Neurale Netwerk Zelfrijdende Auto

Portfolio

Angelo Moestadja
500752222 Hogeschool van Amsterdam
17-01-2024

Inhoudsopgave

1 INLEIDING

In dit project heb ik een zelfrijdende auto gemaakt door middel van twee belangrijke algoritmen: het genetische algoritme (GA) en het kunstmatige neurale netwerk (ANN). Het doel was om de auto een racecircuit te laten voltooien zonder te botsen tegen de muren. De combinatie van GA en ANN was een interessante aanpak om de auto zelf te laten rijden. Normaal gesproken is het neurale netwerk voldoende maar ik zocht naar meer uitdaging.

In mijn rapport vertel ik hoe ik gebruik heb gemaakt van de slimme algoritmes die de auto helpen het racecircuit te laten voltooien. Hierbij leg ik uit hoe GA en ANN werken en hoe ik het samen heb gebruikt om de auto goed te laten rijden in dit project. Ik leg uit welke tools ik heb gebruikt van de game engine Unity, laat wat stukjes code zien in C# en ik vertel hoe ik een virtuele wereld heb gemaakt met racebaan stukjes enzovoorts. Ook laat ik de resultaten zien en trek ik een conclusie. Daarin vertel ik een kleine samenvatting hoe het werkt en wat er mogelijk verbeterd kan worden in het project.

Mijn project heb ik in de Unity game engine gemaakt. De code is geschreven in C# en alle dingen die ik nodig had zoals bepaalde hulpmiddelen, extra software, auto en racecircuit assets, gratis waren. Unity kan heel handig zijn, omdat het voorgebouwde algoritmen heeft zoals ML Agents maar ik ben helemaal vanaf het begin begonnen door mijn eigen neurale netwerk en genetische algoritme te coderen behalve de mathematics plugin.

2 THEORETISCHE ACHTERGROND

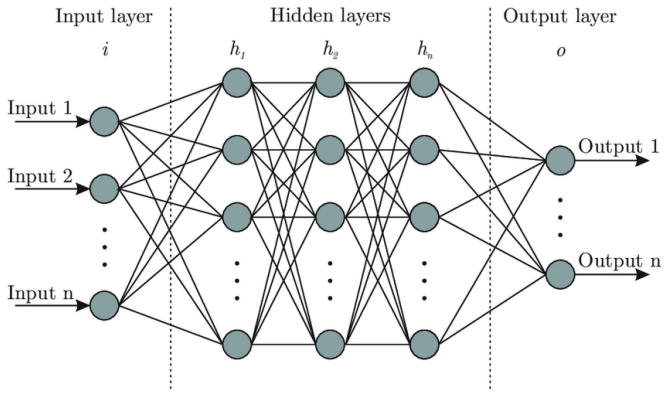
Neural Networks

Kunstmatige neurale netwerken (ANNs) proberen het menselijke brein na te bootsen om problemen op te lossen. Er wordt informatie ontvangen, geanalyseerd, beslissingen worden genomen en er wordt geleerd van hun ervaring. Bij ANN wordt data ingevoerd via de inputlaag, vervolgens gaat het naar een verborgen laag waar wiskundige berekeningen worden uitgevoerd. De data is op dat moment echter een nummer zonder echte betekenis. Na deze wiskundige (mathematics) berekeningen, die de activatiefunctie wordt genoemd, gaan de waarden naar de outputlaag. De neuronen zijn verbonden met synapsen waar gewichten aan vastzitten. Het netwerk past zich aan op basis van het verschil tussen de werkelijke en verwachte waarde. Dit proces wordt herhaald totdat een bepaalde foutmarge of een aantal iteraties zijn voltooid.

Neuronen en Lagen

Een kunstmatig neuraal netwerk bestaat uit neuronen georganiseerd in lagen. Neuronen, de kleinste eenheden, hebben gewichten in de vorm van zwevende komma getallen. Informatie wordt via de inputlaag in het netwerk ingevoerd, waar de inputneuronen zich bevinden. Neuronen zijn verbonden via synapsen, elk met een willekeurige waarde (gewicht) tussen -1 en 1. De inputs worden gecombineerd met de synapsgewichten via het dot product, en daarna wordt de activatiefunctie toegepast om de output te transformeren. Neuronen hebben ook een biaswaarde, willekeurig ingesteld door het netwerk. Er zijn drie lagen in de architectuur van een ANN: de inputlaag, een set verborgen lagen (maximaal twee), en een outputlaag. Tijdens het doorlopen van het netwerk in de feed-forward fase reist de data in deze volgorde. Backpropagation wordt alleen gebruikt tijdens de training van het netwerk.

Neuronen zijn georganiseerd in lagen van een kunstmatig neuraal netwerk. De kleinste eenheden in dit geval de neuronen, hebben gewichten in de vorm van komma getallen. De inputneuronen bevinden zich in het netwerk waar informatie via de inputlaag wordt ingevoerd.



Afbeelding 1 Een standaard ANN architectuur

Voorwaartse sturing (Feedforward) en Achterwaartse foutpropagatie (Backpropagation)

De data wordt tijdens het voorwaartse sturing (feedforward) proces eerst doorlopen bij de inputlaag, dan de verborgen laag en daarna de outputlaag, waar het resultaat wordt verkregen. Achterwaartsefoutpropagatie (backpropagation) is nodig wanneer een netwerk ongetraind is.

Bij de achterwaartsefoutpropagatie (backpropagation) proces worden de fouten bepaald en de neuronen geïdentificeerd, die verantwoordelijk zijn voor de onjuiste output, door het netwerk. De gewichten van de synapsen worden aangepast met behulp van deze fouten. In het trainingsproces is het correcte resultaat bekend bij het netwerk dus kan het worden gebruikt als leidraad. De fout wordt eerst berekend en vervolgens over alle neuronen in de outputlaag verdeeld. De afgeleide van de activatiefunctie van het outputneuron wordt toegepast op de oorspronkelijke output van het neuron zonder de toepassing van de oorspronkelijke activatiefuntie. Om zijn delta te vinden, wordt vervolgens dit resultaat vermenigvuldigd met de fout van het neuron. In de verborgen lagen (hidden layers) wordt de delta van elk neuron berekend, waardoor er bepaald kan worden door het netwerk hoeveel impact een neuron had bij het prodcueren van de onjuiste output.

Activatiefuncties

Voor het oplossen van specifieke problemen, zijn er verschillende activatiefuncties die kunnen worden gebruikt waarbij sommige geschikter zijn dan andere. In verschillende lagen van neurale netwerken kunnen verschillende activatiefuncties worden gebruikt. In de verborgen lagen (hidden layers) kan een netwerk bijvoorbeeld een sigmoid-functie hebben en in outputlagen, een hyperbolische tangensfunctie (tanh function).

Stapfunctie

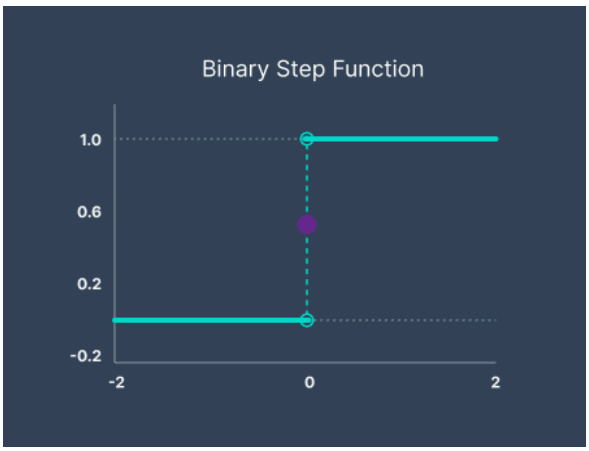
Dit is de eenvoudigste activatiefunctie. Een drempelwaarde is ingesteld; als de uitvoer groter is dan of gelijk is aan de drempel, wordt de neuron geactiveerd. De uitvoer is altijd 0 of 1, afhankelijk van de voorwaarde:

De stapfunctie is de eenvoudigste activatiefunctie. Er is hierbij een drempelwaarde ingesteld en als de uitvoer groter of gelijk aan de drempel is, wordt de neuron geactiveerd. De uitvoer is altijd 0 of 1, afhankelijk van de voorwaarde.

