Trabalho final de Inteligência Artificial

Igor Fabri Cervelin
Ciência da computação
Instituto Federal Catarinense
Videira - SC
Email: Igorcervelin@hotmail.com

***Abstract*—Este artigo tem como objetivo principal implementar os conhecimentos adquiridos na matéria de inteligência artificial. O trabalho em questão consiste em uma rede neural que aprenda a jogar Space Dash, jogo de plataforma desenvolvido em Python e baseado em Geometry Dash e Flappy Bird.**

***Keywords*—Rede, Neural, Inteligência, Artificial.**

I. Introdução

Com a evolução dos computadores, a ciência investe cada vez mais em fornecer inteligência a esses dispositivos, visando que os mesmos tenham capacitade e autonomia para solucionar problemas de raciocínio lógico utilizando métodos de aprendizagem. A linguagem de programação Python vem sendo a mais popular para implementar projetos voltados para machine learning. "Python é uma linguagem completa para o cientista de dados que atende bem a este profissional. Além de contém as bibliotecas necessárias para realizar a análise dos dados, também permite melhorar sua pesquisa através das bibliotecas de Machine Learning e de Deep Learning." [CATARINO, 20XX]

O trabalho a seguir visa implementar uma rede neural artificial em um jogo de plataforma, fazendo com que a máquina aprenda a desviar de obstáculos gerados em uma sequência aleatória. Para isso, foram utilizadas bibliotecas em Python como a pygame, responsável pelo desenvolvimento da mecânica e cenário do jogo, além da biblioteca numpy e scipy, responsáveis por gerenciar as matrizes de entrada, intermédio e saída, possibilitando o funcionamento da rede neural.

II. Desenvolvimento

A. Space Dash

Space Dash é o nome do jogo desenvolvido para esse trabalho. Baseado na jogabilidade e conceitos dos jogos Flappy Bird e Geometry dash, o jogo consiste em uma nave espacial cujo o objetivo é esquivar de obstáculos randômicos e sobreviver o maior tempo possível.

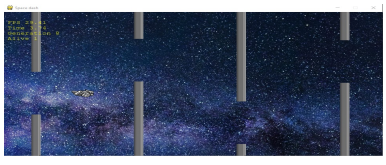

Fig. 1. Space dash em execução

Para simular a física da nave, são definidas as variáveis de altura de pulo e gravidade, de forma que a mesma consiga se manter no ar com pulos constantes. Os obstáculos são criados em loop a partir da lateral direita da tela, e movem-se em uma velocidade constante em direção a nave, que deve utilizar pulos e a própria gravidade para atravessá-los. A distância entre os obstáculos é fixa, assim como o espaço entre os obstáculos superiores e inferiores, por onde a nave deve passar, porém o tamanho dos obstáculos e definido por um intervalo entre 80px e 500px, fazendo com que a passagem localize-se em alturas diferentes em cada obstáculo.

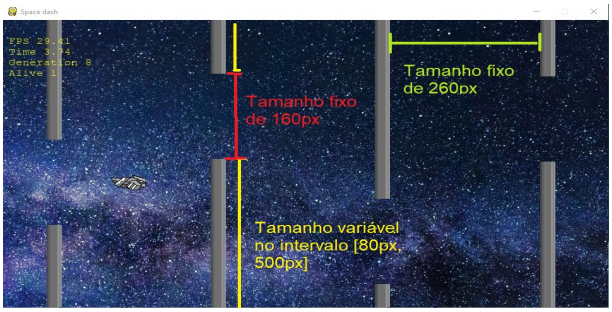

Fig. 2. Mapeamento das medidas

A colisão entre a nave e o obstáculo é calculada por uma função check-hits, utilizando a função img.get-rect() da biblioteca pygame, que permite identificar a área retangular abrangente por uma determinada imagem, como demonstra a figura abaixo.

Fig. 3. Área retangular ocupada pelo objeto definida pela função img.get-rect

Nesse caso, a função rect.colliderect() é responsável por verificar se a área de contato da nave está ocupando um mesmo pixel que a área de contato do obstáculo. Caso ocorra colisão com qualquer lateral do obstáculo, o status da nave é modificado de ALIVE para DEAD e o jogo é resetado. O jogo

não contém um final, ou seja, o objetivo é manter-se vivo pelo maior tempo possível.

B. Rede Neural

A rede neural foi desenvolvida utilizando as bibliotecas numpy para realizar as operações entre as matrizes, e scipy para utilizar a função de ativação. A leitura das matrizes deve-se ao fato de que os valores da rede neural podem ser interpretados no formato matricial, citando como exemplo três imputs de valor 1, 4 e 7, que formam uma matriz 1x3 [1, 4, 7].

