



**UNIVERSIDADE FEDERAL DE MATO GROSSO
FACULDADE DE ENGENHARIA - FAENG**

**DOCUMENTAÇÃO DO 1º TRABALHO
INTELIGÊNCIA ARTIFICIAL
REDES NEURAIS**

ALUNOS:

Luiz Felipe da Silva Marian

RGA:

202021901036

CUIABÁ

2023

ALUNOS:

Luiz Felipe da Silva Marian

RGA:

202021901036

DOCUMENTAÇÃO DO 1º TRABALHO:
INTELIGÊNCIA ARTIFICIAL
REDES NEURAIS

Essa Documentação é apresentada como forma de obtenção de nota parcial da disciplina Inteligência Artificial, ministrada pelo professor Frederico S. Oliveira

**CUIABÁ
2023**

SUMÁRIO

| | |
|---|-----------|
| 1. INTRODUÇÃO | 3 |
| 2. DESCRIÇÃO GERAL | 4 |
| 3. DESENVOLVIMENTO | 10 |
| 3.1. BREVE EXPLICAÇÃO | 10 |
| 3.2. PROPOSTA | 10 |
| 3.3. SOLUÇÕES | 10 |
| 3.3.1. INTERFACE GRÁFICA. | 11 |
| 3.3.2. CONSTRUÇÃO DO CENÁRIO. | 11 |
| 3.3.3. LOCOMOÇÃO E SENSORES DOS INDIVÍDUOS. | 13 |
| 3.3.4. CONSTRUÇÃO DA REDE NEURAL | 14 |
| 3.4. ESCOLHA DO MELHOR INDIVÍDUO | 16 |
| 3.5. DESTRUIR OS INDIVÍDUOS RUINS | 18 |
| 4. DESCRIÇÃO E RESULTADO DOS TESTES. | 19 |
| 5. CONCLUSÃO | 20 |
| 6. BIBLIOGRAFIA | 21 |
| 7. APÊNDICES | 21 |

1. INTRODUÇÃO

Esta documentação tem o objetivo de introduzir e explicar de maneira simples e didática todas as técnicas e ferramentas utilizadas para o desenvolvimento do Software “Neural Networking Tesla Simulation”, no qual consiste em um jogo desenvolvido na linguagem Python 3, mais especificamente utilizando a biblioteca “pygame”, tal jogo possibilita que o player construa estradas com obstáculos com o intuito de ensinar a IA progredir, para isso é possível efetuar uma série de mudanças em “Run Time”, assim alterando o processamento interno da rede neural, produzindo desta maneira melhores ou piores resultados.

A escolha das técnicas de aprendizagem de máquinas utilizadas teve grande influência devido ao fato de que considero fascinante que uma inteligência artificial tome decisões nas quais o desenvolvedor não previu ou até mesmo nem programou.

É válido ressaltar que esse documento é apenas uma breve síntese do vídeo explicativo com link anexado no apêndice. Assistir vídeo é fundamental pois os mesmos demonstram de maneira fluida e extremamente profunda o funcionamento do software e toda sua criação.

2. DESCRIÇÃO GERAL

Para iniciar nossa discussão sobre o tema, é fundamental compreender todo o contexto envolvendo o software, portanto, afirmo que “Neural Networking Tesla Simulation” nada mais é do que um mero jogo, no qual você como jogador tem com objetivo principal ensinar a IA a progredir por um dado caminho.

Figura 1 - Interface Gráfica Inicial do Jogo.



