Yapay Zeka Projesi

Proje Başlığı: **Yapay Zeka İle Mario**

Proje Özeti: Genetik Algoritma ve Sinir Ağı kullanarak Super Mario Bros oyununda Mario'nun bir bölümü en kısa sürede bitirmesini sağlamak.

Raporun Bölümleri:

Bölüm 1: Oyun ile Etkileşim

Bizim istediğimiz şey marionun hızlıca oyunu bitirmesini sağlamak. Bu amacı gerçekleştirmenin önkoşulu ise oyunla etkileşime geçmek.

Öncelikle oyun hakkında bilgi almamız ve oyuna müdahale etmemiz gerekiyor.

Bu oyunun kodu açık şekilde elimizde olsaydı eğer işimiz biraz daha basit olurdu. Oyunun kodunda tanımlanmış değişkenlere ve tutulan verilere direkt erişimimiz olurdu ve marionun bir aksiyon almasını istediğimizde bunu doğrudan sağlayabilirdik.

Tabi oyunun farklı dillerde yazılmış veya farklı oyun motorları ile yapılmış replikaları da var ve bunlardan birini kullanmak da bir seçenek olabilirdi. Ama biz öyle yapmadık.

Not: Oyunun koduna veya (varsa) kullandığı oyun motoruna direkt erişimin olması eğitimi daha hızlı hale getireceğini düşünüyorum.

Sebepler:

- **Ram değerlerine erişim:** Oyunun iç mekaniklerini (ör. karakterin konumu, düşmanların durumu, kalan süre) doğrudan alabiliriz. Bu, ekran görüntüsü işleme gibi zaman alıcı yöntemlere gerek kalmadan **doğru ve hızlı veri** sağlar.
- **Render gereksinimi olmadan çalıştırma**: Görsel çıktıya ihtiyaç duymadan sadece oyun motorunun mantığını çalıştırabiliriz. Bu, simülasyon hızını artırır.

- **Hızlandırılmış döngüler**: Oyunun hızını artırarak, normalden çok daha hızlı şekilde milyonlarca deneme yapabiliriz.
- **Direkt input gönderimi**: Klavye ya da fare taklidi yapmak yerine, motor seviyesinde giriş yapabiliriz. Bu, aksiyonların gecikmesiz ve doğru bir şekilde uygulanmasını sağlar.

Eğer (varsa) oyunun kullandığı motora veya oyunun kaynak koduna erişimimiz yoksa oyunu çalıştırmak ve oyunun ekranından toplayacağımız görsel verileri işlemek gerekir. Ki bu görsel verilerin işlenmesi kısmında da bir yapay zeka kullanabiliriz.

Son olarak şunu söyleyeyim ki bizim oyunumuz nes (nintendo entertainment system) platformu için yazılmış dolayısıyla bu oyunu bir emulator yardımıyla çalıştırmak gerekir. Fakat biz bu yolu da izlemedik.

Biz python için yazılmış bir kütüphane kullandık. **retro kütüphanesi**, klasik video oyunlarını çalıştırmak ve bu oyunlarla yapay zeka (AI) algoritmaları geliştirmek için kullanılan bir python kütüphanesi. Sega Genesis, NES, SNES gibi klasik konsolların oyunlarını çalıştırabiliyor. Oyunun ram değerlerine erişip okuyabiliyoruz ve oyuna input sağlayabiliyoruz.

Oyunun ram değerlerine erişebiliyoruz ama ramde hangi bilgilerin nerede tutulduğunu tam olarak bilmiyoruz.
Sağolsun birileri bizim için bu ram adreslerinin ve alabileceği değerlerin neyi ifade ettiğini listelemiş.

Evet bu bölümün sonunda oyunun verilerine erişebilir haldeyiz ve inputlarımızı de oyuna verebiliyoruz.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	170	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

verileri
alışımız

```python
import retro

env = retro.make(game='SuperMarioBros-Nes', state='Level1-1')
env.reset()
ram = env.get_ram()

# ['B', None, 'SELECT', 'START', 'UP', 'DOWN', 'LEFT', 'RIGHT', 'A']
buttons = [0,0,0,0,0,0,0,1,1]

env.step(buttons) # tuşların durumundan sonraki frame'i oluşturuyor.
```

girdileri verişimiz

Bölüm 2: Probleme Yaklaşım

Problemi basitleştirirsek; Elimizde veriler var ve bu verilerden hareketle doğru çıktılar almalıyız. Peki bunu nasıl yaparız?
Bu girdileri çıktılara çeviren fonksiyonu birkaç farklı şekilde tasarlayabiliriz. Aklıma gelenleri yazayım. Daha farklı çözümler de elbet vardırlar.

