UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Novidade na Emergência da Comunicação em Robôs Sociais

Filipe Gomes Marques dos Santos

Mestrado em Informática

Dissertação orientada por:
Prof. Doutor Paulo Jorge Cunha Vaz Dias Urbano

2025

Agradecimentos

Gostaria de começar por agradecer ao meu orientador, Paulo Urbano, por me ter dado a oportunidade de trabalhar neste projeto, por me ter ajudado e incentivado ao longo de todo o processo. A sua orientação foi inestimável para a elaboração desta tese e para o meu crescimento como investigador. Agradeço também à FCUL por me ter dado as capacidades de conhecimento que tenho hoje, ao LASIGE, ao departamento de informática e aos seus professores por me terem proporcionado um ambiente académico de apoio e acesso a recursos essenciais. A título pessoal, gostaria de agradecer à minha família pelo apoio incondicional e pela compreensão ao longo deste percurso. Um agradecimento especial pela sua paciência e encorajamento.

Resumo

A comunicação é um processo fundamental usado na troca de informação, crucial na resolução de problemas, e intimamente ligado à evolução . No campo da inteligência artificial, na área da computação evolutiva e na robótica evolutiva, o uso da comunicação tem provado ser grande contributo para o desempenho de modelos evoluídos para problemas que requerem a coordenação de multi-agentes. A Novidade (*Novelty*) como critério de avaliação da evolução, é uma métrica recente que, substituindo o *Fitness*, pode facilitar atingir a solução ótima em problemas complexos em que o uso do *Fitness* induz as populações a cair em ótimos locais, levando ao estagnamento da evolução, para além de reduzir significativamente o custo de execução em relação a outros métodos de evolução como a evolução incremental, que poderá vir a substituir. Neste trabalho propomos a hipótese que o uso da Comunicação em agentes homogéneos e a Novidade em modelos de neuro-evolução pode melhorar consideravelmente os resultados obtidos por estes modelos, comparativamente a modelos centralizados heterogéneos com evolução incremental, e por conseguinte proporcionar avanços em algoritmos evolutivos e na robótica evolutiva. Nesse sentido demonstramos o impacto destas novas soluções em sistemas multi-agentes, no conhecido domínio do problema da captura com robôs sociais, com predadores e presas em ambientes 1D e 2D.

Palavras-chave: Evolução Neuronal, Aprendizagem Reforçada, Redes Neuronais Artificiais, Novidade, Comunicação

Abstract

Communication is a fundamental mechanism for information exchange, playing a central role in problem solving and closely intertwined with evolutionary processes. In artificial intelligence, particularly within evolutionary computing and evolutionary robotics, communication has been shown to significantly enhance the performance of evolved models in tasks that require coordinated behaviour among multiple agents. Novelty-based evaluation is a relatively recent evolutionary metric which, by replacing traditional fitness functions, can mitigate premature convergence to local optima and reduce evolutionary stagnation in complex problem domains. In this work, we propose that incorporating both communication and novelty-driven methods into neuro-evolutionary models can substantially improve their performance, thereby contributing to advances in evolutionary algorithms and evolutionary robotics. To support this hypothesis, we analyse the impact of these approaches in multi-agent systems using the well-known predator–prey capture problem with social robots in one-dimensional and two-dimensional environments.

Keywords: Neural Evolution, Reinforced Learning, Artificial Neural Networks, Novelty, Communication

Conteúdo

Lista de Figuras **13**

Lista de Tabelas **15**

1 Introdução **1**

1.1 Motivação 1

1.2 Objetivos 2

1.3 Estrutura do documento 3

2 Contexto **5**

2.1 Robótica Evolucionária 5

2.2 Aprendizagem Reforçada (*Reinforced Learning* [RL]) 5

2.3 ANNs (*Artificial Neural Networks*) 6

2.4 Comunicação 8

2.5 Aptidão (*Fitness*) 8

2.6 Novidade 8

3 Trabalho Relacionado **10**

3.1 Evolução Neuronal (*Neural-Evolution*) 10

3.2 NEAT 12

3.3 Evolução Incremental (*Incremental Evolution*) 15

3.4 Domínio de captura com predadores e presas (*Capture Domain*) 15

3.5 Procura por novidade (*Novelty search*) 17

4 Experiências de neuro-evolução **23**

4.1 1ª experiência 23

4.1.1 Metodologia 23

4.1.2 Resultados das experiências baseadas no Objetivo 25

4.1.3 Avaliação dos resultados baseados no objetivo 28

4.1.4 Metodologia da novidade 28

4.1.5 Resultados da novidade 29

4.1.6 Avaliação de resultados da 1ª experiência 29

4.2 2ª experiência . . . 30
4.2.1 Metodologia . . . 30
4.2.2 Resultados das experiências baseadas no objetivo . . . 33
4.2.3 Avaliação dos resultados baseados no objetivo . . . 34
4.2.4 Metodologia da novidade . . . 36
4.2.5 Primeiros resultados da novidade . . . 37
4.2.6 Segundos resultados . . . 38
4.2.7 Experiência adicional . . . 38
4.3 3ª experiência . . . 42
4.3.1 Metodologia . . . 42
4.3.2 Resultados baseados no objetivo . . . 42
4.3.3 Avaliação dos resultados baseados no objetivo . . . 43
4.3.4 Metodologia da novidade . . . 44
4.3.5 Resultados da novidade . . . 45
4.3.6 Avaliação de resultados da novidade . . . 45
4.4 Conclusões dos resultados das experiências baseadas no objetivo . . . 46
4.5 Avaliação da novidade . . . 46

5 Conclusões **49**
5.1 Conclusão . . . 49
5.2 Trabalho futuro . . . 50

Bibliografia **54**

Índice **55**

A Funções utilizadas nas experiências efetuadas **56**

B Classe que define os agentes usados nas experiências **69**

C Outro código usado **71**
C.1 Movimento presa segunda experiência . . . 71
C.2 Função de cálculo da novidade . . . 71
C.3 Avaliação dos genomas . . . 72
C.4 Anexos . . . 73
C.4.1 Anexo A . . . 73
C.4.2 Anexo B . . . 75
C.4.3 Anexo C . . . 77

Lista de Figuras

2.1 Exemplo de rede neuronal artificial de população homogénea 7
2.2 Exemplo de rede neuronal artificial heterogénea 7

3.1 Exemplo de esquema do processo neuro-evolucionário 11
3.2 Exemplo esquemático de rede evoluída com NEAT. As setas representam conexões com pesos ilustrados com espessura da linha, conexões vermelhas não estão ativas, verdes estão, os círculos indicam nós (entradas, ocultos e saída). 14
3.3 Exemplo esquemático de um labirinto em que o circulo a azul é a posição inicial do agente a ser treinado e a estrela a amarelo é o objetivo. 20

4.1 Ambiente de execução 1ª Experiência em que Dim=5. A presa é imóvel durante toda a duração da experiência 24
4.2 Topologia inicial da ANN para os predadores com comunicação predefinida . . . 25
4.3 Topologia inicial da ANN para os predadores com com evolução da comunicação 25
4.4 Simulação passo a passo do comportamento evoluído no modelo com comunicação livre 27
4.5 ANN do melhor indivíduo do modelo com comunicação predeterminada com camada oculta desenvolvida 27
4.6 ANN do melhor indivíduo do modelo com comunicação livre com camada oculta desenvolvida 27
4.7 Gráfico com todos os resultados da 1ª experiência 29
4.8 Ilustração do domínio de captura usado em que os predadores representados por círculos estão a cercar a presa representada por um elipse 30
4.9 Ambiente de execução com partições numeradas 32
4.10 Resultados da 2ª experiência 34
4.11 ANN resultante do modelo homogéneo comunicação predefinida pM 34
4.12 ANN resultante do modelo homogéneo comunicação livre pM 34
4.13 ANN resultante do modelo heterogéneo comunicação predefinida pM 35
4.14 ANN resultante do modelo heterogéneo 5o comunicação predefinida pM 35
4.15 Resultados da 2ª experiência novidade 1 38
4.16 Resultados da 2ª experiência novidade 2 39
4.17 Resultados da experiência novidade bónus 1 avaliação 40

4.18 Resultados da experiência novidade bónus 9 avaliações 41
4.19 ANN resultante do modelo homogéneo comunicação predefinida pM 43
4.20 ANN resultante do modelo homogéneo comunicação livre pM 43
4.21 ANN resultante do modelo heterogéneo comunicação predefinida pM 44
4.22 ANN resultante do modelo heterogéneo 5o comunicação predefinida pM 44

Lista de Tabelas

4.1 Resultados da primeira experiência de evolução *objective based* com Dim=10 com 10 ensaios (E1..E10) 26
4.2 Resultados da primeira experiência evolução *novelty based* com Dim=10 com 10 ensaios (E1..E10) 29
4.3 Parâmetros da 2ª experiência baseada no objetivo 33
4.4 Resultados da 2ª experiência 33
4.5 Parâmetros da 2ª experiência novidade 37
4.6 Resultados da 2ª experiência novidade, segunda abordagem 37
4.7 2º Resultados da segunda experiência novidade arquivo comportamentos aleatórios 39
4.8 Resultados da experiência novidade adicional gens/capturas 40
4.9 Resultados da 3ª experiência 43
4.10 Parâmetros da 3ª experiência novidade 1 avaliação e 9 avaliações 45
4.11 Resultados da 3ª experiência de novidade, 1 avaliação 45
4.12 Segundos resultados da 3ª experiência da novidade, 9 avaliações 46

Capítulo 1

Introdução

1.1 Motivação

O domínio da captura e fuga, predador e presa no espaço, tem sido amplamente estudado no contexto de algoritmos evolucionários, devido à sua complexidade e à necessidade de cooperação entre agentes. Este problema consiste num ambiente com uma ou mais presas e um ou mais predadores, onde os predadores se deslocam no espaço tentando capturar as presas, enquanto estas procuram escapar. As tarefas de perseguição e evasão são particularmente relevantes por serem omnipresentes no mundo natural, oferecerem um objetivo claro que permite medir o sucesso com precisão, e possibilitarem a análise e visualização das estratégias que evoluem. Estas tarefas geralmente não podem ser resolvidas com técnicas de aprendizagem supervisionada tradicionais, como a retro-propagação, uma vez que as decisões corretas ou ótimas em cada instante não são conhecidas, e o desempenho apenas pode ser avaliado após a tomada de várias decisões ao longo do tempo. São, por isso, necessários algoritmos mais complexos, capazes de aprender sequências de decisões com base em reforço esparso, bem como de coordenar o comportamento com o ambiente, com outros agentes cooperantes e com adversários [1].

Com o objetivo de garantir cooperação entre os agentes deste domínio, surgiu a comunicação como um método promissor, permitindo a coordenação, especialmente entre modelos de agentes homogéneos, através do envio e receção de sinais arbitrários definidos por cada modelo. Estes sinais asseguram comportamentos de equipa vantajosos para alcançar soluções ótimas em problemas multi-agente [2]. O uso de modelos neuronais no treino de agentes heterogéneos, isto é, diferenciados, com recurso à evolução incremental, revelou-se eficaz na promoção de comportamentos coordenados e sofisticados [1]. Neste trabalho, propomos uma abordagem alternativa, baseada em agentes homogéneos, com comunicação evoluída livre embutida na rede neuronal. Mesmo treinados com a mesma rede, estes agentes são capazes de apresentar comportamentos diferenciados conforme o contexto.

Para abordar este tipo de problemas, recorre-se aos algoritmos evolucionários, que são modelos probabilísticos aplicados a populações de agentes ou controladores, inspirados no processo de evolução biológica. Cada agente ou modelo é avaliado em função da sua proximidade relativamente ao objetivo desejado, medida por uma métrica designada por *Fitness* ou aptidão. A evolução

começa com uma população de indivíduos (modelos) que enfrentam o ambiente durante um número definido de gerações. Em cada geração, os indivíduos com melhor *Fitness* são selecionados, sofrendo mutações aleatórias que geram novos comportamentos. No final, é escolhido o modelo que obtém o melhor desempenho na tarefa.

Contudo, o uso de *Fitness* em tarefas complexas levanta várias limitações, como a facilidade com que a evolução estagna em ótimos locais e a dificuldade de encontrar soluções intermédias eficazes. Uma das soluções propostas foi a evolução incremental, que consiste em treinar os modelos progressivamente em problemas de dificuldade crescente, até estes conseguirem resolver a tarefa final pretendida [3]. No entanto, este método torna o processo de desenvolvimento dos modelos excessivamente supervisionado e controlado.

Mais recentemente, surgiu uma abordagem alternativa baseada na Novidade (*Novelty Search*), que substitui a métrica de *Fitness* por uma medida de diversidade comportamental. Em vez de recompensar a proximidade ao objetivo, esta abordagem recompensa comportamentos diferentes dos já observados na população. A novidade é medida pela dispersão — a distância entre o indivíduo e os seus vizinhos comportamentais mais próximos — promovendo a descoberta de comportamentos inovadores que, eventualmente, conduzem à solução ótima [4]. Assim, a Novidade surge como uma alternativa promissora à *Fitness* e à evolução incremental, permitindo a exploração de uma gama mais vasta de comportamentos e estratégias úteis.

Deste modo este trabalho explora o uso de redes neuronais evolutivas, com comunicação evoluída, aplicadas a populações homogéneas de agentes artificiais, comparando o seu desempenho com abordagens heterogéneas já exploradas, no cenário complexo e cooperativo do domínio da captura e fuga. Utiliza-se a Evolução Neuronal como algoritmo evolucionário e analisa-se o impacto da métrica de Novidade na superação das limitações associadas à *Fitness* e à evolução incremental, no treino de modelos capazes de encontrar soluções eficazes para problemas multi-agente. Em relação à plataforma de execução, a escolha entre simulação e ambiente físico real tem implicações críticas. As simulações oferecem custo reduzido, rapidez de iteração, ausência de desgaste físico e flexibilidade para testes de parâmetros extremos, mas têm limitações como o risco de desenvolver soluções que não generalizam para o mundo real devido ao *reality gap*, a diferença entre o modelo simulado e a física real (atrito, ruído sensorial, atrasos, etc.). O uso de robôs reais, embora dispendioso, permite testar comportamentos em ambientes com estocasticidade realista, o que é crucial para validar soluções aplicáveis fora do domínio académico. Alguns trabalhos (por exemplo, Jakobi, 1997) propõem estratégias como o uso de *noise injection* nas simulações para reduzir esse *gap* [5].

1.2 Objetivos

A abordagem proposta neste trabalho recorre a modelos neuronais evoluídos aplicados a agentes homogéneos com comunicação livre emergente. Esta abordagem contrasta com soluções anteriores que dependem de populações heterogéneas para alcançar comportamentos complexos. A inovação reside em testar a hipótese de que uma população homogénea, mas dotada de capaci-

dade de comunicar, pode atingir níveis comparáveis ou superiores de desempenho coletivo, sem necessidade de especialização estrutural inicial. Testar a sua integração neste novo enquadramento poderá revelar novos caminhos para diferenciação funcional em populações inicialmente idênticas. [1]. Propomos uma abordagem diferente de treino de agentes homogéneos com comunicação evoluída que, como alternativa ao modelo de agentes heterogéneos especializados e diferenciados, também permite uma diferenciação no comportamento dos agentes num dado contexto, apesar de treinados com a mesma rede neuronal e de serem treinados como agentes idênticos, a comunicação especifica, dado o contexto, a função de cooperação de cada um. A aprendizagem orientada a objetivos, muito comum em sistemas com agentes heterogéneos, raramente foi aplicada a populações homogéneas. Este trabalho por isso explora a hipótese do uso de modelos neuronais numa população homogénea com comunicação livre evoluída (fazendo parte da rede neuronal), que também permita uma diferenciação no comportamento dos agentes num dado contexto, apesar de treinados com a mesma rede neuronal, garantindo especialização contextual em que cada um agente assume uma função, comparando-o com a abordagem de rede neuronal numa população heterogénea já explorados[1], num cenário de equipa no bem conhecido e complexo problema de multi-agentes, o domínio da captura e fuga, predador e presa no espaço [6], usando agentes artificiais, onde é impossível só a iniciativa individual para a solução ser obtida, usando Evolução Neuronal como algoritmo evolucionário adotado. A Novidade também é explorada como alternativa emergente na procura de uma nova solução que remedeie os problemas de estagnação da evolução por ótimos locais, e remedeie também os problemas encontrados quando se aplica *Fitness* e Evolução Incremental neste espaço de problemas multi-agente, e treine modelos capazes de achar a solução. Este trabalho é todo desenvolvido em cenários simulados desprovidos de robôs reais.

