UNIVERSIDADE FEDERAL DO PARANÁ

Curso de Graduação em Ciência da Computação

Arthur Henrique Canello Vilar - GRR20197153

Vinicius Tikara Venturi Date - GRR20190367

RELATÓRIO PROJETO 4 - INTELIGÊNCIA ARTIFICIAL

CURITIBA

2023

SUMÁRIO

Introdução

Para o quarto trabalho da disciplina de Inteligência Artificial escolhemos o NEAT(NeuroEvolution of Augmenting Topologies), um algoritmo genético que visa evoluir redes neurais artificiais. Nós nos baseamos no artigo oficial do NEAT [1] e na documentação do NEAT-Python [4] para fazer nosso trabalho.

Para nosso estudo iremos fazer com que uma rede aprenda a jogar o “jogo do Dinossauro” presente no Google Chrome.

Primeiro será feita uma introdução resumida sobre o algoritmo NEAT, e posteriormente mostraremos detalhes da implementação do código e os resultados obtidos.

NEAT

O que é o NEAT:

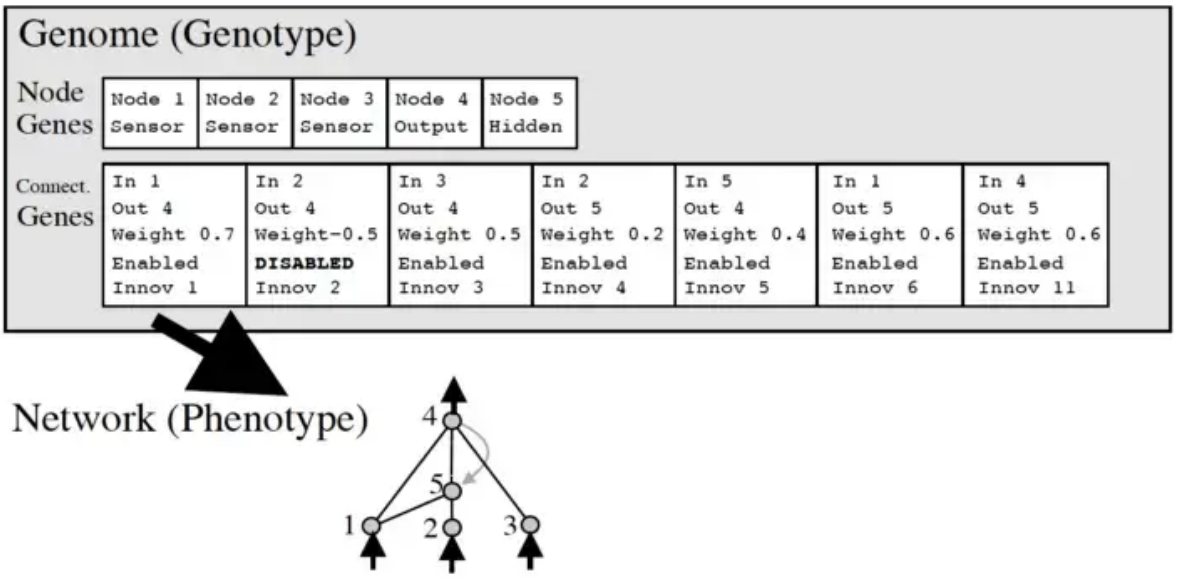
O NEAT consiste em utilizar a Evolução Neural (Neuroevolution), a evolução artificial utilizando algoritmos genéticos, com o seguinte método incluso: evoluir também a topologia da rede em conjunto.

Para nosso uso o NEAT funciona, apesar de sua convergência rápida. O nosso sistema possui um objetivo bem definido: sobreviver o máximo possível, ou seja, evitar qualquer obstáculo que apareça na tela.

Como o objetivo desse algoritmo é evoluir as redes neurais, o ideal é começar com a rede neural mais simples possível para o problema, uma rede que possui apenas nodos de input e nodos de output, e a partir dela evoluir para redes que geram resultados melhores através de mutações e crossover. Os nodos de input e output é uma decisão de projeto que varia de acordo com o problema a ser resolvido.