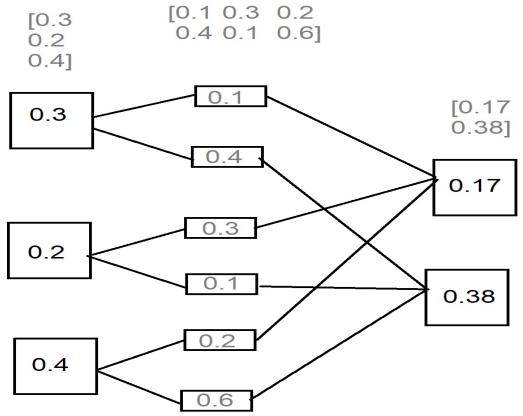

Fig. 4. Exemplo de matrizes convertidas da rede neural

Levando em consideração que o objetivo do jogo é passar no meio dos obstáculos, foram escolhidos dois inputs para a rede neural, sendo o Input 1 a distância horizontal da nave com o próximo obstáculo e o Input 2 a distância vertical da nave com a abertura entre os obstáculos, conforme ilustrado na imagem abaixo:

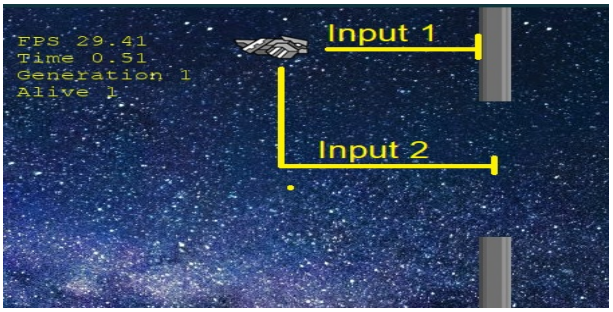

Fig. 5. Inputs da rede neural

O cálculo realizado no Input 1 consiste na distância entre o obstáculo e a lateral esquerda do display subtraido pela distância entre a nave e a lateral esquerda do display. Já para o Input 2, foi identificado a posição de colisão inferior do obstáculo superior e somado ao tamanho do espaço da abertura dividido pela metade. Isso garante que a rede neural consiga detectar exatamente o centro da abertura, aumentando

Fig. 6. Ilustração do cálculo realizado no Input 2

as chances de sucesso da nave atravesá-lo. A imagem abaixo ilustra de forma mais clara o cálculo do Input 2:

Definidos os inputs, o próximo passo é multiplicar a matriz de valores por um valor randômico denominado como ”weight”. Esse valor é o reponsável por fazer as variações na posição da nave entre os indivíduos. O valor obtido será o input da camada intermediária denominada ”Hidden”. Essa camada receberá os valores e aplicará a função de ativação, responsável por definir se um neurônio será ativado ou não. Nesse trabalho, foi utilizada a função sigmoid fornecida pela biblioteca scipy (lambda x: scipy.special.expit(x)). Após retornar da função de ativação, as matrizes obtidas serão novamente multiplicadas pelo ”weight”, formando a matriz de output da camada hidden. Essa matriz passará pela função de ativação novamente, definindo o output final como 0 ou 1. Existe apenas um output nesse cenário, sendo ele responsável por decidir se a nave deve realizar o movimento de pulo ou não. A figura abaixo é uma representação gráfica da explicação realizada neste parágrafo.

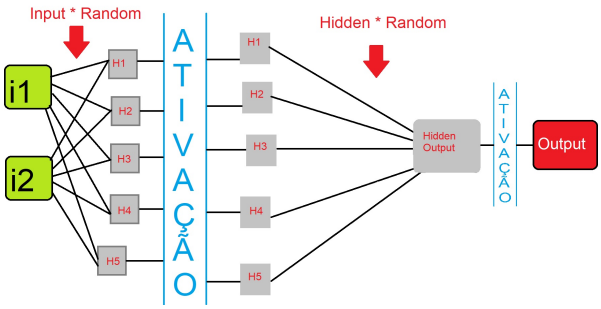

Fig. 7. Rede Neural

O aprendizado da rede é feito coletando as informações da melhor nave entre os indivíduos da geração e replicando-as com uma leve margem de erro. Seguindo a lógica, a melhor nave é a que foi mais longe, porém essa definição pode prejudicar o aprendizado, visto que a nave que foi mais longe pode não ter sido a nave que chegou mais perto de passar do obstáculo. Dito isso, é necessário levar em conta o quão próximo da passagem entre os obstáculos a nave estava quando foi eliminada. Nesse caso, é adicionado um bônus para a nave que estava mais perto da passagem.