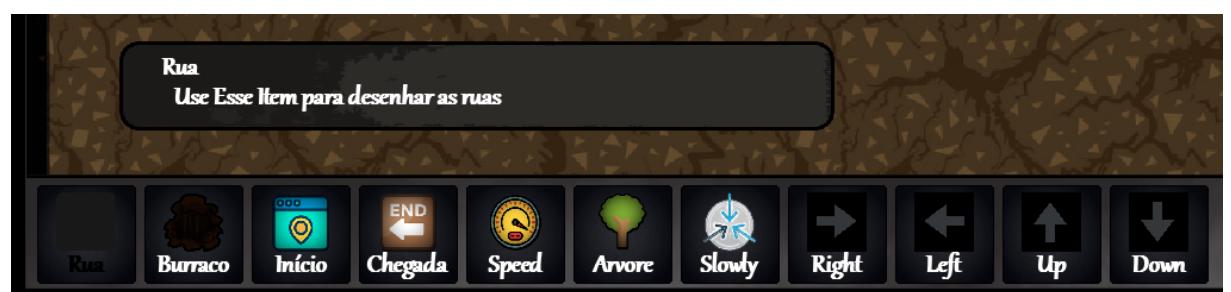
Porém para tal, é necessário construir um caminho pelo qual a rede neural irá aprender, sendo assim o player tem a possibilidade de construir e editar diversos caminhos distintos, com obstáculos ou itens de ajuda, então o limite é a imaginação como também é claro, o tempo que o player está disposto a deixar o jogo rodando com a rede processando e adaptado os pesos, criando assim um universo dinâmico dentro do jogo, possibilitando testar diversos distintos caminhos e comparar o desempenho do algoritmo.

Figura 2 - Construção da Estrada.



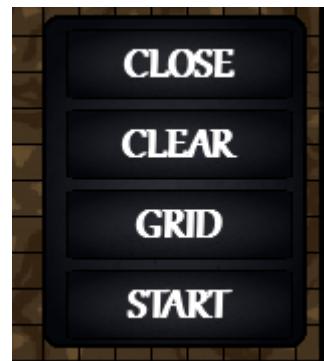
No menu de construção o player deverá construir as ruas, definir o ponto de partida e a chegada e caso queira poderá implementar obstáculos ou setas de ajuda.

Figura 3 - Itens de Construção do Cenário.



O menu interativo no canto superior direito disponibiliza uma gama de opções, dentre elas, a possibilidade de limpar as construções, fechar o menu, habilitar ou desabilitar uma guia de ajuda para construção e por fim iniciar o jogo.

Figura 4 - Menu Superior Direito.



Ao inicializar o jogo, diversos menus são habilitados, alguns meramente visuais, porém outros permitem que o jogador faça mudanças extremamente significativas no processamento da rede neural, alterando seu raciocínio.

Figura 5 - Run Time



Basicamente existem 6 alterações drásticas, Todas elas influenciam no desempenho de aprendizagem da rede neural, permitindo a melhora ou a piora brutal do algoritmo.

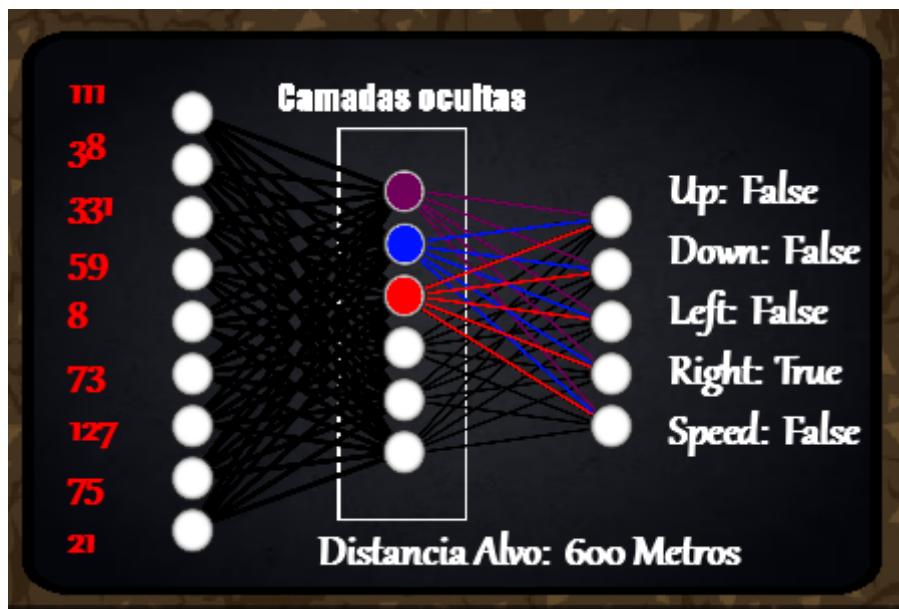
Figura 6 - Menu de Configuração.