İlk yöntem: Tüm durumlarda tam olarak ne yapması gerektiğini söyleyen bi eşleme yapmak. Bu yöntem daha küçük problemlerde işe yarar. Mesela XOX oyununda olası 9! = 362.880 durum için doğru çıktıları tutabilir ve sunabiliriz. Ama bu yöntem böylesi basit bi problemde bile uygulaması oldukça zor. Bunu bizim problemimize uygulamak isteseydik yaklaşık 70x(3^69) gibi devasa bir sayı için çıktıları bulmalıydık. Bellek ayırmak bir yana tüm bu durumlar için en iyi çıktının ne olduğunu nasıl hesaplamak bile sorun.

İkinci yöntem: Rule-Based AI yazmak. Önceden tanımlanmış kurallara ve mantıksal ifadelere dayanarak çalışan bir yapay zeka. **"eğer-bu olursa-şunu yap" (if-then)** mantığına bağlı. Örneğin;

Eğer önünde düşman veya duvar varsa;
    sağa doğru koşarken zıpla.
Değilse;
    sağa doğru koş.
Bu yöntemle yazılmış bi yapay zeka daha az bellek tutar. Fakat bu yöntemde marionun ne zaman ne yapması gerektiğini yeterince iyi tanımlamalıyız. Mario gibi görece basit bir oyunda bile çok farklı durumlarda ne yapılması gerektiğini yazmak bile çok karmaşık bi hal alabilir. Oldu ya sonucu elde ettik daha iyi yazılıp yazılamayacağından emin de olamayabiliriz.
Üçüncü yöntemimiz belki makine öğrenmesiyle ilk yöntemde yaptığımız gibi başta görece küçük bir miktar ve oldukça çeşitli girdileri ve “doğru” çıktıları bir makine öğrenmesi modelinin eğitiminde kullanmak ve kalan olası bütün durumlar için doğru çıktılar vermesini ummak olabilir. Bu problemde de doğru olduğunu varsaydığımız bazı çıktılar var. Ama o çıktıların en iyi çıktılar olduklarından emin miyiz. Emin olduğumuzu varsayarsak bile mario gibi görece basit bir oyunda doğru hareketleri

hesaplamak ve daha karmaşık belki 3 boyutlu bir oyunda da aynısını yapabilir miyiz. Veya yapmaya değer mi? Doğru çıktıları bilmeden karar verecek bir ai yok mudur acaba?

Ve sonunda dördüncü yöntemimize gelelim. Bizi bütün bu sorunlardan kurtaran fakat yine de mükemmel olmayan, genetik algoritma ve sinir ağının birlikte kullanımı.

Bölüm 3: Genetik Algoritma

Genetik algoritma yapay zekadan bağımsız bir algoritmadır. Yapay zeka haricinde de kullanılır.

Genetik algoritma biyolojik evrimden ilham alan bir optimizasyon yöntemidir. karmaşık problemlerde, geleneksel yöntemlerin yetersiz kaldığı durumlarda kullanılır. Bu algoritma çözüm üretmek için doğal seçilim, genetik çaprazlama ve mutasyon gibi biyolojik olayları taklit eder.
Birey veya Kromozom (Individual, Chromosome): Optimize etmek istediğimiz fonksiyonun olası girdi değerleridir. Genelde dizi olarak tutulurlar.

Gen: Bir bireyde bulunan girdi değerlerinden sadece bir tanesidir.

Popülasyon veya Nesil(Population, Generation): Aynı anda değerlendirilen, yaşayan bireylerin oluşturduğu topluluk. Olası girdilerin birden fazlası.

Genetik algoritma birden fazla fonksiyonun birleşiminden oluşur. Bildiğimiz kadarıyla anlatalım.

İlk popülasyonun üretimi: İlk popülasyon genelde rastgele değerlerle oluşturulur fakat belirli bir problemi çözmek için önceden bilgiye dayalı bir başlangıç da yapılabilir.