1.3 Estrutura do documento

Este documento está organizado em quatro capítulos. O capitulo 2 aborda os conceitos principais deste trabalho, *ANNs*, *ANNs* para agentes Homogéneos e *ANNs* para agentes Heterogéneos, Comunicação, Aptidão(*Fitness*), Novidade (*Novelty*), Aprendizagem Reforçada e Robótica Evolucionária. O capitulo 3, contextualiza os principais estudos na área deste projeto, desde contributos em algoritmos evolucionários, avanços no campo da evolução neuronal, com evolução incremental, novidade, estudos que usaram e testaram tecnologia NEAT para o desenvolvimento de redes neuronais evolucionárias e progressos no domínio de problemas de perseguição predadores presa. O capitulo 4 trata de delinear em maior detalhe como os objetivos deste trabalho são abordados e os seus principais contributos, testar o NEAT, evolução com modelos de agentes heterogéneos vs modelos de agentes homogéneos com comunicação, o papel da comunicação e a novidade e evolução baseada na novidade vs evolução baseada no objetivo e evolução incremental. O capitulo 5 termina o trabalho com a análise do que é retirado dos resultados obtidos no capitulo anterior e propostas para continuação do trabalho aqui feito no âmbito da novidade e comunicação em modelos neuro-evolucionários de agentes homogéneos no domínio de perseguição de predadores e presa.

Capítulo 2

Contexto

2.1 Robótica Evolucionária

Robótica evolucionária consiste na aplicação de princípios evolucionários, como a seleção, herança e variação a robôs com inteligência, para a criação automática de robôs [7], conferindo-lhes maior versatilidade, robustez e adaptabilidade [8]. Consiste sobretudo na aplicação de modelos e algoritmos neuro-evolucionários a robôs físicos na execução das mais variadas tarefas. Uma das ideias centrais de robótica evolucionária reside na forma como é considerada toda a morfologia e características do robô, pois estes são o verdadeiro limite das capacidades do mesmo.

2.2 Aprendizagem Reforçada (*Reinforced Learning* [RL])

A RL é um paradigma de aprendizagem em que um agente aprende com as interações entre um ambiente desconhecido e ele próprio [9]. A interação pode ser descrita da seguinte forma. O agente observa um estado $s_t^{\rightarrow}$ e toma uma ação no espaço de tempo t. Subsequentemente, faz uma transição e observa um novo estado $s_{t+1}^{\rightarrow}$, enquanto recebe um *feedback* numérico de recompensa r_t do ambiente. Esse estado observado transição é tratado como uma amostra. Muitas vezes, a RL é modelada como um Processo de Decisão Markov (MDP) que contém um espaço de estados S, um espaço de ações discretas A, uma função de probabilidade de transição P, uma função de recompensa R e um fator de desconto y [10]. A RL tem como objetivo procurar uma política ótima que possa maximizar a recompensa acumulada a longo prazo num horizonte infinito. À semelhança de muitos trabalhos existentes, é considerada a política estocástica $\pi : S \times A \rightarrow [0, 1]$. Assim, a recompensa acumulativa a longo prazo, que um agente pode obter seguindo qualquer política π, é definida

$$J^{\pi} = V^{\pi}(s^{\rightarrow}) = E_{\pi}[\sum_{t=0}^{\infty} Y^t r_t | s_0^{\rightarrow} = s^{\rightarrow}],$$

em que V π é designada por função estado-valor. Representa as recompensas acumulativas esperadas a partir de qualquer estado $s_0^{\rightarrow}$, seguindo a política π. Em alternativa, podemos definir a função ação-valor Q π para representar as recompensas acumulativas esperadas da realização da

ação **a** no estado **s**, ou seja,

$$Q^{\pi}(s \rightarrow, a) = E_{\pi}[\sum_{t=0}^{\infty} Y^{t} r_{t} | s_{0}^{\rightarrow} = s^{\rightarrow}, a_{0} = a].$$

Assim, podemos estabelecer o objetivo da RL que é aprender a política ótima, ou seja,

$$\pi^{*} = argmax\pi V^{\pi}(s^{\rightarrow}).$$

2.3 ANNs (*Artificial Neural Networks*)

O termo Redes Neuronais Artificiais refere-se a um grafo de nós ligados entre si por conexões, em que cada ligação possui um peso específico. O nó neuronal funciona como um operador de limiar, permitindo a passagem do sinal apenas depois de aplicada uma determinada função de ativação. De certa forma, assemelha-se ao modo como os neurónios no cérebro estão organizados. Normalmente, o processo de treino de uma rede neuronal artificial consiste em selecionar os valores adequados dos pesos para todas as ligações da rede. Assim, as redes neuronais artificiais podem aproximar qualquer função e são consideradas aproximadores universais, como estabelece o Teorema da Aproximação Universal [11]. As Redes Neuronais Artificiais (RNA) são um sub-campo da aprendizagem automática no domínio de investigação da inteligência artificial. A investigação sobre o desenvolvimento das RNA começou depois de McCulloch e Pitts (1943) [12] terem proposto um modelo matemático da atividade neuronal no cérebro e de Hebb (1949) [13] ter criado um mecanismo de aprendizagem baseado no reforço para explicar a aprendizagem no cérebro humano. Rosenblatt (1958) [14] criou então um modelo computacional de elementos de processamento do cérebro chamados percetrões e a investigação sobre as RNA começou. O objetivo da investigação sobre as RNA é desenvolver sistemas de aprendizagem automática baseados num modelo biológico do cérebro, especificamente a atividade bio-eléctrica dos neurónios no cérebro. A rede neuronal artificial é um sistema de processamento adaptativo, cuja essência é a transformação da rede e o comportamento dinâmico de funções de processamento de informação distribuídas em paralelo, e em diferentes graus e níveis de imitação do processamento. Está envolvida em vários domínios da neuro ciência, da ciência do pensamento, da inteligência artificial, da informática e de outras áreas interdisciplinares. Uma rede neural artificial (ou simplesmente rede neural) é constituída por uma camada de neurónios (ou nós, unidades) de entrada, várias camadas ocultas de neurónios e uma camada final de neurónios de saída [15]

Redes neuronais de populações homogéneas: Redes neuronais de populações homogéneas seguem o conceito de evolução da rede em que cada agente é regido por uma rede neuronal que é igual entre eles, logo evoluem individualmente começando pelo mesmo ponto de partida. 2.1

Redes neuronais de populações heterogéneas:
Redes que regem vários agentes de uma vez evoluem em equipa visando o desenvolvimento do comportamento de todos os agentes com uma só rede neuronal, ou redes distintas para cada agente que permitem diferenciação de comportamentos entre agentes. 2.2.

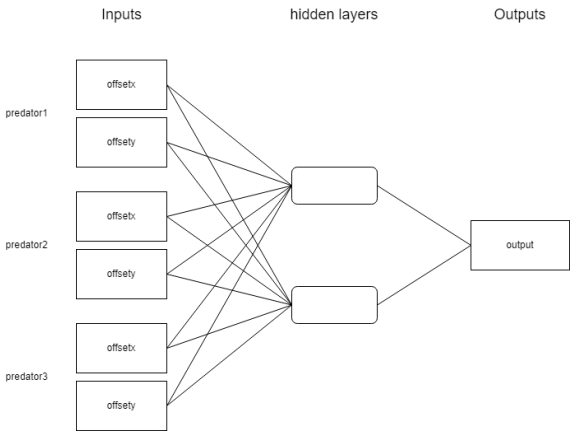

Figura 2.1: Exemplo de rede neuronal artificial de população homogénea

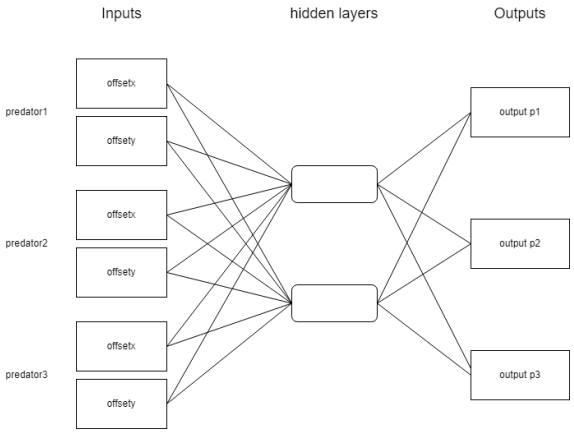

Figura 2.2: Exemplo de rede neuronal artificial heterogénea

2.4 Comunicação

Comunicação tem sido usada como uma potente ferramenta na área de algoritmos evolucionários por permitir maior coordenação entre agentes na procura de objetivos em sistemas multi-agentes [2]. Há exemplos de uso da comunicação com evidentes melhorias resultantes nos modelos que a usam em sistemas multi-agentes [16, 17]. Esta comunicação pode se dar de forma livre, podendo assumir diferentes formas, o que lhe confere uma maior versatilidade quando aplicada a todo o tipo de problemas orientados a um objetivo [2]. comunicação pode ser altamente benéfica, ou mesmo crucial, para a resolução de determinadas tarefas. No entanto, o custo da comunicação, como o gasto de energia na sinalização deve ser tido em conta. De facto, mesmo em domínios em que a comunicação contribui para a resolução de uma tarefa, as características comunicativas podem não evoluir se envolverem um custo significativo [18]. Em vez disso, podem evoluir outros tipos de estratégias de cooperação, dependendo da natureza da tarefa, da densidade da população e da disponibilidade de recursos.

2.5 Aptidão (*Fitness*)

Métrica que permite avaliar a proximidade, num dado problema, do resultado à solução. Em geral em algoritmos evolucionários, a aptidão de uma população vai aumentando com o tempo pela recombinação e mutação de indivíduos e pela seleção dos indivíduos com maior aptidão ao longo de gerações [19]. Em muitos casos com o uso desta métrica, problemas como o da melhor seleção de pressão à população, o problema do ótimo local, ou do desvio genético [19] tornam este método insuficiente na procura da solução ótima aos problemas propostos.

2.6 Novidade

Nova técnica evolucionária que substitui o modelo da aptidão (*Fitness*) baseado no objetivo, por um método baseado na novidade dos indivíduos isto é, selecionar por preferência os indivíduos da população com novos e diferentes comportamentos [20]. A Novidade é no fundo uma métrica que caracteriza a distância comportamental de um individuo relativamente à população calculando a dispersão num ponto. A dispersão pode ser determinada a partir da média das distâncias dos k-vizinhos mais próximos ao ponto, se for elevado então a dispersão é elevada e vice-versa [20].

Capítulo 3

Trabalho Relacionado

3.1 Evolução Neuronal (*Neural-Evolution*)

A evolução-neuronal é uma técnica de aprendizagem reforçada que utiliza algoritmos evolutivos para otimizar a estrutura e os pesos das redes neuronais artificiais. O seu objetivo é conceber automaticamente redes neuronais adequadas a uma tarefa específica, como controlar um robô ou jogar um jogo, selecionando e combinando iterativamente as arquiteturas de rede com melhor desempenho.
À semelhança da seleção natural, que é orientada apenas pelo *feedback* do sucesso reprodutivo, a neuro evolução é guiada por uma medida do desempenho global [21]. Enquanto os algoritmos de aprendizagem de redes neuronais artificiais mais comuns funcionam através de aprendizagem supervisionada e, por conseguinte, dependem de um corpus rotulado de pares de entrada-saída, uma das principais vantagens da neuro-evolução é o facto de permitir a aprendizagem mesmo quando esses corpora não estão disponíveis, com base apenas em *feedback* esparso. Por exemplo, nos jogos, no controlo de veículos e na robótica, as ações ótimas em cada momento nem sempre são conhecidas; só é possível observar como uma sequência de ações funcionou, por exemplo, resultando numa vitória ou numa derrota no jogo. A neuro-evolução permite encontrar uma rede neuronal que otimiza o comportamento apenas com esse *feedback* esparso, sem informação direta sobre o que deve fazer exatamente. o esquema 3.1 de seguida exemplifica o processo de evolução neuronal:

Cada genoma corre o problema x ensaios, é guardado o comportamento dos agentes controlados pelo genoma, no final de cada geração é calculado o *Fitness* de cada genoma comparando o comportamento entre genomas, é feita a seleção de acordo com *Fitness score* dos genomas para avançar para a próxima geração, e isto é feito por n gerações até atingir o limite de gerações definido ou achada a solução.

Ao longo das últimas décadas surgiram muitos métodos de treino de redes neuronais artificiais, mas o mais popular foi o proposto por Geoffrey Hinton, baseado na retro-propagação do erro e na descida do gradiente. Apesar do sucesso no treino de redes profundas, este método tem limitações, como a necessidade de grandes quantidades de dados, dificuldades na transferência de conhecimento para outros domínios e arquiteturas de rede fixas, muitas vezes ineficientes.

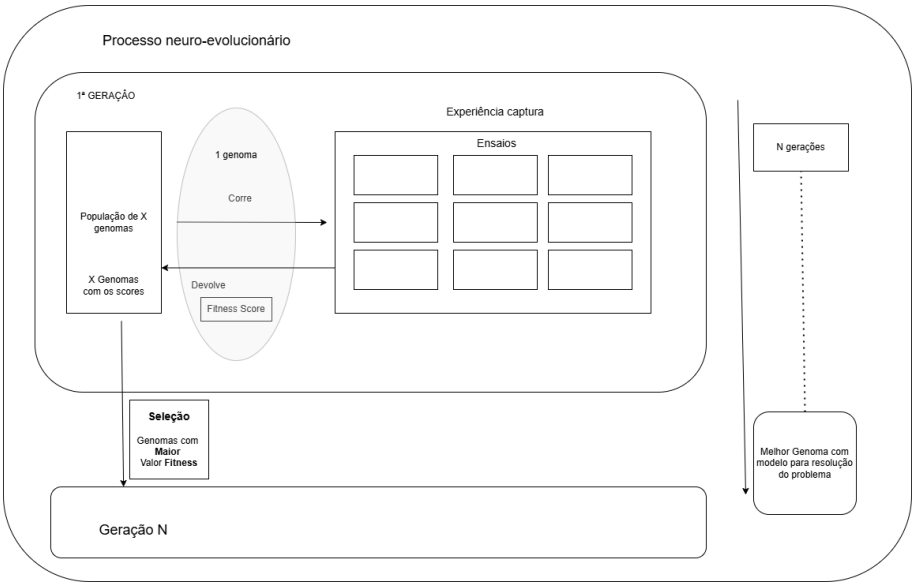

Figura 3.1: Exemplo de esquema do processo neuro-evolucionário

Em alternativa, os algoritmos evolutivos oferecem uma abordagem promissora, inspirada na teoria de Darwin. Estes métodos utilizam operadores genéticos, como mutação e cruzamento, para evoluir populações de redes neuronais, permitindo criar arquiteturas otimizadas e eficientes. As mutações garantem diversidade genética e evitam mínimos locais, enquanto os cruzamentos permitem combinar boas soluções de diferentes indivíduos.

Existem duas formas principais de codificação genética: direta e indireta. A codificação direta, onde os pesos das ligações são representados explicitamente no genótipo, é a base do algoritmo NEAT (*NeuroEvolution of Augmenting Topologies*), desenvolvido por Kenneth Stanley. O NEAT é particularmente eficaz em tarefas de aprendizagem por reforço e no controlo de agentes artificiais, embora esteja limitado a redes de dimensão moderada.

Outros conceitos estudados incluem a co-evolução, onde populações evoluem em interação (competitiva, cooperativa ou comensalista), e a modularidade e hierarquia, inspiradas na organização do cérebro humano, que permitem criar redes mais adaptativas e escaláveis.

3.2 NEAT

O algoritmo NEAT (NeuroEvolution of Augmenting Topologies) foi desenvolvido por Kenneth O. Stanley com o objetivo de evoluir redes neuronais artificiais complexas de forma progressiva e eficiente. A sua principal característica é a complexificação gradual, isto é, a evolução começa com populações de redes simples (apenas nós de entrada, saída e viés) e, ao longo das gerações, são introduzidos novos genes que acrescentam conexões ou nós. Este processo reduz a dimensionalidade inicial do espaço de procura e permite que fenótipos complexos surjam de forma incremental, otimizando o desempenho evolutivo [11].

