Aplicação de Algoritmos Evolutivos em Jogos

Rodrigo A. S. de Araújo - 9763064, Alex Sander R. Silva 9779350

ICMC– Universidade de São Paulo(USP)

SCC0230 - Inteligência Artificial

Resumo. *O trabalho consiste na abordagem do algoritmo NEAT para aprendizado e adaptabilidade em jogos. Foram escolhidos os jogos Super Mario Bros (1985) e Tibia. O grupo modificou tais jogos inserindo controladores que recebem inputs vindos do jogo, e tomam ações baseadas em redes neurais. Foi também criado um processo Master, que produz novas redes neurais (indivíduos) e se comunica com os Slaves, que são os processos que executam o jogo e aplicam a rede neural sobre ele. Os Slaves testam as redes neurais e informam ao Master a eficiência de tais redes no jogo. Com tal informação, o Master gera um conjunto novo de redes neurais, visando produzir indivíduos mais adaptados ao jogo do experimento.*

1. Introdução

O programa implementado é organizado, do nível mais baixo para o mais alto, nos seguintes grupos: O indivíduo, ou genoma, que produz de forma determinística a rede neural associada a ele, a partir de uma estrutura de dados contidos nele. Ele é testado e avaliado pelo jogo, e depois reproduzido ou mutado, gerando novos indivíduos, ou então sendo excluído sem gerar prole.

Os indivíduos que compartilham certa semelhança quanta à herança genética, topologia da rede e comportamento geral, se agrupam entre si em espécies. Os indivíduos só podem se reproduzir com outros da mesma espécie, para evitar a geração de filhos defeituosos.

Uma geração é formada por todo o conjunto de espécies atuais, ou seja, abrange toda a população de indivíduos naquele momento. Uma vez que toda a população de uma geração é testada e avaliada, o algoritmo NEAT é aplicado sobre a geração e produz a geração seguinte. É esperado que as gerações gradualmente se aperfeiçoem na tarefa que lhes foi dada, e possuam indivíduos cada vez mais eficientes.

Todo o histórico de gerações, bem como um conjunto de parâmetros específicos estão contidos em no experimento. Tais parâmetros são por exemplo o modo que a avaliação ou a reprodução serão feitas, qual a quantidade limite de indivíduos por geração ou, no caso do Mario o tamanho da grade de inputs em torno dele. Nos dois experimentos foi utilizada uma biblioteca de NEAT já implementada em c++[4].

2. Fundamentação Teórica

Algoritmos Genéticos são uma forma de algoritmos evolutivos que se inspiram nos conceito de seleção natural, reprodução e genética. De modo geral, um grupo de indivíduos são aleatoriamente gerados e avaliados. Os melhores são selecionados e o

próximo grupo de indivíduos é gerado a partir deles. As técnicas de seleção e reprodução, bem como o ajuste de seus parâmetros, compõem uma rica área de estudos.

Redes Neurais Artificiais, ou simplesmente Redes Neurais, são um modelo inspirado no sistema nervoso central, principalmente no funcionamento dos neurônios. De modo geral, a rede neural consiste em receber diversos inputs e conectá-los aos neurônios que processam o peso que cada input deve ter sobre si, e então geram um output a partir dessas informações. Esse output pode de input para outros neurônios ou pode ser o output final do programa. Em jogos, é comum ligar os outputs às estatísticas do personagem e do ambiente, e os outputs aos botões a serem pressionados. Entretanto, é preciso construir a rede neural, sua topologia, suas conexões e o peso dessas conexões.

O NEAT é uma das formas de utilizar algoritmos genéticos para construir redes neurais. Essa estratégia gera aleatoriamente um grupo de redes neurais, as avalia, e então o próximo grupo é gerado a partir de uma seleção do grupo inicial. O NEAT começa com topologias simples, ou seja, poucos nós e poucas conexões entre nós, e vai se tornando mais complexo com a adição de mais nós e mais conexões. Pela variedade de indivíduos com topologias bem diferentes em uma geração, haverá o conceito de especiação, que é a verificação da semelhança e compatibilidade entre indivíduos antes de tentar cruzá-los. Novos parâmetros específicos ao NEAT são adicionados, tornando ainda mais desafiador configurá-los de maneira eficiente. O artigo “Evolving Neural Networks through Augmenting Topologies”[2] Kenneth O. Stanley e Risto Miikkulainen, de 2002 é tido como um estudo base para abordagens e implementações do NEAT.

