CENTRO UNIVERSITÁRIO NORTE DO ESPÍRITO SANTO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

AILTON JOSE BRANDAO JUNIOR

Dinossauro chrome

São Mateus – ES

2022

1 - Panorama da área.

1.1 - Contextualização

Quando um usuário tenta navegar para uma página web no Google Chrome enquanto está offline, o navegador notifica o usuário de que ele não está conectado à Internet, com uma ilustração de um Tyrannosaurus rex pixelizado mostrado na página. O jogo pode ser iniciado pressionando espaço ou na área de trabalho, ou tocando no dinossauro em dispositivos móveis Android ou IOS. Além disso, o jogo pode ser acessado digitando ou na barra de pesquisa “chrome://dino”.

Durante o jogo, o Lonely T-Rex se move continuamente da direita para esquerda em uma paisagem desértica em preto e branco, com o jogador tentando evitar obstáculos como cactos e pterodáctilos pulando ou abaixando. Pressionar espaço ou tocar no dinossauro em dispositivos móveis fará com que o dinossauro "pule", enquanto pressionar a tecla PgDn fará com que o dinossauro "se abaixe". À medida que o jogo avança, a velocidade do jogo aumenta gradualmente até que o usuário atinja um obstáculo, solicitando um fim instantâneo do jogo.

Para simular esse jogo, foi utilizado um código disponível no github. Tal software faz uma simulação do jogo usando Python com a biblioteca pygame e o paradigma orientado a objetos.

Python é uma linguagem de programação de alto nível, interpretada de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991. Atualmente, possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem, como um todo, não é formalmente especificada.

Pygame é uma biblioteca de jogos multiplataforma (independente de sistema operacional) feita para ser utilizada em conjunto com a linguagem de programação Python. O seu nome tem origem da junção de *Py*, proveniente de Python e *Game*, que significa *Jogo*, ou seja, Jogos em Python.

Programação orientada a objetos (POO, ou OOP segundo as suas siglas em inglês) é um paradigma de programação baseado no conceito de "objetos", que podem conter dados na forma de campos, também conhecidos como *atributos*, e códigos, na forma de procedimentos, também conhecidos como métodos.

1.2 - Exemplo de problemas, técnicas e aplicações

Os algoritmos genéticos têm aplicações em diversas áreas. por exemplo:
Telecomunicações: Segundo Blanchard (1994), no WCCI'94 – World Congress on Computational Intelligence – ocorrido em Orlando, na Flórida, mostrou uma série de soluções promissoras a situações reais utilizando Algoritmos Genéticos. Blanchard mostrou o caso da US West, uma companhia regional de telecomunicações do estado do Colorado, que vem usando um sistema baseado em AGs que possibilita projetar, em duas horas, redes óticas especializadas, trabalho que levaria seis meses utilizando especialistas humanos. O sistema produz resultados ainda 10% (dez por cento) melhores que os realizados pelo homem. A companhia estimava, naquele momento, que o sistema possibilitará uma economia de 100 milhões de dólares até o final do século.
Música: Foi apresentado em 1999 na CEC99 – IEEE – International Conference on Evolutionary Computation um ambiente interativo, utilizando Algoritmos Genéticos, para a avaliação de músicas (sequências de acordes) tocadas em arquivos MIDI. No caso, os indivíduos da população foram definidos em grupos de quatro vozes (soprano, contralto, tenor e baixo) ou coros. Cada um é avaliado segundo três critérios: melodia, harmonia e oitavas. A composição destes três critérios definia a aptidão (fitness) definida pela função de seleção, que retorna o melhor indivíduo ou melhor coro. Um ciclo genético é operacionalizado, criando novos indivíduos dos anteriores e procurando sempre pelo melhor. Quando um novo grupo é selecionado, ele é tocado em MIDI. A duração do ciclo genético determina o ritmo da evolução. O sistema criado foi denominado Vox Populi. (FUKUSHIMA, 1999).

As redes neurais artificiais podem ser aplicadas para resolver uma grande quantidade de problemas. Um bom exemplo de aplicação são softwares de reconhecimento de voz, que precisam aprender a conhecer a voz de determinadas pessoas. Redes neurais também são usadas em robôs que desarmam bombas. Se você já usou um scanner para retirar um texto de um jornal, por exemplo, saiba que o software de OCR, que é responsável por isso, precisa aprender a reconhecer caracteres da imagem. Logo, ele certamente possui algoritmos de rede neural. Existem até alguns softwares que aprendem a identificar SPAMs em e-mails e apagá-los (e conseguem uma margem aceitável de acertos). Mas no geral as redes neurais são usadas principalmente em aplicações mais complexas, como em usinas ou mercado financeiro.