Als bijvoorbeeld de invoer positief is of gelijk aan 0, wordt de uitvoer 1, anders wordt de uitvoer 0.

f(x) = 1, if x >= 0

f(x) = 0, if x < 0



Afbeelding 2 Stapfunctie

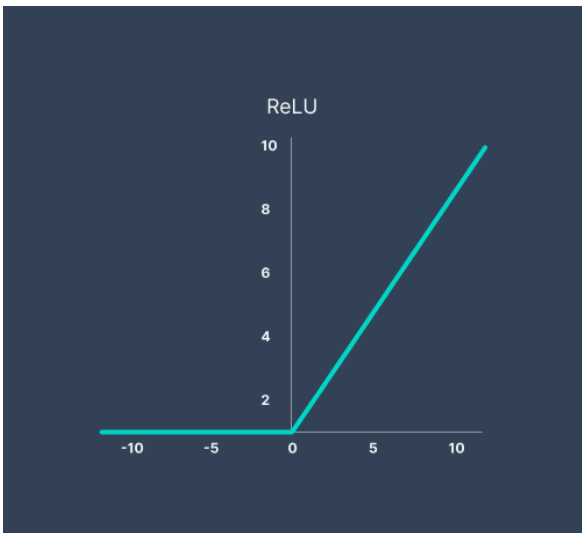
Sigmoïde Functie

De sigmoïde functie is een niet-lineare functie waarbij neuronen verschillende waarden als output geven wanneer het wordt toegepast. Hierdoor wordt het resultaat niet-linear en is het beter geschikt voor complexere problemen. De uitvoer varieert tussen 0 en 1.

Afbeelding 3 Sigmoïde Functie

Rectified Linear Unit

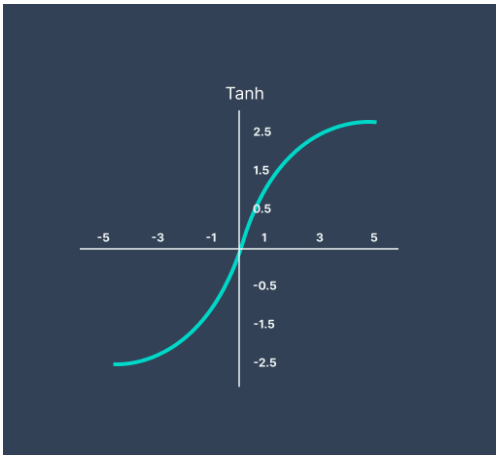
ReLU (Rectified Linear Unit) is de meest populaire activatiefunctie die alleen rekening houdt met neuronen die een uitvoerwaarde groter dan 0 hebben. Neuronen met een uitvoerwaarde van 0 blijven inactief en alles wat minder is dan 0 wordt ingesteld op de waarde 0.



Afbeelding 4 ReLu

Tangens Functie

Qua eigenschappen is de tangens hyperbolicus functie, ook wel tanh functie genoemd, vergelijkbaar met de sigmoïde functie. De tangens functie geeft een waarde binnen het bereik van -1 tot 1. Het belangrijkste verschil tussen de tangens en sigmoïde functie is, dat bij backpropagation de afgeleide van de twee functies verschillend zal zijn. Tanh voorkomt onjuiste weergaven van de werkelijkheid.

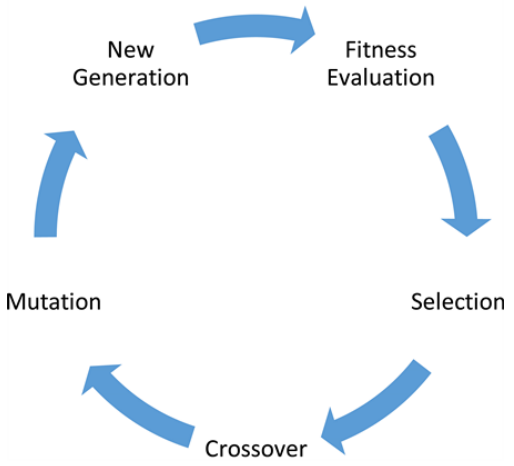


Afbeelding 5 Tangens Functie

Genetische algoritmen

De reden waarom ik genetische algoritmen heb gebruikt met het neurale netwerk, is omdat genetische algoritmen samen met andere algoritmen gebruikt kunnen worden om efficiëntie te vergroten.

Eerst wordt er een groep individuen gemaakt die vertegenwoordigd wordt door een chromosoom. Elk individu wordt dan beoordeeld met een fitheid (fitness) functie. De ouders die worden geselecteerd voor de volgende generatie, zijn de beste individuen bepaald door de fitheid. De kinderen van deze ouders, erven echter niet precies de eigenschappen van hun ouders want dat zal niet helpen bij het vinden van nieuwe oplossingen. Hiervoor wordt er iets genaamd “crossover” gebruikt om ervoor te zorgen dat de kinderen eigenschappen van beide ouders erven, maar niet exact hetzelfde zijn. Om elke generatie te laten evolueren en verschillende mogelijke oplossingen te vinden, wordt er een kleine kans op “mutatie” toegevoegd, waardoor af en toe wat willekeurige veranderingen optreden.



Afbeelding 6 GA cyclus

Fitheid Functie

De fitheid functie wordt gebruikt om te vertellen hoe goed een oplossing is en helpt het algoritme om op de beste oplossing te komen. Er zijn twee soorten gedrag voor een fitheid functie: één die hetzelfde blijft, en één die kan veranderen. De fitheid functie probeert de beste oplossing in de huidige groep te vinden die geselecteerd kan worden voor het maken van nieuwe generaties.

Selectie

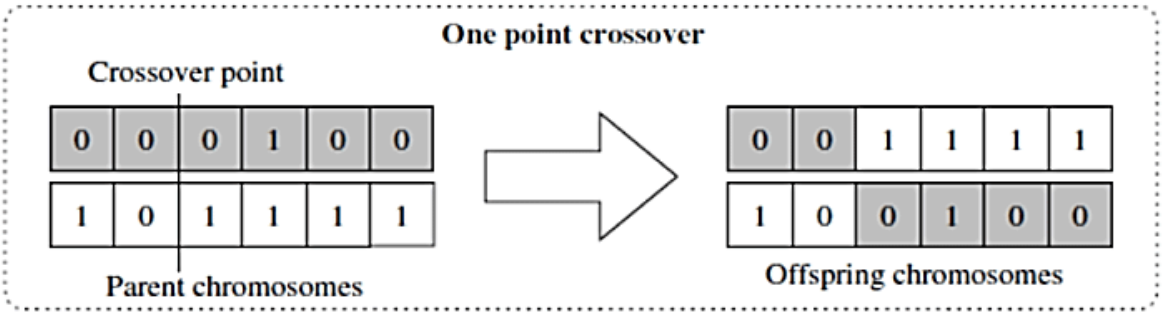
Bij genetische algoritmen wordt er een selectie gemaakt van de sterkste individuen per generatie. De individuen worden gekozen die de bijna perfecte oplossingen hebben. Bij genetische algoritmen worden deze individuen, chromosomen genoemd.

Cross-over

Stel je voor dat de gekozen chromosomen als ouders fungeren en samen kinderen maken. De kinderen erven de eigenschappen (genen) van hun ouders, maar het mag niet precies hetzelfde zijn. We willen immers nieuwe oplossingen vinden voor de volgende generatie. Er zijn verschillende manieren om dit te doen, zoals enkelpunts-, tweepunts-, k-punts- of uniforme crossover.

De gekozen chromosomen fungeren als de ouders die samen kinderen maken. Voor het vinden van nieuwe oplossingen voor de volgende generatie, erven de kinderen de eigenschappen (genen) van hun ouders, maar het mag niet precies hetzelfde zijn. De volgende verschillende manieren om dit te doen zijn zoals de enkelpunts-, tweepunts-, k-punts- of uniforme crossover.

