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

public class NormClass : MonoBehaviour {
    //Reference
    Globals globalsRef;
    NormMovement normMovementRef;

    //Norm Bio
    public int ID = 1;
    public string gender;
    public bool isBorn = true;

    //DNA
    public string[][] DNA;
    public string stringDNA = "";

    //Genes
    public string[] speedGene = new string[] {"S", "Dom:##:", "Dom:##:"}; // {GeneLetter, Dominant:Value(Nomrelized):ID,
Recessive:Value(Nomrelized):ID}
    public string[] lifespanGene = new string[] {"L", "Dom:##:", "Dom:##:"};
    public string[] hungerRateGene = new string[] {"R", "Dom:##:", "Dom:##:"};
    public string[] hungerMaxGene = new string[] {"H", "Dom:##:", "Dom:##:"};


    //Actual attributes
    public float normSpeed = 0;
    float normLifespan = 0; //inseconds
    float normHungerRate = 0; //Rate of which hunger decreases
    float normHungerMax = 0; //Aka stomach content

    //Standard values (* (Genevalue / 10)) Set here default which will be used for calculation
    float normDefaultSpeed = 2f;
    float normDefaultLifespan = 60; //inseconds
    float normDefaultHungerRate = 4f; //Rate of which hunger decreases
    float normDefaultHungerMax = 100; //Aka stomach content

    //Dynamic attributes
    public float currentHunger; //In %
    public float currentLifeLeft;

    //Mating Male
    public NormClass matingPartner; //Male only
    public float hungerMatingThreshold;
    float matingDistanceThreshold = 10; // How close needs a female be before it becomes a potential mating target
    float sexDistanceThreshold = 0.1f; // How close needs a female be before sex is enabled
    //public bool followingMate = false;
    float maleMatingCoolDownCal = 10;


    //Mating Female
    public NormClass babyNormPrefab;
    public int matingAmountLeft = 20; //How many times can a female mate CURRENTLY ABSOLITE
    float femaleMatingCoolDownCal = 5;


    //General Mating
    float matingCooldown; // How long it will take until male will search for a female to mate again. Should be linked to life span e.g. 10% if the lifespan is the
cooldown time
    public float currentMatingCooldown;
    bool canMate = false;
    float genderThreshold = 0.8f; //When will genders be random and when will it be controlled. Currently if at least 80% of the checked gender is that gender
than it will be randomised

    //General go to target
    public  bool turnActive = false; //If is true it will not go towards target

    //Predor
    public bool inDanger = false;

    //Mutation
    bool hasMutated = false;
    float mutationTime;
    int maxMutationAmount = 3;

//Movement
public GameObject moveTarget; //Used to set rotation

//Food
public bool isHungry = false;
float foodDistanceThreshold = 5; //Dynamicly set
NormFoodClass foodTarget;

//Icons
public GameObject deathSkull;
public GameObject birthIcon;
public GameObject DNAicon;

void Awake(){
    SetReferences ();
}

// Use this for initialization
void Start () {
    if (isBorn == false) {
        //Create the Array of Arrays
        CreateDNA ();

        //------------------Set ID nr------------------
        globalsRef.normTotalAmount += 1;
        ID = globalsRef.normTotalAmount;

        //Create random genes if born false
        for (int i = 0; i < DNA.Length; i++) {
            DNA [i] [1] = Random.Range(0,2) + ":" + SetD2(Random.Range(8,13)) + ":" + SetD4(ID);
            DNA [i] [2] = Random.Range(0,2) + ":" + SetD2(Random.Range(8,13)) + ":" + SetD4(ID);
        }

        //------------------Creates the DNA string for visual reference------------------
        CreateStringDNA ();
```

```
        //------------------Set random rotation------------------
        transform.eulerAngles = new Vector3 (0, 0, Random.Range(0, 360));
    }

    //------------------Calculate actual var with genes and default value------------------
    SetStaticValues();

    //------------------Set dynamic var start values to max------------------
    SetStartDynamicValues();

    //------------------Set name------------------
    name = "N:" + gender + "" + SetD4(ID);