Como uma simples explicação, posso afirmar que o “**Fator X**” é responsável por gerar novos valores aleatórios em relação ao melhor conjunto de pesos, a “**População**” é a quantidade de distintos individuais aleatórios por geração. “**Tempo Máximo**” é o limite de tempo que uma geração pode durar, para caminhos muito grandes é necessário aumentar esse tempo. “**Camadas Ocultas**” é a quantidade de camadas ocultas da rede Neural. “**Tipo de Viés**” define se a escolha da ia vai ser tomada exclusivamente pelos pesos ou haverá uma influência externa. Por fim, “**Mov Final**” define se o indivíduo deverá morrer ao falhar ou tentar novamente por 15 vezes, até desistir.

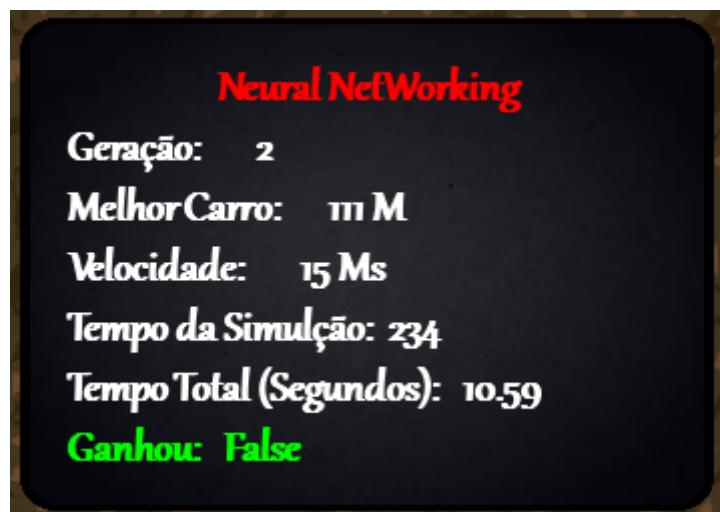
Há também um menu para visualizar o processamento da rede neural do melhor indivíduo na geração atual, a camada de entrada recebe 9 valores, os quais são os sensores, na camada oculta os valores são calculados, porém como você pode modificar a quantidade de camadas ocultas, essa parte é apenas ilustrativa, por fim temos a camada de saída, responsável por dizer qual deverá ser a ação do indivíduo.

Figura 7 - Menu das Camadas.



O menu no canto inferior esquerdo mostra informações relevantes, como a geração atual, a distância do melhor indivíduo, a velocidade, o tempo total da geração, o tempo total de jogo, e por fim, informa se a IA venceu ou não.

Figura 8 - Menu de Informações.



Para finalizar, existe o menu básico, que apenas habilita ou desabilita os outros menus, possibilita pausar o game, ou reiniciar a geração.

Figura 9 - Menu Principal



Quando a la Vencer, Seus pesos são clonados e o resultado é esse:

Figura 10 - la Vencendo.



Após essa breve explicação do jogo e como joga, no proximo topico discutirei como tudo isso foi implementado é válido afirmar que apenas mostrarei os resultados concretos, erros e problemas no algoritmo serão discutidos no vídeo.

3. DESENVOLVIMENTO

3.1. BREVE EXPLICAÇÃO

A proposta e a solução do estudo da inteligência artificial envolvida na concepção desse jogo é vasta, muito ampla e complexa, portanto devo ressaltar que os próximos tópicos apenas demonstraram de maneira simples e didática as técnicas utilizadas, não aprofundado em nenhum algoritmo, pois fugiria do escopo deste trabalho, e causaria um dificuldade na correção, pois facilmente essa documentação ultrapassaria 100 páginas.