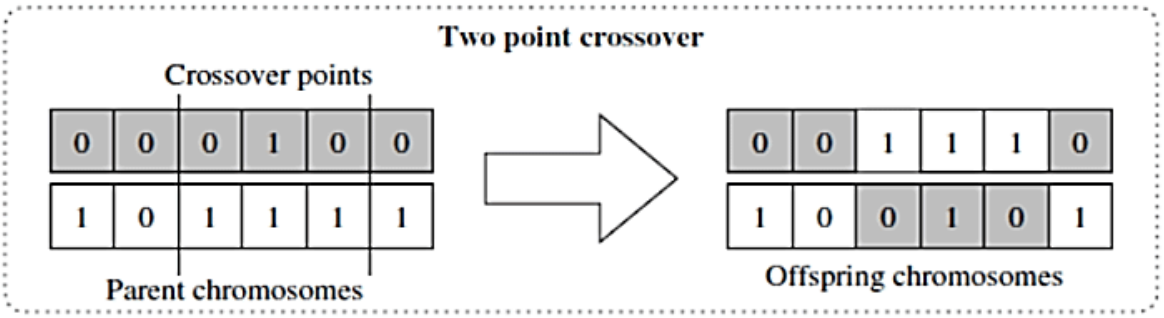
enkelpunts-crossover: Bij deze manier krijgt het ene kind bepaalde genen van de ouder en de andere kind bepaalde genen van de andere ouder. Zo worden er nieuwe combinaties gemaakt voor de volgende generatie. Zie afbeelding 2.



Afbeelding 7 Enkelpunts-crossover

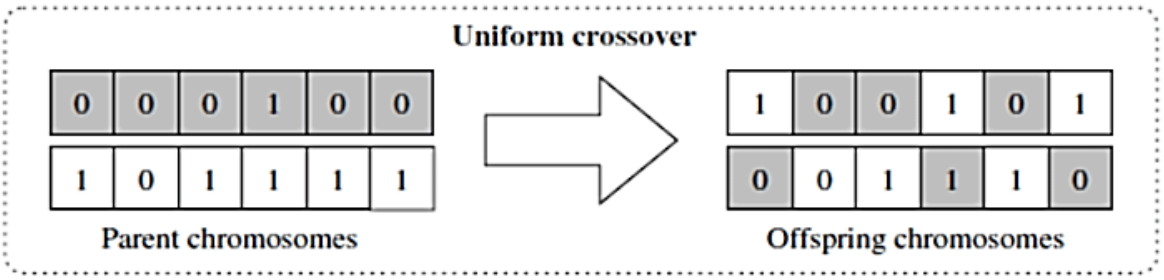
Tweepunts-crossover: Deze methode werkt eigenlijk hetzelfde als de enkelpunts-crossover, maar hierbij worden twee willekeurige punten geselecteerd waar de genen worden uitgewisseld in plaats van één. Zie afbeelding 3.

K-punt crossover is een methode waarbij er meer dan twee willekeurige punten worden geselecteerd waar de genen worden uitgewisseld en K een positief geheel getal is. Deze manier is geschikter voor grotere genenpools.



Afbeelding 8 Tweepunts-crossover

Uniforme Crossover: Bij deze manier heeft elk gen kans gekozen te worden voor de crossover. De kinderen kunnen heel willekeurige eigenschappen hebben waarbij meerdere crossover-punten of zelfs helemaal geen kunnen ontstaan. Het hangt allemaal af van het toeval. Nieuwe en diverse combinaties voor de volgende generatie worden hierbij verkregen door de genen van ouders willekeurig te mixen.



Afbeelding 9 Uniforme Crossover

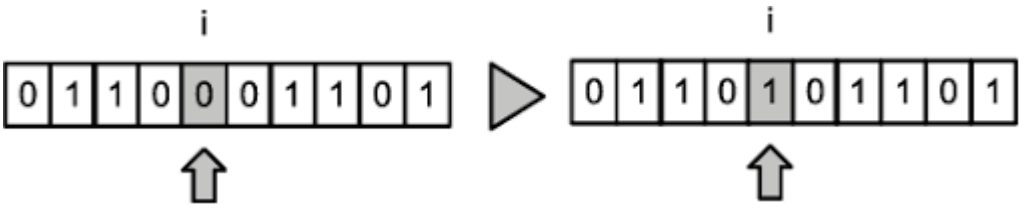
Mutatie

Bij het selecteren van de kinderen erven de kinderen eigenschappen van hun ouders maar niet exact dezelfde eigenschappen. Dit noemen wij mutatie. Bij dit crossover-process wordt minstens één gen in het chromosoom willekeurig veranderd, zonder rekening te houden met wat de ouders hebben of wat hun broertjes en zusjes hebben. Door mutatie worden verschillende mogelijkheden verkend en is daarom heel belangrijk in genetische algoritmen. Maar de kans op mutatie moet wel goed ingesteld worden want als de kans op mutatie te

hoog is, wordt het algoritme te willekeurig en als het te laag is worden de beste oplossingen gemist.

De verschillende manieren om mutatie te doen zijn onder andere bit-flip mutatie, swap-mutatie, scramble mutatie en inversie mutatie.

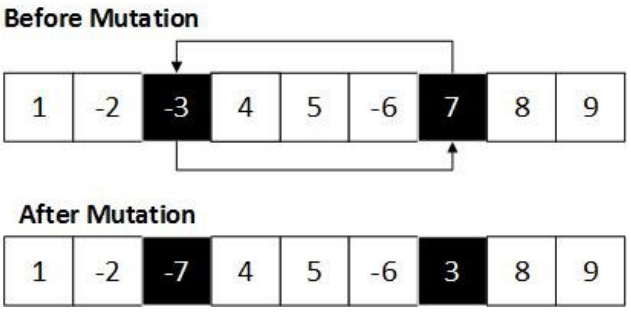
Bit-flip mutatie: hierbij verandert een ’bit’ van 0 naar 1, of van 1 naar 0. Deze manier is vooral handig als de genen binaire getallen zijn. Zie afbeelding 5.



Afbeelding 10 Bit-flip Mutatie

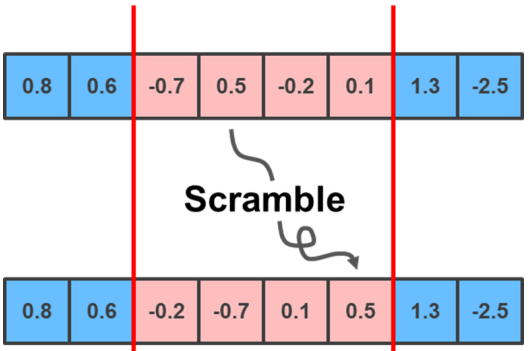
Maar bij genetische algoritmen kan ook gemuteerd worden met decimale getallen afhankelijk van het probleem dat moet worden opgelost.

Swap-mutatie: Dit kan een handige methode zijn als er wordt gemuteerd met decimale getallen. in de reeks pakt het twee willekeurige genen in het chromosoom en wisselt hun posities daarin. Zie afbeelding 6.



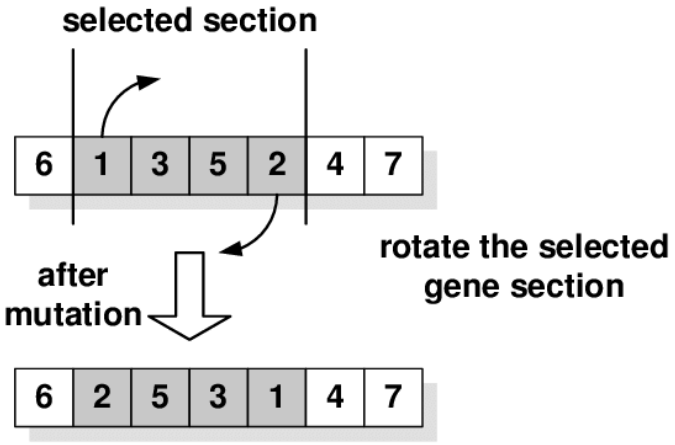
Afbeelding 11 Swap Mutatie

Scramble-mutatie: Hierbij worden twee willekeurige indexen binnen een subset van genen gekozen en alle genen binnen dat bereik worden door elkaar gehusseld. Zie afbeelding 7.