3. Desenvolvimento

Os experimentos do jogo do Mario foram realizados sobre um código fonte que implementa o jogo em c++ e SDL [1]. Tal código fonte foi modificado para utilização no experimento.

No jogo do Mario, foi dada ênfase no paralelismo dos processos e na otimização da velocidade de execução do teste. Para agilizar os testes, sempre que o Mario permanece mais de 5 segundos em tela sem mudar sua posição X no mapa ele morre automaticamente. Dessa forma, se ele ficar preso em um obstáculo ou parar de se mover, o processo não precisa esperar o tempo do nível zerar para poder abrir outro teste.

Para executar mais de 60 processos em concorrência o código foi modificado para poder desabilitar o processamento e renderização de imagem e a geração de sons, diminuindo a carga da CPU. A duração dos frames do jogo também foi diminuída para 75%. Tornando o jogo levemente mais acelerado.

Nos testes do jogo os inputs das redes neurais correspondem a uma matriz quadrada de pequenas áreas quadradas chamada de grade, que está centralizada no personagem do Mario. Estas áreas quadradas detectam se há um bloco sólido ou um inimigo dentro delas. Desta forma a grade tem um mapa do entorno do personagem. Ela pode variar de tamanho para cada experimento, sendo as grades com dimensões 5 e 7 as mais eficientes pelos experimentos realizados.

Os outputs são as teclas A, S, D e Espaço, que correspondem aos movimentos para esquerda, abaixar, para direita e pular respectivamente. Esse outputs variam de -1 a

1, e um dos parâmetros escolhidos para cada experimento é o threshold, que é o valor mínimo de um output para a tecla ser apertada. Isto é, se um threshold é de 0.5, então a tecla correspondente só será apertada caso o output esteja na faixa de valores de [0.5, 1].

O critério de avaliação do jogo Mario é simplesmente a coordenada X de sua posição no mapa no nível. Ou seja, até onde ele conseguiu progredir para a direita.

No jogo Tibia, por sua vez, foi escolhido um escopo limitado dentro da infinidade de ações possíveis em um MMORPG: sobreviver a uma onda de monstros podendo apenas se curar. Uma vez que isso é aprendido, pode-se ensinar à máquina aspectos mais avançados como aprender a derrotar as criaturas, e depois a andar e utilizar o terreno a seu favor.

Para poder realizar os testes da forma mais eficiente possível, foi criado um servidor privado offline com uma série de modificações, como a remoção da penalidade de morte ou do ganho de experiência de qualquer tipo, além da criação de um sistema de testes com a menor variação possível, dentro das limitações do jogo. Esse teste consiste em seis ondas de monstros cada vez mais fortes e com diferentes características para estimular um possível surgimento de estratégias únicas para sobreviver a eles.

Os inputs utilizados foram: porcentagem de vida e mana (pontos de magia) atuais; e o nome, as distâncias relativas ao personagem no eixo x e y, e a direção para a qual o cada um dos no máximo doze monstros estão olhando. Essas informações sobre os monstros são importantes porque uns são mais fortes do que outros, e algumas de suas habilidades atingem somente a direção para a qual estão virados, e em alguns casos, somente em linha reta. Dessa forma, o personagem pode aprender que o nível de dificuldade varia também de acordo a posição dos monstros.

Os outputs utilizados foram: um para cada um dos três maiores tamanhos de poção de vida, um para a poção de mana, um para a magia de cura simples, um para a magia de cura especial de uso único por teste.

O critério de avaliação, ou fitness, é dado por: fitness = t, se t < 120, ou fitness = 720 - (gasto/100), se t = 120. Ou seja, um ponto por segundo vivo, e caso o teste seja concluído, recebe maior pontuação o indivíduo que utilizou a menor quantidade de recursos. O maior (gasto/100) possível é 600, de modo que o personagem sobreviver a qualquer custo é apenas um pouco melhor do que morrer.

Já que manter o servidor privado, o jogo, e os programas rodando ao mesmo tempo se provou custoso demais para nossos computadores, não foi possível realizar testes com mais do que um Tibia em execução concorrente sem comprometer o resultado dos testes com lentidão no servidor. Como também não é possível acelerar a passagem de tempo artificialmente, o personagem ainda não conseguiu completar nenhum teste, em um total de quarenta gerações testadas.