Uma inovação central do NEAT é o esquema de codificação genética linear, que facilita a correspondência entre genes homólogos durante o cruzamento. Cada genoma é representado como uma lista de genes de conexão e de nó:

os genes de nó descrevem identificadores, tipo de nó e função de ativação;

os genes de conexão armazenam nó de entrada, nó de saída, peso da ligação, estado ativo/inativo e um número de inovação.

O número de inovação é um identificador único, atribuído sequencialmente a cada mutação estrutural, permitindo rastrear a origem histórica de cada gene. Assim, durante o cruzamento, genes com o mesmo número de inovação são considerados equivalentes e podem ser recombinados corretamente, enquanto genes disjuntos ou excedentes são herdados do progenitor com maior aptidão. Este mecanismo elimina a necessidade de comparações topológicas complexas.

As mutações no NEAT podem ser de dois tipos:

Alteração de pesos de conexão;

Mutação estrutural, que adiciona novas conexões ou novos nós à rede.

Contudo, redes recém-complexificadas podem ter, temporariamente, menor aptidão e ser eliminadas prematuramente. Para evitar a perda de inovações estruturais promissoras, o NEAT introduz o conceito de especiação: a população é dividida em espécies com base na similaridade topológica, e os indivíduos competem apenas dentro da sua espécie. Isto protege as inovações, dando tempo para que sejam refinadas.

Graças a estas propriedades, o NEAT permite gerar topologias de redes adaptadas ao problema em causa, evitando o excesso de camadas ocultas típico de abordagens clássicas baseadas em retropropagação. As redes resultantes apresentam desempenho competitivo em tarefas de otimização de controlo, aprendizagem não supervisionada e problemas complexos onde o espaço de procura é extenso e enganador.

O NEAT otimiza simultaneamente os parâmetros de ponderação e a topologia das redes neuronais artificiais. Começa a evolução com uma população de redes pequenas e simples e complexifica a topologia da rede em diversas espécies ao longo das gerações. Isso leva a população a um comportamento cada vez mais complexo. Uma caraterística chave do NEAT é a sua abordagem distinta de manter uma diversidade saudável de estruturas em crescimento simultaneamente. São atribuídas marcas históricas únicas a cada novo componente estrutural. Durante o cruzamento, os genes com as mesmas marcas históricas são alinhados, produzindo descendentes válidos de forma

eficiente, sem ter de recorrer a comparações topológicas complexas. A especiação no NEAT protege as inovações estruturais ao reduzir a competição entre redes com topologias distintas, dando tempo tempo para que as novas estruturas tenham os seus pesos otimizados. As redes são atribuídas a espécies com base na medida em que partilham marcas históricas. A complexificação é assim apoiada por marcas históricas. Com efeito, o NEAT procura uma topologia de rede compacta e adequada, começando com uma topologia mínima e aumentando progressivamente a complexidade das estruturas existentes [20].

O NEAT tem-se realçado como uma ferramenta de uso acessível que permite a criação de redes neuronais artificiais face a um qualquer domínio de forma intuitiva e fácil [11] [1]. A relevância prática do NEAT tem sido confirmada em múltiplos contextos. Por exemplo, o TensorNEAT [22] propõe uma implementação paralela em GPU que acelera drasticamente o treino em ambientes de robótica e controlo contínuo. Já o PropNEAT [23] introduz uma integração entre NEAT e retropropagação, permitindo encontrar redes menores, mais eficientes e de alto desempenho em tarefas de classificação. Estes exemplos ilustram como a filosofia original do NEAT — evolução simultânea de pesos e topologia — continua a inspirar variantes de sucesso em cenários contemporâneos.

A Figura 3.2 mostra uma rede típica evoluída com NEAT. Inicialmente composta apenas por nós de entrada, saída e bias, a rede complexifica-se com o surgimento de nós ocultos e novas conexões. Cada elemento desempenha um papel essencial:

Nós de entrada (retângulos cinzentos): representam as variáveis fornecidas ao modelo. Conexões: possuem pesos ajustáveis, podendo ser ativas (verde) ou desativadas (vermelhas) ao longo da evolução. Nós ocultos (círculos a branco): surgem por mutação e permitem a descoberta de representações intermediárias. Nós de saída (Círculos azuis): correspondem às decisões ou previsões finais da rede.

[1] neat smarter, ai solutions

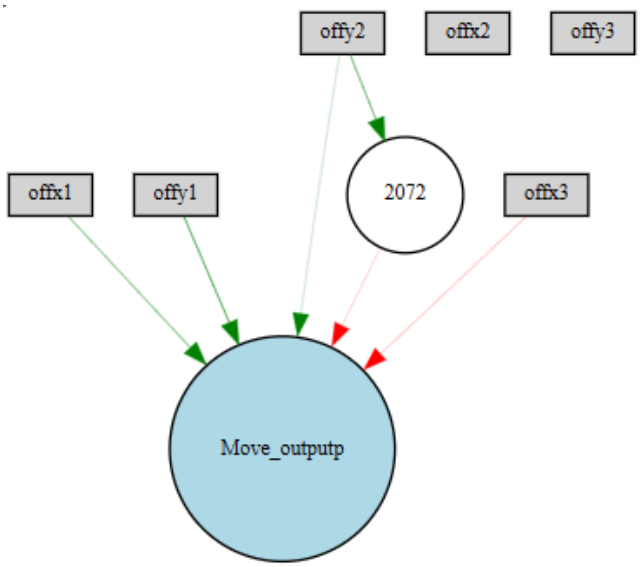

Figura 3.2: Exemplo esquemático de rede evoluída com NEAT. As setas representam conexões com pesos ilustrados com espessura da linha, conexões vermelhas não estão ativas, verdes estão, os círculos indicam nós (entradas, ocultos e saída).

Apesar da sua eficácia, o algoritmo NEAT original foi estendido e adaptado para lidar com problemas mais complexos, especialmente em contextos com inputs espaciais, múltiplos agentes ou ambientes de aprendizagem profunda. Eis algumas das variações e subtipos mais relevantes que não são usados neste estudo:

HyperNEAT (*Hypercube-based NEAT*) - é uma extensão que permite que o NEAT evolua redes neuronais com padrões de conectividade espacial-mente coerentes. Baseia-se na ideia de que os padrões de ligação podem ser descritos por uma função contínua (CPPN – *Compositional Pattern Producing Network*) que mapeia coordenadas espaciais dos neurónios para pesos de ligação [24]. É ideal para tarefas como visão artificial ou controlo motor, onde a estrutura dos dados é espacial.

CoDeepNEAT - é uma versão híbrida que aplica os princípios do NEAT à evolução de arquiteturas de redes profundas (*Deep Learning*), incluindo parâmetros como número de camadas, tipo de ativação, regularização, etc. Foi pensado para problemas em que é desejável automatizar a escolha de arquitetura e hiper-parâmetros, atuando como uma alternativa à pesquisa exaustiva manual [25].

Parallel NEAT / Distributed NEAT - Para cenários de grande escala ou simulações com múltiplos agentes, variações de NEAT distribuído permitem executar a evolução em paralelo, reduzindo significativamente o tempo de execução. Existem implementações em *Clusters* ou GPUs que aproveitam esta abordagem para ambientes de jogos ou robótica em tempo real.

Multi-Objective NEAT - Algumas adaptações do NEAT incorporam funções objetivo múltiplas, permitindo otimizar redes em função de várias métricas, como desempenho, simplicidade e consumo energético. Esta abordagem é útil, por exemplo, em aplicações móveis ou embarcadas.

3.3 Evolução Incremental (*Incremental Evolution*)

Evolução incremental é um método de evolução que visa treinar modelos a resolver problemas complexos, problemas que devido à sua generalidade e dificuldade em atingir o objetivo, impossibilitam a chegada a este pelos métodos mais convencionais. Este método de evolução foi concebido principalmente para o treino eficaz de modelos evolucionários na resolução de problemas mais complexos em que só a evolução que parte do zero não consegue resolver, e demonstrou algum sucesso nesse objetivo [4] e também na robótica evolucionária no treino de um robô com 6 pernas a mover-se [26]. Esta resolução de problemas complexos é alcançável na evolução incremental por passos que permitem o treino progressivo do modelo começando pelo treino de comportamentos mais simples, na resolução de problemas mais simples que vão crescendo em maior complexidade, até que o modelo, demonstrando capacidade na resolução destes problemas mais simples, possa interagir com o problema central inicialmente envisionado e idealmente chegar à solução ótima ou adotar o ou os comportamentos sofisticados pretendidos [3]. De outro modo, dada uma tarefa difícil como a abordada neste projeto como a perseguição e captura de uma presa que foge por parte de um ou vários predadores. O raciocínio proposto é que a tarefa é demasiado complexa para exercer uma pressão seletiva significativa sobre a população durante as fases iniciais da evolução. Todos os indivíduos têm um desempenho fraco, e consequentemente o AG fica preso numa região infrutífera do espaço de soluções. Se uma população evoluir primeiro numa versão mais fácil t0 da tarefa complexa t, pode ser possível descobrir uma região do espaço da rede a partir da qual t é mais acessível. Se não for esse o caso, pode ser possível desenvolver uma população numa versão ainda mais fácil t0 0 de t, a partir da qual t0 é mais acessível, e assim por diante. Desta forma, a tarefa final pode ser alcançada evoluindo gradualmente numa sequência de tarefas, começando por uma tarefa que pode ser evoluída diretamente [3]. Este método tem sido amplamente usado em muitos estudos envolvendo o domínio de perseguição predadores presas com co-evolução para garantir uma evolução possível de alcançar as soluções mais completas evitando, por isso, máximos locais [27, 1].

3.4 Domínio de captura com predadores e presas (*Capture Domain*)

Este domínio de problemas consiste num ambiente com uma ou mais presas e um ou mais predadores. Os predadores deslocam-se no ambiente com o objetivo de apanhar as presas e as presas de fugir aos predadores. As tarefas de perseguição e evasão são interessantes porque são omnipresentes no mundo natural, oferecem um objetivo claro que permite medir o sucesso com exatidão, e permitem analisar e visualizar as estratégias que evoluem. Estas tarefas geralmente não podem ser resolvidas com técnicas de aprendizagem supervisionada padrão, como a retro-propagação. As decisões corretas ou ótimas em cada momento não são normalmente conhecidas e o desempenho só pode ser medido após várias decisões serem tomadas. São necessários algoritmos mais complexos que possam aprender sequências de decisões com base no reforço esparso. As tarefas de perseguição e evasão são um desafio mesmo para os melhores sistemas de aprendizagem porque

requerem coordenação com o ambiente, outros agentes com objetivos compatíveis e agentes adversários [3].

O problema clássico foi introduzido por Benda et al [28], que consistia num mundo em grelha, onde quatro perseguidores tentavam capturar um fugitivo, cercando-o nos quadrados adjacentes. O fugitivo movia-se aleatoriamente, os perseguidores apenas podiam mover-se horizontal ou verticalmente, e não havia sobreposição de posições. O objetivo era mostrar que comportamentos cooperativos complexos podiam emergir de regras simples aplicadas a múltiplos agentes.

Evolução do domínio Depois deste modelo inicial, vários investigadores expandiram o problema, introduzindo novas regras e ambientes para estudar estratégias coletivas.

Duas grandes linhas de investigação surgiram: Estratégias emergentes em cenários 1 contra 1 (um perseguidor e um fugitivo). Estratégias em equipas de perseguidores, onde a captura só é possível através de cooperação.

Haynes e Sen Aplicaram Programação Genética Fortemente Tipada (STGP) para evoluir programas que controlavam perseguidores [29]. Testaram equipas homogéneas (todos os perseguidores com a mesma estratégia) e heterogéneas (cada perseguidor com estratégia distinta).

Achados principais: Emergência de papéis especializados: *Chasers* perseguiam diretamente o fugitivo. *Blockers* posicionavam-se estrategicamente para cortar rotas de fuga. A cooperação emergiu sem comunicação explícita entre perseguidores. Curiosamente, quando comunicação foi introduzida, o desempenho piorou, pois reduziu a pressão evolutiva para a especialização.

Equipas heterogéneas revelaram-se mais eficazes que as homogéneas, demonstrando que a diversidade de comportamentos dentro da equipa potência a cooperação. As estratégias emergentes superaram quase todas as heurísticas pré-programadas, mostrando a eficácia da evolução artificial. Yong e Miikkulainen testaram controladores baseados em redes neuronais evoluídas
[1]. Compararam duas arquiteturas: Centralizada: uma rede única controlava todos os perseguidores. Distribuída: cada perseguidor tinha a sua própria rede. Introduziram também um processo de evolução incremental, começando com tarefas simples (fugitivo parado) e gradualmente mais difíceis (fugitivo mais rápido). O modelo distribuído sem comunicação produziu cooperação muito eficaz, com clara especialização funcional (*chasers* e *blockers*). Concluíram que quando havia comunicação, as equipas tornaram-se mais flexíveis, mas menos eficientes, pois faltava consistência nos papéis especializados. Controladores centralizados funcionaram, mas tiveram piores resultados que os distribuídos, mostrando que a descentralização favorece a emergência de cooperação.

Denzinger e Fuchs desenvolveram perseguidores com comportamentos definidos por pares situação-ação, evoluídos por algoritmos genéticos [30]. O método mostrou-se versátil, permitindo gerar estratégias cooperativas em várias configurações do jogo. Contudo, o sistema continuava limitado pelo ambiente artificial de mundo em grelha, com situações relativamente simples e discretas.

Nishimura e Takashi fizeram estudos (predador-presa em larga escala) em que modelaram populações grandes de predadores e presas em mundos bidimensionais [31]. Usaram parâmetros

evolutivos para controlar interações sociais e observaram padrões coletivos como o *swarming* aleatório (aglomerados dinâmicos e instáveis), formações de grelha, marchas unidirecionais e *clusters* rotativos. Um resultado central foi a importância do *random swarming* que diminuiu as hipóteses de extinção, pois os predadores não conseguiam sincronizar facilmente a perseguição às presas, favoreceu a coexistência estável entre predadores e presas, beneficiando ambos. Os autores ligaram estes resultados a estudos biológicos de peixes em cardume, sugerindo que dinâmicas instáveis e não organizadas podem gerar cooperação espontânea e simbiose.

Também Lehman usou abundantemente modelos de redes neuronais heterogéneos com evolução incremental de modo a que a experiência não encalha-se em ótimos locais, na procura de soluções ótimas para este problema e inclusive fez experiências com a Novidade [32, 33] e encontrou bastante sucesso especialmente com redes de equipas heterogéneas.

Recentemente também houve um estudo que demonstrou a eficácia de modelos de redes neuronais para agentes heterogéneos centralizados com novidade, mais especificamente *NS-TEAM* [34]. O *NS-Team* é uma abordagem que combina a ideia de *Novelty Search* (Procura por Novidade) com co-evolução cooperativa para a seleção dos melhores indivíduos por geração. No processo de seleção dos agentes mais originais no espaço comportamental, são tidos em conta os comportamentos no contexto em equipa e não individualmente. No cenário de perseguição predadores presa, mostrou que o método NS-Team, que recompensa a novidade nos comportamentos de equipa, evita a convergência prematura em algoritmos de co-evolução cooperativa. A abordagem superou os métodos tradicionais, obtendo maior *Fitness* e diversidade de soluções, ao mesmo tempo que reduz o custo de avaliação por indivíduo. Testado com sucesso em cinco tarefas predadores presas, o *NS-Team* demonstrou ser escalável e uma solução eficaz para superar estagnação (máximos locais) em sistemas multi-agente.

De acordo com estes estudos a cooperação em sistemas de predador-presa pode emergir naturalmente através de evolução artificial, mesmo sem comunicação explícita ou regras pré-programadas. Especialização de papéis (*chasers* e *blockers*) num cenário de agentes heterogéneos surge como uma solução recorrente e eficaz. Modelos distribuídos (cada agente com o seu controlador) superam os centralizados. Comportamentos coletivos complexos como *swarming* podem favorecer a sobrevivência mútua e criar dinâmicas estáveis de coexistência. O *Novelty* veio a expandir o espaço comportamental no processo evolutivo mitigando o problema do máximo local. Mas também limitações, a maioria dos estudos foi feita em ambientes artificiais (*grid worlds*), o que deixa em aberto a questão da generalização para sistemas mais realistas. Para além disso muitas destas experiências recorreram a evolução incremental para atingir boas soluções e fugir aos ótimos locais [6].