3.2. PROPOSTA

A proposta do jogo é muito simples, como já explicado no tópico 2, basicamente a IA tem como objetivo sair do ponto de partida e chegar no ponto final, para isso o algoritmo utilizado foi a construção de uma Rede Neural. Porém diversos outros problemas lógicos e algorítmicos dificultaram drasticamente essa missão.

Destacarei os maiores desafios para no próximo tópico explicar suas correlatas soluções.

- **Interface Gráfica.**
- **Construção do Cenário.**
- **Locomoção e Sensores do Indivíduos**
- **Concepção da Rede Neural**
- **Escolha do Melhor Indivíduo.**
- **Destrução de Indivíduos Ruins.**

3.3. SOLUÇÕES

No desenvolvimento do jogo eu tive dezenas senão centenas de problemas, os mesmo quase fizeram com que eu desistisse da ideia, porém devido a minha natureza eu persisti, o resultado é satisfatório, porém caso eu tivesse mais tempo seria melhor, Já no iniciar do desenvolvimento me deparei com o primeiro problema.

3.3.1. INTERFACE GRÁFICA.

Um jogo necessita ser visualmente atrativo, portanto escolhi a biblioteca Pygame, a qual eu não sabia absolutamente nada, portanto isso por si só já foi um desafio inicial, para solucioná-lo eu criei um modelo padronizado que todos os códigos seguiria.

Figura 11- Exemplo

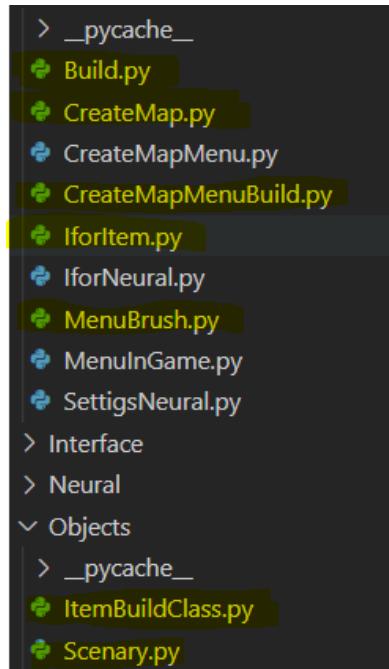
```
#Controlador da Interface
class StartMenuGame:
    def __init__(self) -> None: ...
    def RestartMenu(self): ...
    def setConfigureGame(self): ...
    def createInterfaceMenu(self): ...
    def animateBackGround(self): ...
    def controllerButs(self): ...
    def eventsButs(self): ...
    def modeGame(self): ...
    def update(self): ...
```

Todos os códigos de Interface gráfica, tem 4 estruturas básicas, todos chamam o update para atualizar o jogo a cada frame, todos têm o controlador de botões e o controlador de eventos, por fim todos tem um função com objetivo de iniciar a tela, desta maneira para desabilitar ou habilitar um código inteiro basta controlar a chamada da função update, como tudo é feito em python, manipular os códigos ficou muito fácil e dinâmico.

3.3.2. CONSTRUÇÃO DO CENÁRIO.

Para possibilitar que o jogador possa construir o próprio mapa, eu criei algumas classes, sendo elas.

Figura 12- Classes Responsáveis Pela Construção do Cenário.



A “**Build.py**” permite que você construa os itens escolhidos no menu no cenário. “**CreateMap.py**” cria o mapa de acordo com os pixels da tela. “**CreateMapMenuBuild.py**” é responsável pela interface gráfica da construção, ou seja gera o menu de construção e os itens selecionáveis. “**MenuBrush.py**” controla o menu do pincel, pois você pode aumentar ou diminuir o pincel afetando mais ou menos pixels. “**ItemBuildClass.py**” contém todas as características dos itens, como tamanho, ícone, tipo, etc. “**Scenary.py**” é um banco de dados que contém toda informação dos objetos construídos e do cenário.

Fazer a construção do cenário foi bem mais complexo do que parece, pois havia a necessidade de mapear e definir o que é colisão ou não para a IA, portanto cada item tem um ID, esse ID está associado a todas as centenas de pixel no qual ele está construído. assim toda vez que a ia se locomover em 1 pixel ela busca no banco de dados do cenário o respectivo ID do item associado naquele pixel, solucionando um grande problema.

