Genes to mutate on being Born
    int maxBirthMutations = 2;
    int maxMutationTries = 10; //The system tries 10 times to mutate a random gene.

    //Gene Arrays
    public List<GeneBase> expressedGeneList; //Used for calculating Values
    public Dictionary<string, GeneBase> expressedGenesDic = new Dictionary<string, GeneBase>();
    public GeneBase[] mattingGene;
    public GeneBase[] updatableGenes; //A list of genes that is constantly updated

    //Looked up gene
    GeneBase foundGene = null;


    float inharentExclusiveGeneParent = 75; //A % chance of inharenting a trait that mother and father do not share

    //Chance of adding a gene
    int addGeneChanceCompairedToOtherChanges = 20;
    int addGeneChance = 5; //Chance of a gene being added
```

```
    [Header("Mutation and Evolution Multipliers")]
    public float mutationMulti = 1; //Chance a gene mutates (Is Only set by Mating gene)
    Vector2 minMaxMutationChance = new Vector2(5, 100); //Just in case
    public float geneChangeMulti = 1; //Chance a gene mutates (Is Only set by Mating gene)
    Vector2 minMaxEvolutionChance = new Vector2(5, 100); //Just in case

    //For evolution
    List<GeneBase> potentialEvolutions;
    List<GeneBase> potentialDevolutions;
    List<GeneBase> potentialAddGenes;
    int addGeneToMotherOrFather;
    List<GeneBase> potentialRemoveGenes;

    //Error string
    string errorText = "";



    public void Initialise() {
        SetReferences();
    }

    public void ExecuteGeneSetUp() {
        //Add essentials Genes
        AddEssentialGenes();

        //If does not have a critter class then randomize class and grab genes. Chis is changed as critters now always start
as micro
        //if (critterBaseRef.critterClass == Enums.CritterClass.SystemSet)
        //    SetCritterClass();



        //Instantiate Genes
        InstantiateGenes();
```

```
        //Randomize Genes if not born
        if (!critterBaseRef.isBorn && critterBaseRef.randomizeGenes)
            RandomizeGenes();

        //GeneVariables
        ExpressGenes(); //Sets all the common values correctly
    }

    private void Update() {
        //Update Genes function
        for (int i = 0; i < updatableGenes.Length; i++) {
            updatableGenes[i].GeneUpdate();
        }
    }

    //============================ADD ESSENTIAL GENES============================\\
    //Only done for critters that are not born
    public void AddEssentialGenes() {
        if (!critterBaseRef.isBorn) {
            //Checks if all the esential genes are there and adds them if not
            for (int i = 0; i < GenePool.staticGenePool.essentialGenes.Count; i++) {

                if (!FindGene(GenePool.staticGenePool.essentialGenes[i].nameOnly, motherGenesDic)) {
                    motherGenesDic.Add(GenePool.staticGenePool.essentialGenes[i].nameOnly,
GenePool.staticGenePool.essentialGenes[i].copyGene());

                    if (critterBaseRef.isBorn)
                        Debug.LogError("(" + gameObject.name + ") A Born Critter called '" + gameObject.name + "' is missing
the essential gene in the Mother GeneList: '" + GenePool.staticGenePool.essentialGenes[i].name + "' which is added.");
                }

                if (!FindGene(GenePool.staticGenePool.essentialGenes[i].nameOnly, fatherGenesDic)) {
                    fatherGenesDic.Add(GenePool.staticGenePool.essentialGenes[i].nameOnly,
GenePool.staticGenePool.essentialGenes[i].copyGene());

                    if (critterBaseRef.isBorn)
                        Debug.LogError("(" + gameObject.name + ") A Born Critter called '" + gameObject.name + "' is missing
the essential gene in the Father GeneList: '" + GenePool.staticGenePool.essentialGenes[i].name + "' which is added.");
```

```
            }



        }


        motherGeneList = new List<GeneBase>(motherGenesDic.Values);
        fatherGeneList = new List<GeneBase>(fatherGenesDic.Values);
    }
}


//============================LOCAL INSTANTIATE GENES============================\\
public void InstantiateGenes() {
    if (!critterBaseRef.isBorn) {
        foreach (KeyValuePair<string, GeneBase> gene in motherGenesDic) {
            gene.Value.InstantiateLocalGene(critterBaseRef, "Mother");
        }

        foreach (KeyValuePair<string, GeneBase> gene in fatherGenesDic) {
            gene.Value.InstantiateLocalGene(critterBaseRef, "Father");
        }
    }
}

//============================SET CRITTER CLASS============================\\
public void AddNewGeneGroup() {
    //This adds a basic genegroup which makes the critter a Plant, or creature
    int randomClass = Random.Range(0, GenePool.staticGenePool.critterClassesList.Count);

    critterBaseRef.critterClass = GenePool.staticGenePool.critterClassesList[randomClass].critterClass;
    GenePool.CritterClassGenesStruct newGeneGroup = GenePool.staticGenePool.critterClassesList[randomClass];

    for (int i = 0; i < newGeneGroup.startingGenes.Count; i++) {
        motherGenesDic.Add(newGeneGroup.startingGenes[i].nameOnly, newGeneGroup.startingGenes[i].copyGene());
        fatherGenesDic.Add(newGeneGroup.startingGenes[i].nameOnly, newGeneGroup.startingGenes[i].copyGene());
```

```
        }
    }

    //============================RANDOMIZE GENES============================\\
    public void RandomizeGenes() {
        //CAn be used if critter is not born
        foreach (KeyValuePair<string, GeneBase> gene in motherGenesDic) {
            gene.Value.Randomize();
        }

        foreach (KeyValuePair<string, GeneBase> gene in fatherGenesDic) {
            gene.Value.Randomize();
        }
    }

    //============================MUTATE, EXPRESSION AND DNA============================\\
    public void ExpressGenes() {
        //Developer Only
#if UNITY_EDITOR
        DeveloperOnly();
#endif

        //Mutate ParentGenes
        if (critterBaseRef.canMutateAtStart)
            MutateParentGenes();

        //DNA visuelizer
        CreateDNAString();

        //Create expressed GeneArray
        CreateExpressedGenesArray();


        //Apply the values by expressing each gene in the express gene array
        UpdateCritterBaseValues();

        //Create Gene Update List
        CreateUpdateList();
```

```
    }

#if UNITY_EDITOR
    void DeveloperOnly() {
        if (Global.staticGlobal.devMode == true) {
            /*
            Global.globalMutationMulti = 2.5f;
            Global.globalEvolutionMulti = 5;

            //Add genes here for testing
            motherGenes.Add(GenePool.staticGenePool.GetGene("GeneAmoebaTail", true));
            fatherGenes.Add(GenePool.staticGenePool.GetGene("GenePrimitiveTail", true));
            */
        }
    }
#endif



    //============================CHECK IF SHOULD MUTATE A GENE============================\\
    //Struct with info to keep track of genes that are already mutated
    struct GenesAlreadyMutated {
        public int MotherFather;
        public int index;
    }

    //---------------------------Mutation-----------------------------\\
    void MutateParentGenes() {
        int randomMutationValue;

        int geneIndex = -1;
        int motherOrFatherGene = -1;
        int currentMutationTries = 0;
        int currentGenesMutated = 0;
        float neededMutationChance;

        //To check if gene was already mutated
```

```
        List <GenesAlreadyMutated> alreadyMutatedGenes = new List<GenesAlreadyMutated>();
        GenesAlreadyMutated newAlreadyMutatedStruct;
        int safty = 0;
        bool validMutationChoice = false;
        bool geneWasAlreadyMutated = false;