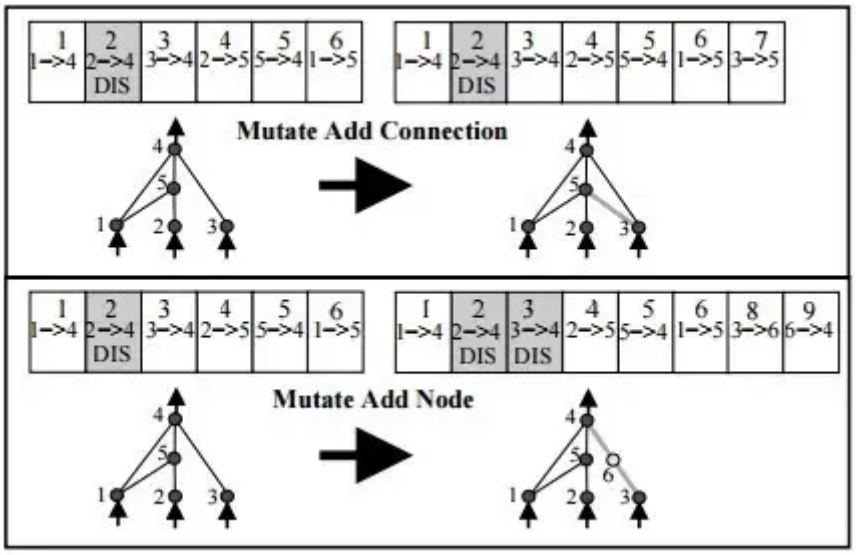
Codificação genética do NEAT:

Na biologia temos o genótipo e o fenótipo, o genótipo (genoma) é a representação genética de um ser, e o fenótipo é a representação física desse ser. No NEAT o genótipo consiste em duas listas de genes, uma lista com uma série de nodos (perceptrons) e uma lista com uma série de conexões. O fenótipo, por outro lado, é a topologia de rede neural em si.



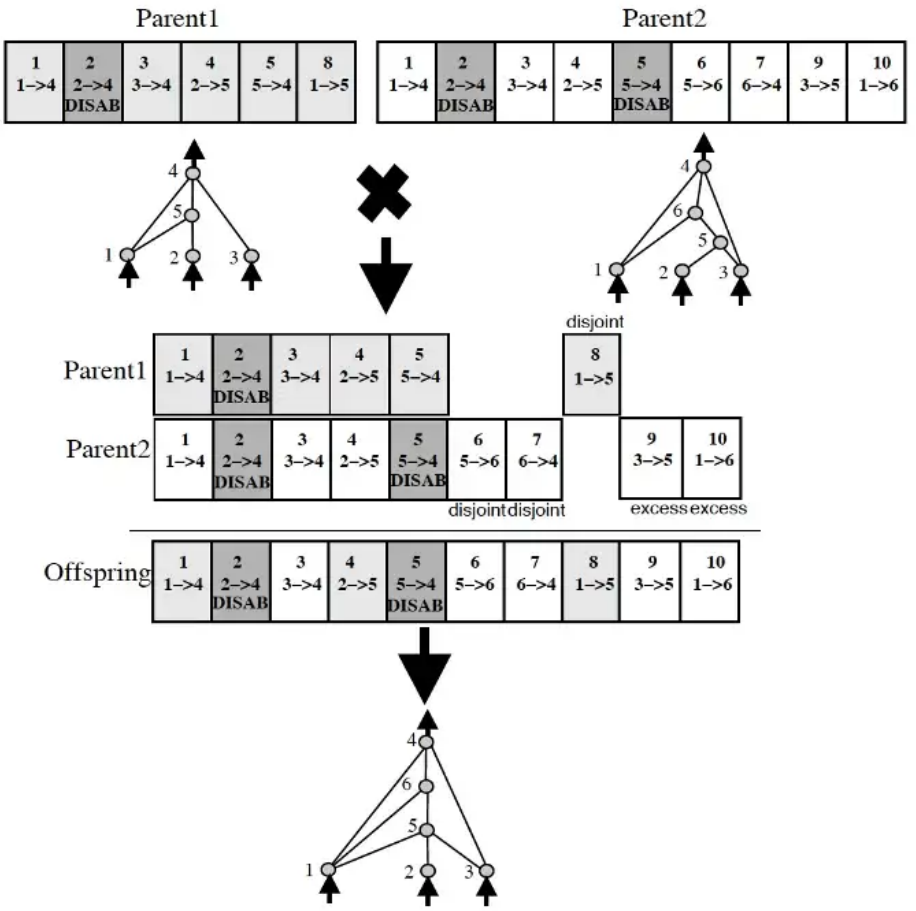
Mutação:

As mutações podem modificar conexões existentes ou adicionar novas estruturas na rede, sendo essas estruturas novas conexões ou novos nodos.



Crossover:

O crossover consiste em recombinar duas redes com topologias diferentes para gerar uma terceira. Nessa etapa o NEAT usa os identificadores de inovação, que são os IDs das mutações, de modo que eles fiquem alinhados. Desse modo evitando que algum gene seja perdido no crossover.



Especiação:

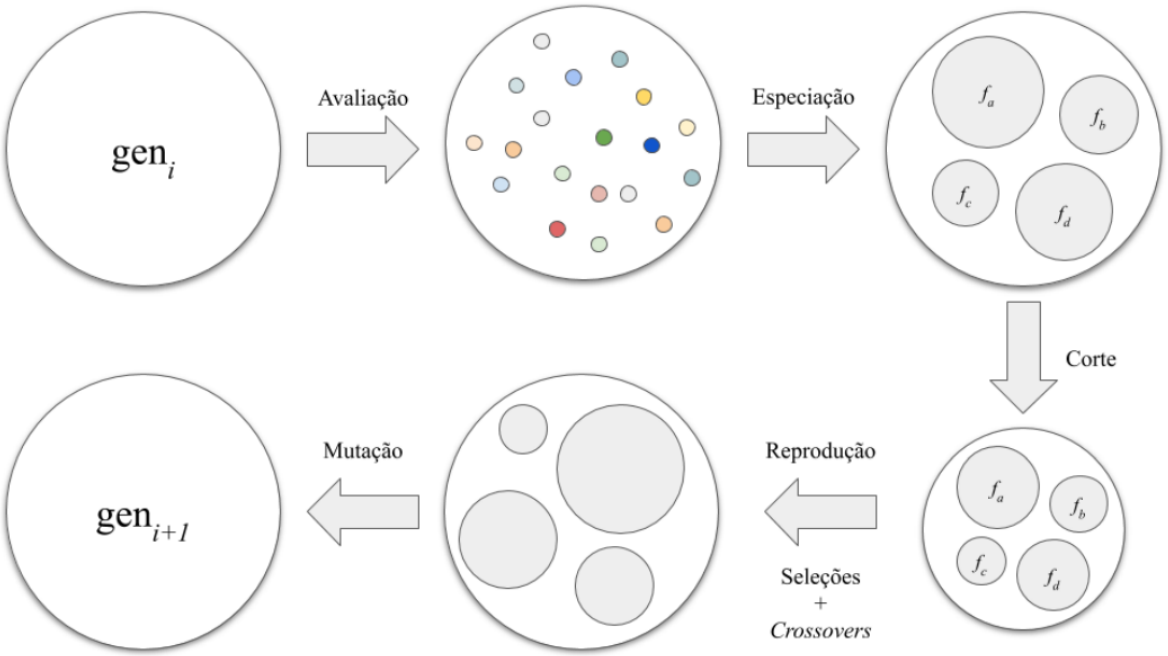
Como o objetivo do NEAT é conseguir uma rede de menor tamanho possível, uma rede que sofreu uma mutação estrutural e recebe um novo nodo está em desvantagem por causa do seu tamanho, por mais que essa mutação seja essencial para chegar na rede ideal.

Ao surgir uma nova população com novas topologias, a população é dividida em espécies para que cada estrutura possa competir com suas semelhantes, de tal forma que evita a eliminação prematura de estruturas (genomas) que estejam em desvantagem. Essa é a ideia de especiação sugerida pelo NEAT, e isso permite com que as novas estruturas tenham um tempo para evoluir e serem testadas antes de serem eliminadas.