3.3.3. LOCOMOÇÃO E SENSORES DOS INDIVÍDUOS.

A locomoção da IA está entre um dos problemas mais complexos, pois como já dito anteriormente havia a necessidade de saber sobre qual item ela está em dado momento, e como a tela é feita de pixel, ela deveria se locomover 1 pixel de cada vez, fazendo assim milhares de solicitação para saber o ítem do pixel em questão, agora imagine 1000 Indivíduos fazendo milhares de solicitações para uma lista, porém não é só isso, a IA necessita saber a distância da colisão mais próxima para todas as direções, então para cada direção há um loop gigantesco verificando o pixel e o ID, para no final enviar todos esse valor para rede neural processar. desta maneira o pc não aguentaria processar.

A solução foi mapear apenas os locais que têm rua construída, e assumir o (ID = 1), todos os outros pixels automaticamente receberam (ID = -1), só isso reduz em milhares as verificações, as demais construções substituir o (ID = -1) pelo seu respectivo ID. em seguida eu utilize a estrutura de dados Biblioteca, assim cada pixel com rua é salvo na biblioteca com sua “KEY” definida sendo o (X,Y) respectivo do pixel, e o “VALUE” recebe uma lista de dados do pixel, como ID do Item e distância. solucionado o problema da verificação.

Para solucionar o problema da quantidade de Indivíduos foi mais simples, eu simplesmente defini um valor limite de 15, assim quando tiverem mais de 15 Indivíduos vivos na mesma geração, o algoritmo espera alguém morrer para gerar um novo, limitado o processamento em apenas 15 IA por vez. Isso é apenas um breve resumo do funcionamento, eu levei considerável tempo tentando solucionar esse problema e testei diversas técnicas para chegar nessa solução aceitável.

Figura 13 - Código - Configuração Colisão do Cenário.

```
def createMapId(self):
    #Configurar Colisão no Cenario
    for x in range(ct.gameWidth):
        for y in range(ct.gameHeight):
            ct.mapId[(x,y)] = 1
```

3.3.4. CONSTRUÇÃO DA REDE NEURAL

Construir a rede neural foi outro grande desafio, pois eu simplesmente não sabia nada, e não tinha tempo para estudar de maneira aprofundada, então eu assisti alguns vídeos explicativos da teoria, e busquei por algumas bibliotecas que poderiam me ajudar, porém, cheguei a conclusão que eu deveria projetar a minha rede do zero, desta maneira eu fiz e refiz a rede mais de 15 vezes até compreender realmente seu funcionamento.

Para isso eu criei um arquivo que contém 5 classes, sei que poderia fazer a rede neural de maneira mais dinâmica, porém como é a minha primeira, resolvi fazer mais limitada para realmente compreendê-la.