2 - Técnica de inteligência artificial escolhida

2.1 - Descrição das técnicas

Para resolução desse problema, utilizei duas técnicas de inteligência artificial. A primeira, uma rede neural artificial. A segunda, algoritmo genético.
A rede neural foi usada para aproximar uma função que define o problema de maneira eficiente. Dados os parâmetros, essa função deve retornar um valor que será interpretado pelo jogo como uma ação do agente.
A função do algoritmo genético neste contexto foi treinar a rede neural. Com a utilização de vários agentes e um determinado grau de aleatoriedade, pode-se ajustar os pesos da rede levando em consideração o desempenho de cada geração de agentes.

2.1.1 - Rede neural artificial

Em ciência da computação e campos relacionados, redes neurais artificiais são modelos computacionais inspirados pelo sistema nervoso central de um animal (em particular o cérebro) que são capazes de realizar o aprendizado de máquina bem como o reconhecimento de padrões. Redes neurais artificiais geralmente são apresentadas como sistemas de “neurônios interconectados, que podem computar valores de entradas”, simulando o comportamento das redes neurais biológicas. As redes neurais são compostas por unidades ou nós conectadas por ligações direcionais. Cada tem a função de propagar a ativação e contém um peso.
Quando a informação chega na unidade é aplicada uma soma e uma função sob a soma chamada de função de ativação.
As funções de ativação determinam a saída de cada nó em função das entradas. Logo após, essas informações são propagadas até o fim da rede.
Normalmente, as redes são organizadas em camadas de tal forma que uma camada recebe as saídas da camada anterior. Por padrão, todas as redes têm a primeira camada, também conhecida como camada de entrada, e a última camada, também conhecida como camada de saída. A rede pode contar com mais camadas entre essas fundamentais, tais camadas são conhecidas como camadas ocultas.

2.1.2 Algoritmo genético

Um algoritmo genético (ou AG) é uma variante de busca em feixe estocástica na qual os estados sucessores são gerados pela combinação de dois estados país, em vez de serem gerados pela modificação de um único estado. A analogia em relação à seleção natural é a mesma que se dá na busca em feixe estocástica, exceto pelo

fato de agora estarmos lidando com a reprodução sexuada, e não com a reprodução assexuada.
Como a busca em feixe estocástico, os algoritmos genéticos combinam uma propensão para subir a encosta com a exploração aleatória e com a troca de informações entre processos de busca paralelos. A principal vantagem dos algoritmos genéticos, se houver, vem da operação de cruzamento. Pode ser demonstrado matematicamente que, se as posições do código genético forem permutadas inicialmente em ordem aleatória, o cruzamento não trará nenhuma vantagem. Intuitivamente, a vantagem vem da habilidade do cruzamento de combinar grandes blocos de genes que evoluem de forma independente para executar funções úteis, elevando assim o nível de granularidade em que a busca opera.
Na prática, os algoritmos genéticos tiveram amplo impacto sobre problemas de otimização, como leiaute de circuitos e escalonamento de prestação de serviços. No momento, não está claro se o interesse pelos algoritmos genéticos surge de seu desempenho ou de suas origens estéticamente atraente na teoria da evolução. Ainda há muito trabalho a ser feito para identificar as condições sob as quais os algoritmos genéticos funcionam bem.

2.2 - PseudoCódigo

2.2.1 - Pseudocódigo rede neural

Algoritmo de propagação da informação:

1. Um padrão (instância ou exemplo) é apresentado à camada de entrada da rede;
2. Essa entrada é multiplicado pelos pesos e chega na camada de neurônios;
3. O neurônio artificial realiza soma entre das entradas e aplica uma função de ativação sob o resultado;
4. A saída da função de ativação é a entrada da próxima camada. Se a próxima camada for a saída, seguimos para o passo 5. Caso contrário, voltamos ao passo 2.
5. A camada de saída retorna os resultados da rede.