        //mutationTries
        while (currentMutationTries < maxMutationTries && currentGenesMutated < maxBirthMutations) {

            randomMutationValue = Random.Range(0, 101);
            motherOrFatherGene = Random.Range(0,2); //0 = pick mother and 1= pick father


            //Check if not already has mutated this gene (Tries 10 times just to be safe)
            validMutationChoice = false;
            geneWasAlreadyMutated = false;
            safty = 0;

            while (validMutationChoice == false && safty <= 10) {


                if (motherOrFatherGene == 0)
                    geneIndex = Random.Range(0, motherGenesDic.Count);
                else
                    geneIndex = Random.Range(0, fatherGenesDic.Count);

                for (int i = 0; i < alreadyMutatedGenes.Count; i++) {
                    if (alreadyMutatedGenes[i].index == geneIndex) {
                        if (alreadyMutatedGenes[i].MotherFather == motherOrFatherGene) {
                            geneWasAlreadyMutated = true;
                            Debug.LogWarning("(" + gameObject.name + ") tried to mutate a gene but it was already mutated once
before (MotherOrFather: '" + motherOrFatherGene + "' GeneIndex: '" + geneIndex + "'");
                            break;
                        }
                    }
                }
```

```
                if (!geneWasAlreadyMutated) {
                    validMutationChoice = true;
                }
                safty++;
            }



            //Actual should mutate check
            if (validMutationChoice) {
                if (motherOrFatherGene == 0) {
                    neededMutationChance = (motherGeneList[geneIndex].mutationChance * mutationMulti) *
Global.globalMutationMulti;

                    if (neededMutationChance < minMaxMutationChance.x) {
                        Debug.LogError("(" + gameObject.name + ") '" + gameObject.name + "' gene: '" +
motherGeneList[geneIndex].name + "'(Mother) has tried to mutate but the mutation chance is lower then the min. Therefore the
mutation chance is temp set to Min");
                        neededMutationChance = minMaxMutationChance.x;
                    }
                    else if (neededMutationChance > minMaxMutationChance.y) {
                        Debug.LogError("(" + gameObject.name + ") '" + gameObject.name + "' gene: '" +
motherGeneList[geneIndex].name + "'(Mother) has tried to mutate but the mutation chance is higher then the max. Therefore the
mutation chance is temp set to Max");
                        neededMutationChance = minMaxMutationChance.y;
                    }

                    if (randomMutationValue <= neededMutationChance) {
                        motherGeneList[geneIndex].Mutate();
                        ChangeGeneOnMutate(motherGeneList[geneIndex], motherGenesDic, motherGeneList, geneIndex,
neededMutationChance);
                        currentGenesMutated++;

                        //Keep track of which ones are mutated
                        newAlreadyMutatedStruct = new GenesAlreadyMutated();
                        newAlreadyMutatedStruct.MotherFather = motherOrFatherGene;
                        newAlreadyMutatedStruct.index = geneIndex;
                        alreadyMutatedGenes.Add(newAlreadyMutatedStruct);
```

```
                }


            }
            else if (motherOrFatherGene == 1) {
                neededMutationChance = (fatherGeneList[geneIndex].mutationChance * mutationMulti) *
Global.globalMutationMulti;

                if (neededMutationChance < minMaxMutationChance.x) {
                    Debug.LogError("(" + gameObject.name + ") '" + gameObject.name + "' gene: '" +
fatherGeneList[geneIndex].name + "'(Father) has tried to mutate but the mutation chance is lower then the min. Therefore the
mutation chance is temp set to Min");
                    neededMutationChance = minMaxMutationChance.x;
                }
                else if (neededMutationChance > minMaxMutationChance.y) {
                    Debug.LogError("(" + gameObject.name + ") '" + gameObject.name + "' gene: '" +
fatherGeneList[geneIndex].name + "'(Father) has tried to mutate but the mutation chance is higher then the max. Therefore the
mutation chance is temp set to Max");
                    neededMutationChance = minMaxMutationChance.y;
                }


                if (randomMutationValue <= neededMutationChance) {
                    fatherGeneList[geneIndex].Mutate();
                    ChangeGeneOnMutate(fatherGeneList[geneIndex], fatherGenesDic, fatherGeneList, geneIndex,
neededMutationChance);
                    currentGenesMutated++;


                    //Keep track of which ones are mutated
                    newAlreadyMutatedStruct = new GenesAlreadyMutated();
                    newAlreadyMutatedStruct.MotherFather = motherOrFatherGene;
                    newAlreadyMutatedStruct.index = geneIndex;
                    alreadyMutatedGenes.Add(newAlreadyMutatedStruct);
                }
            }
            else
```

```
                    Debug.LogError("(" + gameObject.name + ") Something went wrong with trying to decide which Genelist to
grab for random mutation");
                }
                else
                    Debug.Log("(" + gameObject.name + ") Tried to mutate a gene but did not succeed. It could be luck that it just
happenend to randomize the same gene of the same parent. It could be that there are only a very few genes which makes the prev
explanation more frequent. Or something is going wrong. If this happens a lot then change the 'validMutationChoice' loop");


                currentMutationTries++;
            }
        }


        //============================CHANGING GENE ON MUTATE============================\\
        //On Mutation a Gene can evolve or devolve it can also be removed or a new gene can be added that has nothing to do with
this gene
        void ChangeGeneOnMutate(GeneBase geneBase, Dictionary<string,GeneBase> holderDic, List<GeneBase> holderList, int index,
float mutateChance) { //MutateChance is used for debug DebugLog
            int randomPercentage = Random.Range(0, 101);
            float neededChangeChance = (geneBase.mutatedGeneChangeChance * geneChangeMulti) * Global.globalGeneChangeMulti;

            if (neededChangeChance < minMaxEvolutionChance.x) {
                Debug.LogError("(" + gameObject.name + ") '" + gameObject.name + "' gene: '" + geneBase.name + "' has tried to
check for evolution but the evolution chance is lower then the min. Therefore the evolution chance is temp set to min");
                neededChangeChance = minMaxEvolutionChance.x;
            }
            else if (neededChangeChance > minMaxEvolutionChance.y) {
                Debug.LogError("(" + gameObject.name + ") '" + gameObject.name + "' gene: '" + geneBase.name + "' has tried to
check for evolution but the evolution chance is higher then the max. Therefore the evolution chance is temp set to max");
                neededChangeChance = minMaxEvolutionChance.y;
            }


            if (randomPercentage <= neededChangeChance) {
                //Debug.Log("(" + gameObject.name + ") Gene: '" + geneBase.name + "' has mutated and also changed (MutateChance:
'" + mutateChance + "' | ChangeChance: '" + neededChangeChance + "')");
```

```
int canAddGene = 0;
int canRemoveGene = 0;
int canAddBodyPart = 0;
int canRemoveBodyPart = 0;
int canEvolveGene = 0;
int canDevolveGene = 0;

int totalChangeChance = 0;

//Check if should add a gene
int randomAddChance = Random.Range(0, 101);
if (randomAddChance <= addGeneChance) {
    //Check if there are any valid genes to add
    CheckIfCanAddGene();
    if (potentialAddGenes.Count > 0) {
        canAddGene = addGeneChanceCompairedToOtherChanges;
        totalChangeChance += canAddGene;
    }
}