3.5 Procura por novidade (*Novelty search*)

A pesquisa de novidades tem sucesso onde a pesquisa baseada em objetivos falha, recompensando os degraus que eventualmente levam ao topo da escada, isto é, os passos intermédios da evolução que levam idealmente à solução ótima. Ou seja, tudo o que é genuinamente diferente é recompen-

sado e promovido como um ponto de partida para uma maior evolução. Embora não possamos saber quais são os degraus corretos, se assumirmos que o principal problema da pesquisa baseada em objetivos é que não consegue detetar os degraus de todo, então esse problema é remediado.
Estes algoritmos cobrem naturalmente uma vasta gama de comportamentos em expansão. De facto, A procura de novidades requer poucas alterações a qualquer algoritmo evolutivo, para além de substituir a função de aptidão por uma métrica de novidade.
A métrica da novidade mede o quão diferente um indivíduo é dos outros indivíduos, criando assim uma pressão constante para fazer algo novo. A ideia chave é que em vez de recompensar o desempenho num objetivo, a recompensa a divergência em relação a comportamentos anteriores. Por isso, a novidade precisa de ser medida. Há muitas formas potenciais de medir a novidade, analisando e quantificando os comportamentos para caraterizar as suas diferenças. É importante notar que, tal como a função de aptidão, esta medida deve ser ajustada ao domínio.

A novidade de um indivíduo recém-gerado é calculada em relação a comportamentos (i.e. não os genótipos) de um arquivo de indivíduos passados e da população atual. O objetivo é caraterizar a distância que o novo indivíduo é do resto da população e dos seus predecessores no espaço comportamental, ou seja, o espaço de comportamentos únicos. Uma boa métrica deve, portanto, calcular a dispersão em qualquer ponto do espaço comportamental.As áreas com grupos mais densos de pontos visitados são menos novas e, portanto, menos recompensadas [32].

Uma medida simples da dispersão num ponto é a distância média aos vizinhos mais próximos desse ponto, em que k é um parâmetro fixo que é determinado experimentalmente. Intuitivamente, se a distância média aos vizinhos mais próximos de um dado ponto mais próximos de um ponto é grande, então ele está numa área esparsa; está numa região densa se a distância média é pequena.

A dispersão p no ponto x é dada por:

$$p(x) = \frac{1}{k} \sum_{i=0}^{k} dist(x, i)$$

em que μi é o vizinho mais próximo de x no que respeita à métrica de distância (dist), que é uma medida dependente do domínio da diferença comportamental entre dois indivíduos no espaço de pesquisa. O cálculo dos vizinhos mais próximos deve ter em consideração os indivíduos da população atual e do arquivo permanente de novos indivíduos. Os candidatos de regiões mais esparsas deste espaço de pesquisa comportamental recebem pontuações de novidade mais elevadas. Note-se que este espaço de novidade não pode ser explorado propositadamente; não se sabe a priori como entrar em áreas de baixa densidade, tal como não se sabe a priori como construir uma solução próxima do objetivo. Assim, mover-se no espaço de comportamentos novos requer exploração.
A geração atual e o arquivo fornecem uma amostra abrangente de onde a pesquisa esteve e onde se encontra atualmente; desta forma, ao tentar maximizar a métrica de novidade, o gradiente de pesquisa é simplesmente em direção ao que é novo, sem qualquer objetivo explicitamente especificado no espaço de pesquisa.
Em geral, a pesquisa de novidades permite qualquer caraterização de comportamento e qualquer

métrica de novidade. Embora de aplicação geral, a pesquisa de novidades é particularmente adequada para domínios com paisagens de aptidão enganadoras, caraterização comportamental intuitiva e restrições de domínio sobre possíveis comportamentos expressáveis. De uma forma mais geral, a pesquisa de novidades pode ser aplicada mesmo quando um objetivo não está claro em mente. Por exemplo, em alguns domínios, em vez de otimização, o objetivo pode ser recolher todos os comportamentos interessantes no espaço.
Quando a aptidão baseada em objetivos é substituída pela novidade, o modelo subjacente funciona normalmente, selecionando os indivíduos com maior pontuação para se reproduzirem. Ao longo das gerações, a população espalha-se pelo espaço de comportamentos possíveis, ascendendo continuamente a novos níveis de complexidade para criar novos comportamentos à medida que as variantes mais simples se esgotam [32].

Embora muitas abordagens de *Novelty Search* substituam completamente a função de aptidão, estratégias híbridas têm-se revelado eficazes. Estas combinam as duas métricas, por exemplo:

$$score = a \times novelty + (1 - a) \times fitness$$

onde "a"é um parâmetro ajustável num intervalo entre [0,1]. Este tipo de abordagem permite beneficiar da exploração promovida pela novidade e da exploração orientada pela aptidão, sendo útil em tarefas com metas específicas, mas com múltiplos caminhos possíveis.

Novelty tem encontrado bastante sucesso na evolução de modelos em algoritmos evolucionários, em vários tipos de problemas [32, 33].

Um desses casos de sucesso é uma experiência com *Novelty* envolvendo um labirinto [35]. A experiência do labirinto foi concebido para avaliar a eficácia da busca por novidade em comparação com a busca tradicional por *Fitness* em domínios enganadores. Nestes contextos, seguir apenas o gradiente de *Fitness* pode levar a becos sem saída locais, enquanto a busca por novidade ignora o objetivo final e foca em comportamentos inéditos, evitando esse problema. O domínio do problema envolve um robô controlado por uma rede neuronal artificial (RNA) que deve navegar de um ponto inicial até ao objetivo, dentro de um tempo fixo, usando sensores de distância e radares direcionais. O labirinto contém becos sem saída que criam ótimos locais enganosos 3.3.

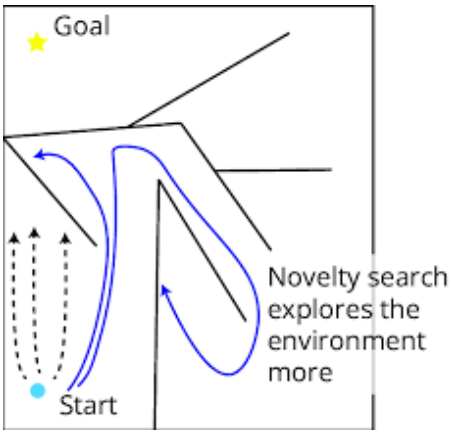

Figura 3.3: Exemplo esquemático de um labirinto em que o circulo a azul é a posição inicial do agente a ser treinado e a estrela a amarelo é o objetivo.

Foram testados dois mapas, um mapa de dificuldade médio, com becos próximos do objetivo mas que exigem trajetórias alternativas mais difíceis e um segundo mapa mais difícil, ainda mais enganador, que obriga a explorar zonas de baixo *Fitness* antes de encontrar o objetivo. Como resultados, o NEAT baseado em *Fitness* foi mais lento e falhou quase sempre no mapa difícil (apenas 3 soluções em 40 execuções). Já o NEAT com busca por novidade obteve desempenho muito superior, resolvendo o mapa difícil em 39 de 40 execuções, com menos avaliações e redes menos complexas. Seleção aleatória confirmou-se ineficaz, validando que a busca por novidade não equivale a exploração ao acaso. Foi ainda demonstrado que limitar o tamanho do arquivo de comportamentos (que armazena trajetórias para calcular novidade) não prejudica significativamente o desempenho, o que reduz o custo computacional sem perda de eficácia. No caso de labirintos não delimitados, a busca por novidade perdeu eficácia, porque muitos comportamentos "novos" correspondiam apenas a movimentos aleatórios fora do labirinto, sem informação útil para a navegação. Tanto a busca por *Fitness* como a busca por novidade falharam, mostrando a importância das restrições do domínio.

Para além desta experiência base foram feitas outras. Uma de alta dimensionalidade em que a localização do robô foi amostrada várias vezes durante a avaliação (até 400 dimensões). Apesar da vastidão do espaço comportamental, a busca por novidade continuou a encontrar soluções de forma consistente, sem perda significativa de desempenho. Este resultado sugere que a complexificação gradual do NEAT ajuda a guiar a exploração, evitando que comportamentos caóticos surjam prematuramente. Uma outra de Discretização (*Controlled Conflation*) que consistiu na divisão em grelha de diferentes resoluções, atribuindo a mesma caracterização a navegadores que terminassem na mesma célula. Mesmo com discretizações grosseiras, a busca por novidade manteve bom desempenho, resolvendo quase sempre o labirinto, ainda que com mais avaliações nas grelhas mais pequenas. Assim, a *conflation* moderada não compromete a eficácia, pois os "*stepping stones*" ainda são respeitados. Um último de caracterização do comportamento com a mesma métrica usada para o *Fitness*. Quando o comportamento foi reduzido a um único valor (distância ao objetivo), navegadores em locais muito diferentes mas igualmente distantes do objetivo foram

confundidos. Esta *conflation* revelou-se prejudicial: a busca por novidade baseada em *Fitness* só resolveu 11 de 40 execuções (melhor do que *Fitness* puro, mas pior do que caracterizações mais informativas).

A experiência mostra que a busca por Novidade é muito eficaz em domínios enganadores, explorando comportamentos de forma mais equilibrada do que a busca por *Fitness*. O impacto da *conflation* depende de como esta afeta os *stepping stones* necessários para evoluir soluções: *Conflation* que respeita *stepping stones* (e.g. discretização espacial) mantém a eficácia. *Conflation* que os oculta (e.g. via *Fitness*) compromete a busca. Além disso, verificou-se que mesmo em espaços comportamentais de alta dimensão, a busca por novidade não falha, contrariando a intuição de que tal vastidão seria intransponível.

Capítulo 4

Experiências de neuro-evolução

Nesta secção está todo o trabalho e experiências feitas no sentido de comparar as metodologias *objective oriented* com o *novelty search*, o uso de comunicação ou não em agentes homogéneos e heterogéneos, a fim de testar a hipótese de uso de modelos neuro-evolucionários para o treino de agentes homogéneos com comunicação e uso de novidade métodos mais eficazes na solução de problemas que requerem a coordenação entre multi-agentes como o caso do problema domínio captura predadores presa. Foram feitas 3 experiências de captura no total, uma num espaço uni-dimensional com dois predadores, e duas experiências distintas num espaço bidimensional, a primeira com três predadores e a última com quatro predadores. Para todas as experiências usei a biblioteca NEAT para linguagem de programação python, implementado e pronto para uso e desenvolvimento de modelos neuro-evolutivos de forma fácil e eficaz. Ela permite a definição, treino, execução de rede neuronais artificiais face a um qualquer problema a que foram concebidos. É uma das abordagens mais usadas e uma das mais bem sucedidas para a implementação de redes neuronais artificiais[1].

4.1 1ª experiência

A primeira experiência consiste na captura da presa imóvel por parte de predadores homogéneos com redes neuronais idênticas num espaço discreto uni-dimensional síncrono que requer comunicação e coordenação para o sucesso da captura [27]. Foram comparados os desempenhos entre dois modelos homogéneos, um com comunicação livre em que há evolução da comunicação e outro em que a comunicação é predefinida.

4.1.1 Metodologia

Temos 2 predadores, um em cada lado da presa a uma distância aleatória 4.1. O desafio da tarefa é que cada predador apenas tem acesso à sua distância à presa e não acedem à distância a que está o outro predador e por isso terão de comunicar entre si para coordenar os seus movimentos. Em cada instante cada um deles recebe a mensagem do outro predador e mede a sua distância à presa, podendo mover-se uma casa no sentido da presa ou manter-se na mesma posição. Os predadores

[1] `https://neat-python.readthedocs.io/en/latest/neat_overview.html`

também terão de decidir o que comunicar ao outro.
O problema é resolvido com sucesso caso ambos os predadores atinjam a posição da presa ao mesmo tempo e sem sucesso caso só um dos predadores atinja a posição da presa ou o limite de ações que é determinado pela seguinte expressão seja atingido:

$$2 \times DIM$$

em que DIM é a distância máxima inicial que um predador pode estar da presa.
Para evitar que a evolução se adapte a uma situação inicial realizam-se uma série de ensaios com distâncias iniciais aleatórias para os dois predadores. Em qualquer dessas situações os predadores podem estar a qualquer distância das presas, entre 1 e DIM.
Em cada ensaio o *fitness* é calculado da seguinte maneira: caso os predadores apanhem a presa ao mesmo tempo o *fitness* é 1, caso só um predador apanhe a presa o *fitness* é 0, caso nenhum dos casos anteriores se suceda o *fitness* é:

$$\frac{1}{X}$$

, em que X é a média das distâncias dos dois predadores à presa. O *fitness* final da execução é o *fitness* médio resultante dos ensaios.

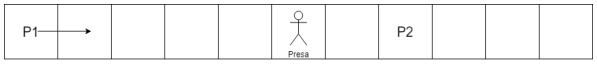

Figura 4.1: Ambiente de execução 1ª Experiência em que Dim=5. A presa é imóvel durante toda a duração da experiência

O objetivo desta experiência é verificar o desempenho da evolução da linguagem numa tarefa simples que exige coordenação e compara-la com o caso em que a linguagem está predefinida, i.e., definimos por design que informação comunicar ao outro predador, não sendo objeto de evolução. Na abordagem com comunicação predefinida, cada um dos predadores está sempre a comunicar a sua distância à presa. Assim, em cada instante, cada um dos predadores controlados por redes neuronais idênticas, terá como inputs: a sua distância à presa e a distância do outro à presa (que lhe foi comunicada), e terá como output único a ação (avançar ou manter-se parado). Como a posição do outro predador é sempre comunicada, não faz parte dos outputs da rede neuronal. Na figura 4.2 apresento a topologia da rede inicial NEAT do predador um que tem 2 inputs, o primeiro corresponde ao número de casas do predador 1 à presa e o segundo input corresponde ao número de casas do predador dois à presa, e um output sendo completamente ligada. Os valores dos neurónios de input e de output têm de ser normalizados, pois o NEAT exige valores no intervalo [0,1]. Assim, as distâncias à presa são sempre divididas pela distância máxima possível Dim e, tendo apenas um neurónio de output, com valores em [0,1]. Para codificar as duas ações consideramos que os valores abaixo de 0.5 correspondem a ficar parado e acima de 0,5 ao movimento de avançar para a presa.
Na segunda abordagem, evolução da linguagem, os predadores enviam em cada instante uma mensagem um ao outro, mas não predeterminamos o que é comunicado entre predadores, existindo

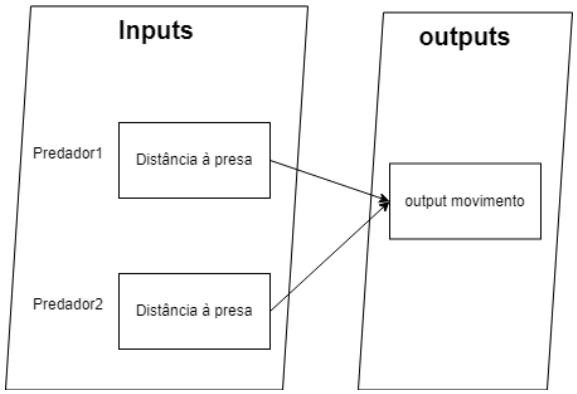

Figura 4.2: Topologia inicial da ANN para os predadores com comunicação predefinida

por isso mais um neurónio de output para além do relativo ao movimento e mantendo os mesmos dois neurónios de input (um para a distância normalizada à presa e o outro para o valor que foi acabado de ser comunicado pelo outro predador). Na figura 4.3 é da do o exemplo de ANN do primeiro predador logo o seu primeiro input corresponde ao número de casas da sua posição à presa, e o segundo input ao sinal do predador dois. No caso da ANN do segundo predador o primeiro input corresponde ao número de casas do predador dois à presa, e o segundo input ao sinal enviado pelo predador um . Na situação inicial o input relativo ao sinal comunicado pelo outro assume o valor de 0.