Afbeelding 12 Scramble-mutatie

Inversiemutatie: Deze mutatie techniek lijkt op de scramble-mutatie maar hierbij worden twee willekeurige indexen gekozen waarvan de genen binnen het aangegeven bereik worden verdeeld in een omgekeerde volgorde van de oorspronkelijke reeks (eerste wordt laatste, laatste wordt eerste, enzovoort). Zie afbeelding 8.



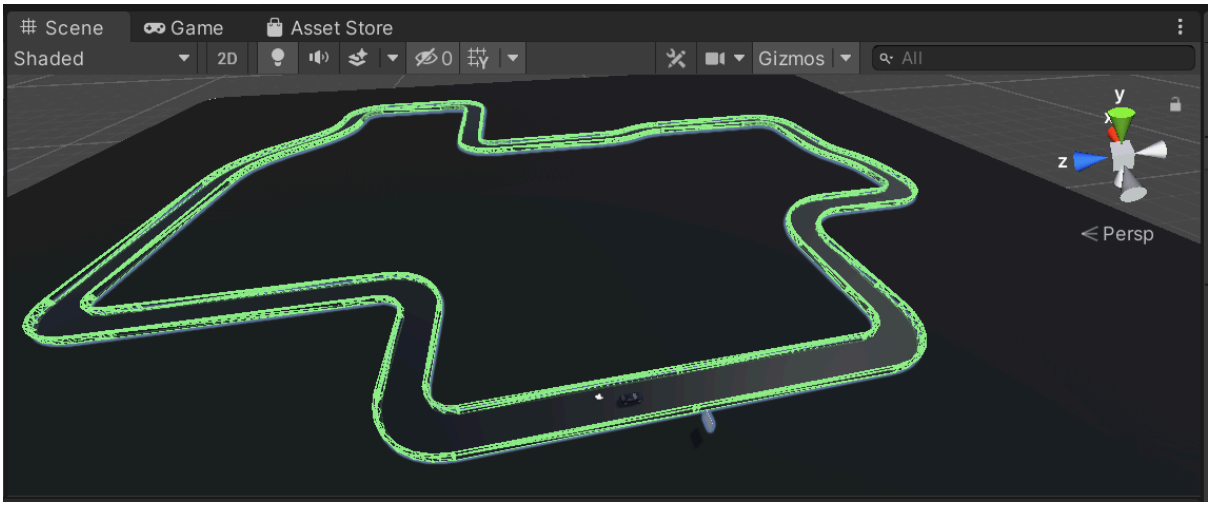
Afbeelding 13 Inversiemutatie

3 UNITY ENGINE

Ik heb gekozen voor Unity boven andere engines omdat er een overvloed aan gratis en open-source assets direct beschikbaar zijn voor gebruik. Naast dat ondersteunt de engine zowel C#, C++ en JavaScript waarbij ik de programmeertaal C# heb gebruikt. Het project is opgezet uit een racebaan, een vlakte en een auto waarbij ik de gratis assets van de Unity asset store heb gebruikt. Verder had ik niks nodig uit de asset store aangezien het doel van het project was de auto zelfstandig op de baan te laten rijden.

De Baan

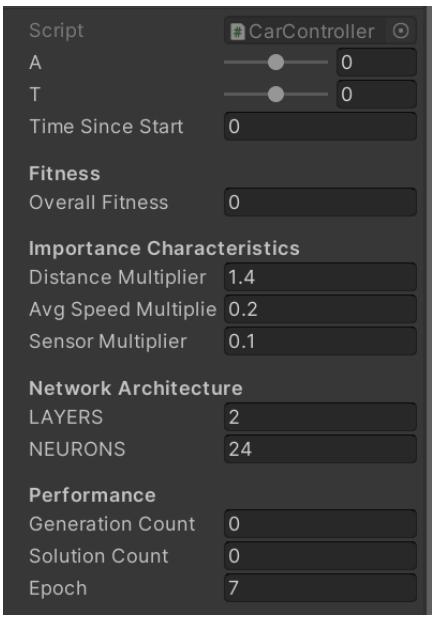
De baan heb ik gratis via de Unity asset store gedownload en geïmporteerd. De baan delen heb ik wel zelf aan elkaar gebouwd totdat het een racecircuit werd. Aan de weerszijden van de baan heb ik zijkanten gevoegd die werken als colliders. Als de auto tegen de zijkanten van de baan botst, resulteert dit in de eliminatie van die specifieke auto. Elke auto is één chromosoom in de populatie.



Afbeelding 14 Baan met de auto

De Auto

De auto wordt aangestuurd door de Car Controller script waarin enkele variabelen zichtbaar zijn in de inspector. Dit maakt het voortdurend experimenteren met trial en error met de waarden mogelijk.



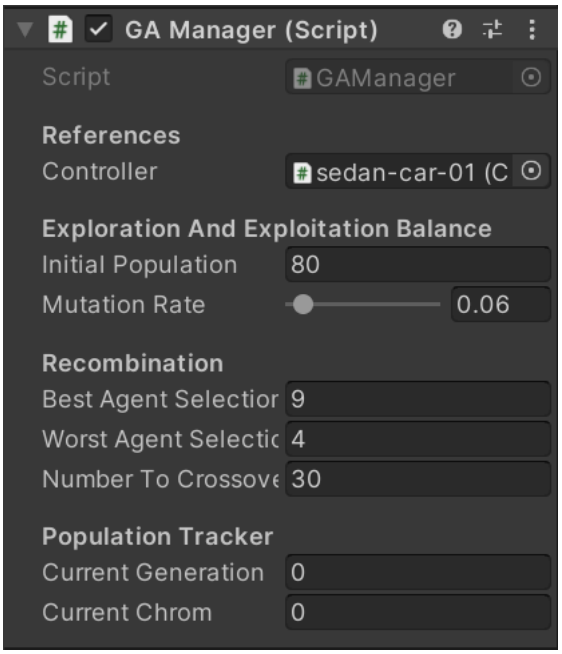
Afbeelding 15 Variabelen CarController inspector

Afbeelding 14 toont de parameters in de inspector. Er is een timer die automatisch wordt verhoogd, en een fitnesswaarde die wordt berekend op basis van de tijd en de afstand sinds het begin. De afstands- en gemiddelde snelheidsvermenigvuldigers worden gebruikt in de fitnessberekening. De snelheid wordt ook willekeurig aangepast bij het creëren van het chromosoom en verandert willekeurig. De sensormultiplier helpt de auto bij het navigeren.

In afbeelding 14 worden de parameters getoond in de inspector. Er is een timer die afgaat en automatisch verhoogd wordt en een fitnesswaarde die berekend wordt op basis van de tijd en de afstand sinds het begin. In de fitnessberekening worden de afstands- en gemiddelde snelheidsvermenigvuldigers gebruikt. Bij het creëren van het chromosoom wordt ook willekeurig de snelheid aangepast. De auto wordt geholpen bij het navigeren door de sensormultiplier.

Genetic Algorithm Manager Object

Om het GAManager script correct te laten werken met het kunstmatige neurale netwerk (ANN) script, moet een apart game-object gemaakt worden die GAManager wordt benoemd. Vervolgens moet dat object gekoppeld worden aan het auto-object. In hoofdstuk 4 wordt uitgelegd welke verschillende methoden en functies worden aangeroepen tussen de twee objecten. In afbeelding 17 worden de waarden getoond van het project tijdens runtime.



Afbeelding 16 Variabelen GAManager inspector

De populatie is ingesteld op 80 met een mutatiekans van 0.06 op elk chromosoom. Voor het creëren van de volgende generatie worden de beste gekozen en de slechtste 4. Het crossover variabele geeft aan hoe vaak deze chromosomen worden gebruikt om nakomelingen te maken.

De populatie tracker geeft weer hoeveel chromosomen zijn geëlimineerd per generatie. Als er meer dan 80 chromosomen zijn geëlimineerd, wordt de generatie verhoogd van 0 naar 1 en begint de current chromosoom weer bij 0. Dit proces herhaalt zich continu en wordt continu bijgewerkt in de inspector tijdens de runtime van het project.

4 DE UITVOERING

In mijn project heb ik in plaats van backpropagation, het neurale netwerk laten samen werken met een genetisch algoritme.