//Can the gene be removed if true then add to randomizer
if (!geneBase.essential) {
    canRemoveGene = geneBase.removeGeneChance;
    totalChangeChance += geneBase.removeGeneChance;
}


//Can there be a body part added
if (geneBase.bodyPartsAmount < geneBase.minMaxBodyPartAmount.y) {
    canAddBodyPart = geneBase.bodyPartAddChance;
    totalChangeChance += geneBase.bodyPartAddChance;
}

//Can there be a body part removed
if (geneBase.bodyPartsAmount > geneBase.minMaxBodyPartAmount.x) {
    canRemoveBodyPart = geneBase.bodyPartRemoveChance;
    totalChangeChance += geneBase.bodyPartRemoveChance;
}
```

```
            //Can the body part be evolved (Evolve and devolve are properly checked) < Needs Testing
            if (geneBase.allowedToEvolve)
                CheckCanEvolveDevolveBodyPart(geneBase, true);

            if (geneBase.allowedToEvolve && potentialEvolutions.Count > 0) {
                canEvolveGene = geneBase.evolveChance;
                totalChangeChance += geneBase.evolveChance;
            }

            //Can the body part be Devolved (Evolve and devolve are properly checked) < Needs testing
            if (geneBase.allowedToDevolve)
                CheckCanEvolveDevolveBodyPart(geneBase, false);

            if (geneBase.allowedToDevolve && potentialDevolutions.Count > 0) {
                canDevolveGene = geneBase.devolveChance;
                totalChangeChance += geneBase.devolveChance;
            }

            if (totalChangeChance != 0) {

                int randomEvolveType = Random.Range(0, totalChangeChance + 1);

                //Debug.Log("(" + gameObject.name + ") RandomEvolve: " + randomEvolveType);

                //Debug.Log("(" + gameObject.name + ") totalEvolveChance: " + totalEvolveChance + " and randomEvolveType: " +
randomEvolveType);

                if (canAddGene != 0 && randomEvolveType <= canAddGene){
                    AddGene();
                }
                else if (canRemoveGene != 0 && randomEvolveType <= (canAddGene + canRemoveGene) && randomEvolveType >
canAddGene){
                    RemoveGene(holderDic,holderList, index);
                }
                else if (canAddBodyPart != 0 && randomEvolveType <= (canAddGene + canRemoveGene + canAddBodyPart) &&
randomEvolveType > (canAddGene + canRemoveGene)){
                    AddBodyPart(geneBase);
```

```
                }
                else if (canRemoveBodyPart != 0 && randomEvolveType <= (canAddGene + canRemoveGene + canAddBodyPart +
canRemoveBodyPart) && randomEvolveType > (canAddGene + canRemoveGene + canAddBodyPart)){
                    RemoveBodyPart(geneBase);
                }
                else if (canEvolveGene != 0 && randomEvolveType <= (canAddGene + canRemoveGene + canAddBodyPart +
canRemoveBodyPart + canEvolveGene) && randomEvolveType > (canAddGene + canRemoveGene + canAddBodyPart + canRemoveBodyPart)){
                    EvolveDevolveBodyPart(geneBase, holderDic, holderList, index, true);
                }
                else if (canDevolveGene != 0 && randomEvolveType <= (canAddGene + canRemoveGene + canAddBodyPart +
canRemoveBodyPart + canEvolveGene + canDevolveGene) && randomEvolveType > (canAddGene + canRemoveGene + canAddBodyPart +
canRemoveBodyPart + canEvolveGene)){
                    EvolveDevolveBodyPart(geneBase, holderDic, holderList, index, false);
                }
                else {
                    Debug.LogWarning("(" + gameObject.name + ") '" + geneBase.name + "': Evolution/Devolution failed as non of
the checks were valid.");
                }
            }
            else {
                //Debug.Log("(" + gameObject.name + ") has mutated but did not evolve as nothing to add, remove, evolve,
devolve or change body part amount. (MutateChance: '" + mutateChance + "' | EvolveChance: '" + neededChangeChance + "')");
            }

        }
        else {
            //Debug.Log("(" + gameObject.name + ") has mutated but did not evolve. (MutateChance: '" + mutateChance + "' |
EvolveChance: '" + neededChangeChance + "')");
        }
    }


    //============================CHECK IF CAN ADD GENE============================\\
    public void CheckIfCanAddGene() {
        //If mother has less genes then father add a gene here else add to father. If they have the same gene count then pick
random
        if (motherGenesDic.Count < fatherGenesDic.Count)
            addGeneToMotherOrFather = 0;
```

```
else if (fatherGenesDic.Count < motherGenesDic.Count)
    addGeneToMotherOrFather = 1;
else
    addGeneToMotherOrFather = Random.Range(0, 2);



potentialAddGenes = new List<GeneBase>(GenePool.staticGenePool.primitiveGenes);

for (int v = 0; v < potentialAddGenes.Count; v++) {
    if (potentialAddGenes[v].bodyPart != Enums.BodyPart.Na) {

        //Mother genes, if same body part found then remove from potential add
        if (addGeneToMotherOrFather == 0) {
            for (int m = 0; m < motherGeneList.Count; m++) {
                if (motherGeneList[m].bodyPart == potentialAddGenes[v].bodyPart) {
                    potentialAddGenes.RemoveAt(v);
                    v--;
                    break;
                }
            }
        }
        //Father Genes
        else {
            for (int f = 0; f < fatherGeneList.Count; f++) {
                if (fatherGeneList[f].bodyPart == potentialAddGenes[v].bodyPart) {
                    potentialAddGenes.RemoveAt(v);
                    v--;
                    break;
                }
            }
        }

    }
    //For all NA bodyparts check for name. If name found then remove from potential add
    else {
        //Mother Genes
        if (addGeneToMotherOrFather == 0) {
```

```
                    if (motherGenesDic.ContainsKey(potentialAddGenes[v].nameOnly)) {
                        potentialAddGenes.RemoveAt(v);
                        v--;
                    }
                }
                //Father Genes
                else {
                    if (fatherGenesDic.ContainsKey(potentialAddGenes[v].nameOnly)) {
                        potentialAddGenes.RemoveAt(v);
                        v--;
                    }
                }
            }
        }
    }

    //============================ACTUAL ADD GENE============================\\
    public void AddGene() {
        print("Add Gene needs testing as it adds it from the list");

        GeneBase geneToAdd = potentialAddGenes[Random.Range(0, potentialAddGenes.Count)].copyGene();

        if (geneToAdd == null) {
            Debug.LogError("(" + gameObject.name + ") Trying to add a new gene but geneToAdd is null for: '" + gameObject.name
+ "' (potentialAddGenes Count:'" + potentialAddGenes.Count + "')");
            return;
        }
        if (addGeneToMotherOrFather == 0) {
            motherGeneList.Add(geneToAdd);
            motherGeneList[motherGeneList.Count - 1].geneValue = (int)Random.Range((int)motherGeneList[motherGeneList.Count -
1].newlyAddedGeneValue.x, (int)motherGeneList[motherGeneList.Count - 1].newlyAddedGeneValue.y + 1);
            motherGeneList[motherGeneList.Count - 1].InstantiateLocalGene(critterBaseRef, "Mother");
            motherGenesDic.Add(geneToAdd.nameOnly, motherGeneList[motherGeneList.Count - 1]);


            Debug.Log("(" + gameObject.name + ") Added new gene to Mother genelist: '" + geneToAdd.name + "'");

