Carro autônomo utlizando NEAT

Bruno Libardoni
Ciência da Computação
Instituto Federal Catarinense
Videira, Santa Catarina
Email: brunolibardoni16@gmail.com

***Resumo*—Múltiplos algoritmos para gerar comportamentos de agentes de jogo foram produzidos nesses últimos anos. Grande parte deles se baseiam em Inteligência Artificial que precisam de treinamento. Neste contexto, este proposto trabalho apresenta uma estratégia de treinamento para desenvolver carros autônomos utlizando o algoritmo revolucionário NEAT capaz de completar o circuito sem que o carro se locomova para fora da pista. Com o algoritmo NEAT foi usado para encontrar a arquitetura de rede neural mais simples que consiga jogar o jogo impecavelmente. O algoritmo teve um tempo de convergência baixa atingindo o comportamento perfeito no jogo.**

***Palavra-chave*—Algoritmo; Rede; Inteligência Artificial; Arquitetura;**

I. Introduction

Python é a linguagem na vanguarda de pesquisa da AI, onde será possível encontrar a maioria das estruturas de machine learning e deep learning [1]. Deste modo, é possível usufruir de bibliotecas, métodos ou função prontas que auxiliam no momento do desenvolvimento. Com o auxílio de ferramentas já desenvolvidas e aplicadas em rede neural o proposto projeto final será desenvolvido na linguagem de programação Python.

O presente projeto final utiliza um algoritimo que faz a criação das redes neurais artificiais chamado de NEAT, juntamente com o pygame e outras bibliotecas padrões que foram necessárias na hora da implementação do trabalho. O projeto é um carro autônomo utilizando uma inteligência artificial que por meio de gerações faz a apredizagem do circuito de corrida, como dirigir e que por meio deste aprendizado a IA consiga completar o circuito de maneira que o carro não desvie da rota e da pista.

Neste trabalho, é aplicado um algoritmo de Neuroevolução conhecido como Neuroevolução de Topologias Aumentadas do acrônimo NEAT no ambiente do jogo de corrida. Com este algoritmo, além de encontrar um agente simples para jogar o jogo, encontra-o rapidamente, ou seja, poucos gerações e jogando cenários curtos. NEAT é um algoritmo originário da Rede Neural Artificial (RNA), mas possui um otimizador de arquitetura dentro dele. O otimizador é feito de Algoritmo Genético (GA) e é necessário para encontrar a arquitetura mais eficaz para RNA, uma arquitetura que tem baixo custo em termos de tempo de cálculo e complexidade [2] .

Na sessão II, será analisado trabalhos relacionados que usam neuroevolução (NEAT) em outros jogos. Na sessão da metodologia será detalhada o propósito da estruturação do jogo e da configuração do NEAT. Nos resultados será apresentados as gerações, fitness e o tempo levado que o carro autônomo levou até a colisão.

II. Trabalhos relacionados

A aplicação da Neuroevolução em jogos vem sendo utilizada há algum tempo com resultados satisfatórios na criação de agentes inteligentes capazes de jogar em nível humano e até super-humano [3]. Assim, NEAT é uma das técnicas mais promissoras no âmbido de jogos.

Flappy Bird é um jogo popular originalmente designado para dispositivos móveis. Hoje, o jogo já foi implementado em diversos tipos de linguagem de programação e tornou-se um meio ambiente de diferentes campos de teste com a inteligência artificial. A literatura [5] contém referencia de treinamento mínimo do jogo Flappy Bird indefinidamente com NEAT. No trabalho, propôs um desenvolvimento de um player virtual autônomo usando NEAT algoritmo neuroevolicionário capaz de jogar o jogo Flappy Bird. Desse modo, foi utilizado NEAT para encontrar a arquitetura de rede mais simples que consiga jogar perfeitamente o jogo. A modelagem do cenário a função de fitness foram definidas para garantir uma representação adequada do problema em comparação com o jogo real.

NEAT também pode ser aplicado ao paradigma multiobjetivo em certos jogos, onde NPCs (personagens não jogáveis) devem realizar mais de uma tarefa, como Ms. Pac-Man [7]. Neste jogo, é utilizada uma variação conhecida como Modular Multiobjetivo NEAT em que os tipos de busca sempre buscarão um agente com comportamento multimodal [8].