Figura 14 - Código - Rede Neural

```
#Gerar Valor Aleatorio.
def getRandoValue(valueLimit = 2000): ...

#Controlador de Pesos das camadas Ocultas
class Weights():

#Controlador de Peso da Camada de Saída
class WeightsLeave():

#Controlador Cerebro dos Carros
class NeuronController():

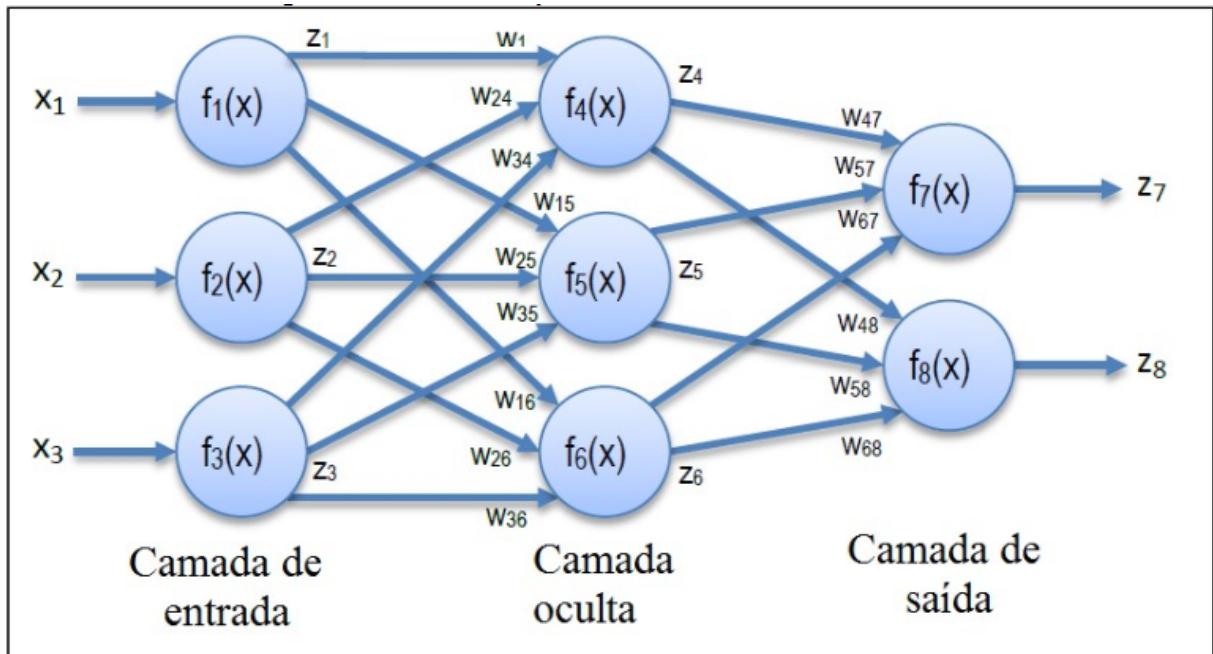
#Neuronio da Camada de Saída
class NeuronLeave():

#Neuronio Camada Oculta
class Neuron():...
```

Cada indivíduo da geração tem acesso a sua própria consciência, cada consciência tem camadas, definidas pelo Jogador, para cada camada, há (N) neurônios, para cada neurônio existe um conjunto de pesos associados às suas entradas. Na primeira geração todos recebem pesos aleatórios, os pesos são multiplicados pelos valores das entradas, que no caso são os sensores, em seguida os resultados das

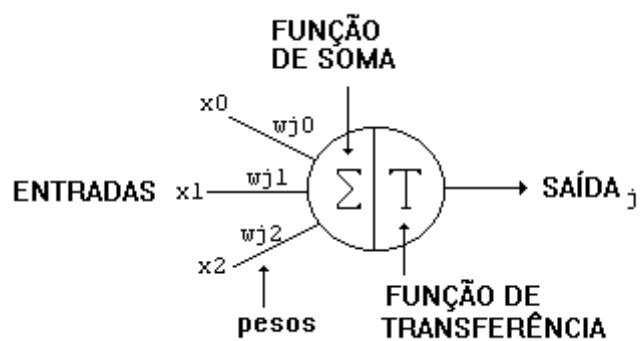
multiplicações são somados e associados a um viés, sendo esse um valor para não permitir que os resultados de saída sejam unicamente reflexo dos valores de entrada, depois o neurônio valida o valor, caso seja positivo ele propaga para próxima camada, a partir da segunda geração os pesos são herdados e pequenamente modificados do indivíduo mais eficiente.

Figura 15 - Rede Neural



Esse conjunto de regras seguidas de maneira lógica e algorítmica, faz com que os pesos a cada nova geração sofra pequenas mudanças e gradativamente vão se aproximando de um limiar chamado de máximo global.

Figura 16 - Neurônio Teoria - Rede Neural



Abaixo está o código implementado na teoria da rede neural.