3.1. Comunicação entre Master Slave e Concorrência de Processos

A base da comunicação entre processos é feita através de arquivos. No início de uma nova geração, o Master produz os arquivos com as redes neurais dos indivíduos no formato nn<idx> sendo idx o índice do indivíduo, que varia de 0 a (população total - 1). Ele reseta o arquivos de tests_available e tests_finished colocando 0 em seus conteúdo.

Os Slaves que estão em estado de espera checam o tests_available periodicamente. Se um Slave consegue acesso ao arquivo e verifica que o conteúdo do

arquivo é um índice menor do que a população total da geração, ele incrementa o índice no arquivo de tests_available e acessa a rede neural nn<idx> correspondente ao índice. Como o arquivo de tests_available é uma região crítica do sistema concorrente, um sistema de lock impede que dois processos acessem o arquivo simultaneamente, impedindo erros de concorrência de processos.

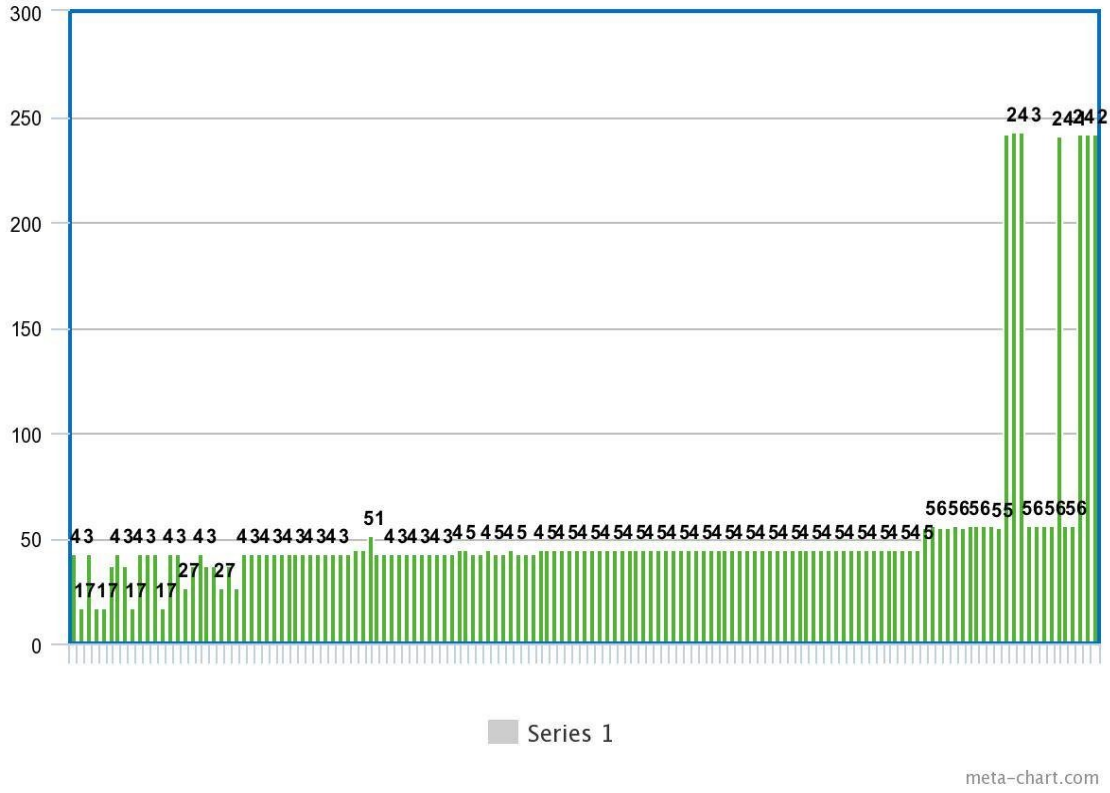
Depois de capturar a rede neural, o Slave realiza o teste, obtém o fitness score e escreve-o no arquivo nn<idx>. Logo em seguida acessa o arquivo tests_finished e incrementa-o, com os mesmos cuidados de concorrência do tests_available.

O Master verifica o arquivo tests_finished periodicamente até que ele chegue no número igual ao da população de indivíduos. Quando isso ocorre, todos os testes já foram concluídos. O Master captura o fitness score de cada indivíduo, disponível no arquivo nn<idx> e cria uma nova geração com base nesses resultados.

