Um ambiente de jogo eletrônico para avaliar algoritmos coevolutivos

Karine da Silva Miras de Araujo
Centro de Matemática, Computação e Cognição (CMCC)
Universidade Federal do ABC (UFABC)
Santo André, Brasil
E-mail: karine.smiras@gmail.com

Fabrício Olivetti de França
Centro de Matemática, Computação e Cognição (CMCC)
Universidade Federal do ABC (UFABC)
Santo André, Brasil
E-mail: folivetti@ufabc.edu.br

***Resumo*—Uma das aplicações de inteligência artificial em jogos eletrônicos consiste em fazer um agente artificial aprender a executar uma determinada tarefa com sucesso. Para isso faz-se necessário o uso de um algoritmo que possa aprender a determinar as sequências de ações de acordo com o ambiente observado. Para esse fim existem diversas técnicas de aprendizado supervisionado que permitem aprender a resposta correta através de exemplos. Porém, no caso de jogos eletrônicos a resposta correta pode ser apenas mensurada após a completude da sequência de ações, sendo então impossível determinar a resposta correta de cada instante de tempo. Uma forma de circunvir esse problema é através da Neuroevolução, que treina uma Rede Neural Artificial através de algoritmos evolutivos de tal forma que ela tenha sucesso no resultado final de suas ações. Nesse artigo, introduzimos um novo ambiente de referência para testar algoritmos de aprendizado de agentes autônomos em jogos eletrônicos, chamado EvoMan, inspirado no jogo de plataforma Mega Man II. Essa plataforma compreende situações de: aprendizado em ambiente estático e dinâmico, aprendizado contínuo de coevolução e aprendizado generalizado. Como experimentos iniciais, aplicamos uma Neuroevolução utilizando Algoritmos Genéticos e o algoritmo NEAT no contexto de coevolução para demonstrar os desafios da plataforma proposta.**

***Index Terms*—neuroevolução, coevolução, jogos eletrônicos.**

I. Introdução

A busca por controladores autônomos é uma tarefa importante da área de Inteligência Artificial, que tem como objetivo a criação de sistemas que tomem decisões automáticas em ambientes incertos. Existem diversas aplicações possíveis para tais agentes, desde indústrias de manufaturas até veículos não tripulados para exploração de ambientes inóspitos.

Um problema dessa área é a falta de ambientes de referência para testar novos algoritmos. Recentemente foram propostos ambientes de teste através de jogos eletrônicos [6][7][8][9]. Nesses ambientes, deve-se criar um algoritmo que aprenda a controlar o jogador principal para atingir objetivos diversos. Esse tipo de ambiente é interessante para projetos dessa natureza, pois é possível simular diferentes níveis de incerteza e diferentes números de controles de entrada.

Além disso, esse ambiente é de interesse prático da indústria de jogos na qual se busca a criação de adversários que se adaptam ao estilo e experiência do jogador, de tal forma a criar um ambiente desafiador, mas sem frustrações. Nesse caso é necessário não apenas que o adversário mude seu comportamento com o tempo, para aprender a combater o jogador, mas também que essa evolução aconteça no tempo certo: se evoluir muito rápido, o desafio levará ao descontentamento do jogador; se for muito lento, o controlador não conseguirá atingir o desafio necessário.

Alguns trabalhos de neuroevolução realizam seus experimentos em ambientes de jogos clássicos. Em [15] os autores utilizam um ambiente de desenvolvimento criado em um software de CAD/CAM (Computer Aided Engineering/Computer Aided Manufacturing), que permite a modelagem de objetos do mundo real e seus mecanismos, como engrenagens etc. Neste trabalho, redes neurais artificiais foram evoluídas para controlar esses modelos realísticos simulados. Nos experimentos, não só as redes para a tomada de decisão do agente são evoluídas, mas ainda o modelo físico simulado do mesmo, através de uma dinâmica de coevolução entre estes.

O objetivo desse trabalho é introduzir um novo ambiente de referência para testar algoritmos de aprendizado de agentes autônomos em um jogo inspirado no Mega Man II [1], aqui denominado EvoMan. Além disso, faremos experimentos iniciais para verificar se tal ambiente de testes não apresenta os mesmos problemas que outros ambientes clássicos (e simplificados), se as incertezas introduzidas ao ambiente não impedem o aprendizado do agente autônomo, e se em um ambiente de coevolução é possível atingir o ponto em que o jogador e o adversário passam a realizar uma queda de braços, tendo uma evolução contínua.