O número de genes disjuntos e excessivos entre dois genomas é um meio natural de medir a distância de compatibilidade entre eles. Quanto mais diferente dois genomas são, menos evoluções eles compartilham e menos semelhantes eles são. O cálculo dessa distância é usado para separar os genomas nas diferentes espécies. Uma nova espécie é criada caso o genoma não se encaixe em nenhuma espécie existente.

Na imagem abaixo podemos ver uma demonstração de uma geração do neat. Onde a população da geração i é avaliada de acordo com sua fitness e separada em espécies. Após a especiação os piores indivíduos de cada espécie são retirados e é feita a

reprodução entre os melhores indivíduos, acontece uma mutação em cima desses novos indivíduos e eles se tornam a nova população da geração i+1.



Jogo

O jogo do dinossauro tem como objetivo ficar o maior tempo possível vivo, e para isso é necessário se esquivar de obstáculos. Os obstáculos do jogo são cactos e pterodáctilos. Os cactos são objetos terrestres que precisam ser pulados sobre, e os pterodáctilos são obstáculos aéreos que podem surgir tanto na altura do chão quanto no ar, sendo possível pular ou passar por debaixo dele.

Na nossa implementação os cactos e pterodáctilos surgem em ordem aleatória um por vez na tela. Os cactos possuem dois tamanhos, pequeno e grande, e os pterodáctilos surgem em duas alturas, no céu ou no chão.

Outro fator que aumenta a dificuldade é que a velocidade do jogador aumenta conforme mais tempo se passa no jogo, diminuindo o tempo de resposta para a esquiva dos obstáculos.

Implementação

O programa foi feito em Python, nós utilizamos uma implementação[2] opensource básica feita em pygame do jogo no github. Nessa implementação o jogo contava apenas com cactos e pterodáctilos fixos em uma altura.

Alteramos esta implementação adicionando uma nova altura para o pterodáctilo de forma que haja mais variedade de obstáculos. Os obstáculos agora são: cacto grande, cacto pequeno, pterodáctilo alto e pterodáctilo baixo. Outra mudança foi o limite da velocidade

máxima alcançável pelo jogo uma vez que a partir de determinada velocidade fica impossível pular um obstáculo sem bater no próximo.

A parte do NEAT foi implementada usando a biblioteca **neat-python** do python. Para configurar os parâmetros do neat foi utilizado um arquivo config.txt, onde é possível fazer alterações nos valores que influenciam o treinamento da rede. Os parâmetros neste arquivo já vem com valores padrão predefinidos. Para o nosso trabalho foi necessário a alteração de algum desses valores de acordo com a modelagem do problema.

Os parâmetros que foram alterados são: o tamanho da população inicial de cada geração (pop_size), o critério de fitness (fitness_criterion), as funções de ativação (activation_default e activation_options), e por fim os parâmetros da rede inicial (num_hidden, num_inputs e num_outputs). Os valores escolhidos foram:

- pop_size = 100
- fitness_criterion = max
    - Significa que quanto maior a fitness melhor.
- activation_default = tanh
- activation_options = tanh
- num_hidden = 0
- num_inputs = 6
- num_outputs = 2

Sendo os inputs usados na rede: coordenada y do dinossauro, coordenada y do obstáculo, largura do obstáculo, altura do obstáculo, distância entre o dinossauro e o próximo obstáculo e a velocidade do jogo.

Fluxo de uma execução

O algoritmo precisa de preparo antes da execução. Os parâmetros citados acima são carregados e é então criada a população de acordo.
Tendo a população criada, ela é então treinada e avaliada, utilizando o jogo como este meio. No laço principal do jogo, cada dinossauro da população aumenta seu fitness a cada passo do jogo em que ele sobrevive.
Seus neurônios são ativados de acordo com os elementos presentes no jogo:

- A posição Y do dinossauro
- A posição Y do obstáculo
- A largura do obstáculo
- A altura do obstáculo
- A distância entre o dinossauro e o obstáculo
- A velocidade atual do jogo

Baseado nessa ativação são recebidos dois outputs, sendo que um deles determina se o dinossauro irá agachar ou continuar correndo e o outro determina se ele deve pular ou não.
Quando todos os dinossauros de determinada execução perdem, a próxima geração se inicia, utilizando as técnicas explicadas anteriormente para gerar os indivíduos dessa população.
Portanto é possível treinar quantas gerações forem necessárias seguindo esse procedimento.