        }
```

```
        else if (addGeneToMotherOrFather == 1) {
            fatherGeneList.Add(geneToAdd);
            fatherGeneList[fatherGeneList.Count - 1].geneValue = (int)Random.Range((int)fatherGeneList[fatherGeneList.Count -
1].newlyAddedGeneValue.x, (int)fatherGeneList[fatherGeneList.Count - 1].newlyAddedGeneValue.y + 1);
            fatherGeneList[fatherGeneList.Count - 1].InstantiateLocalGene(critterBaseRef, "Father");
            fatherGenesDic.Add(geneToAdd.nameOnly, fatherGeneList[fatherGeneList.Count - 1]);

            Debug.Log("(" + gameObject.name + ") Added new gene to Father genelist: '" + geneToAdd.name + "'");
        }
        else
            Debug.LogError("(" + gameObject.name + ") AddGeneToMotherOrFather not set correctly for: '" + gameObject.name +
"'");


    }

    //============================ACTUAL REMOVE GENE============================\\
    public void RemoveGene(Dictionary<string,GeneBase> holderDic, List<GeneBase> geneList, int removeIndex) {
        //Essential safty
        if (geneList[removeIndex].essential) {
            Debug.LogError("(" + gameObject.name + ") '" + geneList[removeIndex].name + "' tried to remove an essential gene!.
This is canceled instead");
            return;
        }

        Debug.Log("(" + gameObject.name + ") '" + geneList[removeIndex] + "': has been removed from the '" +
geneList[removeIndex].motherOrFather + " Gene List'");

        geneList[removeIndex].OnRemoveGene();
        holderDic.Remove(geneList[removeIndex].nameOnly);
        geneList.RemoveAt(removeIndex);

    }

    //============================ACTUAL ADD A BODYPART============================\\
    public void AddBodyPart(GeneBase gene) {
        Debug.Log("(" + gameObject.name + ") '" + gene.name + "' has: '" + gene.bodyPartsIncrements + "' more BodyParts.");
```

```
        gene.bodyPartsAmount += gene.bodyPartsIncrements;

        //Safty
        if (gene.bodyPartsAmount > gene.minMaxBodyPartAmount.y) {
            Debug.LogError("(" + gameObject.name + ") '" + gene.name + "' added a bodyPart but now the total amount is greater
than the max allowed amount! Check if bodyPartsIncrements and or minMaxBodyPartAmount.y are set correctly.");
            gene.bodyPartsAmount = (int)gene.minMaxBodyPartAmount.y;
        }



        //Add all the geneHolders of the pool in one array to randomly select
        //Check if the body part can be added: EG: Are the requirments met: having the ones they need and not having the ones
they don't. If this is false then randomize again for a max of 10 times until a suitable gene is found and added.
    }

    //============================ACTUAL REMOVE BODYPART============================\\
    public void RemoveBodyPart(GeneBase gene) {
        Debug.Log("(" + gameObject.name + ") '" + gene.name + "' has: '" + gene.bodyPartsIncrements+ "' less BodyParts.");

        gene.bodyPartsAmount -= gene.bodyPartsIncrements;

        //Safty
        if (gene.bodyPartsAmount < gene.minMaxBodyPartAmount.x) {
            Debug.LogError("(" + gameObject.name + ") '" + gene.name + "' removed a bodyPart but now the total amount is
smaller than the min allowed amount! Check if bodyPartsIncrements and or minMaxBodyPartAmount.x are set correctly.");
            gene.bodyPartsAmount = (int)gene.minMaxBodyPartAmount.x;
        }

    }

    //============================EVOLUTION GENE AND DEVOLUTION GENE CHECK LOGICS============================\\
    public void CheckCanEvolveDevolveBodyPart(GeneBase gene, bool checkForEvolve) {

        if (checkForEvolve)
            potentialEvolutions = new List<GeneBase>();
        else
            potentialDevolutions = new List<GeneBase>();
```

```
        List<GeneBase> validGeneList = new List<GeneBase>();

        //Can the body part evolve?
        if (gene.evolvesTo.Count <= 0)
            return;


        //Check if correct gene value
        if (checkForEvolve) {
            if (gene.geneValue < gene.minEvolutionGeneValue) {
                Debug.LogWarning("(" + gameObject.name + ") Cannot evolve gene: '" + gene.name + " as geneValue not high
enough. (Needs to be: '" + gene.minEvolutionGeneValue + "' is: '" + gene.geneValue + "'");
                return;
            }
        }
        else {
            if (gene.geneValue > gene.minDevolutionGeneValue) {
                Debug.LogWarning("(" + gameObject.name + ") Cannot devolve gene: '" + gene.name + " as geneValue not low
enough. (Needs to be: '" + gene.minDevolutionGeneValue + "' is: '" + gene.geneValue + "'");
                return;
            }
        }


        //It can evolve/Devolve create list of potential evolutions/devolutions
        if (checkForEvolve) {
            for (int i = 0; i < gene.evolvesTo.Count; i++) {
                validGeneList.Add(GenePool.staticGenePool.GetGene(gene.evolvesTo[i]));
            }
        }
        else {
            for (int i = 0; i < gene.devolvesTo.Count; i++) {
                validGeneList.Add(GenePool.staticGenePool.GetGene(gene.devolvesTo[i]));
            }
        }
```

```
        //Check if all the requirments are met
        for (int i = 0; i < validGeneList.Count; i++) {

            //If critterClass is needed check if critter is the correct class. If false remove from list
            if (validGeneList[i].onlyForClass != Enums.CritterClass.Na && validGeneList[i].onlyForClass !=
critterBaseRef.critterClass) {
                Debug.LogWarning("(" + gameObject.name + ") Cannot evolve/devolve: '" + validGeneList[i].name + "' as it isn't
the correct class. (onlyForClass)");
                validGeneList.RemoveAt(i);
                i--;
            }
            //Is the wrong class
            else if (validGeneList[i].cannotBeClass != Enums.CritterClass.Na && validGeneList[i].cannotBeClass ==
critterBaseRef.critterClass) {
                Debug.LogWarning("(" + gameObject.name + ") Cannot evolve/devolve: '" + validGeneList[i].name + "' as it isn't
the correct class.(cannotBeClass)");
                validGeneList.RemoveAt(i);
                i--;
            }

            //getGeneHasOneReq Needs to go through this and check all the genes in father and mother
            else if (HasOneOfTheGenes(validGeneList[i].getGeneHasOneReq.ToArray()) == false) { //False as didn't find any of
the genes
                Debug.LogWarning("(" + gameObject.name + ") Cannot evolve/devolve: '" + validGeneList[i].name + "' as it has
non of the atLeastOne gene.");
                validGeneList.RemoveAt(i);
                i--;
            }

            //Does not have any of the cannot have genes
            else if (HasOneOfTheGenes(validGeneList[i].getGeneCannot.ToArray(), true) == true) { //True as did find a gene
                Debug.LogWarning("(" + gameObject.name + ") Cannot evolve/devolve: '" + validGeneList[i].name + "' as it has a
cannot have gene.");
                validGeneList.RemoveAt(i);
                i--;
            }

            //Has all of the genes required
```

```
            else if (HasAllOfTheGenes(validGeneList[i].getGeneReq.ToArray()) == false) { //False as did not find one of the
genes
                Debug.LogWarning("(" + gameObject.name + ") Cannot evolve/devolve: '" + validGeneList[i].name + "' as it has
not all req genes.");
                validGeneList.RemoveAt(i);
                i--;
            }