Este artigo está organizado da seguinte maneira: Seção II descreve o ambiente do jogo desenvolvido e utilizado no trabalho, assim como seus parâmetros e configurações; Seção III revisa conceitos de Neuroevolução e trabalhos relacionados; Seção IV descreve a Coevolução e a forma como foi aplicada; Seção V descreve a metodologia dos experimentos que aplicamos, e, finalmente a Seção VI conclui e aponta um direcionamento para trabalhos futuros.

II. Descrição do Ambiente de Testes EvoMan

EvoMan [2] é um jogo em que o personagem pode escolher entre as ações de andar para frente, para trás, pular e atirar, em um ambiente 2D. O personagem principal luta contra oito

[1] https://www.megaman.capcom.com
[2] http://youtu.be/38UjeHfJ1gw

inimigos diferentes, posicionados em extremos da tela no início da luta. Esse jogo é inspirado no jogo Mega Man II, e foi desenvolvido para dar suporte à realização de experimentos de neuroevolução, utilizando a biblioteca de criação de jogos pygame [3], da linguagem de programação Python [4]. Neste jogo, as partes que foram reproduzidas consistem apenas nas lutas contra os oito inimigos principais do Mega Man. A estrutura do ambiente permite realizar diferentes experimentos com a combinação de diferentes modos:

- Jogador humano: em que o jogador é controlado através de um periférico de entrada de dados (i.e., joystick).
- Jogador IA: em que o jogador é controlado por um algoritmo de aprendizado de máquina.
- Inimigo Estático: em que o inimigo utiliza uma estratégia predefinida de ataque baseado em regras. Esse modo simula o comportamento original do jogo Mega Man II.
- Inimigo IA: em que o inimigo é controlado por um algoritmo de aprendizado de máquina.

Nesse artigo utilizaremos as combinações "Jogador IA VS Inimigo Estático"e "Jogador IA VS Inimigo IA"para testar o processo de aprendizado de um controlador capaz de vencer cada um dos oito chefes. A *arena* de combate de cada um dos oito inimigos é definida como uma área retangular podendo ou não conter obstáculos. Os personagens iniciam o jogo com 100 pontos de energia que é decrementado toda vez que um tiro do oponente o atinge ou os personagens entram em contato físico.

O jogo é segmentado em passos de tempo discreto e, a cada passo, o ambiente provê 68 variáveis sobre o estado do jogo, que agem como sensores. A lista a seguir descreve os sensores dos personagens do jogo:

1) Coordenadas do retângulo que envolve cada personagem na tela, totalizando 8 variáveis (4 para cada personagem).
2) *Flag* indicando se o personagem está sob uma superfície, totalizando 2 variáveis.
3) Quantidade de passos de tempo até ser possível atirar novamente, totalizando 2 variáveis.
4) *Flag* indicando que o personagem está atirando, totalizando 2 variáveis.
5) Variáveis de aceleração vertical e horizontal dos personagens, totalizando 4 variáveis.
6) Direção em que cada personagem está olhando, totalizando 2 variáveis.
7) *Flag* indicando que o personagem está sendo atacado naquele instante, totalizando 2 variáveis.
8) As coordenadas dos retângulos envolvendo até 3 tiros do jogador, totalizando 12 variáveis.
9) As coordenadas dos retângulos envolvendo até 8 tiros do inimigo, totalizando 32 variáveis.
10) *Flag* indicando que o inimigo está imune aos ataques do jogador naquele instante, totalizando 1 variável.
11) Contador de passos de tempo do jogo, totalizando 1 variável.

[3]http://www.pygame.org/
[4]https://www.python.org

Alguns dos sensores relacionados aos ataques talvez não sejam úteis para o agente quando jogando contra determinados inimigos, isso porque a quantidade máxima de balas que podem existir simultaneamente na tela é limitada a oito para o inimigo e três para o personagem, mas nem todos utilizam o limite máximo. Além disso, existem ainda alguns sensores que são específicos para determinados inimigos. Quando há em uma iteração do jogo menos balas que o máximo permitido, ou ainda sensores que não se aplicam ao inimigo corrente, os sensores relacionados às balas inexistentes assumem valor zero.

A saída do agente é representada por cinco variáveis de decisão: *esquerda*, *direita*, *atirar*, *pular* e *soltar*. A variável *soltar* é utilizada para diferenciar quando o agente está apertando o botão *pular* repetidamente ou apenas quer aumentar a altura do pulo. A altura do pulo dos personagens é proporcional ao tempo em que o botão é apertado, mas limitado a uma altura máxima.