    //------------------Set colour------------------
    if (gender == "Male") {
        GetComponent<SpriteRenderer> ().color = Color.blue;
        GetComponent<TrailRenderer> ().material.SetColor("_TintColor", Color.blue);
    } else {
        GetComponent<SpriteRenderer> ().color = Color.red;
        GetComponent<TrailRenderer> ().material.SetColor("_TintColor", Color.red);
    }
}


void Update () {
    //------------------Updates dynamic vars------------------
    UpdateDynamicValues();

    //------------------Check if Norm should die------------------
    DeathCheckF();


    //------------------AI behaviours------------------
    if (gender == "Male")
        maleBehaviour ();
    else
        femaleBehaviour ();

    //------------------Check if should mutate------------------
    if (hasMutated == false)
        MutateGene ();
}


////////////////////////////////////-----AI-----\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
//====================MALE AI MOVE BEHAVIOURS====================
void maleBehaviour(){
    //------------------Male Mating Cooldown------------------
    //Calculate current cooldown timer
    if (canMate == false) {
        currentMatingCooldown -= Time.deltaTime;
        if (currentMatingCooldown <= 0) {
            canMate = true;
            currentMatingCooldown = matingCooldown;
        }
    }

    //------------------Male in danger------------------
    if (inDanger == true) {
        return;
    }

    //------------------Male seek food if hungry------------------
    if (isHungry == true && turnActive == false) {
        moveTarget = null;
        SeekFood ();
        //print ("isHungry");
        if (foodTarget != null)
            moveTarget = foodTarget.gameObject;
        return;
    }
    //------------------Male seek mate if not hungry------------------
    if (canMate == true && turnActive == false && globalsRef.allowedToMate == true) {
        moveTarget = null;
        SeekMateMale ();
        if (matingPartner != null)
            moveTarget = matingPartner.gameObject;
        return;
    }

    moveTarget = null; //Sets move to target to null
}


//====================FEMALE AI MOVE BEHAVIOURS====================
void femaleBehaviour(){
    //------------------Female Mating Cooldown------------------
    if (matingAmountLeft > 0 && canMate == false) {
        currentMatingCooldown -= Time.deltaTime;
        if (currentMatingCooldown <= 0) {
            canMate = true;
            currentMatingCooldown = matingCooldown;
        }
    }

    //------------------Female in danger------------------
    if (inDanger == true) {
        return;
    }

    //------------------Female seek food if hungry------------------
    if (isHungry == true && turnActive == false) {
        moveTarget = null;
        SeekFood ();
        if (foodTarget != null)
            moveTarget = foodTarget.gameObject;
        return;
    }
    moveTarget = null; //Sets move to target to null
}
```

```
//====================SEEK FOOD IF HUNGRY====================
void SeekFood(){
    foodTarget = null;
    NormFoodClass potentialFood;
    NormFoodClass closest = null;
    float distance = Mathf.Infinity;
    Vector3 position = transform.position;

    for (int i = 0; i < globalsRef.normFoodsL.Count; i++) {
        if (globalsRef.normFoodsL [i].isActive == true) {
            potentialFood = globalsRef.normFoodsL [i];
            Vector3 diff = potentialFood.transform.position - position;
            float curDistance = diff.sqrMagnitude;
            if (curDistance < distance) {
                closest = potentialFood;
                distance = curDistance;
            }
        }
    }

    if (distance <= foodDistanceThreshold) {
        foodTarget = closest;
        return;
    }
    if (distance > foodDistanceThreshold) {
        foodTarget = null;  //Food is not close enough
        return;
    }
    foodTarget = null; //There is no food
    return;
}