Het neurale netwerk die ik heb geïmplementeerd heeft een standaard structuur met een invoerlaag, een verborgen laag en een uitvoerlaag. Elk neuron heeft een willekeurige bias en elke synaps heeft een willekeurige gewicht. Met behulp van de sigmoïd-activatiefunctie wordt de snelheid bepaald, terwijl de hyperbolische tangens functie gebruikt wordt voor het sturen. De snelheid varieert tussen 0 en 1, waarbij 0 stilstaan betekent en 1 volle snelheid. De tangensfunctie gebruikt -1 om naar links te sturen, 0 om rechtuit te gaan en 1 om naar rechts te sturen. Het gebruik van float waarden voor snelheid en sturen is belangrijk, bijvoorbeeld voor scherpere bochten of rijden langs de rand voor het circuit, wat in sommige gevallen kan leiden door toename van de afstand in de loop van de tijd.

Met behulp van het genetisch algoritme wordt er tijdens het trainen de best presterende netwerken geselecteerd uit een populatie van netwerken en een nieuwe generatie creëert. Er worden ook een klein aantal van de slechtste oplossingen vermengd met de beste, om te voorkomen dat het netwerk de oplossing voor het probleem opslaat en hergebruikt. Om de oplossingen te verkennen wordt er een stabiele populatie behouden waarbij er wordt gecontroleerd welke chromosomen voortplanten. Dit proces wordt herhaald tot een periode (epoch) is behaald.

Voor dit project heb ik de tweede methode gebruikt. Het is een veiligere aanpak en minder zwaar voor het systeem maar het kost wel meer tijd. De reden waarom de tweede methode veiliger is dan de eerste, is omdat alle chromosomen niet kunnen worden uitgeschakeld voordat er een optimale oplossing is gevonden. Anders bestaat er een kans dat het netwerk wordt geperfectioneerd binnen een aantal iteraties gelijk aan het maximale aantal mogelijke generaties. Er kan bij de eerste methode ook het risico ontstaan van overtraining van het model doordat er een perfecte oplossing wordt gevonden binnen een bepaalde tijd en het trainen doorgaat.

De CarController-klasse

In de CarController-klasse wordt het gedrag van elke auto (chromosoom) bepaalt. De auto heeft drie sensoren (links, vooruit en rechts) en een box collider die reageert op de mesh colliders van de muren van het racecircuit. De waarden van de sensoren worden die samenwerkt met de omgeving, worden doorgegeven naar de NeuralNetwork-klasse. Als de auto botst, wordt die specifieke auto uitgeschakeld en wordt er een nieuwe auto aangemaakt die het racecircuit vanaf de start positie weer probeert te voltooien. Het neurale netwerk geeft de snelheid en stuurwaarden door aan de auto controller. De totale afstand en de gemiddelde snelheid van elk auto (chromosoom) worden beoordeeld door de fitnessfunctie.

```
private Vector3 startPosition, startRotation;
private NNet network;

private GAManager gaManager;

[Range(-1f, 1f)]
public float a, t;

public float timeSinceStart = 0f;

[Header("Fitness")]
public float overallFitness;

[Header("Importance Characteristics")]
public float distanceMultiplier = 1.4f;
public float avgSpeedMultiplier = 0.2f;
public float sensorMultiplier = 0.1f;

[Header("Network Architecture")]
public int LAYERS = 1;
public int NEURONS = 10;

private Vector3 lastPosition;
private float totalDistanceTravelled;
private float avgSpeed;

private float aSensor, bSensor, cSensor;

[Header("Performance")]
public int generationCount = 0;
public int solutionCount = 0;
public int epoch = 200;
```

Afbeelding 17 Variabelen CarController

Om de auto te kunnen kunnen verplaatsen worden 3 variabelen als vectoren opgeslagen, namelijk: de startpositie (startPos), de startrotatie (startRot) en de bewegingswaarden (moveInput). De float-variabelen a en t hebben beide een bereik tussen -1 en 1, en staan voor de versnelling en sturing van de auto.

De afstand en de snelheid worden gebruikt om de fitness te berekenen. De afstand die de auto legt zonder te botsen tegen de muren is het belangrijkste. Om de verschillende oplossingen van de chromosomen te onderscheiden is er een snelheid variabele aangemaakt. Dit maakt het mogelijk de meest optimale oplossingen te vinden door te kijken naar hoe snel een bepaalde chromosoom de afstand heeft kunnen leggen.

Het netwerk van elk chromosoom kan aangepast worden in de inspector. Elk chromosoom heeft standaard een verborgen laag met 24 neuronen als het ongewijzigd blijft. Als een chromosoom langer dan 30 seconden leeft en de overallFitness is maar onder 45 dan wordt de chromosoom uitgeschakeld en als de overallFitness gelijk of boven 1000 is. Wanneer de overallFitness boven de 1000 is, heeft het chromosoom (de auto) het racecircuit voltooid.

Fitness Berekening

De fitness van elke auto wordt berekend met de afstand, snelheid en de waarden van de sensoren. Elk variabel wordt vermenigvuldigd met een multiplier. Hoe hoger de afstand multiplier, hoe meer auto’s worden gekozen die lange afstanden leggen, hoe hoger de snelheid multiplier, hoe meer auto's die snel zijn en hoe hoger de sensor multiplier, hoe meer auto's die in het midden van de baan blijven.

```
private void CalculateFitness()
{

    totalDistanceTravelled += Vector3.Distance(transform.position, lastPosition);
    avgSpeed = totalDistanceTravelled / timeSinceStart;

    overallFitness = (totalDistanceTravelled * distanceMultiplier) + (avgSpeed * avgSpeedMultiplier) + (((aSensor + bSensor + cSensor) / 3) * sensorMultiplier);

    if (timeSinceStart > 20 && overallFitness < 45)
    {
        Death();
    }

    if (overallFitness >= 1000)
    {
        generationCount = gaManager.getGenCount() + 1;
        solutionCount++;
        generationList.Add(generationCount);
        solutionList.Add(solutionCount);
        Death();
    }

}
```

Afbeelding 18 Fitness functie

Input Sensoren

De input sensoren zorgen ervoor dat de auto zijn eigen positie weet, hoe dichtbij hij bij de rand staat en wat er voor hem is. Op basis van de sensorische input moet de auto de juiste beslissing nemen en is dus belangrijk voor het trainen van de auto. Door middel van raycast op te roepen in de functie kan de auto de afstand tot een ander object meten.

```
private void InputSensors()
{

    Vector3 a = (transform.forward + transform.right);
    Vector3 b = (transform.forward);
    Vector3 c = (transform.forward - transform.right);

    Ray r = new Ray(transform.position, a);
    RaycastHit hit;

    if (Physics.Raycast(r, out hit))
    {
        aSensor = hit.distance / 20;
        Debug.DrawLine(r.origin, hit.point, Color.red);
    }

    r.direction = b;

    if (Physics.Raycast(r, out hit))
    {
        bSensor = hit.distance / 20;
        Debug.DrawLine(r.origin, hit.point, Color.red);
    }

    r.direction = c;

    if (Physics.Raycast(r, out hit))
    {
        cSensor = hit.distance / 20;
        Debug.DrawLine(r.origin, hit.point, Color.red);
    }
}
```

Afbeelding 19 Sensoren functie

De MoveCar() functie gebruikt twee floats als argumenten die de auto helpen te sturen en te versnellen. Als de auto botst of een te hoge fitheid heeft, wordt de Death() functie opgeroepen en wordt de auto vernietigd. Daarnaast wordt de ResetNetwork() functie opgeroepen die het neurale netwerk reset en een NeuralNetwork object als argument heeft. De auto wordt door de Reset() functie terug naar de startpositie gezet.

NeuralNetwork-klasse

Netwerken worden door de NeuralNetwork-klasse gemaakt door de inputlaag, verborgen laag(lagen) en outputlaag toe te voegen, en door willekeurige gewichts- en biaswaarden te bieden. De hyperbolische tangent-functie heeft C# ingebouwd, maar de sigmoïd-functie niet, dus deze wordt gemaakt in het script.

```
// Input layer with 1 row and 3 columns
public Matrix<float> inputLayer = Matrix<float>.Build.Dense(1, 3);
public List<Matrix<float>> hiddenLayers = new List<Matrix<float>>();
public Matrix<float> outputLayer = Matrix<float>.Build.Dense(1, 2);
public List<Matrix<float>> weights = new List<Matrix<float>>();
public List<float> biases = new List<float>();

public float fitness;
```

Afbeelding 20 Variabelen NeuralNetwork

Als een chromosoom “sterft”, moet de fitnesswaarde ervan worden opgeslagen voor verdere berekeningen. De float-variabele “fitness” zorgt hiervoor. Het wordt namelijk doorgegeven aan de GAManager-klasse, waar de prestaties van het netwerk worden beoordeeld op basis van die waarde.