A interface do jogo com o algoritmo de aprendizado permite a leitura do estado atual do jogo a cada instante de tempo, enviando ao algoritmo todas as variáveis de ambiente, de modo que o jogo então aguarda pela quíntupla de variáveis de decisão, enviada pelo algoritmo de aprendizado. A cada passo de tempo, o algoritmo tem acesso também ao status da energia dos personagens. O jogo termina após a energia de um dos personagens chegar a zero ou até um máximo de 1000 passos de tempo, quando então o desempenho do agente pode ser avaliado. O limite de tempo foi determinado de tal forma que o agente não prolongue a duração do jogo indefinidamente ao aprender somente a ficar fugindo do inimigo.

III. Neuroevolução do Controlador

Nesta seção faremos uma revisão bibliográfica sobre neuroevolução e suas aplicações em jogos. Vamos descrever ainda como o NEAT foi utilizado neste trabalho.

A. Redes neurais

Uma rede neural artificial é um algoritmo de aprendizado de máquina inspirado no funcionamento do cérebro humano e suas características adaptativas[1]. Neste modelo computacional as variáveis de entrada são representadas como neurônios que se relacionam através de pesos não interpretáveis. Cada neurônio age como uma unidade de processamento, e essas conexões valoradas de pesos entre ele representam sinapses [2]. Isso pode ser usado para classificar ambos os alvos qualitativos e quantitativos. Para problemas de aprendizado supervisionado, é comum usar o algoritmo Retropropagação como método de treinamento para a rede encontrar a solução procurada. Todavia, quando a resposta é desconhecida e o objetivo é maximizar/minimizar alguma medida de sucesso, é mais apropriado usar técnicas de aprendizado por reforço, ou algoritmos de otimização caixa preta.

B. Algoritmos genéticos

Um algoritmo genético é um método evolutivo de aprendizado de máquina que aplica os princípios da seleção natural para encontrar soluções de problemas multidimensionais.

Genes são representados como variáveis e, então, cada genoma é um grupo de variáveis assumindo determinados valores. Os genomas que alcançam um melhor *fitness* referente à solução que buscam têm maior probabilidade de sobreviver, cruzar e produzir prole [3]. A prole, que provém dos pais mais adaptados ao ambiente do jogo, sofre mutação, permitindo explorar o espaço de busca das variáveis (genes), e desta forma, possivelmente encontrar soluções melhores através da diversidade. As soluções que têm baixa performance tendem a morrer, e as com alta performance tendem a viver e gerar descendentes similares a elas. A repetição desse processo por um número razoável de gerações permite se aproximar do ótimo global da solução. Isto ocorre através da convergência da população para um indivíduo que se aproxima das caraterísticas que levam à solução ótima.

C. Neuroevolução

Um algoritmo evolutivo é um tipo de método de busca estocástico, baseado-em-população, inspirado pelas ideias evolucionistas de Darwin. Na neuroevolução, em vez de usar algoritmos convencionais de aprendizado, como Retropropagação em modo supervisionado, para treinar uma rede neural, as características das redes neurais como pesos e topologia podem ser codificadas como genomas artificiais que serão evoluídos e selecionados de acordo com sua performance em achar as soluções esperadas [2].

De acordo com a pesquisa de literatura feita em [4], na maior parte dos casos a neuroevolução é usada para evoluir um agente na tarefa de jogar um jogo. É importante notar que existem diferentes tipos de desafios ao se evoluir um agente, e isto pode variar de acordo com o jogo. Alguns destes desafios são: avaliação de estado/ação, geração de conteúdo, seleção de estratégias e seleção de ação direta [4], sendo este último o tipo de desafio do jogo que serviu de ambiente para os experimentos realizados neste artigo.