//====================MALE SEEK MATE AND ENABLE MATING(SEX)====================
void SeekMateMale(){
    NormClass potentialMates;
    NormClass closest = null;
    float distance = Mathf.Infinity;
    Vector3 position = transform.position;

    for (int i = 0; i < globalsRef.normFemalesL.Count; i++) {
        potentialMates = globalsRef.normFemalesL [i];
        if (potentialMates.canMate == true){
            Vector3 diff = potentialMates.transform.position - position;
            float curDistance = diff.sqrMagnitude;
            if (curDistance < distance) {
                closest = potentialMates;
                distance = curDistance;
            }
        }
    }

    if (distance <= matingDistanceThreshold) {
        matingPartner = closest; //Set mating target
        if (matingPartner != null && distance <= sexDistanceThreshold) //Actually have sex
            MaleMating ();
        return;
    }

    if (distance > matingDistanceThreshold) {
        matingPartner = null; //Mating partner is not close enough
        return;
    }
    matingPartner = null; //There are no females
}

//====================MATING MALE (SEX)====================
//Only male calls the mating
void MaleMating(){
    if (canMate == true && matingPartner != null) {
        normMovementRef.isMating = true;

        List<string> maleGenes = new List<string>();

        for (int i = 0; i < DNA.Length; i++) {
            maleGenes.Add (DNA [i] [(int)Random.Range (1, 3)]);
        }

        matingPartner.FemaleMating (maleGenes);
        canMate = false; //Activates the matting cooldown

        //Random Direction
        transform.eulerAngles = new Vector3 (0, 0, Random.Range(0, 360));
        return;
    }
    if (matingPartner == null) {
        print ("No partner set");
        return;
    }
    if (canMate == false){
        print ("Current in cooldown");
        return;
    }
}

//====================MATING FEMALE (SEX)====================
//Called by Male
public void FemaleMating(List<string> maleGenes){
    if (gender == "Female" && canMate == true) {
        normMovementRef.isMating = true;
        canMate = false;

        NormClass newborn;
        List<string> femaleGenes = new List<string>();

        for (int i = 0; i < DNA.Length; i++) {
            femaleGenes.Add (DNA [i] [(int)Random.Range (1, 3)]);
        }
```

```
//Giving Birth
//Set the first Gene i in DNA to malegene and the second to female
newborn = Instantiate(babyNormPrefab, transform.position, Quaternion.Euler(0,0, Random.Range(0, 360))) as NormClass;

//Create the Array of Arrays
newborn.CreateDNA ();

for (int i = 0; i < newborn.DNA.Length; i++) {
    newborn.DNA [i] [1] = maleGenes [i];
    newborn.DNA [i] [2] = femaleGenes [i];
}

//------------------Creates the DNA string for visual reference------------------
newborn.CreateStringDNA ();

//------------------Set Gender------------------
newborn.gender = calculateBornGender ();
globalsRef.AddnewNormToLists (newborn.gender, newborn); //Add to current norms


//------------------Born Icon------------------
GameObject birthIconInstance = Instantiate (birthIcon, transform.position, Quaternion.identity) as GameObject;
if (newborn.gender == "Male")
    birthIconInstance.GetComponent<IconToAppear>().colorVector = new Vector3 (0, 0.5f, 1);
else
    birthIconInstance.GetComponent<IconToAppear>().colorVector = new Vector3 (1, 0.25f, 0.25f);

//------------------Set ID and add total norms------------------
newborn.ID = globalsRef.normTotalAmount;

//------------------Have mating cooldown------------------
newborn.canMate = false;

//------------------Just born cooldown------------------
        newborn.GetComponent<NormMovement>().isJustBorn = true;

        //------------------Check if female can still mate------------------
        //If there is no mating amount left than the female will no longer be target by males.
        matingAmountLeft -= 1;
        if (matingAmountLeft <= 0) {
            canMate = false;
        }
    }
}


////////////////////////////////////-----DNA-----\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

//NOTE DOES NOT SEEM TO HAVE CAP. VALUES SHOULD BE BETWEEN 05 AND 15
void MutateGene(){
    if (currentLifeLeft <= mutationTime) {
        int indexDNA = Random.Range (0, DNA.Length);
        int indexGene = Random.Range (1, 3);
        int currentGeneValue = int.Parse(DNA [indexDNA] [indexGene] [2] + "" + DNA [indexDNA] [indexGene] [3]);
        int mutationvalue = Random.Range (-maxMutationAmount, maxMutationAmount);