Er wordt voor elke laag een matrix gemaakt. De "Build.Dense()" methode zorgt ervoor dat de "inputLayer"-matrix een 1x3-matrix bouwt. Echter is de structuur van de "hiddenLayer" op dit moment niet ingesteld, omdat deze niet mag zijn voorgecodeerd. Een 1x2-matrix wordt ten slotte gemaakt door de “outputLayer”. Voor het instellen van de snelheid is de ene neuron, en voor het bepalen van hoeveel de auto moet draaien, de andere neuron bedoeld. Van elke neuron wordt de bias-waarde opgeslagen in een lijst.

Netwerk Initialiseren

Alle waarden moeten worden gereset wanneer een nieuw netwerk wordt gemaakt. Eerst reset de functie de waarden van de inputlaag, de verborgen laag, de outputlaag, de gewichten en de biases. Aangezien de input- en outputlagen al zijn gebouwd, bouwt de "Initialise()" functie de verborgen laag(lagen). De functie neemt twee integers als argumenten: één voor het aantal verborgen lagen en een ander voor het aantal neuronen in een verborgen laag. Het maakt dan de verborgen lagen met behulp van een for-lus, waarin de bias-waarden worden gerandomiseerd en toegewezen aan elke neuron, evenals het gewicht van elke verbinding. De gewichtswaarden tussen de laatste verborgen neuronen en de output neuronen moeten ook worden ingesteld.

Wanneer een nieuw netwerk wordt gemaakt moeten alle waarden worden reset. Eerst worden de waarden gereset van de inputlaag, de verborgen laag, de outputlaag, de gewichten en de biases. De “Initialise()” functie bouwt de verborgen laag aangezien de input- en outputlagen al zijn gebouwd. Twee integers worden door de functie gebruikt als argumenten. Eén voor het aantal verborgen lagen en de andere voor het aantal neuronen in een verborgen laag.

```
public void Initialise(int hiddenLayerCount, int hiddenNeuronCount)
{

    inputLayer.Clear();
    hiddenLayers.Clear();
    outputLayer.Clear();
    weights.Clear();
    biases.Clear();

    for (int i = 0; i < hiddenLayerCount + 1; i++)
    {

        Matrix<float> f = Matrix<float>.Build.Dense(1, hiddenNeuronCount);

        hiddenLayers.Add(f);

        biases.Add(Random.Range(-1f, 1f));

        //WEIGHTS
        if (i == 0)
        {
            Matrix<float> inputToH1 = Matrix<float>.Build.Dense(3, hiddenNeuronCount);
            weights.Add(inputToH1);
        }

        Matrix<float> HiddenToHidden = Matrix<float>.Build.Dense(hiddenNeuronCount, hiddenNeuronCount);
        weights.Add(HiddenToHidden);

    }

    Matrix<float> OutputWeight = Matrix<float>.Build.Dense(hiddenNeuronCount, 2);
    weights.Add(OutputWeight);
    biases.Add(Random.Range(-1f, 1f));

    RandomiseWeights();

}
```

Afbeelding 21 Initialisatie functie

De functie "RandomiseWeights()" bevat een reeks geneste loops. De buitenste lus telt het totale aantal gewichten. De eerste geneste lus telt de rijen, de meest geneste lus telt de kolommen van de matrix. De methode binnen de meest geneste lus wijst aan elk gewicht in het netwerk een willekeurige float-waarde toe tussen -1 en 1.

De functie “RandomiseWeights()” heeft een reeks geneste loops. De buitenste loop telt het aantal gewichten, de eerste geneste loop telt het het aantal rijen en de meest geneste loop telt de kolommen van de matrix. De methode binnen de meest geneste loop, kiest voor elk gewicht in het netwerk een willekeurige float-waarde tussen -1 en 1.

```
public void RandomiseWeights()
{

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

        for (int x = 0; x < weights[i].RowCount; x++)
        {

            for (int y = 0; y < weights[i].ColumnCount; y++)
            {

                weights[i][x, y] = Random.Range(-1f, 1f);

            }

        }

    }

}
```

Afbeelding 22 Randomiseer gewichten functie

De sigmoïd functie is vrij makkelijk toe te passen. In het script neemt de functie een float-argument, dat wordt doorgegeven aan "Math.Exp()" -methode als parameter.

```
private float Sigmoid(float s)
{
    return (1 / (1 + Mathf.Exp(-s)));
}
```

Afbeelding 23 Sigmoïde functie

Het Netwerk Starten

StartNetwork is een functie die het netwerk uitoefent en die bestaat uit drie argumenten en twee retourwaarden. Van de invoersensor komen de drie argumenten plus de uitvoerwaarden, deze bepalen hoe de auto roteert, beweegt en hoe snel de auto gaat.
In de invoerlaag worden de waarden van de invoersensor ingesteld. Hierna volgt het feedforward- proces.
Ook worden er in de invoerlaag de waarden, met behulp van de hyperbolische tangent-functie, getransformeerd. Deze getransformeerde waarden worden doorgegeven aan de verborgen laag waarbij de doorgegeven waarde wordt vermenigvuldigd met het gewicht van de snyaps, waarna de bias wordt toegevoegd. Vervolgens wordt door de tnagens functie de waarde getransformeerd zodat het binnen het bereik van -1 en 1 past. Bij de uitvoerlaag wordt de eerste waarde gezien als de snelheid waar een float getal tussen 0 en 1 wordt gegeven door middel van de sigmoid-functie.En bij de tweede waarde wordt er een float waarde door de hyperbolische tangent functie gegeven tussen -1 en 1 voor het sturen van de auto.

```
public (float, float) RunNetwork(float a, float b, float c)
{
    inputLayer[0, 0] = a;
    inputLayer[0, 1] = b;
    inputLayer[0, 2] = c;

    inputLayer = inputLayer.PointwiseTanh();

    hiddenLayers[0] = ((inputLayer * weights[0]) + biases[0]).PointwiseTanh();

    for (int i = 1; i < hiddenLayers.Count; i++)
    {
        hiddenLayers[i] = ((hiddenLayers[i - 1] * weights[i]) + biases[i]).PointwiseTanh();
    }

    outputLayer = ((hiddenLayers[hiddenLayers.Count - 1] * weights[weights.Count - 1]) + biases[biases.Count - 1]).PointwiseTanh();

    //First output is acceleration and second output is steering
    return (Sigmoid(outputLayer[0, 0]), outputLayer[0, 1]);
}
```

Afbeelding 24 Feedforward functie

De klasse GAManager (Genetische Algoritme Manager)

Voor het trainen van het netwerk wordt de klasse GAManager gebruikt. Het beoordeelt elk netwerk en selecteert de beste populatie voor reproductie.

```
{
    [Header("References")]
    public CarController controller;

    [Header("Exploration And Exploitation Balance")]
    public int initialPopulation = 85;
    [Range(0.0f, 1.0f)]
    public float mutationRate = 0.06f;

    [Header("Recombination")]
    public int bestAgentSelection = 9;
    public int worstAgentSelection = 4;
    public int crossoverQuantity;

    private List<int> genePool = new List<int>();

    private int naturallySelected;

    private NNet[] population;

    [Header("Population Tracker")]
    public int currentGeneration = 0;
    public int currentChrom = 0;
```

Afbeelding 25 Variabelen GAManager

In afbeelding 17 worden de variabelen getoond die worden gebruikt in GAManager. Er zijn categorieën gemaakt om de voortgang makkelijk te volgen. De population header toont de initialPopulation en mutationRate in de inspector. De waarden hiervan kunnen worden aangepast in de inspector omdat ze openbaar zijn. Hierdoor worden de hardcoded waarden wel overschreven.