Um outro gênero da aplicação de neuroevolução é em jogos de lutas. Na literatura [2], é usado NEAT para resolver o problema do algoritmo KNN chamado MizunoAI da incapacidade de ajustar seus parâmetros automaticamente. Na implementação, NEAT recebe 6 entradas de distância no eixo X e Y, assim a saída sendo a probabilidade de todos os movimentos que o oponente seja capaz de realizar. Deste modo, todas as saídas serão processadas pelo simulador para determinar a melhor contramedida baseadas nas predições.

III. Metodologia

Nesta sessão será apresentada a metodologia das fases na hora da implementação da IA, do circuito e do carro. Na primeira fase no desenvolvimento do projeto final, foi estabelecer quais ferramentas usar na implementação. Deste modo, foi escolhido NEAT pela simplicidade da estrutura.

NEAT utiliza especiação para evitar convergência prematura para soluções subótimas. As inovações potenciais são protegidas, dando às estruturas mais novas uma oportunidade melhor de se desenvolver, em vez de serem descartadas no início em favor de estruturas existentes mais desenvolvidas. Isso é feito permitindo que os indivíduos competam principalmente com outros membros de sua espécie, em vez de com a população inteira. O número de descendentes permitido por espécie é proporcional à média aptidão dessa espécie [9].

Para o uso do algoritmo NEAT é preciso de apenas dois arquivos, main e arquivo de configuração. Na main encontra-se o desenvolvimento do código fonte e o arquivo de configuração é representada como, uma linguagem de análise de arquivo de configuração básica que fornece uma estrutura semelhante à que você encontraria em arquivos INI do Microsoft Windows [5]. Para fazer o uso do mesmo, é possível obter através da documentação ou pelo download do próprio site do NEAT.

No arquivo de configuração do NEAT, tem-se diversos parâmetros que na própria documentação é explicado a função de cada um deles. Estes parâmetros deverão ser estabelecidos para o funcionamento da IA e na ordem de como é proporcionada na documentação. É possível fazer modificações dos parâmetros de acordo com sua necessidade. Neste projeto foi proporcionado os seguintes parâmetros como mostrado na Fig 1.

```
fitness_criterion     = max
fitness_threshold     = 100000
pop_size              = 30
reset_on_extinction   = True

[DefaultGenome]
# node activation options
activation_default      = tanh
activation_mutate_rate  = 0.01
activation_options      = tanh

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.01
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power       = 0.5
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.1

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.5

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.01

feed_forward            = True
initial_connection      = full
```

Fig. 1. Arquivo de configuração NEAT.

Com a abordagem selecionada, será feito o treinamento do carro e a construção das redes neurais mais complexas usando técnica de aprendizado evolutiva. Foi identificado os parâmetros de entrada apropiados e sua função de aptidão para as redes neurais a serem empregra.

Em grande parte dos experimentos foi deixado intacto arquivo de configuração NEAT. Por exemplo, há várias funções de ativação disponíveis no NEAT, porém neste projeto foi implementada a tangente, de modo que esta função de ativação permita-nos obter nosso valor para a camada de saída seja qual for o resultado e esmagar este valor em 2 valores definidos (-1;1). Desse modo, com o valor da saída será feita uma verificação de caso o valor seja menor de 0,5 neste caso o carro seguirá para o sentido esquerdo, senão para o sentido direito.

Na segunda fase é a implementação do código para a estruturação da IA juntamente com o jogo. Na função main, como mostrado na Fig 2, é possível analisar o caminho até o arquivo de configuração NEAT, a criação do core da evolução da classe do algoritimo e logo em seguida a parte da impressão no terminal dos status do melhor fitness, geração e a inicialização do NEAT com o máximo de threshold. Para iniciar o algoritmo NEAT é necessário declarar estes parâmetros.

```
__name__ == "__main__":

caminho = "./config-feedforward.txt"
config = neat.config.Config(neat.DefaultGenome,
neat.DefaultReproduction, neat.DefaultSpeciesSet,
neat.DefaultStagnation, caminho)

# Criando o core da evolução da classe do algoritmo
p = neat.Population(config)

p.add_reporter(neat.StdOutReporter(True))
stats = neat.StatisticsReporter()
p.add_reporter(stats)

# Rodando o NEAT
p.run(run_car, 1000)
```

Fig. 2. Função Main.