Figura 16 - Código - Rede Neural

```
class NeuronLeave():
    def __init__(self,neuronValue1,neuronValue2,neuronValue3,neuronValue4,neuronValue5,neuronValue6,weights,theFinal = False) -> None:
        #Iniciar Neurônio
        self.valueFinal = 0
        self.listWithValueFromNeurom = []
        self.listWithValueFromNeurom.append(neuronValue1.valueFinal)
        self.listWithValueFromNeurom.append(neuronValue2.valueFinal)
        self.listWithValueFromNeurom.append(neuronValue3.valueFinal)
        self.listWithValueFromNeurom.append(neuronValue4.valueFinal)
        self.listWithValueFromNeurom.append(neuronValue5.valueFinal)
        self.listWithValueFromNeurom.append(neuronValue6.valueFinal)
        self.theFinal = theFinal
        self.allWeights = weights
        self.multiplyWeights()
        self.sumValues()
        pass

    def multiplyWeights(self):
        #Multiplicar Valores
        index = 0
        for ss in self.listWithValueFromNeurom:
            self.listWithValueFromNeurom[index] = ss * self.allWeights.weights[index] * self.allWeights.bias[index]
            index += 1

    def sumValues(self):
        #Somar Valores
        for ss in self.listWithValueFromNeurom:
            self.valueFinal += ss

        self.relu()

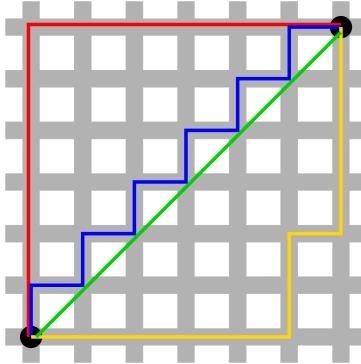
    def relu(self):
        #Função de Ativação

        if(self.valueFinal <= 0):
            self.valueFinal = 0
        else:
            if(self.theFinal):
                self.valueFinal = 2
```

3.4. ESCOLHA DO MELHOR INDIVÍDUO

Como os pesos do melhor indivíduo serão clonados e pequenamente randomificados para a nova geração, é fundamental escolher corretamente a melhor consciência, porém fazer isso foi muito complexo, pois o mapa é construído pelo jogador e o ponto de partida e chegada pode ser construído em qualquer lugar, ou seja eu precisava de um algoritmo que conseguisse calcular a distância entre os dois pontos levando em consideração a estrada construída. Eu desenvolvi diversos algoritmos que solucionam os problemas, porém todos demoravam minutos para calcular, devido à comparação de pixels.

Figura 17 - Distância Euclidiana.



Fiquei dias nesse problema, até que resolvi buscar vídeos na internet sobre o tema, no canal **“Universo Programado”** ele fez algo semelhante porém mais eficiente, no vídeo ele afirma que usou o Algoritmo “WaveFront”, então busquei pela teoria do algoritmo para conseguir implementar em Python, o resultado foi esse:

Figura 18 - WaveFront em Python

```
def CalculeDistaceFromEnd(self):
    #Algoritmo para Calcular distancia ate o Objetivo.
    origemNodeList = [(ct.centerFinalPos,1)]

    def insertNewNodeInList(pos):
        if(pos in ct.mapIdDistaceFromEnd and ct.mapIdDistaceFromEnd[pos] == 0):
            ct.mapIdDistaceFromEnd[pos] = distance
            newNodeList.append((pos,distance))

    if(ct.centerFinalPos in ct.mapIdDistaceFromEnd):
        ct.mapIdDistaceFromEnd[ct.centerFinalPos] = 0

    while origemNodeList:

        newList = []

        for value in origemNodeList:
            posCorrect = value[0]
            distance = value[1] + 1
            ct.mapIdDistaceFromEnd[posCorrect] = value[1]

            #Direita
            insertNewNodeInList((posCorrect[0] + 1,posCorrect[1]))

            #Esquerda
            insertNewNodeInList((posCorrect[0] - 1,posCorrect[1]))

            #Baixo
            insertNewNodeInList((posCorrect[0],posCorrect[1]+1))

            #Cima
            insertNewNodeInList((posCorrect[0],posCorrect[1]-1))

        origemNodeList.clear()

        for ss in newList:
            origemNodeList.append(ss)
```

Finalmente funcionou, mas os meus algoritmos anteriores apresentavam uma solução melhor do que o “WaveFront”, pois sempre o melhor indivíduo era o que ficava no meio da pista, e não nas bordas, assim sempre a rede conseguiu chegar na solução, pois sempre evitava qualquer tipo de colisão, mas demorava muito tempo para calcular, desta maneira optei por utilizar o “WaveFront”, mesmo que ele aumente consideravelmente o tempo para o sucesso da IA.