De startpopulatie wordt bepaald in de initialPopulation. Deze waarde wordt tijdens de hele training bijgehouden door de GA. De beoordeling vindt bijvoorbeeld plaats nadat 80 chromosomen geprobeerd hebben het probleem op te lossen. De kans op mutatie wordt bepaald bij de variabele mutationRate waarbij 1 een 100% kans op mutatie is, en 0.06 een mutatiekans van 6% betekent. In afbeelding 17 wordt de waarde op 6% gehouden.

In de Crossover Controls worden de beste en de slechtste oplossingen geselecteerd voor crossover. Deze waarden kunnen ook in de inspector veranderd worden. In dit geval worden de beste 146 en de slechtste 4 gekozen. Bij de variabele crossoverQuantity wordt de waarde ingesteld voor de k-puntscrossover.

Alle chromosomen worden in een lijst van gehele getallen opgeslagen in genePool, en er wordt een NeuralNetwork-object met de naam population gemaakt.

De variabelen currentGen en currenChrom geven een live-update over het aantal chromosomen en generatie die momenteel wordt uitgevoerd tijdens de run-time.

Bij het starten van het programma, wordt de functie InitialisePopulation() opgeroepen, die een nieuwe populatie chromosomen aanmaakt. Deze functie roept ook de functie RandomPopulate() op die in de GAManager-klasse zit. Hierbij worden neurale netwerk objecten gemaakt met gerandomiseerde waarden. Op deze manier wordt de initiële populatie van chromosomen gemaakt.

Sorteren En Selecteren

De fitheid van een chromosoom wordt samen met de gewichten en biases opgeslagen nadat de chromosoom klaar is met uitvoeren. Dit wordt gedaan door de NeuralNetwork-klasse. Om de chromosomen te ordenen op fitness wordt de bubble sort methode gebruikt. De functie PickBest() maakt gebruik van deze resultaten en de variabelen selectBest en selectWorst om de chromosomen te selecteren voor reproductie.

```
private NNet[] PickBestPopulation()
{
    NNet[] newPopulation = new NNet[initialPopulation];

    for (int i = 0; i < bestAgentSelection; i++)
    {
        newPopulation[naturallySelected] = population[i].InitialiseCopy(controller.LAYERS, controller.NEURONS);
        newPopulation[naturallySelected].fitness = 0;
        naturallySelected++;

        int f = Mathf.RoundToInt(population[i].fitness * 10);

        for (int c = 0; c < f; c++)
        {
            genePool.Add(i);
        }

    }

    for (int i = 0; i < worstAgentSelection; i++)
    {
        int last = population.Length - 1;
        last -= i;

        int f = Mathf.RoundToInt(population[last].fitness * 10);

        for (int c = 0; c < f; c++)
        {
            genePool.Add(last);
        }
    }

    return newPopulation;

}

private void SortPopulation()
{
    for (int i = 0; i < population.Length; i++)
    {
        for (int j = i; j < population.Length; j++)
        {
            if (population[i].fitness < population[j].fitness)
            {
                NNet temp = population[i];
                population[i] = population[j];
                population[j] = temp;
            }
        }
    }
}
```

Afbeelding 26 Functies sorteren en selecteren

De K-punt crossover functie wordt als crossover functie gebruikt in het algoritme. Om de crossover te maken in het neurale netwerk, neemt het programma twee nakomelingen, pakt alle gewichtswaarden van elk van deze objecten (opgeslagen in een lijst),en kiest dan de punten van de crossover. Gewichtswaarden uitgewisseld tussen de neurale netwerkobjecten op het opgegeven indexpunt. Omdat neuronen ook biases hebben, worden deze bias-waarden ook overgedragen. De functie krijgt een neurale netwerk object als argument en werkt de populatie bij met het resultaat van de crossover.

```
private void Crossover(NNet[] newPopulation)
{
    for (int i = 0; i < crossoverQuantity; i += 2)
    {
        int AIndex = i;
        int BIndex = i + 1;

        if (genePool.Count >= 1)
        {
            AIndex = genePool[Random.Range(0, genePool.Count)];
            do { BIndex = genePool[Random.Range(0, genePool.Count)]; } while (AIndex == BIndex);
        }

        NNet Child1 = new GameObject().AddComponent<NNet>();
        NNet Child2 = new GameObject().AddComponent<NNet>();
        Child1.Initialise(controller.LAYERS, controller.NEURONS);
        Child2.Initialise(controller.LAYERS, controller.NEURONS);
        Child1.fitness = 0;
        Child2.fitness = 0;

        for (int w = 0; w < Child1.weights.Count; w++)
        {
            if (Random.Range(0.0f, 1.0f) < 0.5f)
            {
                Child1.weights[w] = population[AIndex].weights[w];
                Child2.weights[w] = population[BIndex].weights[w];
            }
            else
            {
                Child2.weights[w] = population[AIndex].weights[w];
                Child1.weights[w] = population[BIndex].weights[w];
            }
        }

        for (int w = 0; w < Child1.biases.Count; w++)
        {
            if (Random.Range(0.0f, 1.0f) < 0.5f)
            {
                Child1.biases[w] = population[AIndex].biases[w];
                Child2.biases[w] = population[BIndex].biases[w];
            }
            else
            {
                Child2.biases[w] = population[AIndex].biases[w];
                Child1.biases[w] = population[BIndex].biases[w];
            }
        }
        newPopulation[naturallySelected] = Child1;
        naturallySelected++;
        newPopulation[naturallySelected] = Child2;
        naturallySelected++;
    }
}
```

Afbeelding 27 Crossover functie

De mutatiefuncties kunnen de gewichtswaarde van de verbindingen (synaps) in een neurale netwerk veranderen maar niet de biaswaarde van een neuron.

De mutationRate wordt beoordeeld door middel van een if-statement waarin een willekeurige float wordt gekozen in het bereik van 0 en 1. En als die waarde kleiner is dan de mutationRate, dan wordt de gewichtswaarde op een gekozen index verandert met behulp van de MutateMatrix() functie. In deze GA wordt de uniforme mutatie gebruikt. Het lijkt op uniforme crossover maar elke bit (gewichtswaarde) heeft een willekeurige kans om te muteren.

De functie MutateMatrix() neemt als argument een matrix die de gewichtswaarden heeft. Uniforme mutatie wordt gebruikt door n keer willekeurige posities in de matrix te kiezen, waarbij n een willekeurig getal tussen 1 en de grootte van de matrix is. Dit is beter dan een willekeurige kans toepassen op elk gewicht, en het verbetert de prestaties van het programma.

```
private void Mutate(NNet[] newPopulation)
{

    for (int i = 0; i < naturallySelected; i++)
    {

        for (int c = 0; c < newPopulation[i].weights.Count; c++)
        {

            if (Random.Range(0.0f, 1.0f) < mutationRate)
            {
                newPopulation[i].weights[c] = MutateMatrix(newPopulation[i].weights[c]);
            }

        }

    }

}

Matrix<float> MutateMatrix(Matrix<float> A)
{

    int randomPoints = Random.Range(1, (A.RowCount * A.ColumnCount) / 7);

    Matrix<float> C = A;

    for (int i = 0; i < randomPoints; i++)
    {
        int randomColumn = Random.Range(0, C.ColumnCount);
        int randomRow = Random.Range(0, C.RowCount);

        C[randomRow, randomColumn] = Mathf.Clamp(C[randomRow, randomColumn] + Random.Range(-1f, 1f), -1f, 1f);
    }

    return C;

}
```

Afbeelding 28 Mutatiefuncties

Voor het maken van de nieuwe populatie chromosomen, wordt de functie Repopulate() gebruikt. Het roept functies aan die binnen de klasse zijn gemaakt en geeft aan die functies parameters door. Wanneer Repopulate() wordt aangeroepen, verhoogt de generatie teller, de beste geselecteerde kandidaten gaan terug naar 0, populatie wordt gesorteerd en het

past selectie, crossover en mutatie toe op de vorige populatie om de nieuwe populatie te maken. Uiteindelijk wordt de chromosoomteller terug op 0 gezet en verwijderd het chromsoom met ResetToCurrentChrom(), waardoor de repopulatie gebeurt op het laagste niveau door het neuraal netwerk te resetten. In de klassen CarController en NeuralNetwork worden verdere berekeningen uitgevoerd. Wanneer de auto botst of de fitnesslimiet heeft bereikt, sterft het chromosoom door middel van de Death() functie die wordt aangeroepen op dat moment en haalt de eigenschappen en fitnesswaarde van het chromosoom eruit en slaat ze op.