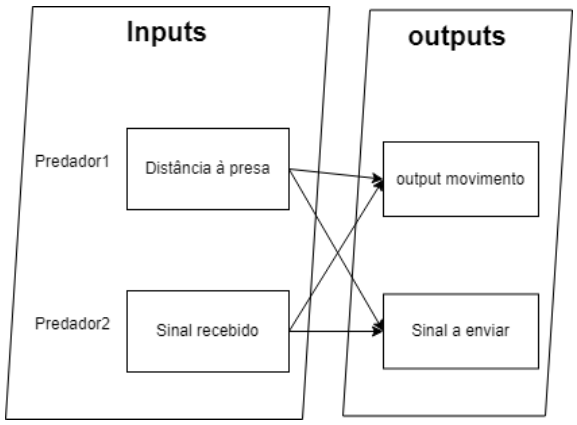

Figura 4.3: Topologia inicial da ANN para os predadores com com evolução da comunicação

4.1.2 Resultados das experiências baseadas no Objetivo

Em ambos os modelos usados a experiência foi executada 10 vezes numa população de 500 indivíduos num máximo de 100 gerações. Em cada execução foram corridos 10 ensaios que consistem em cenários em que os predadores surgem a distâncias aleatórias da presa num intervalo [1, 10], logo Dim = 10. É calculada a média de *fitness* dos 10 ensaios em cada geração para determinar o sucesso de cada modelo resultante de cada execução.

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
MCP	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Geração	2	1	1	1	1	2	1	1	2	1
MCL	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Geração	4	40	96	20	5	5	3	12	5	1

En: Execução n; MCP: Modelo comunicação predefinida; MCL: Modelo comunicação livre; Geração: Geração em que se deu a captura

Tabela 4.1: Resultados da primeira experiência de evolução *objective based* com Dim=10 com 10 ensaios (E1..E10)

Ambos os modelos foram eficazes ao encontrarem sempre a solução da experiência em todas as evoluções. No primeiro modelo o sucesso dá-se quando o predador mais próximo da presa espera que o outro se aproxime da presa até ficarem ambos à mesma distância, mais especificamente quando a sua distância normalizada à presa coincide com a distância normalizada do outro, movendo-se então ambos para a presa acabando por a capturar. No modelo em que houve evolução da comunicação, os sinais enviados entre os predadores seguiram uma lógica interessante: quanto mais próximo estiver o predador da presa maior o valor do sinal enviado, na maior parte dos casos quando um predador se encontra numa posição adjacente à presa este espera até que o outro predador envie um sinal de valor semelhante aquele enviado por este quando primeiro chegou à posição que assumiu, como confirmação de que também o outro predador se encontra a uma ação da captura, para avançar e assim no mesmo turno alcançarem a presa. Noutros casos os sinais seguiram uma lógica muito idêntica ao comportamento observado na primeira abordagem, em vez de o predador mais próximo esperar na casa adjacente à presa este espera numa casa mais longe, a aguardar que o outro envie um sinal com o mesmo valor a indicar que também ele se encontra à mesma distância da presa, para depois prosseguirem na captura da presa como é possível observar na fig. (fig.4.4). Houve casos em que as ANNs resultantes de execuções desenvolveram camadas adicionais e interações mais complexas entre os nós (fig.4.5) (fig. 4.6) É possível visualizar o comportamento do modelo treinado no repositório [2].

[2] `https://www.dropbox.com/scl/fo/ll975494f1gat42ki7g4w/AFJj8eGdi59dzmnFB-e5VsA?rlkey=p7s2ufw0lu9p5l4n0rbqgyebs&st=286i4194&dl=0`

```
simulação 4
ilustration of environment and distance to predator: 5 6
. . . . . 1 . . . . * . . . . . 2 . . . .
ilustration of environment and distance to predator: 4 5
. . . . . 1 . . . * . . . . 2 . . . . .
signal1 sent= 0.0342851176855111 signal2 sent= 0.015527656499295533

ilustration of environment and distance to predator: 3 4
. . . . . 1 . . * . . . 2 . . . . . .
signal1 sent= 0.07291808167186418 signal2 sent= 0.03314339453603113

ilustration of environment and distance to predator: 2 3
. . . . . 1 . * . . 2 . . . . . .
signal1 sent= 0.14812441232704796 signal2 sent= 0.06904996608981012

ilustration of environment and distance to predator: 2 2
. . . . . 1 . * . 2 . . . . . .
signal1 sent= 0.27393091491769694 signal2 sent= 0.13389706441457222

ilustration of environment and distance to predator: 1 1
. . . . . 1 * 2 . . . . . .
signal1 sent= 0.2609451175833774 signal2 sent= 0.23429464653864035

ilustration of environment and distance to predator: 0 0
. . . . . * . . . . . .
signal1 sent= 0.417663547907912 signal2 sent= 0.41105232763170874

fitness: 1
```

Figura 4.4: Simulação passo a passo do comportamento evoluído no modelo com comunicação livre

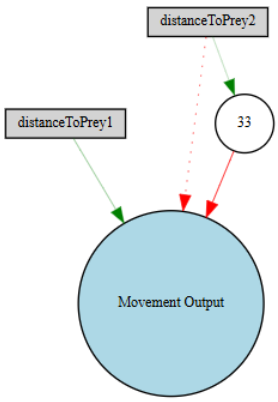

Figura 4.5: ANN do melhor indivíduo do modelo com comunicação predeterminada com camada oculta desenvolvida

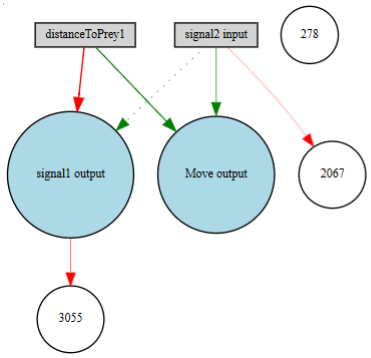

Figura 4.6: ANN do melhor indivíduo do modelo com comunicação livre com camada oculta desenvolvida

4.1.3 Avaliação dos resultados baseados no objetivo

Os resultados demonstram que em ambas as abordagens de modelo usadas a solução foi encontrada sem exceção em todas as execuções. Isto deve-se principalmente à simplicidade e linearidade do problema, bastaram poucas gerações, na maior parte dos casos, para os modelos evoluírem a resolver o problema. Apesar disto o modelo de comunicação livre teve um desempenho visivelmente inferior, demorou mais gerações em média a encontrar a solução, num dos casos até chegou de mais 90 gerações a encontrar a solução numa das evoluções (tabela 4.1). A causa disto está no facto de o modelo de comunicação livre ter de sofrer uma evolução da linguagem e necessitar de mais coordenação para finalmente garantir a comunicação competente entre os predadores 4.4. O modelo de comunicação livre também demorou mais tempo em média na execução de cada geração com tempos a rondar os 0.8seg em comparação com o primeiro modelo que demorou aproximadamente 0.5seg. Isto deve-se principalmente ao facto deste modelo de comunicação livre ter uma topologia de rede neuronal mais complexa com dois outputs em vez de um só do modelo de comunicação predefinida 4.5 4.6.

4.1.4 Metodologia da novidade

Foram feitas mais experiências com o treino de modelos usando novidade para uma comparação direta com a abordagem *objective-oriented*. Nesta abordagem o processo de seleção dos genomas mais aptos ao longo das gerações é feita com o *novelty score* e não o *fitness*. A relembrar que o *novelty score* é a distância do comportamentos dos indivíduos da população aos comportamentos dos K-vizinhos mais próximos. Para isso foi preciso definir o comportamento de cada genoma, o arquivo que guarda os comportamentos de acordo com um critério e K-vizinhos. O comportamento ficou definido como as distâncias finais dos predadores à presa em todos os ensaios que corresponde a um vetor:

`[[E1D1,E1D2], [E2D1,E2D2], ..., [E10D1,E10D2]]`

Em que En é o número do ensaio, D1 e D2 o número de casas de cada predador à presa no final do ensaio. O arquivo ficou a guardar os comportamentos aleatórios de qualquer individuo, uma geração de cada vez. O K assumiu o valor de 10 vizinhos. O *novelty-score* que é calculado percorrendo todo o espaço de comportamentos da geração atual mais os comportamentos que, usando um arquivo, foram guardados ao acaso um por geração de um genoma aleatório, escolhendo os k-vizinhos mais próximos para o calculo da distância de cada comportamento aos seus mais próximos para obter um valor da sua "originalidade"quantificada.

4.1.5 Resultados da novidade

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
MCP	1	1	1	1	1	1	1	1	1	1
Geração	2	2	2	1	1	1	1	2	2	1
MCL	1	1	1	1	1	1	1	1	1	1
Geração	6	1	7	4	8	2	11	1	6	31

En: Execução n; MCP: Modelo comunicação predefinida; MCL: Modelo comunicação livre; Geração: Geração da evolução em que foi encontrada a solução ótima;

Tabela 4.2: Resultados da primeira experiência evolução *novelty based* com Dim=10 com 10 ensaios (E1..E10)

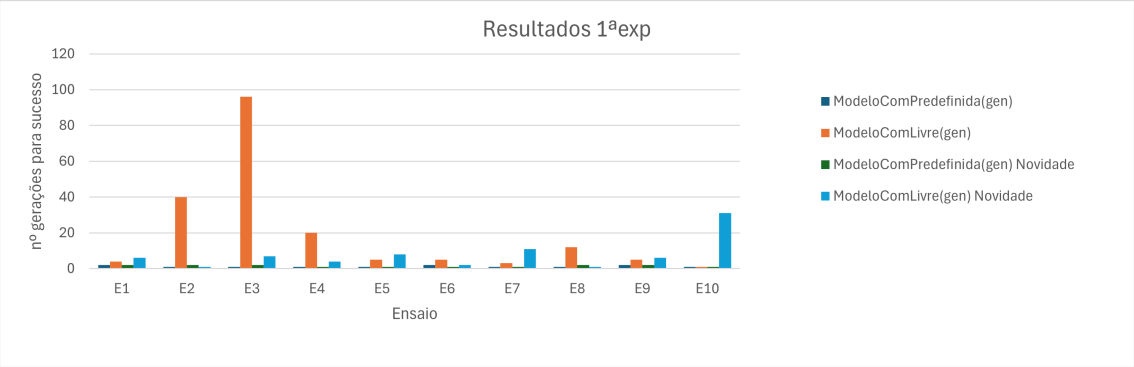

Figura 4.7: Gráfico com todos os resultados da 1ª experiência

4.1.6 Avaliação de resultados da 1ª experiência

Como é possível observar no gráfico 4.7, em todas as experiências a solução foi encontrada e regra geral a novidade precisou de menos gerações que a abordagem *objective-oriented* especialmente no modelo de comunicação livre (evoluída). Nesta experiência o uso da novidade foi melhor que o uso do *fitness* no processo evolutivo dos modelos treinados. O modelo em que não há evolução da comunicação em geral foi mais rápido a encontrar a solução precisando de menos gerações, em grande parte porque a evolução já definida para os agentes é boa o suficiente e não precisa de ser reinventada como no caso de comunicação livre. Estes modelos provaram ser mais do que capazes de resolver problemas desta complexidade.

4.2 2ª experiência

Esta experiência de captura tem como objetivo comparar o desempenho entre modelos com e sem Novidade já num ambiente bidimensional ortogonal síncrono discreto. O foco principal desta experiência foi fazer uma comparação direta entre modelos neuronais em agentes heterogéneos e agentes homogéneos com comunicação assim como comparar o uso ou não da Novidade e verificar se este último é tão ou mais capaz que o primeiro no treino de agentes diferenciados no comportamento, na resolução do problema de captura. O objetivo deste problema é a captura da presa por parte de um dos 3 predadores mas a captura dá-se com maior facilidade quando a presa é cercada, não se podendo mover com os predadores a limitar os movimentos. É importante também referir que os predadores são os agentes a serem treinados por meio de neuro-evolução, e por isso as suas ações são regidas pelas redes neuronais definidas e treinadas. Nesta experiência basta que um predador se desloque sobre a presa para a capturar. No entanto, a presa move-se tão rapidamente como os predadores e afasta-se sempre do predador mais próximo, pelo que não há forma de a apanhar simplesmente perseguindo-a. O principal desafio é por isso coordenar a perseguição: Os agentes têm de se aproximar da presa a partir de diferentes direções para que esta não tenha para onde ir no final 4.8.

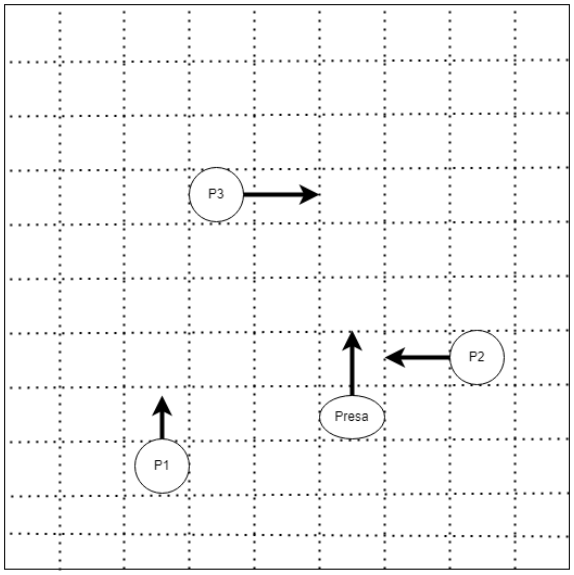

Figura 4.8: Ilustração do domínio de captura usado em que os predadores representados por círculos estão a cercar a presa representada por um elipse

4.2.1 Metodologia

Este exercício envolve um ambiente 2D, discreto, toroidal em forma de quadrado, isto é um ambiente em que os agentes ao moverem para além dos limites do mapa, por exemplo movendo-se para cima no limite superior do mapa, surgem no limite inferior. A experiência conta com 4 agentes, uma presa e três predadores, sendo de novo apenas os predadores o alvo da evolução e a presa assume um comportamento predefinido. Todos os agentes movem-se à mesma velocidade, que

equivale a uma casa adjacente de cada vez (a um centésimo das dimensões do lado do quadrado), em 4 possíveis direções: Norte, Sul, Este e Oeste, ou não se moverem de todo. Todos os agentes neste cenário podem-se atravessar, ou assumir a mesma posição. Cada ensaio começa com os predadores a assumirem as suas posições no canto inferior esquerdo do mapa e acaba caso a presa seja apanhada, ou quando todos os agentes se tenham movido 150 vezes. Há captura quando um dos predadores assume a posição da presa ou quando se atravessam, isto é, um dos predadores e a presa trocam de posição em instantes consecutivos.

Numa primeira versão, na rede neuronal adotada não se dá evolução da comunicação pelo que a comunicação é predefinida e o modelo heterogéneo (uma rede para treinar todos os predadores), e por isso os inputs são os *offsets* (distâncias em x e em y) dos 3 predadores à presa e os outputs são 3, cada um a determinar a ação do respetivo predador. Também foram aplicados 2 modelos homogéneos, um com comunicação predefinida e um com evolução da comunicação. O primeiro inclui comunicação predefinida na rede e por isso só tem os 6 inputs correspondentes aos *offsets* de cada predador à presa e um output a determinar o movimento do predador. O segundo inclui comunicação livre na rede, com 4 inputs (2 inputs que correspondem às distâncias em x e em y do predador em questão e 2 sinais provenientes dos outros 2 predadores) e 2 outputs, um que determina o movimento do predador, e um sinal que será enviado como input para os outros predadores. Estes modelos foram testados em quatro cenários diferentes: O primeiro em que a presa não se move e por isso a tarefa de captura é muitíssimo simplificada. O segundo em que os predadores se movem ao dobro da velocidade da presa facilitando bastante a captura da presa. O terceiro cenário em que a presa só foge caso um predador esteja a uma distância de 10 casas. E o quarto cenário, mais difícil, em que tanto os predadores como a presa se movem à mesma velocidade.

Quando pelo menos um dos predadores captura a presa, o *fitness* é dado pela fórmula seguinte:

$$\frac{2 * (W + H) - X}{10}$$

em que W é o comprimento do ambiente, H a largura, X a média das distâncias ortogonais finais dos predadores à presa caso a presa tenha sido apanhada.

No caso em que não há captura após o nº limite de tiques ou passos:

$$\frac{(Y - X)}{10}$$

em que Y é a média das distâncias ortogonais iniciais dos predadores à presa e X a média das distâncias ortogonais finais dos predadores à presa, caso a presa não tenha sido apanhada.

Deste modo, os indivíduos da população, no processo de evolução, são sempre mais recompensadas caso a presa tenha sido apanhada, e também dá-se um aumento do valor do *fitness* quanto mais próximos estiverem todos os predadores da presa. Como a presa se move à mesma velocidade que os predadores no quarto cenário, é impossível um só predador, perseguindo-a, apanha-la,

precisando por isso cooperação e coordenação entre eles para que se dê a captura, o que torna este cenário mais difícil que os outros. O valor máximo de aptidão (*fitness threshold*) que um individuo pode assumir é 20.