Uygunluk(Fitness) Fonksiyonu: Her bireyin uygunluğunu değerlendiren bir fonksiyondur. Bir çeşit uygunluk skoru döndürür. İyi bireyler daha yüksek fitness değerlerine sahip olur. Bu fonksiyon, algoritmanın doğru çözümü bulma başarısını etkileyen çok önemli bir parçasıdır. Farklı problemler için farklı şekillerde tasarlanır.

Seçim(Selection) Fonksiyonu: Daha yüksek fitness değerine sahip bireyleri üremek, çaprazlama ve mutasyon işlemlerine katılmak için seçer. Seçilemeyen bireyler ölür, silinir, çöpe atılır, veya artık siz ne derseniz. Bir bakıma kısa süreliğine popülasyondaki birey sayısını azaltır. Bu seçim dilediğiniz şekilde olabilir. Yaygın kullanılan seçim yöntemlerine bi bakalım.

> Rulet Seçimi(Roulette Selection): Bireylerin fitness puanlarını tüm popülasyonun toplam fitness değerine bölerek olasılıklar elde eder. Ve bu olasılıkları kullanarak rastgele seçim yapar. Fitness'ı yüksek olan bireylerin üreme ihtimali diğerlerine göre daha fazla

olur.

Turnuva Seçimi(Tournament Selection): Keyfimizce belirlenmiş(parametrik) sayıda rastgele seçilmiş bireyin arasından en iyi olanı seçer. İstediğimiz sayıda bireyi seçene kadar devam eder.

Elitist Seçim(Elitism): En yüksek fitness değerine sahip belli sayıda bireyin doğrudan seçerek korurken daha az fitness'a sahip olan kalan kısmın da bir kısmı turnuva veya rulet seçimi ile seçilir. Bu sayede en iyi bireyler kaybolmaz ve düşük fitness'a sahip bireylere de şan tanınır.

Deterministik Seçim: Sadece en yüksek fitness değerine sahip bireyler seçilir. En güçlü olan hayatta kalır. Elitist seçime göre çeşitlilik açısından kötüdür. Çeşitliliğe daha sonra değineceğim.

Daha fazla seçim yöntemi olsa da sadece bunlara değinmek istedim. Farklı seçim algoritmalarının farklı avantaj ve dezavantajları var. Kimileri en iyi bireyleri korumakta daha iyiyken kimileri de çeşitliliği korumakta daha iyi olabiliyor. Bazıları yavaş bazıları ise daha hızlı(ne pahasına) olabiliyor. Her bir fonksiyonu küçük değişikliklerle farklı şekillerde yazmak mümkün. Nasıl bir seçim algoritması kullanacağımız ise bize kalıyor. En iyisini seçmek için umarım iyi bir seçim algoritmanız vardır.

Çaprazlama(Crossover) Fonksiyonu: Seçim ile azaltılan birey sayısını artırmaya yarar. Daha önce seçilen bireylere ebeveyn(parent) denir. Bu fonksiyon iki ebeveynin (belki daha fazlasının) genetik bilgilerini karıştırarak yeni bireyler(çocuklar) oluşturur. Bu foksiyon da farklı yöntemler kullanılarak yazılabilir. Bu yöntemlerden birkaç tanesini yazayım.

Tekdüze Çaprazlama(Uniform Crossover): Rastgele seçilen iki ebeveynin birbirine karşılık gelen her geninin birinin rastgele seçilmesidir.
Ebeveyn 1: 1 2 3 4 5 6 7 8
Ebeveyn 2: A B C D E F G H
Çocuk : A 2 C 4 E 6 7 H

Tek Noktalı Çaprazlama(Single Point Crossover): İki ebeveynin kromozomu alınır. Kromozom üzerinde rastgele bir kesim noktası belirlenir.
Belirlenen kesim noktasından itibaren iki ebeveynin genleri takas edilir. Örneğin;
Kesim noktası 3. elemandan sonra olsun.
Ebeveyn 1: 1 2 3 | 4 5 6 7 8
Ebeveyn 2: A B C | D E F G H
Çocuk 1: 1 2 3 | D E F G H
Çocuk 2: A B C | 4 5 6 7 8

Aynı mantıkla çalışan iki noktalı çaprazlama ve çok noktalı çaprazlama da yazılabiliyor.

Bunlardan daha farklı olan genlerin ortalamasını alan, bir kromozomun belli bir bölümünü sadece değiştiren veya iki genin değerleri arasında rastgele bir değere sahip gen oluşturan çaprazlama fonksiyonları da var. Yine hepsinin iyi ve kötü tarafları var.