            //Check if correct bodyParts (egthe current gene cannot have less then the minBodyPartAmount of the evolve/devolve
and not more then the Max of the evolve/devolve)
            else if (validGeneList[i].minMaxBodyPartAmount.x > gene.bodyPartsAmount || validGeneList[i].minMaxBodyPartAmount.y
< gene.bodyPartsAmount) { //False as did not find one of the genes
                if (!validGeneList[i].ignoreBodyPartAmount) {
                    Debug.LogWarning("(" + gameObject.name + ") Cannot evolve/devolve: '" + validGeneList[i].name + "' as it
has either a too much or too little bodypart amount.");
                    validGeneList.RemoveAt(i);
                    i--;
                }
            }


        }

        //Create either an evolve or devolve list
        if (checkForEvolve)
            potentialEvolutions = validGeneList;
        else
            potentialDevolutions = validGeneList;
    }


    //============================ACTUAL EVOLVE/DEVOLVE GENE============================\\
    public void EvolveDevolveBodyPart(GeneBase geneToReplace, Dictionary<string,GeneBase> holderDic, List<GeneBase>
holderList, int indexToReplace, bool doesGeneEvolve) {
        GeneBase newGene = null;
        string removeAtMotherOrFather = geneToReplace.motherOrFather;

        if (doesGeneEvolve) {
```

```
            newGene = potentialEvolutions[(int)Random.Range(0, potentialEvolutions.Count)].copyGene();
            Debug.Log("(" + gameObject.name + ") '" + geneToReplace.nameOnly + "' has been evolved into: '" + newGene.nameOnly
+ "' in the '" + removeAtMotherOrFather + " GenesList'");
            newGene.geneValue = (int)Random.Range((int)newGene.newlyEvolvedGeneValue.x, (int)newGene.newlyEvolvedGeneValue.y +
1);
        }
        else {
            newGene = potentialDevolutions[(int)Random.Range(0, potentialDevolutions.Count)].copyGene();
            Debug.Log("(" + gameObject.name + ") '" + geneToReplace.nameOnly + "' has been devolved into: '" +
newGene.nameOnly + "' in the '" + removeAtMotherOrFather + " GenesList'");
            newGene.geneValue = (int)Random.Range((int)newGene.newlyDevolvedGeneValue.x, (int)newGene.newlyDevolvedGeneValue.y
+ 1);
        }


        //BodyPart Amount
        newGene.bodyPartsAmount = ReturnProperBodyPartAmount(geneToReplace, newGene);

        //Evolve/Devolve Gene logics
        if (doesGeneEvolve)
            holderList[indexToReplace].OnEvolveGene(newGene.nameOnly, holderDic, holderList, removeAtMotherOrFather);
        else
            holderList[indexToReplace].OnDevolveGene(newGene.nameOnly, holderDic, holderList, removeAtMotherOrFather);

        //Remove gene from dic and add new
        holderDic.Remove(geneToReplace.nameOnly);

        //Remove gene and add new gene at same index
        holderList.RemoveAt(indexToReplace);
        holderList.Insert(indexToReplace, newGene);

        //Initiate again
        if (removeAtMotherOrFather == "Mother") {
            holderList[indexToReplace].InstantiateLocalGene(critterBaseRef, "Mother");
        }
        else {
            holderList[indexToReplace].InstantiateLocalGene(critterBaseRef, "Father");
        }
```

```
        //Add the new gene to Dic
        holderDic.Add(holderList[indexToReplace].nameOnly, holderList[indexToReplace]);

    }

    int ReturnProperBodyPartAmount(GeneBase geneToReplace, GeneBase newGene) {
        //Safty if it does not check for bodypart amounts
        if (geneToReplace.bodyPartsAmount < newGene.minMaxBodyPartAmount.x)
            return (int)newGene.minMaxBodyPartAmount.x;
        else if (geneToReplace.bodyPartsAmount > newGene.minMaxBodyPartAmount.y)
            return (int)newGene.minMaxBodyPartAmount.y;
        else { //check if the increments are correct
            float calcFloatTemp = geneToReplace.bodyPartsAmount / newGene.bodyPartsIncrements;
            int calcintTemp = (int)calcFloatTemp;

            //Check which bodyPart amount is closest to be assigned
            if (calcFloatTemp - calcintTemp != 0) {
                Debug.Log("(" + gameObject.name + ") '" + geneToReplace.name + "' has been evolved into: '" + newGene.name +
"' but the body part amount increments don't match up so the closest bodyPart amount has been assigned.");

                int checkAmount = (int)newGene.minMaxBodyPartAmount.x;
                int closestBodyPartAmount = 0;
                float diffrence = 99999;
                float newDiffrence = 0;

                while (checkAmount <= (int)newGene.minMaxBodyPartAmount.y) {
                    if (checkAmount < calcFloatTemp)
                        newDiffrence = calcFloatTemp - checkAmount;
                    else
                        newDiffrence = checkAmount - calcFloatTemp;

                    //Found a new closest incriment
                    if (newDiffrence < diffrence) {
                        diffrence = newDiffrence;
                        closestBodyPartAmount = checkAmount;
                    }
```

```
                checkAmount += newGene.bodyPartsIncrements;
            }
            return checkAmount;

        }
        else {
            return geneToReplace.bodyPartsAmount;
        }

    }

}



//============================CREATE EXPRESSED GENES ARRAY============================\\
//Checks all Mother and father genes and creates an array of all the genes to be expressed
void CreateExpressedGenesArray() {
    List<GeneBase> tempMotherGenes = new List<GeneBase>(motherGeneList);
    List<GeneBase> tempFatherGenes = new List<GeneBase>(fatherGeneList);
    List<GeneBase> geneBaseList = new List<GeneBase>();


    //Check for same names and pick father and/or mother Gene
    for (int m = 0; m < tempMotherGenes.Count; m++) {
        for (int f = 0; f < tempFatherGenes.Count; f++) {

            if (tempMotherGenes[m].name == tempFatherGenes[f].name) {
                geneBaseList.Add(ReturnExpressedGene(tempMotherGenes[m], tempFatherGenes[f]));

                //Remove found gene
                tempMotherGenes.RemoveAt(m);
                tempFatherGenes.RemoveAt(f);
                m--;
                f--;
                break;
            }
```

```
            }
        }

        //Check for same bodyPart type. If true this means they do not share the same name and are most likely diffrent
evolutions (Note this does not check if the gene can actually be expressed)
        for (int m = 0; m < tempMotherGenes.Count; m++) {
            if (tempMotherGenes[m].bodyPart != Enums.BodyPart.Na) {
                for (int f = 0; f < tempFatherGenes.Count; f++) {

                    if (tempMotherGenes[m].bodyPart == tempFatherGenes[f].bodyPart) {
                        //Check if both dominant or both resessive
                        if ((tempMotherGenes[m].dominant && tempFatherGenes[f].dominant) || (!tempMotherGenes[m].dominant &&
!tempFatherGenes[f].dominant)) {

                            if (tempMotherGenes[m].tier > tempFatherGenes[f].tier) {
                                Debug.LogWarning("(" + gameObject.name + ") On Express gene: Mother gene '" +
tempMotherGenes[m].name + "' father over '" + tempFatherGenes[f].name + "' as the mother tier is higher.");
                                geneBaseList.Add(tempMotherGenes[m].copyGene());
                            }
                            else if (tempMotherGenes[m].tier < tempFatherGenes[f].tier) {
                                Debug.LogWarning("(" + gameObject.name + ") On Express gene: Father '" +
tempFatherGenes[f].name + "' mother over '" + tempMotherGenes[m].name + "' as the father tier is higher.");
                                geneBaseList.Add(tempFatherGenes[f].copyGene());
                            }
                            else if (tempMotherGenes[m].tier == tempFatherGenes[f].tier) {
                                int randomMotherFather = Random.Range(0, 2);