D. NEAT

Neuroevolução de topologias aumentativas é um método que treina redes neurais artificiais usando algoritmos genéticos que foi proposto por [5] e tem mostrado ótima performance em tarefas complexas do aprendizado. Diferentemente das abordagens tradicionais de neuroevolução, NEAT evolui não apenas os pesos, mas também a topologia da rede incrementalmente. Na primeira geração, as redes têm os neurônios da camada de entrada completamente conectados aos neurônios da camada de saída, e não há camadas intermediárias. Cada rede começa com a estrutura mais simples possível, e desenvolve novos nós e conexões via mutação ao longo das gerações. Essas mudanças estruturais tendem a se manter quando trazem algum benefício para o *fitness* da rede, e redes que sofreram mutações que trazem malefícios tendem a morrer. Assim como mutações que não refletem mudanças significativas podem ser facilmente superadas por outras que sofreram mutação satisfatória. Evoluir a topologia é importante porque não apenas os pesos têm influência na performance da rede, mas também sua estrutura como um todo. Normalmente a tarefa de decidir quantas camadas escondidas e seus números de neurônios vão existir na rede e ainda quais conexões elas terão é feita por trabalho humano, através de testes experimentais. Isso pode consumir bastante tempo, e o NEAT propõe uma alternativa, quando encontra o número adequado de neurônios escondidos e suas conexões valoradas, baseado no benefício que cada item desta configuração de rede traz.

E. Trabalhos relacionados ao NEAT

Tentando criar um controlador para jogar Frogs [6] fez experimentos utilizando o NEAT. Foram usados doze sensores para perceber o estado do ambiente do jogo, e os sensores mediram a proximidade dos objetos à volta do jogador em relação a direções e localizações específicas. Os sapos eram recompensados conforme chegavam mais perto da área objetivada. Cada indivíduo foi testado três vezes no jogo, e o *fitness* final foi a média destes. A aplicação do NEAT para dirigir um carro no jogo TORCS usando entradas mínimas foi discutida por [7]. A implementação destacada no artigo usou para o carro jogador apenas entradas de sensores da linha central da pista na posição corrente do carro, assim como o ângulo relativo à linha central e à velocidade corrente do carro. Usando NEAT [8] tentou e teve sucesso em descobrir uma boa função de *fitness* para um robô jogador de futebol que precisava aprender a carregar uma bola, evitar o oponente e chutar ela contra o gol. Um outra aplicação interessante de NEAT foi a de [9] para jogar Ms.Pacman. Considerando que Ms.Pacman tem que escapar de fantasmas quando eles são ameaças, e pegá-los quando eles são elegíveis (além de comer todas as pílulas em cada level), trataram como um problema de otimização de múltiplos objetivos. Este comportamento multi-objetivo foi alcançado através do que eles chamaram de neurônios de preferência.

F. Configuração do NEAT neste trabalho

Antes de serem providas aos algoritmos, todas as entradas foram normalizadas entre -1 e 1. Os experimentos usaram um algoritmo genético para encontrar a topologia (e seus pesos) da rede neural mais adequada a lutar contra o inimigo e a alcançar o melhor *fitness* possível. As soluções encontradas pelo algoritmo foram tratadas como genomas artificiais, no qual cada um continha a informação que representava uma rede neural (topologia e pesos). Para que os genomas fossem considerados suficientemente parecidos, a medida de distância, demonstrada pela Equação 1, devia estar abaixo de um ponto de corte; para esse trabalho o limiar foi 3.

$$s = (D + E)/n + 0.4\Delta W \tag{1}$$

onde D é a quatidade de genes disjuntos, E é a quantidade de genes em excesso e ΔW é a diferença absoluta média dos pesos.

A reorganização de espécies era feita a cada geração, e quando as espécies eram reorganizadas, antes do cruzamento, um processo chamado *fitness sharing* era plicado ao *fitness* de todos os genomas. Isso era feito dividindo o *fitness* de cada genoma pelo número de genomas da espécie. O total de prole

que era produzido por cada espécie era proporcional à soma de prole calculada para ser produzida por cada genoma da espécie, sendo este valor calculado pelo *fitness* do genoma dividido pelo *fitness* médio de toda a população.

O ponto de corte final das cinco variáveis de decisão (esquerda, direita, tiro, pulo e soltar) foi 0.5.

Diversos testes foram feitos para definir um *fitness* que representasse a qualidade de uma solução. O *fitness* final é definido pela Equação 2.

$$fitness = (100-e)^{\gamma} - (100-p)^{\beta} - ((\sum_{i=1}^{(\vec{t})} 100-p_i)/t)^{\alpha} \quad (2)$$

onde *e*, *p* são as energias do adversário e do jogador respectivamente, que variam de 0 a 100; *t* é a quantidade de passos de tempo utilizada até o término da luta.