3.5. DESTRUIR OS INDIVÍDUOS RUINS

Como o progresso da aprendizagem da rede neural depende de novas gerações, é fundamental gerar novos indivíduos melhorados o tempo todo, mas essa tarefa não é fácil, novamente o fato de o jogo ser dinâmico e o mapa pode ser modificado, causou grande problema, pois quando a IA não bate e fica parada, ela não morre, assim não gera novos indivíduos, e entra em um loop infinito.

A solução foi criar uma lista de posições que o indivíduo já passou, caso ele passe 5 vezes pela mesma posição, significa que ele está parado ou voltado, portanto deverá morrer. Uma segunda opção foi desenvolvida, nesse caso ao invés de morrer o indivíduo gera novos pesos aleatórios, e tenta novamente por 15 vezes.

Figura 19 - Sistema de Morte da IA.

```
if(ct.modeCar == 1):
    if(self.countLimitStop > 5 and self.pos in self.listPosWalked):
        self.deadCar()
        return
    else:
        self.countLimit += 1
        if(self.countLimit > 5 and self.pos in self.listPosWalked):
            self.neuronController.update()
            self.countLimit = 0

            if(self.chanceWeight > 15):
                self.deadCar()

            self.chanceWeight +=1
```

4. DESCRIÇÃO E RESULTADO DOS TESTES.

Podemos concluir analisando os resultados dos testes, e assim eu afirmo que a IA consegue solucionar todos os problemas lineares de maneira rápida, ou seja uma pista sem curvas acima de 180 graus são solucionáveis, porém estradas complexas demandam um tempo considerável em horas, com algumas necessitando de ajuda das setas.

Figura 20 - IA em Pista Linear - Tentando



Figura 21 - IA em Pista Linear - Venceu



Ao utilizar as setas a IA ignora a Rede Neural, e começa a Utilizar um Algoritmo de busca semelhante ao A*. sendo assim ao colocar setas na pista toda, a IA sempre ganha.

Figura 22 - IA com Setas.



Portanto há duas formas de jogar, Utilizando as setas ou a Rede Neural, São duas técnicas diferentes.

5. CONCLUSÃO

Posso concluir afirmando que foi um trabalho desafiador, devido ao fato de eu ter complicando, pois o jogo é inteiro é dinâmico, criar a rede neural foi um grande desafio e tenho certeza que um belo aprendizado, passei muitos dias e muitas horas dedicando nesse trabalho.

6. BIBLIOGRAFIA

<https://www.pygame.org/news>

https://en.wikipedia.org/wiki/Wavefront_expansion_algorithm

<https://sites.icmc.usp.br/andre/research/neural/>

https://pt.wikipedia.org/wiki/Rede_neural_artificial

➡ #01a - Deep Learning - Introdução

➡ #01b Deep Learning - GPU Grátis do Google

➡ DEEP LEARNING #02 CUSTO EM DEEP LEARNING

➡ But what is a neural network? | Chapter 1, Deep learning

➡ Path Planning #2 Wave Propagation, Potential Fields & Modern(ish) C++

➡ A Matemática das Redes Neurais

➡ Introdução a Redes Neurais e Deep Learning

➡ Inteligência Artificial aprendendo a DIRIGIR!! (Deep Cars)

7. APÊNDICES

Vídeos Mostrando o Software e GitHub :

<https://github.com/MushroomAngelsGames/Ai-Neural-NetWorking>

Jogo: <https://youtu.be/SNYwd35Qtdk>

Códigos do Jogo: <https://youtu.be/sSyAjwUbJi0>