Resultados

Nós consideramos uma solução ótima aquela que o dinossauro consegue sobreviver para sempre. Do modo que implementamos o jogo, a velocidade do jogo começa em 20 e aumenta em 1 a cada 100 pontos, chegando a uma velocidade máxima de 40 ao atingir 2000 pontos. Se o dinossauro consegue sobreviver na velocidade 40 até os 4000 pontos podemos considerar que esse dinossauro nunca vai morrer, sendo ele a solução ótima.

Na grande maioria das vezes o código convergiu para um resultado ótimo com uma média de 10 gerações e apenas uma espécie, onde em cada geração a fitness aumentava um pouco em relação a geração anterior. Apesar de em alguns experimentos o algoritmo convergir para uma solução ótima em menos de 5 gerações.

Duas vezes o algoritmo dividiu a população em duas espécies e demorou em média 20 gerações para convergir para uma solução ótima.

Para melhor visualização dos resultados foram gerados os seguintes gráficos:

Gráfico 1:

Este gráfico apresenta um dos resultados dos testes que a divisão foi feita em apenas uma espécie. Nesses testes o dinossauro escolheu andar sempre abaixado como solução ótima, pulando apenas pelos obstáculos terrestres. Para este exemplo foram necessárias 9 gerações para o algoritmo chegar em uma rede neural que conseguisse vencer o jogo.