        DNA [indexDNA] [indexGene] = Random.Range(0,2) + ":" + SetD2(currentGeneValue + mutationvalue) + ":" + SetD4(ID);

        //print (ID + " has mutated");
        SetStaticValues ();
        SetMutationValues ();
        CreateStringDNA ();

        Instantiate (DNAicon, transform.position, Quaternion.identity);
        hasMutated = true;
    }
}

//==================GET DOMINANCE VALUE==================
float getGeneValueF (string [] GeneArray){
    float geneValue = 0;
    int dominanceAmount = 0;

    //------------------Check for dominance------------------
    if (GeneArray [1] [0] != '0'){ //First gene dominance is not 0 (Check first character)
        dominanceAmount += 1;
        geneValue += int.Parse(GeneArray [1] [2] + "" + GeneArray [1] [3]); //Total genevalue + the domance amount
    }
    if (GeneArray [2] [0] != '0'){ //Second gene dominance is not 0 (Check first character)
        dominanceAmount += 1;
        geneValue += int.Parse(GeneArray [2] [2] + "" + GeneArray [2] [3]);  //Total genevalue + the domance amount
    }


    //------------------Calculate return value------------------
    if (dominanceAmount == 1) { //If there is one dominant
        //print ("1Dom: " + geneValue);
        return (geneValue / 10);
    }

    if (dominanceAmount == 2) { //If there both are dominant
        geneValue /= 2; //Get the average between 2

        //print ("2Dom: " + geneValue);
        return (geneValue / 10);
    }

    //Return value calculation
    if (dominanceAmount == 0) { //If there are no dominant
        geneValue = int.Parse (GeneArray [1] [2] + "" + GeneArray [1] [3]) + int.Parse (GeneArray [2] [2] + "" + GeneArray [2] [3]); //Add both resessive
and devide by 2
        geneValue /= 2;

        //print ("0Dom: " + geneValue);
        return (geneValue / 10);
    }
    print ("ERROR: getGeneValue!!!");
    return (geneValue / 10);
```

```
        //Note each value is devided by 10 as than a normalised value is created. This way it can be a value with one decimal but I don have to take this into account in the GeneArray

    }

    //====================CREATE DNA ARRAY OF ARRAYs====================
    void CreateDNA(){
        //Sets the Array of Arrays
        DNA = new string[][] { speedGene, lifespanGene, hungerRateGene, hungerMaxGene };
        //print ("Gene Amount: " + DNA.Length);
    }

    //====================CREATE DNA STRING====================
    void CreateStringDNA(){
        stringDNA = "";
        //Function to calculate the Letters of DNA
        for (int i = 0; i < DNA.Length; i++) {
            string geneLetter = DNA [i] [0]; //Get the geneLetter
            int dominanceAmount = 0;

            if (DNA [i] [1] [0] != '0') //If first gene dominant dominance amount +1.
                dominanceAmount += 1;

            if (DNA [i] [2] [0] != '0') //If second gene dominant dominance amount +1.
                dominanceAmount += 1;

            //Set DNA gene section. For each recessive set geneletter as lowercase (In order of capptial first)
            if (dominanceAmount == 1)
                stringDNA += geneLetter + geneLetter.ToLower ();
            if (dominanceAmount >= 2) //> is safty
                stringDNA += geneLetter + geneLetter;
            if (dominanceAmount == 0)
                stringDNA += geneLetter.ToLower () + geneLetter.ToLower ();
        }
        //print ("ID: " + ID + " lenghtDNA: " + stringDNA.Length);
    }


    ////////////////////////////////-----VARIABLES-----\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

    //====================SET STATIC VARS====================
    void SetStaticValues(){
        normSpeed = 0;
        normLifespan = 0; //inseconds
        normHungerRate = 0; //Rate of which hunger decreases
        normHungerMax = 0; //Aka stomach content


        //var = var * genevalue
        normSpeed = normDefaultSpeed * getGeneValueF (speedGene);
        normLifespan = normDefaultLifespan * getGeneValueF (lifespanGene);
        normHungerRate = normDefaultHungerRate * getGeneValueF (hungerRateGene);
        normHungerMax = normDefaultHungerMax *  getGeneValueF (hungerMaxGene);
    }