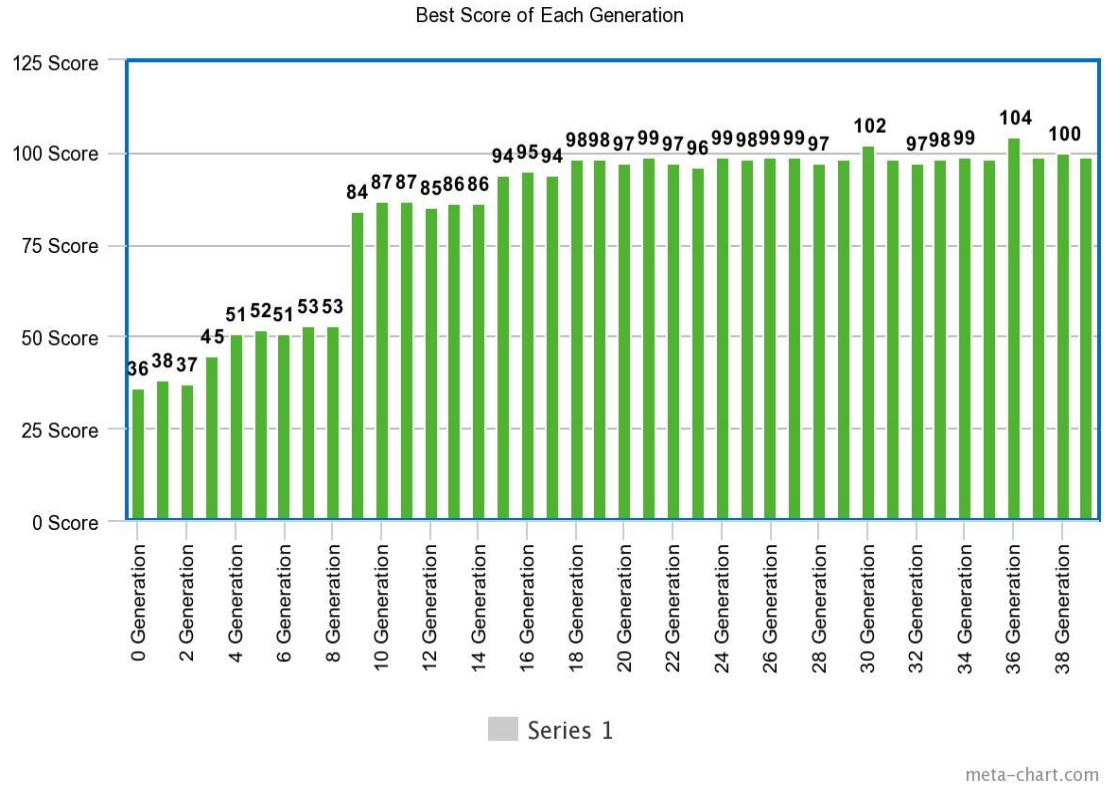
4. Conclusão

Pelos experimentos nos dois jogos foi possível observar a evolução e adaptação desse algoritmo para resolver os problemas. Características observáveis foram a presença de platôs de desempenho. Em que durante várias gerações o fitness score máximo permanecia estacionário. Isso decorria, no caso do Mario por exemplo, de algum obstáculo diferente que a rede neural ainda não estava pronta para enfrentar, fossem estes buracos ou canos muito longos. O experimento obteve sucesso em mostrar a aplicação de algoritmos genéticos sobre jogos.

No caso do Mario, o algoritmo conseguiu resolver o nível após 127 gerações. O gráfico abaixo mostra as gerações no eixo x, e o máximo fitness por geração no eixo y. O fitness para concluir o nível é 243, que corresponde à distância do ponto inicial do nível até a bandeira.



No caso do Tibia, o algoritmo não conseguiu cumprir o seu primeiro objetivo até o momento — sobreviver a todas as seis etapas do teste — em que este documento foi escrito, isso porque, em média, cada teste requer cerca de oitenta segundos, é não é possível executar mais de um teste por vez em nossas máquinas. Entretanto, mesmo em um número pequeno de gerações, o personagem aprendeu a utilizar somente as poções de vida maiores (e mais caras), bem como a usar a magia de cura. Infelizmente, ele não foi capaz de aprender em cerca de 40 gerações a poupar seus recursos, e acaba sucumbindo na fase final do teste, a qual só é possível concluir caso tenha economizado mana (pontos de magia) ou a magia especial de cura, que só pode ser utilizada uma vez por teste.



References

[1] https://github.com/jakowskidev/uMario_Jakowski

[2] http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

[3] https://github.com/simoesusp/SSC0713_Sistemas-Evolutivos-Aplicados-a-Robotica

[4] https://github.com/hav4ik/tinyai