Na própria inicialização do jogo é realizado a criação dos genomas com o tamanho da população que foi configurada no arquivo do NEAT, é preciso criar para cada rede o genomo e o fitness. Para cada passagem do carro na rede irá ser obtido a melhor ação através dos resultados proporcionados da rede e com isso atribuído o angulo em que o carro irá realizar. Para o último carro que estiver ainda ”vivo”, será recompesado com um valor definido para que na próxima geração seja construído uma nova população através do fitness deste último carro.

Na terceira fase, é a implementação da renderização do jogo. A construção do circuito foi baseado no tamanho da tela do autor e na escala do carro autônomo para que o carro conseguisse realizar as curvas sem que houvesse somente colisão. O carro deverá percorrer o circuito sem que saia da rota. A inteligência artificial deverá conseguir pilotar em qualquer estrutura de circuito. Desse modo, foi construido 2 sensores na

parte traseira do carro, 2 sensores na parte frontal do carro e um sensor na parte central totalizando 5 sensores. Desse modo, no arquivo de configuração do NEAT, num_imputs deverá ser 5, pois são os dados que serão colocados na rede.

Para as saídas foi aderida 2 num_outputs, de modo que serão os angulos em que o carro irá realizar (direita ou esquerda). No começo do projeto a idealização era com três saídas distintas, onde esta terceira saída seria o movimento do carro para frente. Para proporcionar uma maior simplicidade na rede, foi implementado como a velocidade constante do carro, assim proporcionando para a rede duas saídas.

O carro autônomo deverá percorrer o circuito sem que saia do mesmo, para alcançar este objetivo foi programado a colisão do carro. A colisão, é ativada quando os sensores encostam na parede, que, neste caso foi configurada na cor branca. Quando os sensores identificar a cor branca no píxel, então a função de colisão será acionada e o carro voltará para o estado inicial.

IV. Resultados

Os experimentos foram divididos em duas fases. Em ambas as fases foram designados em alguns parâmetros específicos do algoritmo do jogo e do arquivo de configuração NEAT para verificar como a rede artificial iria se comportar. O propósito destas duas fases é aveririguar em quantas gerações a rede iria precisar para encontrar o agente quase ou perfeito para realizar o circuito completo. A pista será a mesma até a finalização e a troca do mapa, com isso NEAT conseguirá facilmente aprender a detectar as partes laterais do circuito em branco para evitar a colisão.

No primeiro teste, foi aderido uma população com 30 carros e um fitness de threshold de 1.000.000, e deixado as configurações padrões pré definidas como mostrado na sessão anterior. Neste experimento foi modificado o mapa, como mostrado na Fig 3, para deixar com algumas falhas nas curvas para conseguir visualizar se a rede estava aprendendo a passar nestes locais estreitos.

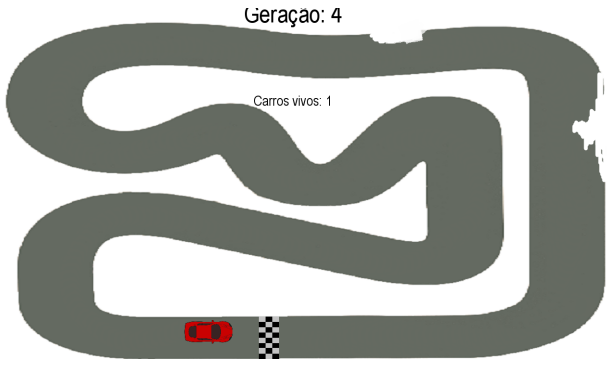

Fig. 3. Teste 1

Os resultados obtidos no primeiro teste não foram os esperados, pois em alguns momentos o carro obtia sempre o mesmo fitness e consequentemente encontrava-se estagnado e com isso alcançava sempre o mesmo ponto de colisão. Deste modo, foi reconfigurado o tratamento de recompensa do fitness, assim foi proporcionado uma recompensa maior. Com o tratamento da recompensa do fitness reconfigurado, com sucesso a rede conseguiu em poucas gerações aprender e realizar o circuito. Neste mesmo experimento foi utizado uma velocidade constante de 20km/h e com um ângulo de virada de 10, deste modo sendo proporcionado uma velocidade constante baixa juntamente com o ângulo de virada proporcional a velocidade.