Mutasyon(Mutation) Fonksiyonu: Kromozomlar üzerinde küçük değişiklikler yaparak çeşitliliği artıran bir fonksiyon. Bi kaç çeşidini yazayım.

Takas(Swap) Mutasyonu: Kromozomdaki genlerin bi kısmının birbiri arasında yer değiştirmesi.
[A B C D] -> [A D C B]

Çevirme(Inversion) Mutasyonu: Kromozomun bir bölümünün ters çevrilmesi.
[A B C D E] -> [A D C B E]

Gen yerleştirme(Gene Replacement): Kromozomdaki bazı genlerin rastgele değerlerle değiştirilmesi.

Sabit Aralıklı(Boundary) Mutasyon: Genin belli bi yüzde değeri arasında rastgele geğiştirilmesi.
yeni_gen =
eski_gen * (1+ random_value_in_range(-0.05, 0.05))

Bunun gibi mutasyon ve bunlardan farklı mutasyon fonksiyonları var. Bu fonksiyonların birkaçının beraber kullanılmasına da çoklu mutasyon deniyor. Üstte bahsedilenler statik(sabit) mutasyonlardı. Bir de dinamik(değişen) mutasyon fonksiyonları var. Mesela nesiller ilerledikçe mutasyon yüzdesini düşüren, çeşitlilik azaldıkça bu yüzdeyi artıran bir fonksiyon gibi zamanla davranışı değişen fonksiyonlar da yazılabilir.
Çeşitlilik(Diversity): Bir popülasyondaki bireylerin birbirinden farklılığını ifade eder. Çeşitliliğin fazla olması daha fazla keşif yapılmasını daha farklı çözümlerin denenmesini sağlar. Çeşitliliğin az olmasıysa genelde kötüdür. Çeşitlilik azaldıkça algoritma yerel optimumlara takılabilir ve çözüm uzayının dar bir noktasına odaklanarak mutlak optimum kaçırılabilir. Fakat mulak optimumu bulduğunuzu düşündüğünüzde azalması daha hassas çözüm bulunabilir.
Çeşitliliği etkileyen şeyler genelde seçim, çaprazlama ve mutasyon fonksiyonlarıdır.

Durdurma Kriteri(Stopping Criterion): Algoritmanın ne zaman durdurulacağını belirleyebilirsiniz. Belirli bir nesil sonrasında durdurmak veya uygunluk değerinde belli bir süre belirlenen değişimin altında değişim yaşanması durumunda durdurmak gibi durdurma kriterleri belirlenebilir.

Şimdi genetik algorityı kısaca özetleylim.

Diyelim ki elimizde bir fonksiyonun optimizasyonu ile ilgili bir problem var.Bu fonksiyonun alacağı girdileri(parametreleri) ve vereceği çıktıları var.

Başlangıçta ilk nesli oluşturmak için rastgele(genelde) değerlere sahip girdi topluluğundan "yeterince" sayıda üretiyoruz.

Probleme uygun bi uygunluk(fitness) fonksiyonu yazıyoruz. Birden fazlaolan çıktıyı tek bir sayıya indiriyo. Bu da karşılaştırmayı kolay yapıyo.

Sonrasında bu ilk neslin girdilerinden alacağımız çıktıları buluyoruz. Bu çıktıları uygunluk fonksiyonuna sokuyoruz ve her girdi topluluğunun uygunluk değerini hesaplıyoruz.

Bu uygunluk değerlerini belirleyeceğimiz seçim fonksiyonuna sokuyoruz. Daha uygun girdileri seçiyor. Ve popülasyondaki birey sayısı azalıyo.

Sonrasında belirleyeceğimiz çaprazlama fonksiyonuyla seçilen girdi topluluklarından daha sonra kullanmak için yeni girdi toplulukları oluşturuyoruz. Burada popülasyondaki birey sayısı önceden belirlediğimiz sayıya tekrar çıkıyo.

En sonunda elimizdeki fonksiyonu belirlediğimiz mutasyon fonksiyonuna sokuyoruz.

Bu şekildeki döngüyü bir süre çalıştırıyoruz(durma kriteri belirlenebilir). Her şeyin sonuna geldiğimizde ise fonksiyonumuzun en optimum girdilerini ve çıktılarrını bulmuş oluyoruz.

Basit bir problemi genetik algoritmayla çözelim.