2.2.2 Pseudocódigo algoritmo genético

Função ALGORITMO-GENÉTICO(população, FN-ADAPTA) retorna um indivíduo
    entradas: população, um conjunto de indivíduos.
        FN-ADAPTA, uma função que mede a adaptação de um indivíduo.

    repita

```
        nova_população ← conjunto vazio

        para i = 1 até TAMANHO(população) faça
            x ← SELEÇÃO-ALEATÓRIA(população, FN-ADAPTA)
            y ← SELEÇÃO-ALEATÓRIA(população, FN-ADAPTA)
            filho ← REPRODUZ(x, y)
            se (pequena probabilidade aleatória) então filho ←
            MUTAÇÃO(filho)
            adicionar filho a nova_população
            população ← nova_população

        até algum indivíduo estar adaptado o suficiente ou até ter decorrido
        tempo suficiente
        retornar o melhor indivíduo em população, de acordo com FN-ADAPTA

função REPRODUZ(x, y) retorna um indivíduo
    entradas: x, y, indivíduos país
    n ← COMPRIMENTO(x)c ← número aleatório de 1 a n
    retornar CONCATENA(SUBCADEIA(x, 1 c), SUBCADEIA(y, c + 1, n))
```

3 - Descrição do problema escolhido

3.1 Descrição do problema

O jogo do dinossauro é teoricamente simples. Durante o jogo encontram-se dois obstáculos: Cactus e pterodátilos.
Cactus tem sempre a mesma largura e altura e se apresentam sempre rente a superfície. O dinossauro se esquiva desse obstáculo pulando sobre ele e caso haja colisão, ele morre.
Pterodátilos têm sempre a mesma largura e altura, porém se apresentam em posições diferentes.
A primeira classe de pterodáctilo é a que se apresenta rente a superfície, como os cactos, o dinossauro deve pular para se esquivar, e caso haja colisão, ele morre.
A segunda classe de pterodáctilo é a que se apresenta um pouco acima da superfície. Nesse caso, o dinossauro pode abaixar ou pular para se esquivar, e se houver colisão, ele morre.
A terceira classe de pterodáctilo é a que se apresenta sob o dinossauro, quase no fim do eixo y do cenário. Para se esquivar desse obstáculo o dinossauro pode abaixar ou não realizar nenhum movimento, caso ele pule, atingirá o pterodáctilo. Havendo colisão, o dinossauro morre.
À medida que o jogo vai avançando a velocidade aumenta proporcionalmente. Isso traz um nível de dificuldade maior porque o dinossauro deve tomar suas decisões com antecedência e o tempo que ele passa no ar ao realizar um pulo aumenta.
O jogo original não tem fim. Mas, para fins didáticos, foi estipulado um limite de velocidade e de score que o dinossauro pode atingir.
O objetivo do algoritmo é que o dinossauro alcance esse score e se torne “imortal”.

A estratégia usada para a resolução do problema foi inserir em cada dinossauro um vetor contendo vários pesos. Esses pesos são usados na rede neural para obter duas saídas.
Caso a primeira saída seja maior que 0, o dinossauro pula.
Caso a segunda saída seja maior que 0 e a primeira menor que 0, o dinossauro se abaixa.
Caso as duas sejam menores que 0, o dinossauro não faz nada.
Os parâmetros de entrada da rede foram a distância do dinossauro para o objeto, a altura do dinossauro, a altura do objeto, a velocidade do jogo e a largura do objeto.
A rede é formada por uma camada de entrada contendo 5 neurônios, uma camada oculta também contendo 5 neurônios e uma camada de saída contendo dois neurônios. A função de ativação utilizada foi a RELU.

Para o treinamento da rede, usou-se uma população inicial de 200 dinossauros, cada um iniciado com pesos aleatórios. Quando todos os dinossauros foram mortos, os dois com maior score foram selecionados para o crossover.

O crossover consiste em gerar um filho usando os dois melhores dinossauros. Para cada peso, calculou-se a média e atribuiu ao dinossauro filho.
Os dois melhores dinossauros continuam para a próxima geração, enquanto todos os outros são substituídos pelo filho.
A próxima geração sempre tem uma população menor, a população diminui um indivíduo por geração.
Além do crossover, os dinossauros são submetidos a um processo de mutação. Há uma função definida como:

aleatoriedade = (2000 + melhorScoreAtual) / scoreGeracaoAnterior

Que nos dá a aleatoriedade aplicada a cada dinossauro após o crossover.
Para cada dinossauro na geração, percorremos o vetor de peso e fazemos mudanças aleatórias. Essas mudanças podem ser somar a (aleatoriedade / pesoOperacao) ao peso, subtrair a (aleatoriedade / pesoOperacao) ao peso ou não mudar o peso.
O “pesoOperacao” descrito acima é uma variável aleatória que varia de 1 a 5. Garantindo assim, ainda mais aleatoriedade no processo de mutação.