O *fitness* deve ser maximizado, aumentando de forma inversamente proporcional à energia do inimigo (primeiro parênteses da equação), e proporcional à energia do jogador (segundo parênteses), sendo penalizado quando o jogador demora muito para terminar a luta, calculando a energia média acumulada do jogador (terceiro parênteses). Quanto maior este for, melhor a performance do jogador em ganhar a luta mais eficientemente. Para este trabalho os valores de γ, β, α foram testados empiricamente e os melhores valores obtidos foram: γ=1, β=2, α=2 para a evolução do jogador e γ=1, β=2, α=3 para a evolução do inimigo.

IV. Coevolução

Nesta seção falaremos sobre coevolução e como esta foi aplicada neste trabalho.

A. A coevolução

Uma adaptação em uma linhagem (predadores) pode forçar a pressão de seleção em outra linhagem (presas), dando impulso a uma contra-adaptação. Se isso for recíproco, uma queda de braços pode ser iniciada [10].

A coevolução é um tipo de dinâmica de aplicação de algoritmos evolutivos. Em vez de evoluir uma população individualmente com um objetivo fixo, duas (ou talvez mais) populações de soluções evoluem simultaneamente, tornando o *fitness* dinâmico de uma para a outra. Isso pode ser feito de duas formas. Na coevolução cooperativa, agentes podem cooperar entre si para ter sucesso na realização de uma determinada tarefa, podendo até haver penalizações para uma solução que não considere o sucesso do grupo como um todo. Já na coevolução competitiva, agentes podem competir para realizar a mesma tarefa, de modo que quanto mais cada um evolui, mais o outro é forçado a buscar novas soluções, sendo comumente aplicada em jogos para problemas do tipo predador versus presa [11].

Coevolução competitiva é um método de inteligência computacional que se mostra promissor, mas ainda tem um número pequeno de casos de sucesso prático [12]. Na tentativa de implementar coevolução competitiva em programas de computador, pesquisadores frequentemente se deparam com alguns problemas limitantes como *cycling*, no qual o melhor indivíduo de alguma geração supera seu inimigo da mesma geração, mas não o competidor de uma geração anterior [12].

B. Trabalhos relacionados a coevolução

Mesmo considerando as dificuldades apresentadas pela evolução, [12] investigou-a através de experimentos com Ms. Pac-man e encontrou algumas evidências de que controladores coevoluídos generalizavam melhor que controladores evoluídos individualmente, levando a uma conclusão de que esse domínio merece mais estudos. Os autores notaram ainda que era significativamente mais fácil evoluir controladores para o Ms. Pac-man do que para os fantasmas, indicando que o sucesso da abordagem pode depender do domínio escolhido.

Em [13], foram aplicadas simultaneamente coevolução competitiva e cooperativa num domínio de predador/presa, o que apresentou sucesso nos resultados através de competições e cooperação de times de predador e presa. A partir disso, comportamentos de alto nível em termos de fuga e ataque emergiram. Ainda [14] criou uma arquitetura robótica para evoluir um comportamento de predador/presa, e observou que os agentes adquiriram comportamentos interessantes como evasão a obstáculos, discriminação de objetos e percepção visual, permitindo concluir que a coevolução competitiva pode ajudar a encontrar estratégias melhores de caça e evasão entre agentes.

C. Coevolução neste trabalho

No presente trabalho, para a evolução competitiva, o *fitness* de cada indivíduo foi construído pela média dos *fitness* de lutas contra o melhor indivíduo da população inimigo e ainda mais quatro indivíduos aleatórios desta mesma população. Outras formas, como lutar contra os melhores indivíduos de N populações anteriores, também foram testadas, mas não mostraram melhora significante na performance.

Como na coevolução as duas populações devem evoluir simultaneamente, é necessário criar algum tipo de chaveamento de predador/presa para o processo de evolução. Esse chaveamento foi feito a cada três gerações, ou seja, a cada três gerações a população-foco sendo evoluída mudava entre jogador e inimigo (predador vira presa e vice versa). Vale ressaltar que esse número de 3 gerações é bem pequeno, dado que normalmente evolui-se uma população ao longo de 100 gerações. Todavia, o número reduzido foi usado para viabilizar a execução dos experimentos. Vale lembrar que quando o jogador sendo evoluído é algum dos inimigos do EvoMan, a dinâmica do contato físico se inverte, de modo que quem perde energia em contato físico com o oponente é sempre o agente sendo treinado.

V. Experimentos

Nesta seção descreveremos a metodologia experimental utilizada. Além disso, será feita uma breve discussão dos resultados obtidos.