Problem: Elimizde 40 metrelik bi çıta var. Biz bu çıtadan dikdörtgen bi çerçeve yapmak istiyoruz. Çerçevenin alanının maksimum olması için çerçevenin kenarları kaçar metre olmalı? Çerçevenin maksimum alanı kaç m^2 olur?

Bu problem türevin geometrik yorumu ile kolayca çözülür. Fonksiyon $f(x) = x * (20-x) = -x^2 + 20x$, $0<x<40$ olur.
Maksimum için f(x)'in maksimumu 100 metre kare olduğu ve kenarların 10'ar metre olduğunu bulmak zor değil.

Bu problemi yine de genetik algoritmayla çözelim.

$f(x) = -x^2 + 20x$, $0<x<40$

maksimumu bulmak istediğimiz için f(x) ne kadar büyük olursa uygunluk değeri o kadar fazla olmalı. Bundan dolayı uygunluk(fitness) fonksiyonumuzu burada f(x)'i f(x)'e götüren birim fonksiyon şeklinde seçebiliriz. Yani fitness(f(x)) = f(x) oldu.

Popülasyon sayımız 4 olsun mesela (keyfimize göre)
Bizim girdilerimiz sadece 1 tane o da x.
İlk popülasyonu rastgele oluşturalım. (0 ile 20 arası)

	birey_1	birey_2	birey_3	birey_4
x :	12,21	17,03	3,03	5,73

çıktılarımız(ve uygunluk değerlerimiz):

f(x):	95,12	50,58	51,42	81,77

Buradan seçim fonksiyonumuzla 2 tane en yüksek çıktıya ve aynı zamanda uygunluğa sahip olanı seçelim. 12,21 ve 5,73 en yüksek uygunluğa sahip x'ler veya genler veya kromozomlar.

Fonksiyonumuzun girdilerinin sayısı 1 olduğundan çaprazlama yapamıyoruz. Çünkü sadece 1 gen var. Yerleri değiştiğinde bir şey değişmiyor. Bu yüzden direkt ±%5 aralığında aralıklı mutasyon uygulayalım ve bu şekilde tekrar 4 tane x değerine çıkalım. Yeni x'lerimiz;

x :	12,21	5,73	5,81	12,28

.

Buradan sonra döngüye girip aynı şeyleri tekrarlarsak:

.

.

.

x :	10,23	9,89	10,11	9,99
f(x):	99,95	99.99	99.99	99.99

gibi değerlere ulaştığımızda durabiliriz ve cevabı tahmin edebiliriz. Görüldüğü gibi bu yöntem kesin sonuçlar vermez. Sadece yaklaşık bir sonuç verir.

Ve bu yaklaşık sonucun doğruluğundan emin olamayız.

Çözdüğümüz problemdeki fonksiyonu sadece 1 tane yerel maksimumu vardı ve bu yerel maksimum aynı zamanda mutlak maksimumdu. Eğer birden fazla yerel maksimum olsaydı bu algoritma bizi o yerel maksimuma da yaklaştırabilirdi.

Ve bu kesin olmayan sorunlu cevaba ulaşmamız da çok uzun sürdü.

Peki bu algoritmayı bunca kötülüğüne rağmen neden kullanalım?

Çözdüğümüz problem oldukça basit bir problemdi. Sadece bir parametresi ve bir çıktısı vardı. Fonksiyonumuzun türevini de rahatlıkla alabiliyorduk. Fakat ne yazık ki tüm problemler bu kadar basit değil. Genetik algoritma da zaten daha karmaşık optimizasyon problemlerin yaklaşık çözümünü bulmak için veya bize çözüm hakkında bir fikir vermek için daha kullanışlı.

Bölüm 4: Yapay Sinir Ağı

Uzun uzun genetik algoritmayı anlattık peki yapay sinir ağlarının bununla ne ilgisi var?

Tekrar hatırlatmak gerekirse genetik algoritma bir "fonksiyonu" optimize ediyordu.

Tekrar projemizin konusu olan mario'ya dönersek şunu farkediyoruz ki elimizde bir fonksiyon yok.
Evet elimizde oyundan alacağımız girdilerimiz var ve çıktı olarak istediğimiz olası değerler var ama aradaki dönüşümü yapacak olan şey eksik.

Bunun için bir fonksiyon yazmanın çok zor olduğunu daha önce söylemiştim. Dolayısıyla bize girdilerimizden çıktılar veren kendimizin yazamadığı bir fonksiyonumsu lazım. İşte tam burada yapay sinir ağları bizim için harika bir seçenek oluyor.