O intuito de todo esse treinamento é tentar gerar um dinossauro melhor pelo cruzamento dos dois melhores dinossauros somado a uma mutação aleatória nos pesos dos mesmos.
Para aumentar a chance de convergência, cada vez que a população chega a um número múltiplo de 10, os melhores dinossauros são mantidos, porém a população é aleatória, na tentativa de escapar de máximos locais.
Para finalizar o treinamento, algum dinossauro precisa fazer o score de 2000 5 vezes, ao término, os pesos desse dinossauro são salvos em um arquivo de texto chamado “saida.txt”.

4. Experimentos computacionais

4.1 Definição dos parâmetros da técnica

A população inicial contém 200 dinossauros, a cada geração, essa população diminui um indivíduo;
A rede contém 5 atributos:

1. Distância do dinossauro para o obstáculo: Para decidir a hora de pular ou abaixar, o rede precisa saber quão longe ele está do próximo obstáculo;
2. Altura do dinossauro: É importante para a rede saber a altura do dinossauro, pois caso ele esteja no alto (logo após pular) e encontre um obstaculo, ele pode não ter tempo de tocar o chão novamente antes de colidir com o obstaculo. Para isso ele pode abaixar, adiantando assim, sua volta ao chão;
3. Altura do obstáculo: Para decidir se pula ou se abaixa, a rede precisa saber qual a altura do obstáculo;
4. Comprimento do obstáculo: Esse parâmetro é importante para a rede ter conhecimento de quando o dinossauro já não pode mais ser atingido por um obstáculo.

A camada de entrada contém 5 neurônios, a camada oculta também contém 5 neurônios e a camada de saída contém dois neurônios.
A função de ativação usada em todos os neurônios foi a função RELU.
A aleatoriedade aplicada na mutação é definida pela função aleatoriedade = (2000 + scoreAtual) / scoreAnterior.

4.2 Descrição dos experimentos

A população foi escolhida de acordo com as limitações do hardware disponível para treinar o modelo. Infelizmente, ao testar mais de 200 dinossauros o jogo apresentava instabilidades e o treinamento não podia continuar. O tamanho da população influencia na chance de obter um agente ótimo e, portanto, no tempo de treinamento.
A rede não apresentou ganhos significativos com o aumento de camadas ocultas nem com o aumento de neurônio nas camadas.
Realizei teste em rede com até 3 camadas ocultas e até 10 neurônios por camadas. Porém, o que apresentou melhor resultado foi uma rede simples com apenas uma camada escondida.
A função de ativação RELU apresentou um desempenho melhor nesse problema. Outras funções como tangente hiperbólica e sigmóide foram testadas, porém não apresentaram bons resultados.
A função para definir a aleatoriedade foi o parâmetro mais complexo de se encontrar. O raciocínio que me levou a essa função foi que precisava avaliar se a população estava evoluindo ou não. Caso esteja, aplico uma aleatoriedade menor, caso não esteja, aplico uma aleatoriedade maior.

Para isso, usei scoreAtual / scoreAnterior.
Porém a aleatoriedade estava relativamente baixa, e a ideia foi acrescentar uma constante para aumentar a aleatoriedade aplicada. Essa constante foi o maior score possível, assim, cheguei a fórmula (2000 + scoreAtual) / scoreAnterior.

4.3 Descrição dos resultados alcançados

O tempo médio de treinamento foi de 1 hora e 10 minutos e são necessárias, em média, 50 gerações. Ao final do treinamento, os pesos do melhor dinossauro são salvos em um arquivo onde podemos reutilizá-lo para testes. O resultado é um dinossauro imortal.