                                if (randomMotherFather == 0) {
                                    Debug.LogWarning("(" + gameObject.name + ") On Express gene: Randomly picked mother gene
'" + tempMotherGenes[m].name + "' over father gene '" + tempFatherGenes[f].name + "' as tiers are the same level.");
                                    geneBaseList.Add(tempMotherGenes[m].copyGene());
                                }
                                else {
                                    Debug.Log("(" + gameObject.name + ") On Express gene: Randomly picked father gene '" +
tempFatherGenes[f].name + "' over mother gene '" + tempMotherGenes[m].name + "' as tiers are the same level.");
                                    geneBaseList.Add(tempFatherGenes[f].copyGene());
                                }
                            }
```

```
                        else {
                            Debug.LogError("(" + gameObject.name + ") Something went wrong with expressing same body part
genes. Genes involved: '" + tempMotherGenes[m].name + "' (mother) & '" + tempFatherGenes[f].name + "' (father)");
                        }
                    }

                    //One is dominant the other resesive
                    //Mother dominant = express this gene
                    else if (tempMotherGenes[m].dominant) {
                        Debug.LogWarning("(" + gameObject.name + ") On Express gene: Mother gene '" +
tempMotherGenes[m].name + "' father over '" + tempFatherGenes[f].name + "' as the mother gene is dominant.");
                        geneBaseList.Add(tempMotherGenes[m].copyGene());
                    }
                    //Mother father = express this gene
                    else if (tempFatherGenes[f].dominant) {
                        Debug.LogWarning("(" + gameObject.name + ") On Express gene: Father '" + tempFatherGenes[f].name +
"' mother over '" + tempMotherGenes[m].name + "' as the mother gene is dominant.");

                        geneBaseList.Add(tempFatherGenes[f].copyGene());
                    }
                    else {
                        Debug.LogError("(" + gameObject.name + ") Something went wrong with expressing same body part
genes. Genes involved: '" + tempMotherGenes[m].name + "' (mother) & '" + tempFatherGenes[f].name + "' (father)");
                    }


                    //Remove found gene
                    tempMotherGenes.RemoveAt(m);
                    tempFatherGenes.RemoveAt(f);
                    m--;
                    f--;
                    break;
                }
            }
        }
    }
```

```
        //Any other genes
        for (int m = 0; m < tempMotherGenes.Count; m++) {
            geneBaseList.Add(tempMotherGenes[m].copyGene());
            tempMotherGenes.RemoveAt(m);
            m--;
        }
        for (int f = 0; f < tempFatherGenes.Count; f++) {
            geneBaseList.Add(tempFatherGenes[f].copyGene());
            tempFatherGenes.RemoveAt(f);
            f--;
        }

        //Safty
        if (tempMotherGenes.Count != 0)
            Debug.LogError("(" + gameObject.name + ") Something went wrong with '" + gameObject.name + "' as mother gene list
still has elements");
        if (tempFatherGenes.Count != 0)
            Debug.LogError("(" + gameObject.name + ") Something went wrong with '" + gameObject.name + "' as father gene list
still has elements");


        //CreateExpressed Gene List
        expressedGeneList = new List <GeneBase>(geneBaseList);

        //Create expressedGenes Dic and Instantiate the local genes for frist time
        for (int i = 0; i < expressedGeneList.Count; i++) {
            expressedGeneList[i].InstantiateLocalGene(critterBaseRef, "ExpressedGene");
            expressedGenesDic.Add(expressedGeneList[i].nameOnly, expressedGeneList[i]);
        }

        //Check if all given genes are valid
        CheckIfExpressedGenesAreValid(expressedGeneList);


    }
```

```
    //============================RETURN EXPRESSED GENE============================\\
    //If only one is dominant return that gene else take a avarege of both
    GeneBase ReturnExpressedGene(GeneBase motherGene, GeneBase fatherGene) {
        GeneBase expressedGene;

        if (motherGene.dominant == true) {
            //Both Dominant
            if (fatherGene.dominant == true) {
                expressedGene = motherGene.copyGene();
                expressedGene.geneValue = (motherGene.geneValue + fatherGene.geneValue) / 2;
                expressedGene.bodyPartsAmount = ReturnBodyPartAmount(motherGene.bodyPartsAmount, fatherGene.bodyPartsAmount,
motherGene.bodyPartsIncrements);
            }
            //Only Mother gene Dominant
            else
                expressedGene = motherGene.copyGene();
        }
        //Only Father gene Dominant
        else if (fatherGene.dominant == true) {
            expressedGene = fatherGene.copyGene();
        }
        else {
            //Both Recessive
            expressedGene = motherGene.copyGene();
            expressedGene.geneValue = (motherGene.geneValue + fatherGene.geneValue) / 2;
            expressedGene.bodyPartsAmount = ReturnBodyPartAmount(motherGene.bodyPartsAmount, fatherGene.bodyPartsAmount,
motherGene.bodyPartsIncrements);
        }

        return expressedGene;
    }

    //Calculate bodyparts amount if both genes are dominant or resesive
    int ReturnBodyPartAmount(int motherBodyPartAmount, int fatherBodyPartAmount, int increments) {
        if (motherBodyPartAmount == fatherBodyPartAmount)
            return motherBodyPartAmount;

        //If it is only one increment away
```

```
        if ((motherBodyPartAmount + increments == fatherBodyPartAmount) || (fatherBodyPartAmount + increments ==
motherBodyPartAmount)) {
            int randomMotherFather = Random.Range(0, 2);

            if (randomMotherFather == 0)
                return motherBodyPartAmount;
            else
                return fatherBodyPartAmount;
        }

        //if it is not equal amount and more then 1 increment away
        else {
            return (((motherBodyPartAmount / increments) + (fatherBodyPartAmount / increments)) / 2) * increments;
        }
    }

    //============================CHECK IF ALL EXPRESSED GENES ARE VALID============================\\
    //If an expressed gene is not valid then either replace it or if this is not possible remove it or set gene value to 0 if
it is an essential gene
    void CheckIfExpressedGenesAreValid(List<GeneBase> createdExpressedGenes) {
        for (int i = 0; i < createdExpressedGenes.Count; i++) {


            //Gene is missing one or more of the requirements.
            if (IsGeneValid(createdExpressedGenes[i], expressedGenesDic) == false) {
                bool replacementFound = false;

                //Search for replacement if is a bodypart
                if (createdExpressedGenes[i].bodyPart != Enums.BodyPart.Na) {

                    //Check if the expressed gene is in Mother list. If not then go through list and find the gene with the
same bodypart
                    if (!motherGenesDic.ContainsKey(createdExpressedGenes[i].nameOnly)) {
                        for (int m = 0; m < motherGeneList.Count; m++) {

                            //Check if same body part, if this is true then it might be able to replace by this
                            if (createdExpressedGenes[i].bodyPart == motherGeneList[m].bodyPart) {
```