Yapay sinir ağlarını genetik algoritma gibi uzun uzun anlatmayacağım. Sadece şunu söylemek lazım ki yapay sinir ağları da bir çeşit ayarlanabilir fonksiyonlardır.

Örnek için;

İleri beslemeli bi 2x3x2'lik sinir ağı düşünürsek;
12 ağırlık ve 5 bias değeri var.

x_1 ve x_2 giriş nöronları, h_1 h_2 h_3 gizli nöronlar ve
y_1 y_2 çıkış nöronları olsun.

f_1 ve f_2 bizim belirleyeceğimiz aktivasyon fonksiyonlarımız (relu, sigmoid, tanh gibi) olsun.

$$y_1 = f_2(w_{h1y1} * f_1(w_{11}*x_1 + w_{12}*x_2 + b_{h1}) + w_{h2y1} * f_1(w_{21}*x_1 + w_{22}*x_2 + b_{h2}) + w_{h3y1} * f_1(w_{31}*x_1 + w_{32}*x_2 + b_{h3}) + b_{y1})$$

$$y_2 = f_2(w_{h1y2} * f_1(w_{11}*x_1 + w_{12}*x_2 + b_{h1}) + w_{h2y2} * f_1(w_{21}*x_1 + w_{22}*x_2 + b_{h2}) + w_{h3y2} * f_1(w_{31}*x_1 + w_{32}*x_2 + b_{h3}) + b_{y2})$$

$f(x_1 , x_2) = (y_1 , y_2)$ yukardaki gibi hesaplanır.
Yapay sinir ağları da girdileri alırlar ve çıktılar verirler. Çıktıları da ağırlık(weight) ve bias sabitlerini kullanarak oluştururlar.
Yapay sinir ağlarının ağırlık ve bias değerleri değiştirilebilen parametrelerdir. Ve eğer bu parametreler yeterince "optimize" edilebilirse girdiler için yaklaşık istenen çıktıları verebilir.
Peki ağırlık ve biasleri optimize etmenin bir yolu var mı?

Sanki bu ağırlık ve biasleri optimize etmek için bir yöntem biliyoruz gibi. Evet, cevap genetik algoritma.

Bölüm 5: Bizim Yaptıklarımız

Şimdi sırayla yaptıklarımızı anlatayım.

Oyunun raminden aldığımız değerlerden ekrandaki pikselleri bloklara ayırdık ve bu bloklardan 70 tanesini girdi değerimiz olarak seçtik. 0 boşluk, 1 yapı, 170 mario, 255 de yaratıkları temsil etti.

Elimizdeki bu 70 uzunluğundaki listedeki değerleri normalize ettik. (sinir ağı kullanacağımızdan)

Çıktılarımız 6 tane tuşun her birine basılıp basılmadığını temsil edecek olan sayılar olacaktı.

Bu bilgilerden yola çıkarak kuracağımız sinir ağının boyutlarını 70x64x32x6 olarak belirledik.

Dolayısıyla her mario için 6720 ağırlık ve 102 bias değerimiz olacaktı. Her mario için 6822 sayı tutacaktık. Bu 6822 değer marionun kromozomu oldu.

Genetik algoritma için popülasyonumuzun boyutunu belirledik. 64 mario olacaktı ve bu 64 mario içinden 16'sı seçilecekti.

Programımızın kapatılıp kaldığı yerden devam etmesini istediğimiz için seçilen mario'ların ağırlık ve bias değerlerini json dosyasına kaydetmek istedik. Bu da 16 seçilecek mario için 109152 sayıya denk geliyordu.

Genetik algoritmadaki ilk popülasyonu oluşturmak yerine ilk seçilmiş mario'ların kromozomlarını rastgele oluşturmak ve bu kromozomları chromosomes.json dosyasına yerleştirmek için chromosome_generator kodunu yazdık.

Programın başlangıcında bu 16 mario'nun kromozomlarını chromosomes.json dosyasından çektik ve bir array'e yerleştirdik.