    //====================SET DYNAMIC VARS====================
    //Sets the starting value of all dynamic vars
    void SetStartDynamicValues(){
        currentHunger = normHungerMax;
        currentLifeLeft = normLifespan;

        //If current life span <= to this than a gene will mutate
        mutationTime = Random.Range(10, normLifespan);

        //Mating coold downs
        if (gender == "Male")
            matingCooldown = normLifespan / maleMatingCoolDownCal;
        else
            matingCooldown = normLifespan / femaleMatingCoolDownCal;
        currentMatingCooldown = matingCooldown; //Sets the value for the first time

        //Mating hunger threshold
        hungerMatingThreshold = normHungerMax / 2; // if current is greater than the threshold than the male will seek a mate
        //hungerMatingThreshold = normHungerMax; //TEMP
    }

    //====================SET MUTATION VARS====================
    //Sets all dynamic values again
    void SetMutationValues(){
        //For now current hunger and currentLifeLeft are not changed

        //Mating coold downs
        if (gender == "Male")
            matingCooldown = normLifespan / maleMatingCoolDownCal;
        else
            matingCooldown = normLifespan / femaleMatingCoolDownCal;
        currentMatingCooldown = matingCooldown; //Sets the value for the first time

        //Mating hunger threshold
        hungerMatingThreshold = normHungerMax / 2; // if current is greater than the threshold than the male will seek a mate
    }

    //====================UPDATE DYNAMIC VARS====================
    void UpdateDynamicValues(){
        currentHunger -= normHungerRate * Time.deltaTime;
        currentLifeLeft -= Time.deltaTime;

        if (currentHunger <= hungerMatingThreshold) {
            isHungry = true;
            //Calculate here the distance it needs before going to food target!
        } else
            isHungry = false;
    }

    //====================DEATH CHECK====================
    void DeathCheckF(){
        //Age death check
        if (currentLifeLeft <= 0) {
            print (ID + " died of old age");
            globalsRef.RemoveCurrentDead (gender, ID);
```

```
            Instantiate (deathSkull, transform.position, Quaternion.identity);
            Destroy (gameObject);
        }

        //Food death check
        if (currentHunger <= 0) {
            print (ID + " died of hunger");
            globalsRef.RemoveCurrentDead (gender, ID);
            Instantiate (deathSkull, transform.position, Quaternion.identity);
            Destroy (gameObject);
        }
    }
    //====================KILL====================
    public void KillF(){
        //print (ID + " is killed");
        globalsRef.RemoveCurrentDead (gender, ID);
        Instantiate (deathSkull, transform.position, Quaternion.identity);
        Destroy (gameObject);
    }

    //====================GENDER====================
    string calculateBornGender(){
        if ((globalsRef.normMalesL.Count / globalsRef.normFemalesL.Count) < genderThreshold) {
            return "Male";
        }
        if ((globalsRef.normFemalesL.Count / globalsRef.normMalesL.Count) < genderThreshold) {
            return "Female";
        }
        string[] babyGender = new string[] { "Male", "Female" };
        return babyGender[(int)Random.Range (0, 2)];
    }

    ////////////////////////////////////-----CALLED FUNCTIONS-----\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

    public void FeedNorm(int foodValue){
        currentHunger += foodValue;
        if (currentHunger > normHungerMax)
            currentHunger = normHungerMax;
    }

    //====================SET VALUE TO STRING FUNCTION====================
    //Converts value to two decimal string to be stored in array
    string SetD2(int inputValue){
        string returnString = inputValue.ToString("00");
        return returnString;
    }
    string SetD3(int inputValue){
        string returnString = inputValue.ToString("000");
        return returnString;
    }
    string SetD4(int inputValue){
        string returnString = inputValue.ToString("0000");
        return returnString;
    }

    //====================REFERENCES====================
    void SetReferences (){
        globalsRef = FindObjectOfType (typeof(Globals)) as Globals;
        normMovementRef = gameObject.GetComponent<NormMovement>();
    }
}
```