Com aproximadamente 4 gerações a rede conseguiu encontrar o agente perfeito, deste modo o carro autônomo conseguiu realizar o circuito inúmeras vezes. Quando o fitness alcançar ou ultrapassar o fitness threshold, por padrão o algoritmo NEAT irá encerrar o programa, e mostrará as estatísticas do algoritmo como: ID, Fitness, Geração e População como mostrado na Fig 4. Como o algortimo e os parâmetros são simples, o algoritmo NEAT consegue rapidamente achar o agente, consequentemente resultando em aproveitamento de tempo.

```
****** Running generation 3 ******

Population's average fitness: 12409.00000 stdev: 31451.61330
Best fitness: 156600.00000 - size: (4, 12) - species 1 - id 106
Average adjusted fitness: 0.079
Mean genetic distance 1.467, standard deviation 0.255
Population of 30 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    3    30  156600.0    0.079     0
Total extinctions: 0
Generation time: 5.223 sec (3.577 average)
```

Fig. 4. Tempo e fitness 1

Durante a primeira fase de teste, foi possível analisar uma forma estranha na locomoção do carro. O mesmo locomovia-se para frente constantemente, porém juntamente com o deslocamento do carro estava movendo-se para os dois lados mesmo quando não continha nenhuma curva e era para apenas andar em linha reta. Mesmo na forma de deslocação estranha e com a pista modificada com caminhos estreitos, a rede conseguiu completar o circuito em um tempo baixo e poucas gerações.

Na segunda fase do teste, foi feita algumas mudanças de parâmetros, tais como, o tamanho da população foi implementada para 50 carros em cada geração e o aumento da velocidade para 25km/h. Com o aumento da velocidade tivemos que aumentar o ângulo de virada do carro, pois com esta velocidade e o mesmo circuito, em algumas curvas da pista o carro não iria conseguir realizar as curvas sem que houvesse colisão, neste caso teríamos duas opções - alterar o ângulo ou refazer o circuito com curvas mais abertas. Neste experimento, foi apenas modificado o valor do ângulo para 15.

Neste segundo teste foi verificado uma maior dificuldade da rede para completar o circuito. A velocidade conseguiu influenciar a rede e dificultando o mesmo no momento do aprendizado, deste modo a rede necessitou de uma quantidade maior de gerações para encontrar o agente perfeito. Em 8 gerações a rede encontrou este agente e o algoritmo finalizou o programa quando ultrapassou o fitness threshold. Na Fig

5 é possível analisar as informações que o algoritmo NEAT retornou do agente que foi encontrado.

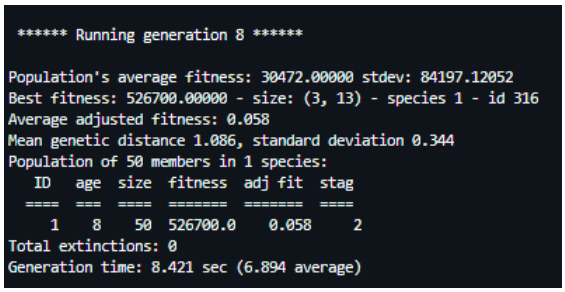

Fig. 5. Tempo e fitness Teste 2

Nesta segunda fase de teste também foi contemplado o modo estranho do carro de se locomover. Porém, o carro ainda consegue realizar impecavelmente o circuito de maneira que não haja colisão. Para a correção do mesmo, teríamos de colocar mais uma entrada na rede, para a parte de locomoção para frente, assim entraríamos em alguns problemas adicionais como: complexidade e estagnção do carro. Como o objetivo do trabalho não era construir uma rede complexa, o presente carro autônomo consegue perfeitamente concluir o circuito numerosas vezes, e assim alcaçar o objetivo do projeto de concluir o mapa.