16 mario'nun kromozomlarından 48 mario oluşturmak için tekdüze çaprazlama fonksiyonu yazdık. (uniform_crossover)

Seçilecek olan 16 mario'dan ilk 4 mario'yu elit olarak seçtik ve 12 tanesini ruletle alacaktık. Bu yüzden elimizdeki 64 mario kromozomunun son 60 tanesine etki edecek mutasyon fonksiyonunu yazdık.(ilk 4 tanesine uygulamamızın sebebi onların elit olması ve genlerini korumak istememiz.) Hangi marionun daha uygun olduğunu hesaplayacak bir uygunluk fonksiyonu yazdık. (zamana, gidilen yola ve bölümün bitirilip bitirilmediğine bağlı bir tane)

64 marionun kendi kromozomlarının tuttuğu ağırlık ve biasleri tek tek sinir ağına yerleştirdik. Bu ağırlık ve bias değerleriyle girdilerden çıktı üretip hareket eden mariolar için uygunluk değerlerini hesapladık.

Her neslin istatistiğini tutmak için bir json dosyası daha oluşturduk. Ve şu anki neslin ortalama uygunluk, zaman, gidilen uzaklık değerlerini bir de bitirme yüzdesini o dosyaya kaydettik.

Bu uygunluk değerlerine göre marioları sıraladık ve ilk 4 mario'yu ayırdık. Kalan 60 mario arasında rulet seçimiyle 12 mario seçtik(elitist rulet seçimi). Elitler ve ruletten gelenleri bir araya getirip yeni neslin ebeveynlerini oluşturmuş olduk.

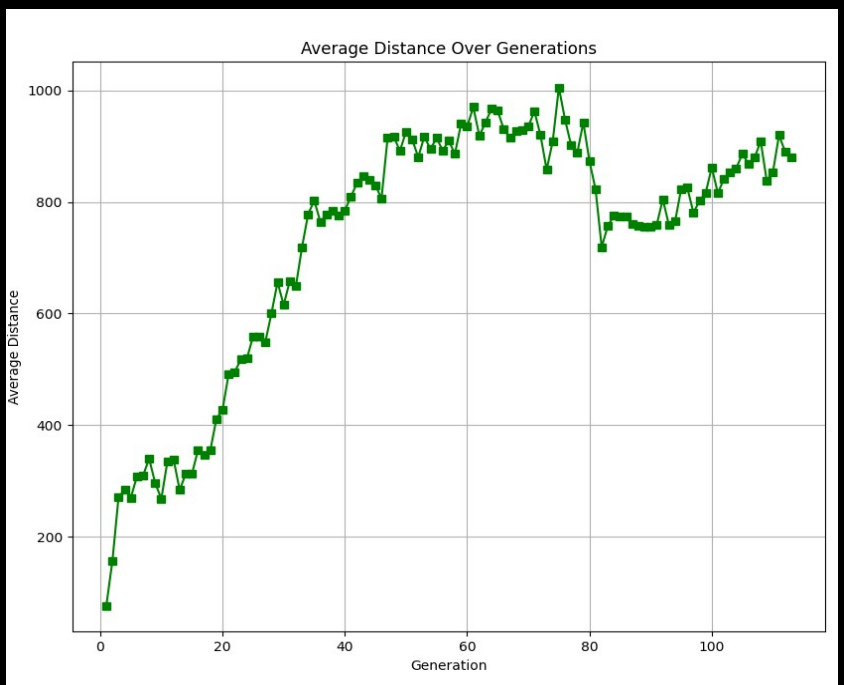
Bu ebeveynlerin kromozomlarının kaybolmaması için tekrar chromosomes.json dosyasına kaydettik.

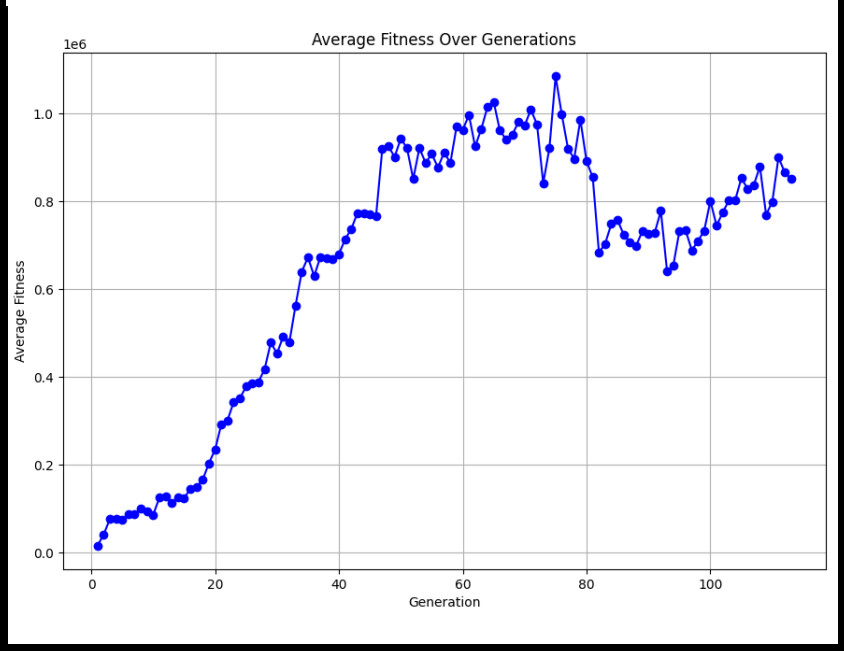
Bu noktadan sonra bir döngüyle aynı işlemleri tekrarladık.