Tabela I
TESTE FINAL DE ENERGIA DOS MÉTODOS DE NEUROVOLUÇÃO

Inimigo	GA simples	NEAT	t-Test
FlashMan	0	92	p <0.01
AirMan	87	92	p <0.01
WoodMan	73	100	p <0.01
HeatMan	48	84	p <0.01
MetalMan	97	99	p <0.11
CrashMan	87	54	p <0.01
BubbleMan	63	67	p <0.01
QuickMan	77	88	p <0.01
Média dos métodos	66.5	84.5	p <0.18

Tabela II
TESTE FINAL DE ENERGIA DOS MÉTODOS ESTÁTICO E EVOLUTIVO

Inimigo	NEAT estático	NEAT evolutivo	t-Test
FlashMan	92	0	p <0.01
AirMan	92	74	p <0.01
WoodMan	100	4	p <0.01
HeatMan	84	0	p <0.01
MetalMan	99	94	p <0.28
CrashMan	54	0	p <0.01
BubbleMan	67	22	p <0.01
QuickMan	88	70	p <0.01
Média dos métodos	84.5	33	p <0.01

A. Metodologia

Nos experimentos, três formas diferentes de treinamento foram aplicadas: algoritmo genético simples, NEAT com evolução individual e NEAT com coevolução.

Contra cada um dos oito inimigos separadamente, o agente jogador foi evoluído e comparado através dos métodos de evolução individual e coevolução competitiva. Ambos os experimentos foram feitos utilizando NEAT, porém no caso da evolução individual, um experimento adicional foi feito evoluindo as redes neurais através da aplicação de um algoritmo genético simples.

A evolução individual foi feita através de lutas contra um inimigo estático, sendo testada no final neste mesmo inimigo. Já a coevolução competitiva foi feita através de lutas com um inimigo em evolução, sendo testada no final no inimigo estático também, a fim de permitir comparação. Ao evoluir contra um inimigo dinâmico, as combinações de situações possíveis (estratégias de ataque do inimigo) são muitas, e provavelmente ao longo do número de gerações escolhido o jogador não tenha experienciado todas, sendo desta forma pouco provável que dentre tantas ele conheça justamente a estratégia do inimigo estático. Assim, o inimigo estático se mostra um bom teste de generalização.

As melhores soluções de cada método para cada chefe foram salvas e testadas 10 vezes, apresentando como resultado final a média das repetições e um *t-Test*.

B. Adversário Estático

O adversário estático é o modo natural do jogo, no qual o inimigo tem sempre o mesmo comportamento estático de ataque e defesa. Neste experimento, tentamos encontrar o ótimo global do problema de duas formas. Primeiro por um algoritmo genético de neuroevolução simples, depois pelo NEAT.

Na Tabela I podemos ver que todos os inimigos foram derrotados pelo agente treinado através de algum dos métodos. O método mais eficiente foi o NEAT, que conseguiu matar todos os chefes, apresentando energia média de 84.5, considerando ainda que no caso do inimigo WoodMan, não houve perda de energia. O método de Algoritmos genéticos simples (GA simples) também teve um resultado favorável, mesmo com média mais baixa, em 66.5, embora a diferença não seja significante, tendo falhado em ganhar apenas do Flashman. É interessante notar que contra apenas um dos inimigos (CrashMan) o método de GA simples foi melhor que o NEAT. A diferença das médias entre os testes do GA e do NEAT foram significantes para todos os inimigos, com exceção do MetalMan.

C. Adversário Estático e Evolutivo

O adversário evolutivo é um inimigo que evolui simultaneamente ao jogador, no processo de coevolução. Neste caso, durante a evolução o papel de predador e presa eram invertidos a cada três gerações entre o jogador e o inimigo. Nesta parte do experimento, evoluímos o agente de dois modos: lutando contra um inimigo de comportamento estático; lutando contra um inimigo evolutivo.