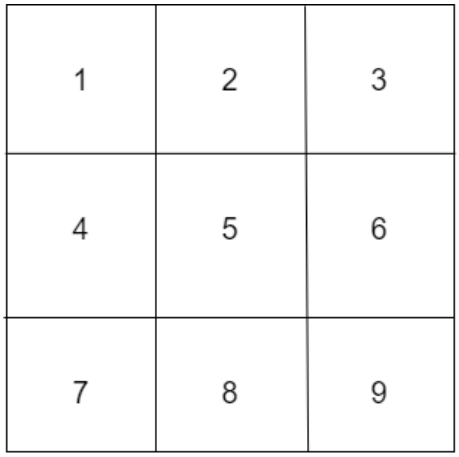

Figura 4.9: Ambiente de execução com partições numeradas

Em todas as experiências deste cenário o ambiente bidimensional é quadrático com dimensões

$$100 * 100$$

casas, e cada agente move-se uma casa de cada vez de modo a poder mover-se de um lado ao outro do mapa em 100 movimentos. Cada genoma da população corre 9 ensaios seguidos, isto é cenários iniciais em que os predadores surgem sempre no canto inferior esquerdo mas a presa começa numa das 9 partições do ambiente quadrático (fig 4.9), sem repetição, por ensaio, e é posteriormente calculado o desempenho do individuo na população com a média de *fitness* das 9 tentativas.
Todos os inputs também foram sujeitos a uma normalização usando a expressão:

$$\frac{\frac{s}{L} + 1}{2}$$

em que s é o offstet em x ou y (distância de Manhattan) e L as dimensões de um dos lados do mapa. Como o valor de s (offset) pode ser negativo, é necessário somar por um e depois fazer a divisão por dois para o valor do input ser sempre positivo e deste modo os inputs ficaren entre os valores [0, 1] como é aconselhado fazer pela biblioteca NEAT[3]. Os outputs que determinam a ação também assumem um valor entre [0,1]. No caso de ser usado somente 1 output para determinar 1 ação, se o valor estiver entre [0, 0.2[não se move, se estiver entre [0.2, 0.4[move-se em direção a Leste, se estiver entre [0.4, 0.6[move-se para Norte, se estiver entre [0.6, 0.8[move-se para Oeste, se estiver entre [0.8, 1] vai para Sul.

[3] `https://neat-python.readthedocs.io/en/latest/neat_overview.html`

Algoritmo de movimento da presa A presa move-se de acordo com um algoritmo que determina que a presa foge movendo-se na direção que a afaste mais do predador mais próximo desta. É guardado o predador mais próximo da presa de acordo com a distância de *Manhattan*, em caso de empate é escolhido um dos predadores empatados ao acaso. Depois de selecionado o predador, é calculada a distância de cada umas das quatro posições adjacentes da presa ao predador, a presa move-se na direção que a afaste mais deste predador, em caso de empate é selecionada uma das direções ao acaso. Cada um dos 4 cenários da experiência alteram ligeiramente este algoritmo. No primeiro cenário 4.2.1 em que a presa não se move, a função de movimento não é usada, logo a presa fica sempre parada. No segundo 4.2.1 e quarto cenários 4.2.1 o algoritmo é aplicado exatamente como está em C.1. No terceiro cenário4.2.1 foi adicionada uma condição que assegura que a presa não se mova a não ser que um ou mais predadores estejam a 10 casas de distância.

4.2.2 Resultados das experiências baseadas no objetivo

Os 4 modelos mencionados foram corridos uma vez em cada um destes 4 cenários com uma população de 500 genomas ao longo de 1000 gerações. Em cada execução foram usados 9 ensaios em que a presa começa numa posição diferente num dos 9 sectores distintos do ambiente bidimensional. Quando se dá captura o *fitness* ronda os 18+.

Tabela 4.3: Parâmetros da 2ª experiência baseada no objetivo

	nº população	nº gerações	nº ensaios
MHCP	500	1000	9
MHCE	500	1000	9
Mhe1o	500	1000	9
Mhe5o	500	1000	9

Tabela 4.4: Resultados da 2ª experiência

Mode/Scenario	pNM	P2M	pP	pM	tempo(s)
CPHo	15.402(107.814)	13.288(93.018)	4.310(30.17)	4.134(28.941)	93.63 sec
CPHe	13.375(93.626)	9.941(69.588)	8.533(59.731)	4.708(32.956)	45.29 sec
CPHe5o	9.938(69.564)	7.348(51.433)	8.387(58.776)	4.936(34.552)	48.42 sec
CLHo	15.890(111.231)	19.385(135.696)	4.235(29.645)	4.281(29.970)	43.053 sec
fitness threshold	20(140)				

pNM: Presa não se move; P2M: Predadores 2 vezes velocidade; pP: presa move-se quando predadores estão próximos; pM: presa move-se normalmente;CPHo: Modelo comunicação predefinida homogéneo ;CPHe: Modelo comunicação predefinida heterogéneo ;CPHe5o: Modelo comunicação predefinida heterogéneo 5 outputs ;CLHo: Modelo comunicação evolutiva homogéneo;

É possível visualizar o comportamento do modelo treinado neste repositório [4].

[4] www.dropbox.com/scl/fo

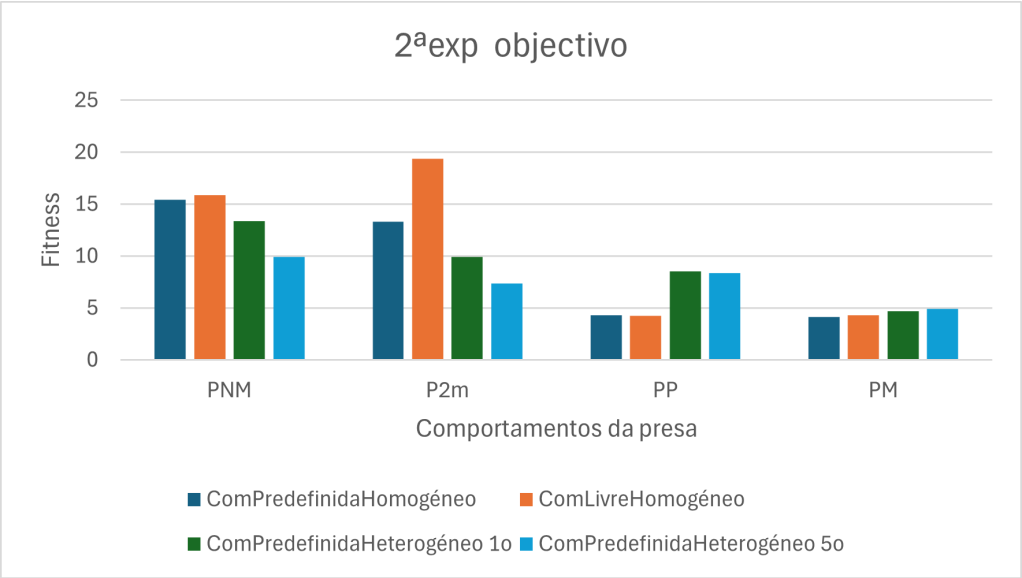

Figura 4.10: Resultados da 2ª experiência

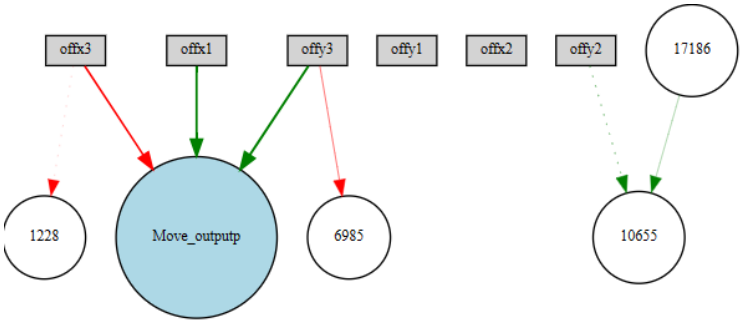

Figura 4.11: ANN resultante do modelo homogéneo comunicação predefinida pM

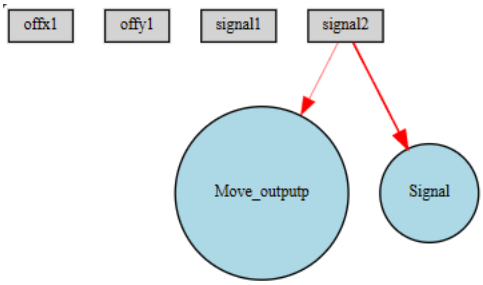

Figura 4.12: ANN resultante do modelo homogéneo comunicação livre pM

4.2.3 Avaliação dos resultados baseados no objetivo

No caso em que as presas não se movem, os resultados refletem um melhor desempenho das redes neuronais homogéneas, uma aptidão (*fitness*) superior a 14.3 (100), em comparação com as redes heterogéneas que só atingiram um valor pouco acima de 11.4 (80), com resultados ligeiramente

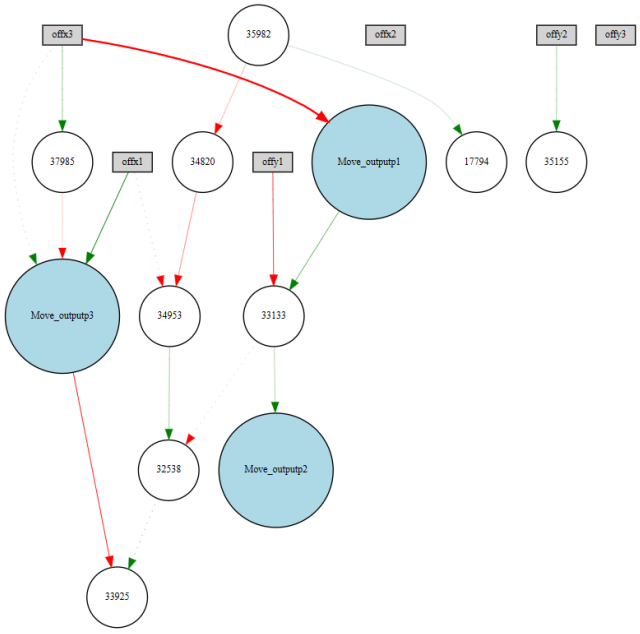

Figura 4.13: ANN resultante do modelo heterogéneo comunicação predefinida pM

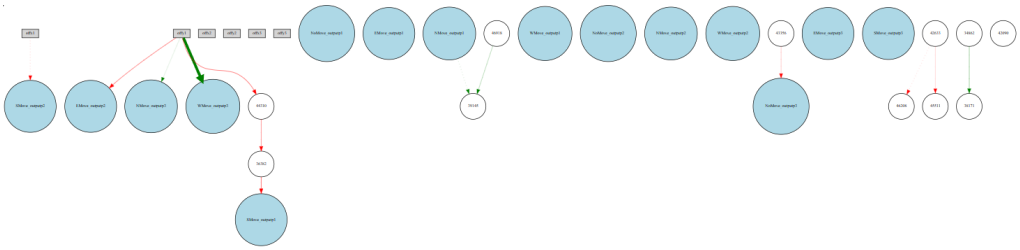

Figura 4.14: ANN resultante do modelo heterogéneo 5o comunicação predefinida pM

melhores no caso em que há comunicação livre. Em todos os casos a presa foi capturada em pelo menos 5 dos 9 ensaios sendo que o modelo de evolução da comunicação capturou 8 das 9 presas.

Quando as presas se movem a metade da velocidade dos predadores mais uma vez os modelos homogéneos obtiveram resultados substancialmente melhores(acima de 14.3[100]) comparando com os modelos heterogéneos que só obtiveram resultados entre os 7.1 (50) e os 11.4 (80). O modelo em que há comunicação obteve o melhor resultado tendo atingido um *fitness* muito próximo do máximo (20)[140]

Em ambos os casos acima referidos em que há uma clara vantagem dos predadores sobre a presa o modelo não precisa de ser muito sofisticado na definição do comportamento dos predadores para eles acharem a solução, é somente necessário no mínimo um predador dirigir-se rumo à presa para que a captura eventualmente se dê. Apesar disso a maior parte dos modelos não achou a solução ótima mas capturou a maior parte das presas (entre 7 e 8 dos 9 ensaios)

Quando as presas se movem ao mesmo ritmo que os predadores, os ANNs treinados obtiveram valores baixos sem exceção a rondar os 4(28) de *fitness*, nenhuma ANN chegou próximo de atingir o objetivo de capturar os predadores nas 9 instâncias diferentes.

Nos modelos em que houve evolução da comunicação os sinais enviados entre os agentes foram ineficazes, independentemente do estado ou posições que os agentes assumiam, os sinais enviados mantêm o mesmo valor do início ao fim dos ensaios, o que prova a irrelevância da linguagem evoluída nos modelos treinados. Nos cenários mais simples em que a presa não se move e os predadores se movem o dobro da distância apesar do desenvolvimento de uma linguagem incapaz, o modelo conseguiu achar a solução apanhando a presa devido principalmente à trivialidade da tarefa. Já nos cenários mais difíceis em que a presa foge tanto quando está ou não está próxima de predadores, o modelo de comunicação não teve sucesso, à semelhança dos modelos sem comunicação, com uma comunicação que foi igualmente incapaz de redimir a impotência do modelo na captura da presa.

As Anns resultantes no modelo homogéneo comunicação predefinida denunciam a adoção de um comportamento por parte dos agentes que ignora a grande maioria dos inputs, dados das distâncias dos outros predadores, o que levou a que os predadores assumissem todos um comportamentos idêntico de pouca cooperação e coordenação (fig. 4.11). No modelo homogéneo com evolução da comunicação um dos predadores já adotou um comportamento distinto dos outros predadores separando-se deles numa tentativa de captura cercando a presa levando-a a aproximar-se dos outros predadores (fig. 4.12). Nos modelos heterogéneos todos os predadores adotaram comportamentos diferentes apesar de ineficientes movendo-se todos em direções diferentes mas não foram capazes de capturar a presa em fuga (fig. 4.13) (fig. 4.14).

4.2.4 Metodologia da novidade

Há três métodos possíveis no cálculo da novidade:

Usar só a população da geração atual: A novidade aqui é calculada somente comparando o comportamento de cada genoma com os outros na geração contemporânea ao seu e não guarda como referência comportamentos passados cujo *novelty score* tenha sido distinto;

Usar a população da geração atual + arquivo que guarde genomas com melhor *novelty score*: O cálculo da novidade é feito, neste caso, juntando os comportamentos de todos os genomas na geração atual mais os comportamentos dos genomas de geração anteriores que tiveram um valor de novidade bom o suficiente para ocuparem uma posição do arquivo que guarda os comportamentos mais originais.

Usar população geração da atual mais arquivo que insere um comportamento ao acaso de um individuo por geração: Trata-se de combinar o registo de comportamentos de todos os genomas da geração atual com um arquivo que contém o comportamento de um genoma ao acaso por geração, para o cálculo da Novidade.

Foram feitas mais experiências com o treino de modelos usando novidade para uma comparação direta com a abordagem *objective-oriented*. Nesta abordagem o processo de seleção dos genomas mais aptos ao longo das gerações é feita com o *novelty score* e não o *fitness*. A relembrar que o *novelty score* é a distância do comportamentos dos indivíduos da população aos comportamentos dos K-vizinhos mais próximos. Para isso foi preciso definir o comportamento de cada

genoma, o arquivo que guarda os comportamentos de acordo com um critério e K-vizinhos.

O comportamento ficou definido como as distâncias finais dos predadores à presa em todos os ensaios que corresponde a um vetor:
`[[E1D1x, E1D1y, E1D2x, E1D2y, E1D3x, E1D3y], ..., [E9D1x, E9D1y, E9D2x, E9D2y, E9D3x, E9D3y]]`
Em que En é o número do ensaio, Dxn e Dn o número de casas do predador n em x e em y à presa no final do ensaio. O arquivo ficou a guardar os comportamentos aleatórios de qualquer individuo, uma geração de cada vez. O K assumiu o valor de 10 vizinhos. O *novelty-score* que é calculado percorrendo todo o espaço de comportamentos da geração atual mais os comportamentos que, usando um arquivo, foram guardados ao acaso um por geração de um genoma aleatório, escolhendo os k-vizinhos mais próximos para o calculo da distância de cada comportamento aos seus mais próximos para obter um valor da sua "originalidade"quantificada.