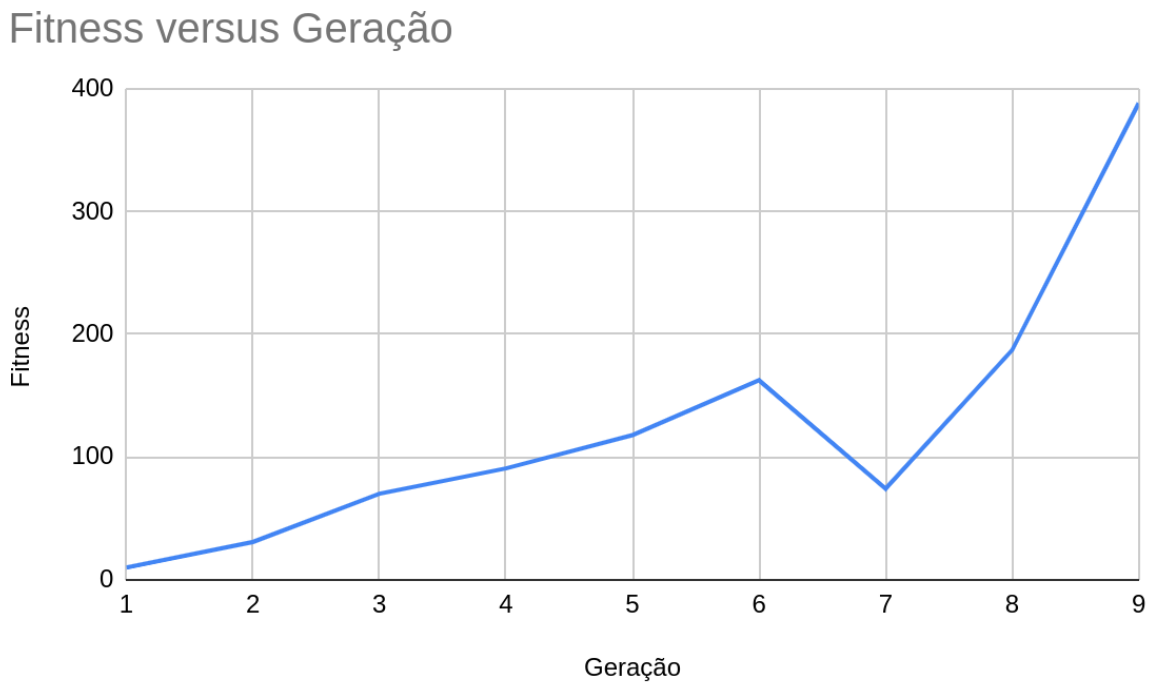
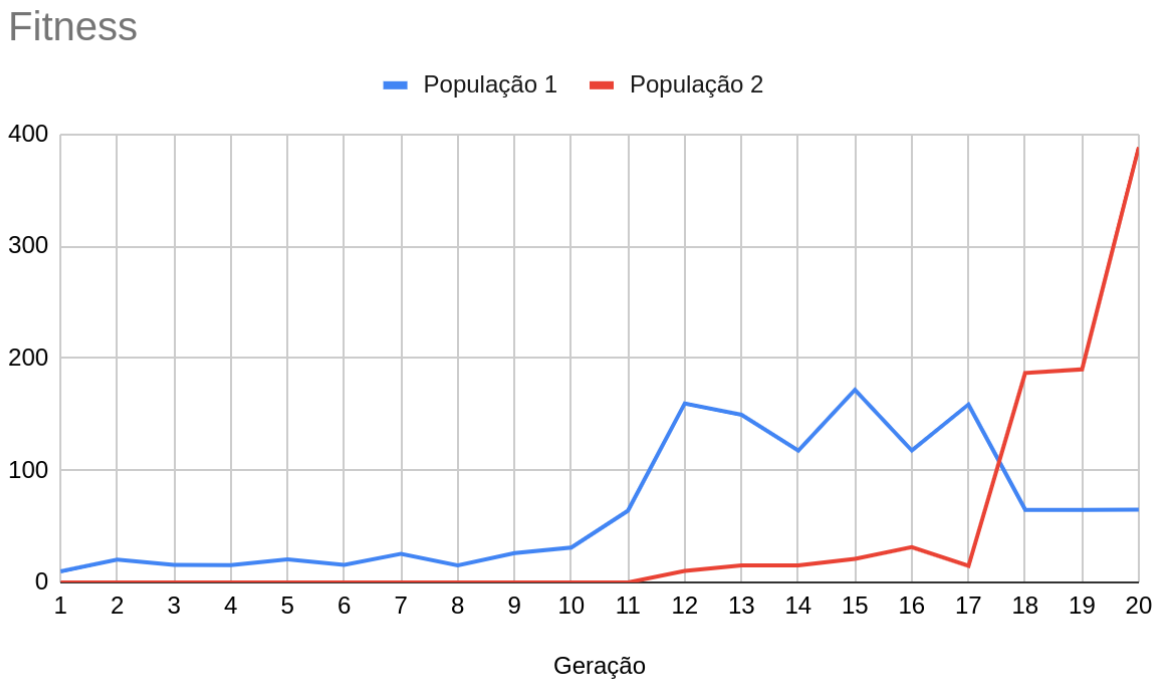
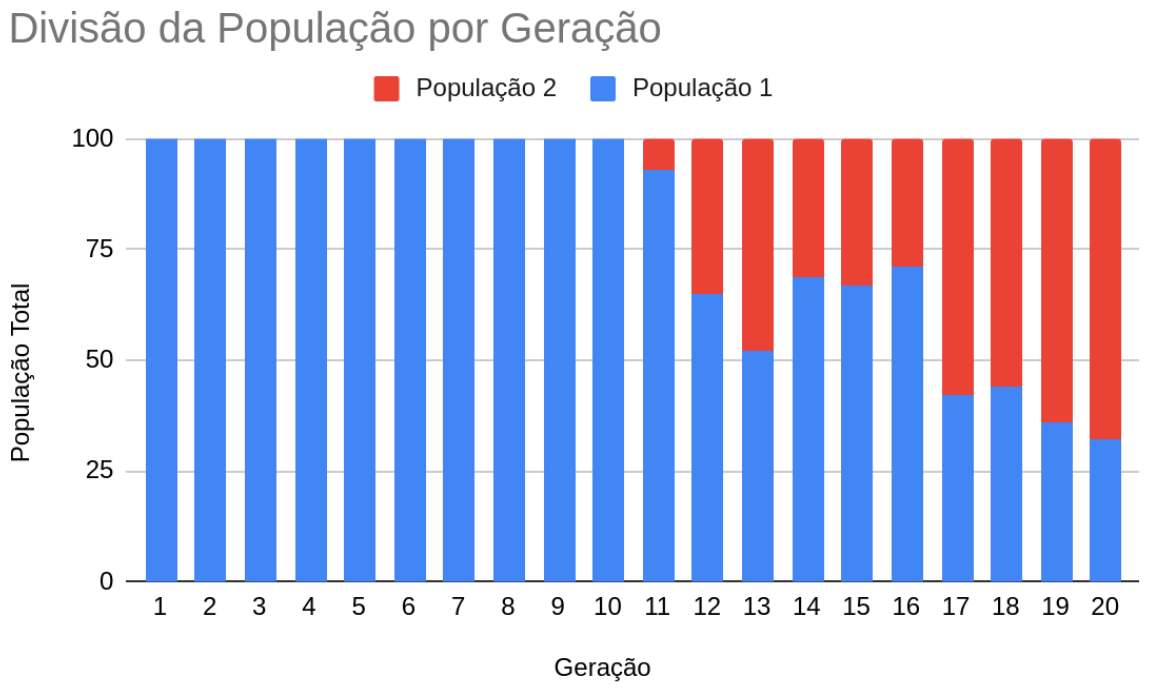