```
                            //Check if the potential replace gene is valid
                            if (IsGeneValid(motherGeneList[m], expressedGenesDic)) {
                                Debug.Log("(" + gameObject.name + ") '" + createdExpressedGenes[i].name + "' has been
replaced with: '" + motherGeneList[m].name + "' as it was not a valid gene to express (Gene was missing one or more of the
requirements)");
                                expressedGenesDic.Remove(createdExpressedGenes[i].nameOnly);
                                createdExpressedGenes.RemoveAt(i);
                                createdExpressedGenes.Insert(i, motherGeneList[m].copyGene());

                                expressedGenesDic.Add(createdExpressedGenes[i].nameOnly, createdExpressedGenes[i]);
                                expressedGenesDic[createdExpressedGenes[i].nameOnly].InstantiateLocalGene(critterBaseRef,
"ExpressedGene");
                                replacementFound = true;
                            }
                            break;
                        }
                    }
                }



                //If did not find replacement in mothergene
                if (!replacementFound) {
                    if (!fatherGenesDic.ContainsKey(createdExpressedGenes[i].nameOnly)) {

                        for (int f = 0; f < fatherGeneList.Count; f++) {

                            //Check if same body part
                            if (createdExpressedGenes[i].bodyPart == fatherGeneList[f].bodyPart) {
                                //Check if same bodypart but not same name (Else it is the same as expressed gene)

                                //Check if the potential replace gene is valid
                                if (IsGeneValid(fatherGeneList[f], expressedGenesDic)) {
                                    Debug.Log("(" + gameObject.name + ") '" + createdExpressedGenes[i].name + "' has been
replaced with: '" + fatherGeneList[f].name + "' as it was not a valid gene to express (Gene was missing one or more of the
requirements)");
                                    expressedGenesDic.Remove(createdExpressedGenes[i].nameOnly);
                                    createdExpressedGenes.RemoveAt(i);
```

```
                                    createdExpressedGenes.Insert(i, fatherGeneList[f].copyGene());

                                    expressedGenesDic.Add(createdExpressedGenes[i].nameOnly, createdExpressedGenes[i]);

expressedGenesDic[createdExpressedGenes[i].nameOnly].InstantiateLocalGene(critterBaseRef, "ExpressedGene");
                                    replacementFound = true;
                                }
                                break;
                            }
                        }
                    }
                }

            }

            if (!replacementFound) {
                if (createdExpressedGenes[i].essential) {
                    Debug.LogError("(" + gameObject.name + ")" + errorText + " NOTE: The gene is essential and could not
be removed so gene value is set to 0");
                    createdExpressedGenes[i].geneValue = 0;
                    expressedGenesDic[createdExpressedGenes[i].nameOnly].geneValue = 0;
                }
                else {
                    Debug.LogError("(" + gameObject.name + ") Gene removed because: " + errorText);
                    expressedGenesDic.Remove(createdExpressedGenes[i].nameOnly);
                    createdExpressedGenes.RemoveAt(i);
                    i--;
                }
            }
        }


    }
}

//============================CHECK IF GENE IS VALID============================\\
//--------------------------Check gene is valid in given list--------------------------\\
bool IsGeneValid(GeneBase gene, Dictionary<string,GeneBase> customSearchDic) {
```

```
        if (gene.onlyForClass != Enums.CritterClass.Na && gene.onlyForClass != critterBaseRef.critterClass) {
            errorText = "Gene: '" + gene.name + "' on Critter: '" + gameObject.name + "' is not the correct class for the
critter! (onlyForClass) (CurrentClass: '" + critterBaseRef.critterClass + "' OnlyForClass: '" + gene.onlyForClass + "')";
            return false;
        }
        else if (gene.cannotBeClass != Enums.CritterClass.Na && gene.cannotBeClass == critterBaseRef.critterClass) {
            errorText = "Gene: '" + gene.name + "' on Critter: '" + gameObject.name + "' is not the correct class for the
critter! (cannotBeClass) (CurrentClass: '" + critterBaseRef.critterClass + "' CannotBeClass: '" + gene.cannotBeClass + "')";
            return false;
        }
        else if (HasOneOfTheGenes(gene.getGeneHasOneReq.ToArray(), false, true, customSearchDic) == false) {
            errorText = "Gene: '" + gene.name + "' on Critter: '" + gameObject.name + "' does not at least has one of the
required genes!";
            return false;
        }
        else if (HasOneOfTheGenes(gene.getGeneHasOneReq.ToArray(), true, true, customSearchDic) == true) {
            errorText = "Gene: '" + gene.name + "' on Critter: '" + gameObject.name + "' has one of the cannot genes!";
            return false;
        }
        else if (HasAllOfTheGenes(gene.getGeneHasOneReq.ToArray(), true, customSearchDic) == false) {
            errorText = "Gene: '" + gene.name + "' on Critter: '" + gameObject.name + "' misses at least one of the required
genes!";
            return false;
        }
        else {
            errorText = "";
            return true;
        }

    }

    //----------------------------Check gene has at least one of req or cannot----------------------------\\
    //For checking if has any of the req genes and has any of the cannot genes
    public bool HasOneOfTheGenes(string[] genes, bool shouldnotHave = false, bool searchCustomGeneBase = false,
Dictionary<string,GeneBase> searchGeneBase = null) {
        if (genes.Length <= 0 && !shouldnotHave) //Ignore as list is to small
            return true;
```

```
        else if (genes.Length <= 0 && shouldnotHave) //Ignore as list is to small
            return false;


        bool geneFound;

        //Check if any of the genes given is found if list has at least 1 elements
        for (int i = 0; i < genes.Length; i++) {
            if (!searchCustomGeneBase) {

                //Check if gene exists in mother Gene Dictonary
                geneFound = motherGenesDic.ContainsKey(genes[i]);

                if (geneFound == true)
                    return true;

                //Check if gene exists in father Gene Dictonary
                geneFound = fatherGenesDic.ContainsKey(genes[i]);
                if (geneFound == true)
                    return true;
            }
            //Searching custom genebase
            else {
                geneFound = searchGeneBase.ContainsKey(genes[i]);

                if (geneFound == true)
                    return true;
            }
        }

        return false;
    }

    //-----------------------------Check gene has all req-----------------------------\\
    //For checking if all the req genes are present
    public bool HasAllOfTheGenes(string[] genes, bool searchCustomGeneBase = false, Dictionary<string, GeneBase>
searchGeneBase = null) {
```

```
    //Check if ALL of the genes given are found
    for (int i = 0; i < genes.Length; i++) {
        bool geneFound = false;

        if (!searchCustomGeneBase) {
            geneFound = motherGenesDic.ContainsKey(genes[i]);

            //Gene was not in mother list
            if (!geneFound) {
                geneFound = fatherGenesDic.ContainsKey(genes[i]);
            }
        }
        else {
            for (int c = 0; c < searchGeneBase.Count; c++) {
                geneFound = searchGeneBase.ContainsKey(genes[i]);
            }
        }

        //Gene was not found in neither mother nor father (or not in custom) thus return false
        if (!geneFound) {
            return false;
        }
    }
    //All genes where found
    return true;
}


//============================DNA STRING============================\\
void CreateDNAString() {
    //Set Capitial Letters Mother
    for (int m = 0; m < motherGeneList.Count; m++) {
        if (motherGeneList[m].dominant) {
            motherGeneList[m].geneLetter = motherGeneList[m].geneLetter.ToUpper();
        }
        else {
            motherGeneList[m].geneLetter = motherGeneList[m].geneLetter.ToLower();
        }
```