Cálculo da novidade Para o cálculo da novidade foi usada uma função C.2, de determinação dos comportamentos do genoma C.3, que guarda todos os comportamentos numa geração antes do cálculo do *novelty score*. Deste modo no primeiro ciclo, os comportamentos de todos os genomas numa geração são guardados numa lista, e no segundo ciclo, o *novelty score* é calculado para cada genoma e retornado para posterior seleção dos genomas mais aptos para a próxima geração. Assim os vizinhos mais próximos são determinados com maior precisão.

4.2.5 Primeiros resultados da novidade

Tabela 4.5: Parâmetros da 2ª experiência novidade

	número de entradas arquivo	nº população	nº gerações
random + gen	1500	500	1500
archive + gen	1500	500	1500
gen only	NULL	500	1500

Tabela 4.6: Resultados da 2ª experiência novidade, segunda abordagem

	G	A+G	R
pM			
MHCP	2.5544(1c)	2.3019(1c)	1.7042(0c)
MHCE	3.0739	2.8027	6.0761
MHe1o	2.772(1c)	3.167	9.1022
MHe5o	2.89776	3.2342	2.0169

A: só arquivo; G: só geração atual; A+G: arquivo mais geração atual; R: comportamento ao acaso por geração para arquivo; pP: presa move-se quando predadores estão próximos; pM: presa move-se normalmente; MHCP: Modelo para população homogéneo com comunicação predefinida; MHCE: Modelo para população homogéneo com evolução da comunicação;

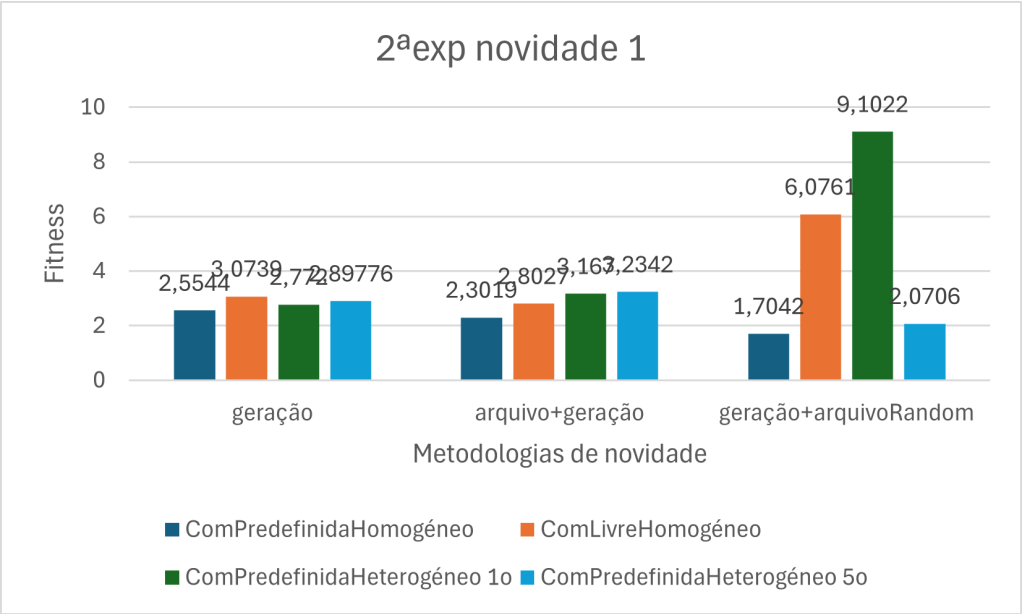

Figura 4.15: Resultados da 2ª experiência novidade 1

Avaliação dos primeiros resultados Depois de testar todos os métodos de cálculo de novidade propostos, como a abordagem de uso de um arquivo com comportamentos aleatórios um por geração mais os comportamentos da geração atual obteve substancialmente melhores resultados que as outras abordagens, com 4 capturas no modelo heterogéneo com um output por predador, esta passou a ser a única abordagem a ser usada em experiências futuras neste âmbito.

4.2.6 Segundos resultados

Após escolher a abordagem de uso do arquivo com comportamentos aleatórios, um por geração, como método de aplicação da novidade, fiz mais experiências aplicando as várias metodologias de rede neuronal para averiguar qual a melhor a achar soluções para o problema.

Avaliação dos segundos resultados Estes resultados apresentam uma melhoria significativa dos modelos treinados na sua generalidade sobre os modelos treinados na primeira abordagem 4.6. O modelo neuronal para equipa heterogéneo foi o que obteve melhores resultados funcionando bastante bem com *novelty search* e obtendo comportamentos mais sofisticados de dispersão de predadores a fim de flanquear a presa e captura-la eficazmente. O modelo neuronal homogéneo com evolução da comunicação também obteve melhores resultados e apresentou comportamento coordenados entre predadores.

4.2.7 Experiência adicional

Numa tentativa de otimizar mais o treino dos modelos, foi mudada a descrição do comportamento dos agentes para que, para além de terem a distância relativa final de cada predador à presa no fim de cada ensaio, também tenham as suas distâncias relativas à presa a meio de cada ensaio, au-

Tabela 4.7: 2º Resultados da segunda experiência novidade arquivo comportamentos aleatórios

	1	2	3	tempo de execução (média)
pP				
MHCP	5.463(2c)	3.159(0c)	3.195(0c)	171.83 sec
MHCE	5.0326(2c)	2.4838(1c g32)	2.9654(1c g127)	29.23sec
MHe1o	4.1591(1c)	9.3747(4c)	6.9522(3c)	31.49sec
MHe5o	7.7965(4c)	7.2500(3c)	6.9107(3c)	33.13sec
pM				
MHCP	4.5386(1c)	2.9902(1c)	1.8711(1c)	110sec
MHCE	6.0761(3c)	4.7879(2c)	5.309(2c)	30sec
MHe1o	9.102(4c)	6.820(3c)	8.450(4c)	34sec
MHe5o	2.017(0c)	2.071(0c)	2.171(0c)	35sec

pP: presa move-se quando predadores estão próximos; pM: presa move-se normalmente; MHCP: Modelo para população homogéneo com comunicação predefinida; MHCE: Modelo para população homogéneo com evolução da comunicação;

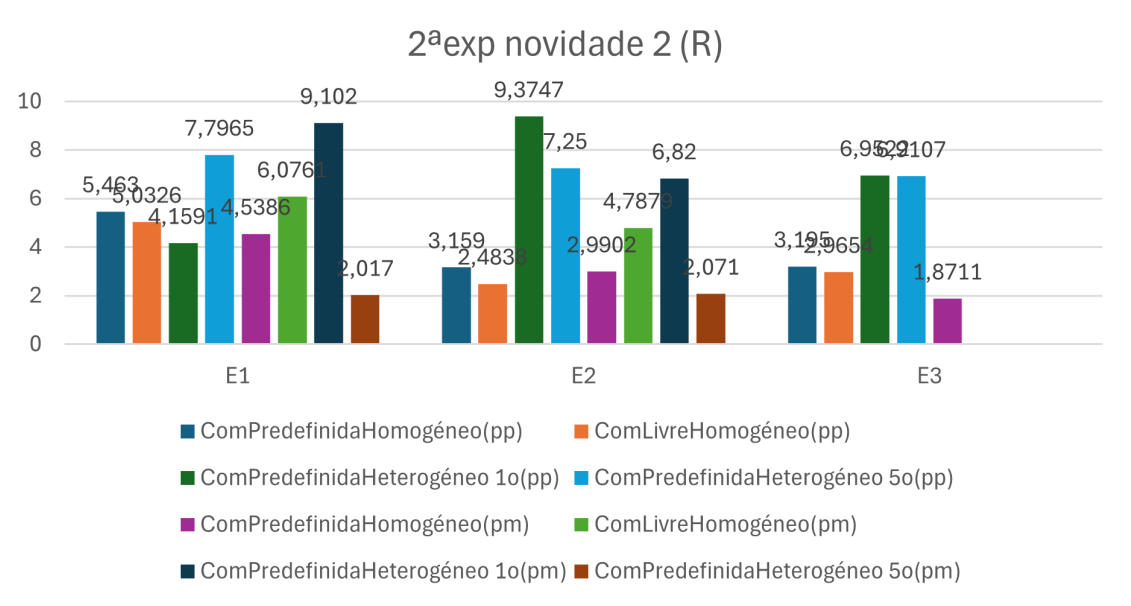

Figura 4.16: Resultados da 2ª experiência novidade 2

mentando o detalhe na descrição do comportamento de cada genoma para o cálculo mais rigoroso da novidade, na expectativa de aumentar o espaço de procura e diversidade de comportamentos resultantes. Assim o comportamento de cada individuo da evolução é representado pelo seguinte vetor:

```
[ [[E1D1xM, E1D1yM, E1D2xM, E1D2yM, E1D3xM, E1D3yM], [E1D1xM, E1D1yF,
E1D2xF, E1D2yF, E1D3xF, E1D3yF]], ..., [[E9D1xM, E9D1yM, E9D2xM, E9D2yM,
E9D3xM, E9D3yM], [E9D1xM, E9D1yF, E9D2xF, E9D2yF, E9D3xF, E9D3yF]]
]
```

Em que En é o número do ensaio, Dxn e Dyn o número de casas do predador n em x e em y, M

corresponde às medições a meio do ensaio e F às medições no final do ensaio.

Para esta experiência foi usada a abordagem de uso de arquivo com comportamentos aleatórios 1 por geração, ao longo de 1500 gerações, numa população de 500 genomas. A presa move-se quando os predadores estão a:

```
10 * STEP
```

De distância desta.

Tabela 4.8: Resultados da experiência novidade adicional gens/capturas

	1E	2E	3E	4E	5E	6E	7E	8E	9E	10E
1 eval										
MHCE	g63	g5	g323	g134	g52	g68	g41	g247	g298	g43
MHe1o	g23	g1	g0	g7	g4	g8	g11	g0	g0	g10
9 evals captures										
	1E	2E	3E	4E	5E	6E	7E	8E	9E	10E
MHCE	2c	4c409g	94c1495g	4c1063g	4c701g	4c938g	4c1314g	4c947g	2c104g	1c0g
MHe1o	6c	6c286g	7c681g	7c381g	7c1070g	7c453g	6c1053g	5c534g	7c1005g	5c1018g

MHe1o: Modelo para população heterogéneo com 1 output por predador; MHCE: Modelo para população homogéneo com evolução da comunicação;

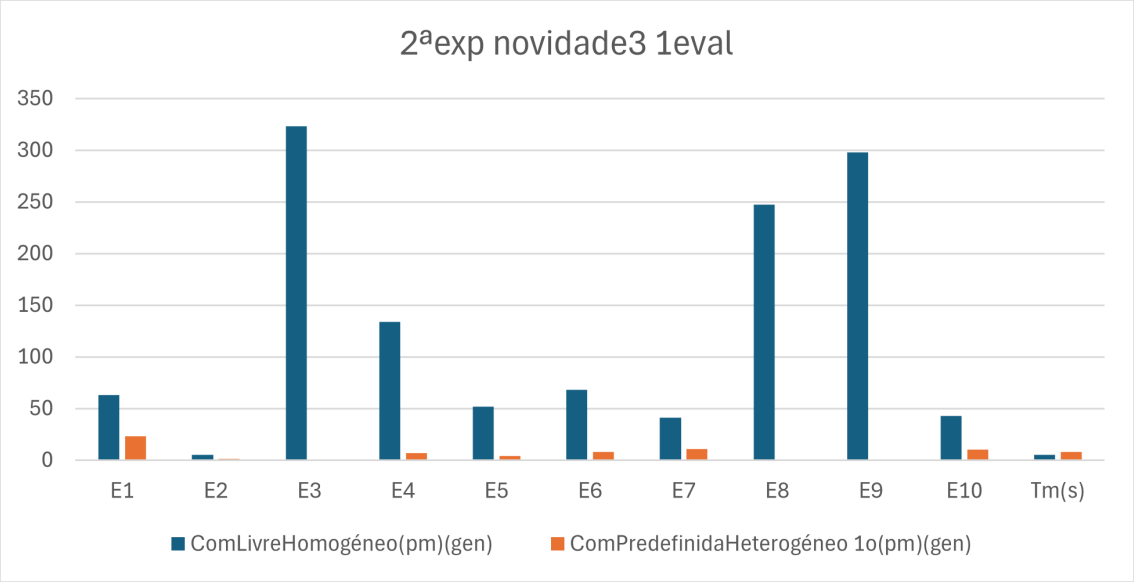

Figura 4.17: Resultados da experiência novidade bónus 1 avaliação

Resultados

Avaliação dos resultados Houve uma melhoria muito significativa dos resultados dos modelos treinados, com a rede neuronal para agentes heterogénea a conseguir capturar 7 das 9 presas possíveis. Somente com esta ligeira alteração o desempenho melhorou bastante o que torna mais

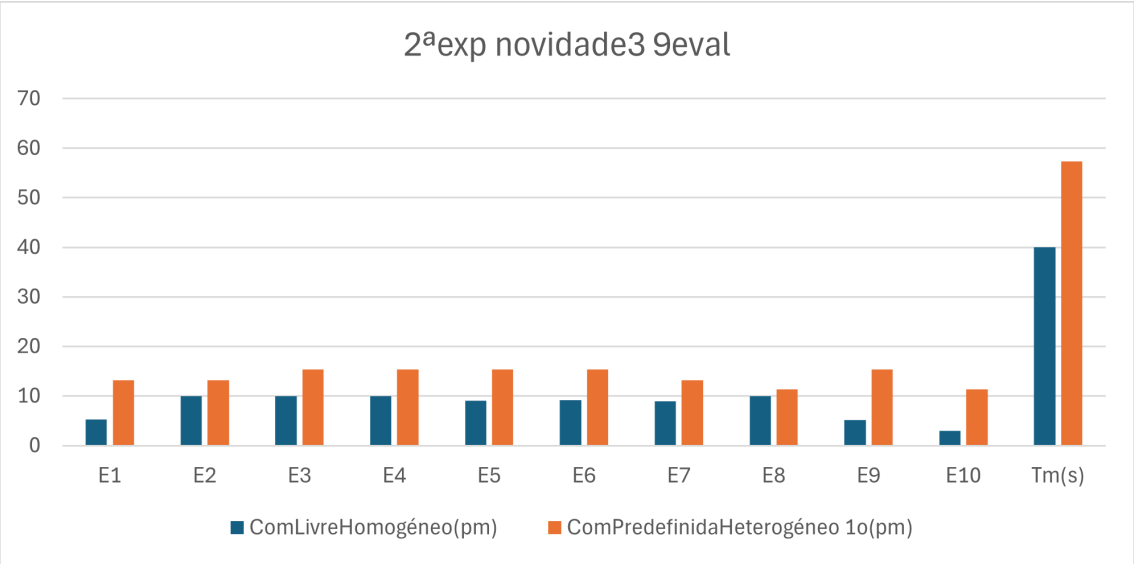

Figura 4.18: Resultados da experiência novidade bónus 9 avaliações

promissor o alcance da solução ótima se forem implementadas mais otimizações ao modelo neuro-evolutivo (e mais testes). O modelo heterogéneo mesmo assim obteve em média melhores resultados que o modelo homogéneo com evolução da comunicação, tanto na experiências com 1 ensaio com menos tempo (gerações) necessário para alcançar a solução, e nas experiências de 9 ensaios com maior número de capturas da presa em média. Tanto o modelo homogéneo com comunicação como os modelos heterogéneos demonstraram comportamentos mais sofisticados de bloqueadores, perseguidores (*blocker/pursuers*) em que uns predadores assumiam o comportamento de tentativa de limitação da presa para que os outros predadores, ao persegui-la, a conseguissem capturar. De notar que no modelo heterogéneo cada agente pareceu desempenhar a mesma função ao longo de todos os ensaios, enquanto que no modelo homogéneo as funções de cada predador mudaram em cada ensaio, dependendo do contexto do ambiente, o que lhe confere maior versatilidade.