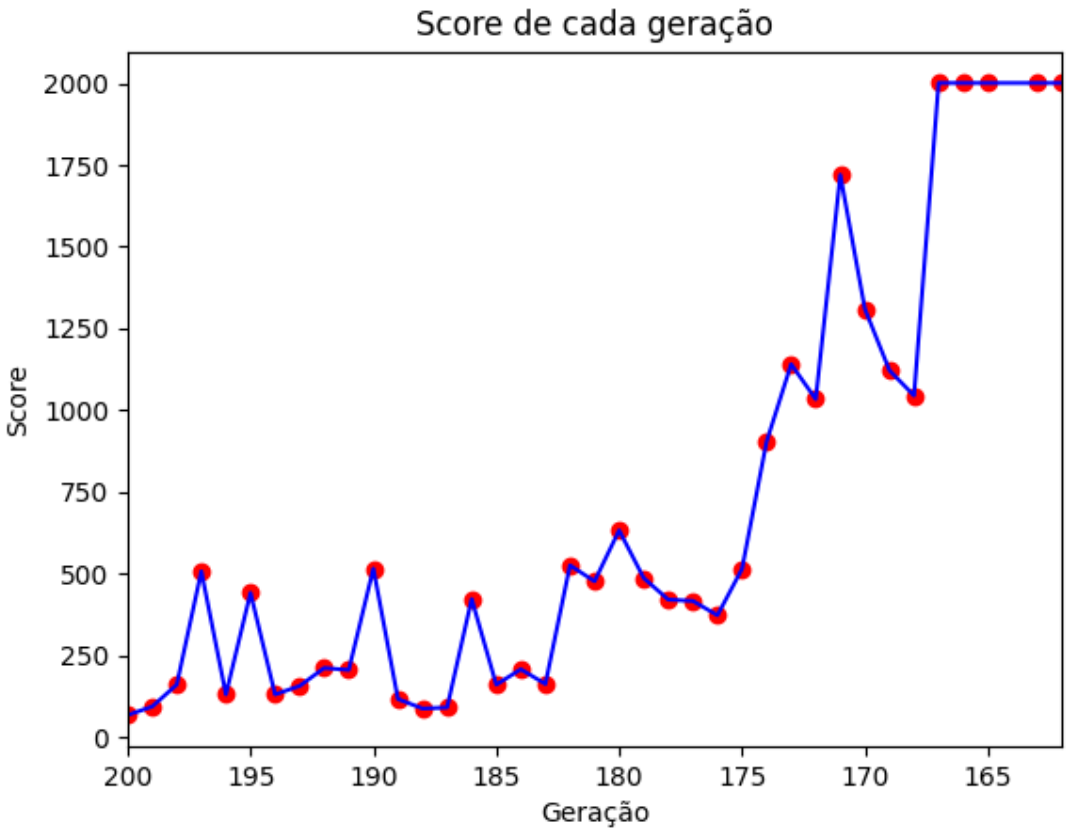

Imagem evidenciando o resultado de um treinamento. Nela, é possível analisar a quantidade de agentes de cada geração e seus determinados scores.

5 Resumo descritivo

5.1 Pontos positivos

Como pontos positivos destaco a experiência de ter sentido na pele como funciona todo o processo do desenvolvimento de um agente inteligente, desde a modelagem até a implementação.
Também destaco todo conhecimento adquirido através de pesquisa para implementação do projeto. Para a realização do mesmo, foi necessário muita pesquisa e troca de informações com os colegas de curso que têm mais afinidade na área, motivando assim, a colaboração entre os estudantes e a capacidade de resolver problemas que você, inicialmente, não tem noção de como atacar.

5.2 Pontos negativos

Um ponto negativo foi que a minha limitada experiência na área somada ao curto tempo do semestre acarretou em um projeto que entregou os resultados esperados, porém poderia ser bem melhor.
A utilização de uma biblioteca de algoritmos genéticos com redes neurais poderia ser a solução para a maioria dos meus problemas. A minha escolha de não utilizar bibliotecas com funções já prontas acabou causando algumas dificuldades que foram difíceis de contornar.

Referências bibliográficas

RUSSEL, Stuart russel. NORVIG, Peter Norvig. ”**Inteligência artificial**”. Terceira edição. 2013.

CÉSAR, Augusto César E. Redusino. “**Aplicações de algoritmos genéticos**”. Faculdade Salesiana Maria Auxiliadora. Rio De Janeiro. http://www.fsma.edu.br/si/edicao3/aplicacoes_de_alg_geneticos.pdf.

AURÉLIO, Marco Aurélio Cavalcanti Pacheco. “**ALGORITMOS GENÉTICOS: PRINCÍPIOS E APLICAÇÕES**”. Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro. https://www.inf.ufsc.br/~mauro.roisenberg/ine5377/Cursos-ICA/MQ-intro_apost.pdf

Python. https://pt.wikipedia.org/wiki/Python. Acessado em 26 de julho de 2022.

Programação orientada a objetos. https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_orientada_a_objetos. Acessado em 26 de julho de 2022.

Pygame. https://pt.wikipedia.org/wiki/Pygame. Acessado em 27 de julho de 2022.