V. Conclusão

Neste trabalho, o autor prôpos uma estratégia de treinamento utlizando NEAT para gerar carros autônomos capazes de dirigir sem haver algum tipo de colisão. O agente conseguiu dominar o circuito de modo que ao passar das gerações o algoritmo sempre pretende buscar a evolução mais implícita para um problema e que o fitness apresentado ajudou o NEAT a encontrar uma arquitetura de rede com a solução mais simples. As técnicas descutidas neste trabalho, auxiliaram o algoritmo a encontrar as soluções em um curto espaço de tempo, demonstrando sua eficácia.

Com o uso do tamanho da população de 30 carros e com velocidade de 20km/h, foi possivel observar que a rede foi capaz de convergir para um estado ideal com poucas gerações e encontrar o agente perfeito. Com uma população de 50 carros e velocidadede 25km/h, foi observado que o algoritmo evolucionário demorou um pouco mais para chegar a um estado ideal. Neste ponto em ambos os testes, o agente poderá jogar o jogo infinitamente.

O algoritmo NEAT é capaz de realizar cálculos complexos e determinísticos rapidamente para encontrar o agente perfeito e com isso adicionando novos elementos estruturais, assim as redes são tendenciosas a serem pequenas. Porém, conforme é aumentada gradativamente a complexidade do jogo com vários fatores a serem analisados durante o jogo, o algoritmo NEAT deixa a desejar, aumentando exponencialmente a complexidade e o tempo das gerações.

Referência

[1] POINTER, Ian. Conheça as 5 melhores linguagens de programação para Inteligência Artificial. 2018. Disponível em: https://computerworld.com.br/2018/07/04/conheca-5-melhores-linguagens-de-programacao-para-inteligencia-artificial/. Acesso em: 12 ago. 2020.

[2] T. Kristo and N. U. Maulidevi, "Deduction of fighting game countermeasures using Neuroevolution of Augmenting Topologies," 2016 International Conference on Data and Software Engineering (ICoDSE), Denpasar, 2016, pp. 1-6, doi: 10.1109/ICODSE.2016.7936127.

[3] THIHA, Phyo. Extending Robot Soccer Using NEAT. 2009. Disponível em: https://www.cs.swarthmore.edu/ meeden/cs81/s09/finals/Phyo.pdf. Acesso em: 13 ago. 2020.

[4] S. Samothrakis, D. Perez-Liebana, S. M. Lucas, and M. Fasli, "Neuroevolution for General Video Game Playing," 2015 IEEE Conference on Computational Intelligence and Games, CIG 2015 - Proceedings, pp. 200–207, 2015.

[5] NEAT-PYTHON. Configuration file description. Disponível em: https://neat-python.readthedocs.io/en/latest/config_file.html. Acesso em: 12 ago. 2020.

[6] CORDEIRO, Matheus G.; SERAFIM, Paulo Bruno S.; NOGUEIRA, Yuri Lenon B.; VIDAL, Creto A.; CAVALCANTE NETO, Joaquim B.. A Minimal Training Strategy to Play Flappy Bird Indefinitely with NEAT. 2019. Disponível em: https://ieeexplore.ieee.org/abstract/document/8924807. Acesso em: 13 ago. 2020.

[7] J. Schrum and R. Miikkulainen, "Discovering multimodal behavior in ms. pac-man through evolution of modular neural networks," IEEE Transactions on Computational Intelligence and AI in Games, vol. 8, no. 1, pp. 67–81, March 2016.

[8] J. Schrum and R. Miikkulainen, "Constructing complex npc behavior via multi-objective neuroevolution," in Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, ser. AIIDE'08. AAAI Press, 2008, pp. 108–113. [Online]. Available: http://dl.acm.org/citation.cfm?id=3022539.3022558

[9] M. Wittkamp, L. Barone and P. Hingston, "Using NEAT for continuous adaptation and teamwork formation in Pacman," 2008 IEEE Symposium On Computational Intelligence and Games, Perth, WA, 2008, pp. 234-242, doi: 10.1109/CIG.2008.5035645.