Gráfico 2:

Este gráfico apresenta um dos resultados dos testes que a divisão foi feita em duas espécies, sendo que a segunda espécie começou junto com a 11ª geração da primeira espécie. Nesta solução o dinossauro vencedor foi da espécie 2, ele escolheu andar a maior parte do tempo abaixado como solução ótima, levantando por um curto período de tempo antes dos obstáculos, e pulando pelos obstáculos terrestres. Para este exemplo foram necessárias 20 gerações para o algoritmo chegar em uma rede neural que conseguisse vencer o jogo.





Conclusão

O NEAT é popularmente usado para treinar agentes de jogos objetivos, mas principalmente jogos de corrida. Várias implementações podem ser encontradas com facilidade no YouTube e no Github. É um algoritmo que performa melhor em problemas simples, pois em problemas mais complexos ele demora muito para convergir, perdendo para métodos de gradiente.

O início de uma população de Redes Neurais densas pode ser muito custoso pois o algoritmo precisaria de muito tempo para otimizar essas redes. O NEAT inicia a execução com redes de apenas nodos iniciais e nodos finais. Dessa maneira ele sempre percorre o espaço de busca explorando primeiro as soluções menores e mais simples, tendo uma vantagem de desempenho em comparação com outros algoritmos que visam resolver o mesmo problema.

A relação da matéria com o trabalho se fez presente tanto na hora de escolher os parâmetros usados no arquivo de configuração do NEAT quanto na hora de escolher como deveria ser o cálculo da fitness do agente (dinossauro).

Durante a primeira parte da matéria nós aprendemos diversos métodos de avaliar a aptidão de um agente de uma IA, esses métodos foram muito úteis para nos ajudar a decidir como calcular a fitness do nosso agente.

Já na segunda parte da matéria nós aprendemos sobre perceptron e multilayer perceptron, e como identificar se um problema pode ser resolvido por um perceptron ou se precisa de uma rede com hidden layers. No nosso trabalho usamos esse conhecimento para decidir quantos nodos de input e quantos nodos de output a nossa rede precisaria inicialmente para funcionar corretamente. Chegando a conclusão de que os 6 nodos de input seriam o suficiente para suprir todas as informações que a rede precisava para tomar uma decisão.

Referências

[1] Kenneth O. Stanley & Risto Miikkulainen (2002). "Evolving Neural Networks Through Augmenting Topologies" (PDF). *Evolutionary Computation*. **10** (2): 99–127. CiteSeerX 10.1.1.638.3910. doi:10.1162/106365602320169811

[2] https://github.com/codewmax/chrome-dinosaur

[3] https://youtu.be/2f6TmKm7yx0

[4] https://neat-python.readthedocs.io/en/latest/