Para evoluir os indivíduos, primeiramente é necessário separar as com bom desempenho dos indivíduos com mau desem-

penho. As naves com baixo desempenho serão descartadas, porém uma parcela predefinida dessas naves será sorteada para serem recicladas, voltando ao array principal para tentar novamente com um novo valor weight. Coletando os dois melhores e reaproveitando alguns dos ruins, o valor da população diminuirá. Para sanar essa situação, são utilizadas childrens, indivíduos criados a partir do produto de pares randômicos entre os melhores indivíduos e fazendo pequenas alterações em seu weight para que tenha uma alta taxa de sucesso.

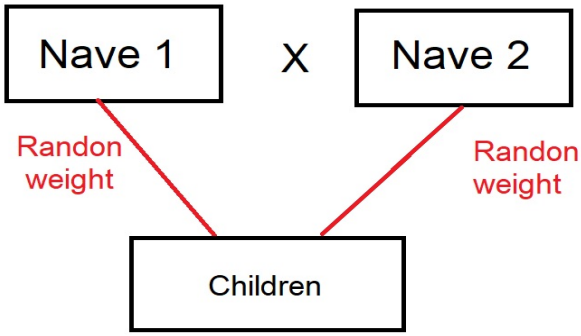

Fig. 8. Criação de uma children a partir de duas naves com bons valores

III. Resultados e discussões

Após realizar diversos testes modificando os parâmetros de entrada da rede neural, pode-se observar que a mesma funcionou conforme o esperado. Conforme o passar das gerações,a rede neural continuou aprendendo e se aperfeiçoando até obter um indivíduo perfeito, ou seja, uma nave que nunca colide. Devido o jogo não conter nenhuma mudança em seu decorrer, como aumento de velocidade ou redução de espaço, após passar pelos primeiros obstáculos, a propabilidade de um indivíduo ficar perfeito é muito alta. Como geralmente existia apenas um indivíduo perfeito, foi implementada uma função de reset a cada 20 segundos de jogo. Dessa forma, o indivíduo perfeito era clonado, e os outros indivíduos da geração conseguem obter uma performance muito melhor, aumentando a taxa de indivíduos perfeitos.

Após a proposta inicial ter funcionado, foram realizados alguns testes para melhorar o jogo. Foram implementadas duas novas funcionalidades: aumento da velocidade e redução da distância entre os obstáculos, ambas de acordo com o tempo de jogo. Apesar de ser possível adicionar o input de detecção de velocidade na rede neural, era necessário fornecer para a rede o controle das variáveis de gravidade e distância de pulo, pois seria necessário modificar ambos os valores conforme o tempo de ação diminui. Devido a isso, as funcionalidades não sairam conforme o planejado, portanto não foram implementadas na versão final do código

IV. Conclusão

Por fim, o trabalho atendeu os requisitos mínimos esperados. A aplicação dos conceitos estudados em inteligência artificial para automatizar um jogo auxiliou na compreensão

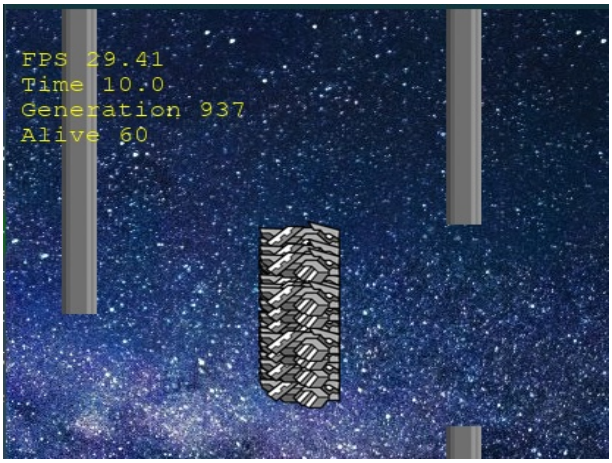

Fig. 9. 60 naves perfeitas de um total de 100 naves em uma geração avançada

do funcionamento de uma rede neural, além de complementar com conhecimentos a respeito de matrizes e lógica. Apesar do cumprimento dos requitos básicos, devido a contratempos com relação ao desenvolvimento, não foi possível implementar novas funcionalidades que tornariam o trabalho mais interessante, como aumento da dificuldade do jogo, que demandariam mais inputs e, consequentemente, mais aprendizado para a IA.

References

[1] CATARINO, Marino. Python: melhor linguagem para Machine learning. Entenda!. Disponível em: ¡https://www.impacta.com.br/blog/2020/04/06/python-melhor-linguagem-para-machine-learning-entenda/¿. Acesso em: 17. Ago 2020.