Programı istediğimiz zaman kapatıp tekrar açıp kaldığımız yerden devam edecek şekilde yazmış olduk.

Bölüm 5: Sonuç ve Eleştiriler

Sonuç:





(not: 80’den itibaren gözüken düşüşün sebebi benim kod üzerinde düzenleme yapmam. Onun haricinde marioların davranışında bir gariplik yok.)

İlk yazdığım kodda deterministik seçim kullandığım için çeşitliliğin azaldığını düşündük. Bundan dolayı da yerel maksimuma takılmış olabileceğimizi düşündük. Zaten grafikteki duraksama da bunu destekliyor.

Bu şüphemizden sonra yeni bir başlangıç popülasyonu oluşturup onu elitist rulet seçimiyle tekrar eğitmeye karar verdik. Ki buna da zamanımız yetmedi.

Sorunun diğer muhtemel nedenleri;

Genetik algoritma tasarımımımız kötü olabilir.

Girdi olarak seçtiğimiz değerler yanlış olabilir.

Başka bir sebep de bizim karar verdiğimiz sinir ağının boyutunun yetersiz olması olabilir. 70X64x32x6 yerine 70x128x64x6 seçebiliriz.

Veya belki de sadece zamana ve daha fazla işlem gücüne ihtiyacımız vardır.

Günün sonunda Mario’nun bölümü en hızlı şekilde bitirmesini bırakın, bölümü bitirmesini bile sağlayamadık. Ama en azından mario’yu biraz yürütüverdik.

Eleştrilere gelince:

Tasarım: Tasarladığımız genetik algoritmayı diğerleriyle kıyaslayamasak da bize çeşitliliği yok ediyor ve yerel optimuma sıkışıyormuş gibi geldi.

CPU Kullanımı: Bizim problemimiz kolay paralellenebilir bir problem olduğundan kolayca GPU üzerinde çalıştırabilirdik ve eğitim daha hızlı olurdu.

Anlık Girdi: Biz oyunun her frame'i için girdi alıp her frame'i ayrı olarak değerlendiriyoruz. Belki daha önceki frame'lerden aldığımız verileri saklayıp hepsini beraber kullanabilirmişiz. Mesela her önceki 60 frame için alacağımız girdileri saklayıp. Sinir ağına girdi olarak verebilirmişiz. İşe yarar mı gerçi bilmiyoruz.

Mario'ların Eğitimi: Mariolar yaşadıkları sürece sinir ağlarını güncellememeleri veya değerlerini değiştirmemeleri bize sorun yaratmış olabilir. Eğitimi uzatmış veya daha kötü hale getirmiş olabilir.

Mario'ların Sıkışması: Çoğu mario bir duvara kafa atıp süresi bitene kadar aynı duvara koşuyor. Bazı mario'lar ise öylece bekliyor. Çevrenin değişip değişmediğini kontrol eden bir kod olsa idi bu bekleme süresince frame üretmemiz gerekmezdi ve program hızlanırdı.

Kod Yazımı: Kodu hiç estetik yazmadık. Optimize değil. Yeniden düzenlemek veya parametreleri değiştirmek kolay değil. GUI yazmadık yani insanların kodla ilişmesi gerekiyor. Paralelliğin olmadığından zaten bahsettik. Performansın bolca gerektiği böyle bir projede python yerine c++ kullanabilirmişiz.

Olumlu Eleştiri: Kod iyi kötü çalışıyor.

https://github.com/NeIseZitti/Mario_AI