5 RESULTATEN

De auto is in staat de baan in één generatie af te maken. Echter is dit afhankelijk van de waarden die in de inspector van de CarController en GeneticManager scripts zijn ingevoerd. Als de auto het racecircuit heeft afgemaakt betekent dat niet dat het algoritme het probleem heeft opgelost, het heeft alleen een mogelijke oplossing gevonden. Snelheid en sturing kunnen nog steeds worden verbeterd bij het zoeken naar andere mogelijke oplossingen bij het doortrainen van de auto.

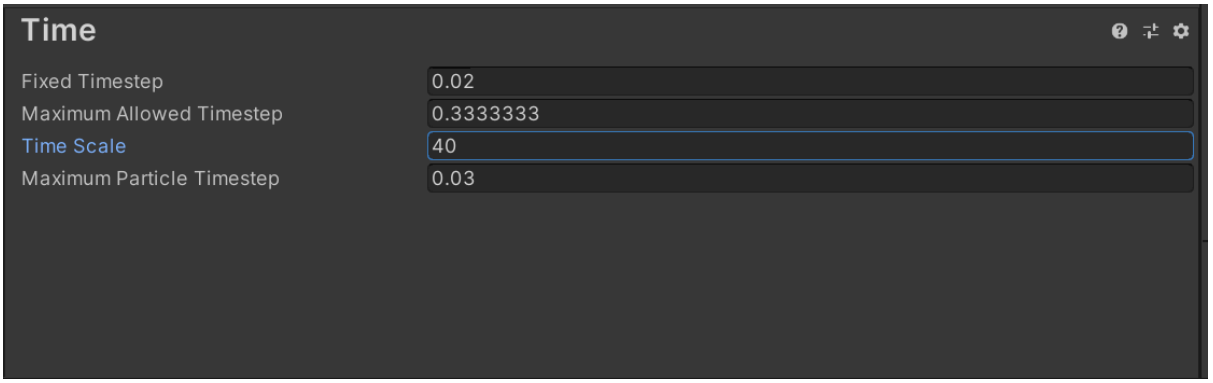
Bij het kiezen van een groot aantal beste chromosomen en een klein aantal slechtste chromosomen, weet de auto al binnen 5 generaties meerdere mogelijke oplossingen te vinden. In de loop der tijd worden de auto's per generatie steeds slimmer. Het verhogen van de verborgen laag helpt, maar het verhogen van de neuronen in de verborgen laag zorgt ervoor dat de prestaties van de auto nog beter worden.

In de epoch kunnen wij bepalen hoeveel generaties wij willen trainen. Als de epoch terecht komt op 0 wordt de tijd stilgezet en de prestaties van de auto opgeslagen in een csv bestand die te openen zijn in excel. In het bestand is te zien hoeveel oplossingen de auto heeft gevonden in een bepaald aantal generaties. De prestaties worden dan aanzienlijk beter omdat de auto steeds meer oplossingen vindt per generatie en ook het tempo van de oplossingen steeds hoger wordt.

	Solution Count, Generation Count, Ratio
2	1,2,0.5
3	2,3,0.6666667
4	3,4,0.75
5	4,4,1
6	5,5,1
7	6,5,1.2
8	7,5,1.4
9	8,6,1.333333
10	9,6,1.5
11	10,6,1.666667
12	11,6,1.833333
13	12,6,2
14	13,6,2.166667
15	14,6,2.333333
16	15,6,2.5
17	16,6,2.666667
18	17,6,2.833333
19	18,7,2.571429
20	19,7,2.714286

Afbeelding 29 Prestaties training

Om de auto's sneller te laten testen, biedt Unity de mogelijkheid de tijd van de simulatie te versnellen, waardoor de training sneller kan worden afgemaakt. De tijd kan aangepast worden als er wordt geklikt op "Bewerken", "Projectinstellingen", "Tijd" en daar de tijd te veranderen naar keuze. Voor mijn simulatie heb ik de tijd op 40 gezet, wat betekent dat het programma 40 keer zo snel zal draaien.



Afbeelding 30 Tijdschaal Unity

6 CONCLUSIE

In plaats van de gebruikelijke backpropagation-techniek voor het trainen van de auto's om zelf de baan af te maken, gebruikt dit project een genetisch algoritme (GA) samen met een neuraal netwerk. Voor elke richting gebruikt de auto een invoer sensor. De eerste invoersensor is schuin links, de tweede, centraal, en de derde, schuin rechts gericht vanaf de auto naar voren. Deze sensoren werken samen met de collider-functie van Unity en verkrijgen waarden die vervolgens doorgegeven worden aan het neurale netwerk, dat de tangent-activatiefunctie gebruikt. De invoer neuronen gebruiken echter de sigmoid-activatiefunctie omdat het sturen kan variëren van -1 tot 1, waar 0 het midden is, en -1 en 1 voor links en rechts staan. De versnelling van de auto kan variëren van 0 tot 1. Bij 0 staat de auto stil, en bij 1 heeft de auto zijn volledige snelheid bereikt.

Het genetisch algoritme past selectie, crossover en mutatie toe op de neurale netwerken om de prestaties per generatie van de auto's te verbeteren. Het netwerk bestaat uit biases met de gewichten van de verbindende synapses waar deze waarden uitgewisseld worden in nakomelingen bij de chromosomen. Mutatie heeft alleen effect op de gewichtswaarden.

Na het bereiken van een bepaalde hoeveelheid getrainde generaties, worden de resultaten van de training opgeslagen in een csv-bestand voor analyse. Dit project kan goed gebruikt worden om te experimenteren met genetische algoritmen en kunstmatige neurale netwerken (ANN) voor verschillende soorten trainingen waar waarden van de invoersensoren nodig zijn.

Het moeilijkste aan dit project was het implementeren van een kunstmatig neuraal netwerk (ANN) en het laten werken met een genetisch algoritme. Het hele proces is vrij complex. Na het coderen van het model moet het getraind worden zodat mogelijke oplossingen gevonden kunnen worden. Neurale netwerken en genetische algoritmen vond ik desondanks zeer boeiend, vooral omdat het geïnspireerd is vanuit de biologie. Net als hoe de sterksten overleven in de natuur, overleven de beste auto's in mijn simulatie per generatie.

8 BIBLIOGRAFIE

Genetic Algorithms + Neural Networks = Best of Both Worlds. (2018, March 26). Towards Data Science. Retrieved December 14, 2023, from https://towardsdatascience.com/gas-and-nns-6a41f1e8146d

Kuo, J. C. (2020, September 15). *Genetic Algorithm in Artificial Neural Network and its Optimization Methods | by Jonathan C.T. Kuo | The Startup*. Medium. Retrieved December 12, 2023, from https://medium.com/swlh/genetic-algorithm-in-artificial-neural-network-5f5b9c9467d0

Newcombe, J. (2015, February 6). *Implementing Genetic Algorithms in C#*. CodeProject. Retrieved Mei 15, 2023, from https://www.codeproject.com/Articles/873559/Implementing-Genetic-Algorithms-in-Csharp

Rahul, K. (n.d.). *Artificial Neural Network - Meaning, Types, Examples, Applications*. WallStreetMojo. Retrieved Januari 4, 2024, from https://www.wallstreetmojo.com/artificial-neural-network/

SHARMA, S. (2017, September 6). *Activation Functions in Neural Networks | by SAGAR SHARMA*. Towards Data Science. Retrieved Januari 2, 2024, from https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

shindhe, d. (2019, March 21). *Genetic algorithm in unity using C# | by dhruv shindhe | Analytics Vidhya*. Medium. Retrieved Mei 2, 2023, from https://medium.com/analytics-vidhya/genetic-algorithm-in-unity-using-c-72f0fafb535c

What are Neural Networks? (n.d.). IBM. Retrieved April 30, 2023, from https://www.ibm.com/topics/neural-networks