Conforme a Tabela II, o método mais eficiente foi o NEAT estático, que conseguiu matar todos os inimigos, apresentando energia média de 84.5. O método NEAT evolutivo teve pior performance, embora ainda razoável, falhando em vencer 3 dos oito chefes, e com média de energia significantemente menor, em 33, tendo sido ainda o NEAT estático melhor que o NEAT evolutivo em todos os inimigos. A diferença das médias entre os testes do estático e do evolutivo foram significantes para todos os inimigos, com exceção do MetalMan. Dessa forma, notamos que a evolução com inimigo estático (evolução individual) teve mais que o dobro da performance do inimigo evolutivo (coevolução competitiva) no domínio de predador/presa do EvoMan. Os gráficos da Figura 1 permitem observar a evolução dos métodos NEAT estático e NEAT evolutivo ao longo das gerações.As séries dos gráficos se referem às medidas de energia dos agentes (inclusive estático) ao final de uma luta. Estas são complementares, de modo que no pior caso o jogador morre e não prejudica em nada o inimigo (energia do jogador em 0 e energia do inimigo em 100), e no melhor caso o jogador mata o inimigo sem se ferir (energia do jogador em 100 e energia do inimigo em 0). A série azul representa o jogador e a série vermelha representa o inimigo, e são referentes à melhor solução de cada geração, devendo estas serem maximizadas e minimizas respectivamente.

Do lado esquerdo temos a evolução dos inimigos com o método estático, e do lado direito com o método evolutivo. Para todos os oito inimigos, podemos observar que há uma forte perturbação nas séries do método evolutivo, enquanto as séries do método estático apresentam razoável tendência. Essa pertubação apresentada demontra a queda de braços da coevolução aplicada. Foi possível encontrar uma solução de aprendizado boa no ambiente dinâmico testado, embora

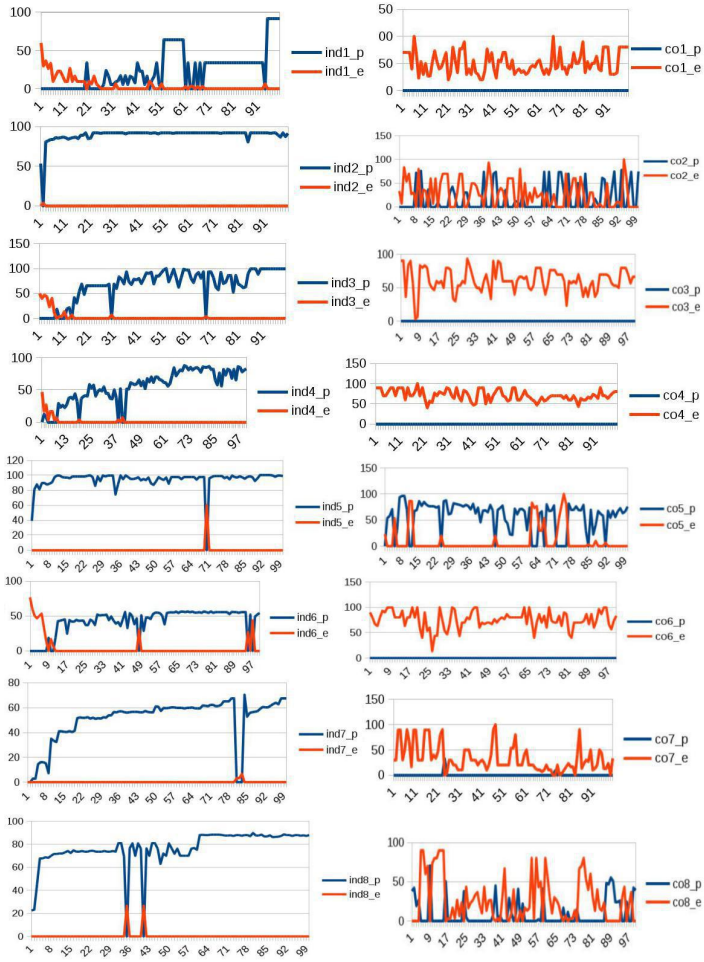

Figura 1. Série da medida de energia do melhor indivíduo de cada geração, média de três lutas de teste para cada inimigo (1-FlashMan, 2-AirMan, 3-WoodMan, 4-HeatMan, 5-MetalMan, 6-CrashMan, 7-BubbleMan e 8-QuickMan), comparando evolução contra inimigo estático (ind) e inimigo evolutivo (co). A abreviação ”p”refere-se a jogador e ”e”a inimigo.

naturalmente a melhor performance tenha sido alcançada com o método estático de evolução individual.

As eventuais quedas repentinas da série do jogador (azul) do método estático se referem aos casos em que a solução encontrada foi a fuga, ferindo parcialmente o inimigo até a luta expirar. Embora esse tipo de comportamento tenha sido desencorajado na construção do *fitness*, ainda ocorreram alguns casos. Nestes casos, para inclusão na série, a medida do jogador foi zerada, usando somente a medida do inimigo ao final da luta.