```
    motherGeneList[m].name = motherGeneList[m].nameOnly + "(" + motherGeneList[m].geneLetter + ")";
}

//Set Capitial Letters Mother
for (int f = 0; f < fatherGeneList.Count; f++) {

    if (fatherGeneList[f].dominant) {
        fatherGeneList[f].geneLetter = fatherGeneList[f].geneLetter.ToUpper();
    }
    else {
        fatherGeneList[f].geneLetter = fatherGeneList[f].geneLetter.ToLower();
    }
    fatherGeneList[f].name = fatherGeneList[f].nameOnly + "(" + fatherGeneList[f].geneLetter + ")";
}

List <GeneBase> tempMotherGenes = new List<GeneBase>(motherGeneList);
List<GeneBase> tempFatherGenes = new List<GeneBase>(fatherGeneList);


for (int m = 0; m < tempMotherGenes.Count; m++) {
    for (int f = 0; f < tempFatherGenes.Count; f++) {

        if (tempMotherGenes[m].geneLetter.ToLower() == tempFatherGenes[f].geneLetter.ToLower()) {
            if (tempMotherGenes[m].dominant && !tempFatherGenes[f].dominant) {
                DNA += tempMotherGenes[m].geneLetter;
                DNA += tempFatherGenes[f].geneLetter;
            }
            else if (!tempMotherGenes[m].dominant && tempFatherGenes[f].dominant) {
                DNA += tempFatherGenes[f].geneLetter;
                DNA += tempMotherGenes[m].geneLetter;
            }
            else {
                DNA += tempMotherGenes[m].geneLetter;
                DNA += tempFatherGenes[f].geneLetter;
            }

            //Remove found gene
            tempMotherGenes.RemoveAt(m);
```

```
                tempFatherGenes.RemoveAt(f);
                m--;
                f--;

                break;
            }
        }
    }

    //Remaining not shared genes
    for (int m = 0; m < tempMotherGenes.Count; m++) {
        DNA += tempMotherGenes[m].geneLetter;
    }
    for (int f = 0; f < tempFatherGenes.Count; f++) {
        DNA += tempFatherGenes[f].geneLetter;
    }
}


//============================EXPRESS THE EXPRESSGENES============================\\
void UpdateCritterBaseValues() {

    //Essential
    for (int i = 0; i < expressedGeneList.Count; i++) {
        if (expressedGeneList[i].essential == true)
            expressedGeneList[i].GeneExpression();
    }

    //Non-Essential
    for (int i = 0; i < expressedGeneList.Count; i++) {
        if (expressedGeneList[i].essential == false)
            expressedGeneList[i].GeneExpression();
    }
}

//============================CREATE UPDATABLE GENE ARRAY============================\\
void CreateUpdateList() {
    List<GeneBase> tempUpdatable = new List<GeneBase>();
```

```
    for (int i = 0; i < expressedGeneList.Count; i++) {
        if (expressedGeneList[i].updates == true) {
            tempUpdatable.Add(expressedGeneList[i]);
        }
    }

    updatableGenes = tempUpdatable.ToArray();

}

//============================CREATE MATTING GENE============================\\
public void CreateMatingGene() {
    List<GeneBase> tempMotherGenes = new List<GeneBase>(motherGeneList);
    List<GeneBase> tempFatherGenes = new List<GeneBase>(fatherGeneList);
    List<GeneBase> randomizedGenes = new List<GeneBase>();

    for (int m = 0; m < tempMotherGenes.Count; m++) {
        for (int f = 0; f < tempFatherGenes.Count; f++) {

            if (tempMotherGenes[m].name == tempFatherGenes[f].name) {
                //0 = mother, 1 = father
                int parent = Random.Range(0, 2);

                //Grab random mother or father
                if (parent == 0)
                    randomizedGenes.Add(tempMotherGenes[m].copyGene());
                else
                    randomizedGenes.Add(tempFatherGenes[f].copyGene());

                //Remove found gene
                tempMotherGenes.RemoveAt(m);
                tempFatherGenes.RemoveAt(f);
                m--;
                f--;
                break;
            }
        }
```

```
}

//Remaining Genes not shared
List<GeneBase> remainingGenes = new List<GeneBase>();

//Remaining Bodypart Genes that contest each other (Picks a body part gene of mother or father but never both)
for (int m = 0; m < tempMotherGenes.Count; m++) {
    if (tempMotherGenes[m].bodyPart != Enums.BodyPart.Na) {
        for (int f = 0; f < tempFatherGenes.Count; f++) {
            if (tempMotherGenes[m].bodyPart == tempFatherGenes[f].bodyPart) {
                //0 = mother, 1 = father
                int parent = Random.Range(0, 2);

                //Grab random mother or father
                if (parent == 0)
                    remainingGenes.Add(tempMotherGenes[m].copyGene());
                else
                    remainingGenes.Add(tempFatherGenes[f].copyGene());

                //Remove found gene
                tempMotherGenes.RemoveAt(m);
                tempFatherGenes.RemoveAt(f);
                m--;
                f--;
                break;
            }
        }
    }
}

//All remaining genes are added not accociated with body parts or not shared
for (int m = 0; m < tempMotherGenes.Count; m++) {
    remainingGenes.Add(tempMotherGenes[m].copyGene());
}
for (int f = 0; f < tempFatherGenes.Count; f++) {
    remainingGenes.Add(tempFatherGenes[f].copyGene());
}
```

```
        /*
        //Inharent Exclusive Parent Trait (Chance of getting a trait that only the mother or father has)
        int randomInherent = Random.Range(0, 101);
        for (int i = 0; i < remainingGenes.Count; i++) {
            randomInherent = Random.Range(0, 101);

            //Gets exclusive trait
            if (randomInherent <= inharentExclusiveGeneParent) {
                randomizedGenes.Add(remainingGenes[i].copyGene());
                Debug.Log("(" + gameObject.name + ") '" + gameObject.name + "' got a parent exclusive Gene: '" +
remainingGenes[i].name + "'" );

                remainingGenes.RemoveAt(i);
                i--;
            }
        }
        */

        mattingGene = randomizedGenes.ToArray();
    }


    //============================UPDATE IDENTIFIER============================\\
    public void UpdateParentToNew(string oldIdentifier, string newIdentifier) {
        //If the class changes it will be done through a gene expression thus genes need to be changed afterwards
        foreach (KeyValuePair<string, GeneBase> gene in motherGenesDic) {
            if (gene.Value.parent == oldIdentifier)
                gene.Value.parent = newIdentifier;
        }

        foreach (KeyValuePair<string, GeneBase> gene in fatherGenesDic) {
            if (gene.Value.parent == oldIdentifier)
                gene.Value.parent = newIdentifier;
        }

        foreach (KeyValuePair<string, GeneBase> gene in expressedGenesDic) {
            if (gene.Value.parent == oldIdentifier)
                gene.Value.parent = newIdentifier;
```

```
        }

    }

    //============================GENE FINDER============================\\
    bool FindGene(string key, Dictionary<string, GeneBase> dic) {
        if (dic.TryGetValue(key, out foundGene)) {
            return true;
        }
        else {
            foundGene = null;
            return false;
        }
    }


    //============================SET REFERENCES============================\\
    void SetReferences() {
        if (critterBaseRef == null)
            critterBaseRef = GetComponent<CritterBase>();

        if (critterBaseRef == null)
            Debug.LogError("(" + gameObject.name + ") '" + gameObject.name + "' has no CritterBase component!");

    }

}
```