VI. Conclusão e trabalhos futuros

Este trabalho propôs um novo ambiente de aprendizado de agentes para jogos eletrônicos baseado no jogo Mega Man II, aqui chamado de EvoMan. Esse ambiente permite diversas combinações distintas de testes de diferentes dificuldades.

Para fins de teste um agente foi evoluído individualmente em ambientes estático e dinâmico através de métodos de neuroevolução para o jogador aprender a lutar contra os oito inimigos. Em ambos os ambientes o agente evoluído teve sucesso em aprender tarefas como esquivar-se de objetos e agentes que podem trazer malefício ao seu *fitness*, assim como atacar o inimigo de forma apropriada para feri-lo.

No ambiente estático usa-se uma estratégia especialista, em que a população de jogadores é evoluída tendo sempre um único e mesmo objetivo, que é alvo de teste no final. O método de coevolução competitiva usa uma segunda estratégia, que se apoia na dinâmica exploratória da coevolução, em que na busca por novas estratégias que permitam vencer o inimigo atual evoluído, o jogador é forçado a explorar situações que antes não tinha experienciado. Para o domínio de aplicação do EvoMan, o método de evolução em ambiente estático se mostrou bastante superior ao método de coevolução competitiva.

Como trabalhos futuros, algumas melhorias serão feitas no ambiente de forma a flexibilizar o processo de introdução de novos agentes e introdução de novos inimigos. Além disso, serão feitos testes de outras combinações de desafios tais quais: generalização do aprendizado, em que o agente aprende a lutar contra n adversários e seu desempenho é mensurado na luta contra os adversários restantes e evolução incremental, em que o adversário evolui gradativamente de acordo com o desempenho de um jogador humano.

Referências

[1] Dreiseitl, Stephan, and L. Ohno-Machado, *Logistic regression and artificial neural network classification models: a methodology review*, Journal of biomedical informatics 35.5 (2002): 352-359.

[2] Floreano, Dario, P. Dürr, and C. Mattiussi, *Neuroevolution: from architectures to learning*, Evolutionary Intelligence 1.1 (2008): 47-62.

[3] Holland, John H, *Genetic algorithms*, journal Scientific american volume 267 number 1 pages 66-72, 1992.

[4] Risi, Sebastian and J. Togelius, *Neuroevolution in Games: State of the Art and Open Challenges*, 2014.

[5] Stanley, O. Kenneth, and R. Miikkulainen, *Evolving neural networks through augmenting topologies*, Evolutionary computation 10.2 (2002): 99-127.

[6] A. Davis and W. Jake, *Developing Frogger Player Intelligence Using NEAT and a Score Driven Fitness Function*

[7] I. James and A. Mohammed, *Training Bots to Drive in TORCS using NEAT*

[8] P. Thiha, *Extending Robot Soccer Using NEAT*, CS81 Final Projects (2009).

[9] J. Schrum and M. Risto, *Evolving multimodal behavior with modular neural networks in ms. pac-man.*, Proceedings of the 2014 conference on Genetic and evolutionary computation. ACM, 2014.

[10] R. Dawkins, and J. Krebs, *Arms races between and within species.*, Proceedings of the Royal Society of London. Series B. Biological Sciences 205.1161 (1979): 489-511.

[11] Pollack, B. Jordan, A. Blair, and M. Land, *Coevolution of a backgammon player*, Artificial Life V: Proc. of the Fifth Int. Workshop on the Synthesis and Simulation of Living Systems. Cambridge, MA: The MIT Press, 1997.

[12] Cardona, A. Borg, J.Togelius, and M. Nelson, *Competitive coevolution in ms. pac-man*, Evolutionary Computation (CEC), 2013 IEEE Congress on. IEEE, 2013.

[13] Rawal, Aditya, P. Rajagopalan, and R. Miikkulainen, *Constructing competitive and cooperative agent behavior using coevolution*, Computational Intelligence and Games (CIG), 2010 IEEE Symposium on. IEEE, 2010.

[14] Gebeová, Gyongyike, et al. *The experimental study of the competitive co-evolution using Predator-Prey tasks*, Intelligent Technologies: Theory and Applications: New Trends in Intelligent Technologies 76 (2002): 245.

[15] J. Pollack, H. Lipson, P. Funes, S. Ficici, G. Hornby, *Coevolutionary Robotics*, Proceedings of the First NASA/DoD Workshop on. IEEE, 1999.