4.3 3ª experiência

A 3ª experiência de captura é a mais exigente do trabalho, também com um foco na comparação direta entre duas abordagens de evolução, uma usando modelo de neuro-evolução numa população de agentes heterogénea e outro usando modelo de neuro-evolução numa população de agentes homogénea com comunicação e o teste com novidade, mas numa tarefa ainda mais difícil em que a captura só se dá caso a presa fique completamente cercada e os agentes não se podem atravessar.

4.3.1 Metodologia

O ambiente desta experiência é igual à anterior 4.2, um espaço bidimensional com 100 casas no comprimento e na largura. Com o objetivo de captura por cerco da presa, o cálculo de *fitness* é feito do mesmo modo que na experiência anterior dando vantagem às soluções em que os predadores estão mais próximos da presa. Neste caso são 4 predadores em vez dos três da 2ª experiência, e nenhum dos agentes pode se atravessar e não pode haver mais do que um agente na mesma posição. Isto é cumprido com uma função que verifica se um agente se está a mover para uma casa já ocupada sempre que se move, se for esse o caso o agente em causa mantém-se na mesma casa onde estava. Nesta experiência foram calculadas as distâncias euclidianas em contraste ao cálculo das distâncias ortogonais da experiência anterior, para tornar o movimento da presa mais determinístico. O ambiente da experiência e os tipos de modelos de redes neuronais usados foram idênticos à experiência anterior, dois modelos homogéneos, um com comunicação predefinida e outro com comunicação livre com evolução da linguagem, e dois modelo heterogéneos, um com um output para determinar o movimento por predador e outro com 5 outputs por predador, com as adaptações necessárias para 4 predadores em vez de os 3 usados na experiência 2. Esta experiência foi sujeita a dois cenários, um primeiro de controlo em que a presa não se move e um segundo em que a presa só se move quando um qualquer predador se encontra a uma distância à presa de 10 casas.

Movimento da presa O comportamento da presa é regido pelo mesmo algoritmo que na segunda experiência 4.2.1.

4.3.2 Resultados baseados no objetivo

Os 4 modelos usados foram treinados uma vez em cada um dos 2 cenários com uma população de 500 ao longo de 1000 gerações. Em cada execução foram usados 9 ensaios em que a presa começa numa posição diferente num dos 9 sectores distintos do ambiente bidimensional (fig 4.9).

É possível visualizar o comportamento do modelo treinado neste repositório [5].

[5] https://www.dropbox.com/scl/fo/ll975494f1gat42ki7g4w/AFJj8eGdi59dzmnFB-e5VsA?rlkey=p7s2ufw0lu9p5l4n0rbqgyebs&st=286i4194&dl=0

Tabela 4.9: Resultados da 3ª experiência

Mode/Scenario	pNM	pP	tempo(sec)
NCHo	Na	3.953	281.201 sec
NCHe	Na	3.345	78.641 sec
NCHe5o	Na	4.279	81.219 sec
CHo	2.974	3.431	74.830 sec
fitness threshold	20		I

pNM: Presa não se move; pP: presa move-se quando predadores estão próximos; NCHo: NoComHomogéneo; NCHe: NoComHeterogéneo; NCHe5o: NoComHeterogéneo com5outputs; CHo: ComHomogéneo

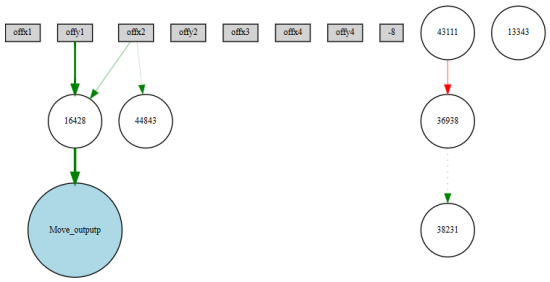

Figura 4.19: ANN resultante do modelo homogéneo comunicação predefinida pM

4.3.3 Avaliação dos resultados baseados no objetivo

Com estes resultados é possível verificar que a metodologia de evolução adotada, neste problema de captura de uso desta simples métrica de avaliação baseada na aptidão (*Fitness*), sem treino prévio em problemas mais simples, partindo do zero, é insuficiente em todos os modelos aplicados desde os homogéneos e heterogéneos sem comunicação até ao modelo com comunicação homogéneo. Mesmo no caso em que a presa não se move a solução só foi parcialmente encontrada. Claramente que um máximo local foi atingido, nos modelos heterogéneos e no modelo homogéneo com evolução da comunicação, que leva os predadores a tentarem se afastar uns dos outros numa tentativa de encurralar a presa mas não têm a coordenação suficiente para a cercarem intei-

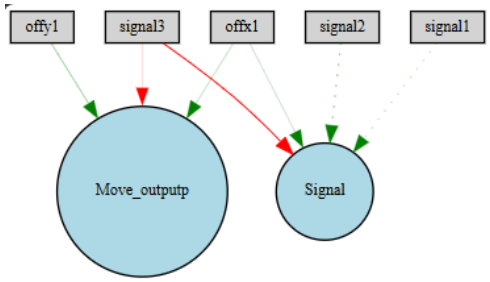

Figura 4.20: ANN resultante do modelo homogéneo comunicação livre pM

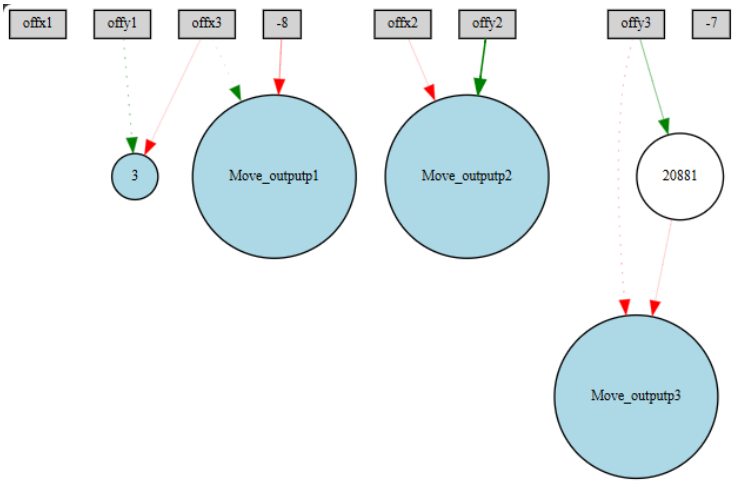

Figura 4.21: ANN resultante do modelo heterogéneo comunicação predefinida pM

Figura 4.22: ANN resultante do modelo heterogéneo 5o comunicação predefinida pM

ramente ao colocarem-se cada um numa das 4 posições adjacentes à presa. Atingiram uma aptidão a rondar o valor 3 muito abaixo do *threshold* de 20.

A comunicação desenvolvida entre os agentes no modelo em que se deu evolução da linguagem foi falida, não ajudou na coordenação dos agentes na captura da presa e não contribuiu para o êxito dos modelos treinados. No cenário em que a presa foge quando na proximidade de predadores, um dos predadores mais autónomos enviou consistentemente um valor de sinal alto para os outros e foi possível verificar que foi o agente com comportamento mais diferenciado dos outros, perseguindo mais ativamente a presa. No caso em que a presa não se move, apesar da maior simplicidade do cenário, os agentes não tiveram coordenação suficiente para atingir o objetivo de cercar a presa nas quatro posições adjacentes a esta. A comunicação resultante da evolução não contribuiu para a coordenação necessária entre os predadores para obtenção da solução. As ANNs resultantes dos modelos são muito semelhantes à 2ª experiência, com os predadores resultantes do modelo homogéneo com comunicação predefinida a assumirem exatamente os mesmos movimentos por turno (fig. 4.19), o modelo homogéneo com evolução da comunicação a possibilitar maior distinção de comportamentos entre predadores (fig. 4.20) e os modelos heterogéneos em que se verifica a maior diferenciação de movimentos entre os predadores (fig. 4.22) (fig. 4.21).

4.3.4 Metodologia da novidade

Neste caso a implementação da novidade deu-se assim:
O comportamento ficou definido como as distâncias finais dos predadores à presa em todos os

ensaios que corresponde a um vetor à semelhança da 2ª experiência mas com 4 predadores em vez de 3:

`[[E1D1x, E1D1y, E1D2x, E1D2y, E1D3x, E1D3y, E1D4x, E1D4y], ..., [E9D1x, E9D1y, E9D2x, E9D2y, E9D3x, E9D3y, E9D4x, E9D4y]]`

Em que En é o número do ensaio, Dxn e Dn o número de casas do predador n em x e em y à presa no final do ensaio. O arquivo ficou a guardar os comportamentos aleatórios de qualquer individuo, uma geração de cada vez. O K, neste âmbito, também assumiu o valor de 10 vizinhos depois de averiguar ser o valor mais consiste com uma análise preliminar.

O *novelty-score* foi calculado percorrendo todo o espaço de comportamentos da geração atual mais os comportamentos que, usando um arquivo, foram guardados ao acaso um por geração de um genoma aleatório, escolhendo os k-vizinhos mais próximos para o calculo da distância de cada comportamento aos seus mais próximos para obter um valor da sua "originalidade"quantificada. Foi escolhido k=10 de número de vizinhos, porque foi o mais consistente após análise preliminar.

Tabela 4.10: Parâmetros da 3ª experiência novidade 1 avaliação e 9 avaliações

	número de entradas arquivo	nº população	nº gerações
random + gen	1000	500	1000
archive + gen	1000	500	1000
gen only	NULL	500	1000

4.3.5 Resultados da novidade

Tabela 4.11: Resultados da 3ª experiência de novidade, 1 avaliação

	1	2	3	tempo de execução(média)
pP				
MHCP	6.3864(0c)	5.2824	5.3468	17sec
MHCE	4.107	5.947	5.8847	6sec
MHe1o	5.109	5.169	5.146	7sec
MHe5o	6.02834(g314)	4.5884	4.9514	8sec
pM				
MHCP	3.8742	6,7131	6,1796	18sec
MHCE	5.951(0c)	5.9847	6.1807	7sec
MHe1o	6.168	5.635	5.959	8sec
MHe5o	4.7473	6.1539	6.2365	9sec

pP: presa move-se quando predadores estão próximos; pM: presa move-se normalmente; MHCP: Modelo para população homogéneo com comunicação predefinida; MHCE: Modelo para população homogéneo com evolução da comunicação;

4.3.6 Avaliação de resultados da novidade

Houve um aumento significativo, em todas as experiências, de diversidade de comportamentos dos predadores ao longo do treino do modelo com aumentos no *fitness* final atingido comparativamente

Tabela 4.12: Segundos resultados da 3ª experiência da novidade, 9 avaliações

	result(fitness)	tempo de execução(média)
pM		
MHCP	4.5079(0c)	156sec
MHCE	4.5543(0c)	87.441 sec
MHe1o	4.5621(0c)	59.083sec
MHe5o	4.1318(0c)	91.842 sec

pP: presa move-se quando predadores estão próximos; pM: presa move-se normalmente; MHCP: Modelo para população homogéneo com comunicação predefinida; MHCE: Modelo para população homogénea com evolução da comunicação;

aos modelos orientado a objetivos. Apesar disso, em todos os modelos neuronais testados, houve uma incapacidade generalizada em atingir o objetivo de cercar por completo a presa em todas as direções, nenhum dos modelos treinados atingiu o nível de sofisticação de comportamento coordenado necessário entre os predadores para chegar ao sucesso da tarefa de captura pretendida. Com resultados que declararam uma insuficiência, perante os parâmetros experimentados, destes modelos de por si só acharem a solução ao problema da captura em cerco, não fez sentido para o âmbito deste estudo prosseguir com uma versão do problema ainda mais complexa e difícil, em que os agentes podem assumir qualquer ângulo de direção, uma aceleração e diferentes velocidades.

4.4 Conclusões dos resultados das experiências baseadas no objetivo

Olhando para o resultado das experiências que envolveram o desenvolvimento e treino de ANNs usando *fitness* e porventura a comunicação no problema da captura e cerca da presa em movimento, concluo que o uso do *fitness* como métrica para a seleção dos indivíduos da população e sem evolução previa incremental é insuficiente, como é possível observar no máximo local atingido nestas experiências a rondar os 30, valor muito baixo. Apesar disto foi possível verificar que o uso de comunicação nos modelos neuro-evolucionários impulsionou uma maior diversidade de comportamentos nos agentes treinados demonstrando, por isso, maior nível de coordenação do modelo para população homogénea com evolução da comunicação, comparável com os comportamentos dos agentes treinados com um modelo para população heterogénea confirmando a hipótese de que é possível treinar em equipa usando um modelo que apesar de homogéneo, aplica comunicação o que permite maior nuance de comportamentos à semelhança dos modelos heterogéneos.

4.5 Avaliação da novidade

Relativamente à segunda experiência de captura houve uma melhoria significativa dos resultados dos modelos treinados com o uso de novidade em comparação com os modelos que só usaram aptidão como métrica de evolução dos modelos neuro-evolucionários. Com esta pequena mas poderosa alteração de uso de novidade os modelos ficaram mais consistentes na captura das presas tendo aumentado o número de capturas nos ensaios, na generalidade dos modelos, com o melhor

modelo treinado a obter 7 capturas em 9 possíveis, resultado nunca obtido com abordagem orientada a objetivos. Assim foi achada uma resolução parcial do problema de captura. De notar que a novidade de facto permitiu uma maior diversificação de comportamentos verificados pelos modelos durante o processo evolutivo quando comparado com o uso do *fitness* como único critério de seleção, o que é muito benéfico na procura da solução ótima e fuga aos ótimos locais. A experiência adicional 4.2.7 foi a que, de longe, alcançou resultados mais promissores, mais próximos do procurado, demonstrado de facto o impacto da novidade no processo evolutivo, e como força que mitiga máximos locais, permite uma evolução mais eficaz sem recorrer a evolução incremental.

Com a observação dos resultados obtidos na 3ª experiência de novidade 4.12, foi aferido que com as condições definidas com que se dá a captura e é atingido o objetivo, uma vez que são tão especificas e exigem elevados níveis de coordenação, é no mínimo improvável que através de uma evolução do modelo neuro-evolucionário partindo do zero que controla 4 predadores, (que usa novidade para uma maximização do número de comportamentos diferenciados entre genomas e que depende de algum acaso), seja encontrada a descrição exata do comportamento coletivo dos predadores para a captura sistemática e ideal da presa. Para a resolução de problemas de acrescida complexidade e dificuldade, como o problema final aqui testado, é necessário, mais do que a evolução sistemática, e aprendizagem reforçada e não supervisionada dos agentes, mesmo com novidade, uma evolução algo incrementada e controlada do modelo treinado. Com as capacidades atuais de treino destes modelos surge o conhecido conflito entre generalizar o modelo mas não achar a solução ótima para os problemas, e especificar o modelo para ser ótimo na resolução de um problema ou caso mas insuficiente para todos os outros.

Capítulo 5

Conclusões

5.1 Conclusão

Concluindo, estes modelos de neuro-evolução baseados na novidade são poderosos e na sua generalidade são melhores do que os modelos neuro-evolucionários *objective-based* para muitos domínios de problemas, mais especialmente aqueles em que há uma maior expressividade de comportamentos por parte dos agentes no espaço do problema, por garantir uma maior exploração de todos os comportamentos possíveis. Os modelos com novidade treinados com um espaço de comportamentos mais descrito obtiveram consistentemente melhores resultados tendo quase atingido a solução ótima 4.2.7, demonstrando o potencial do uso da novidade em evolução reforçada na cobertura de problemas com um espaço de comportamentos muito largo, especialmente se a descrição do comportamento dos indivíduos que constituem a evolução forem muito bem descritos e completos. Os modelos neuronais heterogéneos treinados, no geral, demonstraram melhores resultados que os modelos homogéneos com comunicação livre apesar de tentativas de obter bons comportamentos com o último. Na generalidade, os resultados obtidos ficaram aquém dos resultados obtidos nas experiências deste domínio encontrados [27, 1], com os modelos finais treinados a não conseguirem atingir o objetivo final pretendido e não atingirem o valor de aptidão desejado. Em causa está o facto de estas experiências não terem a ajuda de evolução incremental e partirem do zero na resolução destes problemas complexos aqui apresentados, ao contrário da maioria dos outros estudos [1]. Apesar deste projeto merecer continuidade, o meu trabalho teve sucesso em aferir a utilidade e importância do uso da novidade em contrapartida à aptidão, e parcialmente à evolução incremental, no treino de modelos (neuro-evolucionários), na resolução de problemas complexos em que o espaço de comportamentos é extenso, assim como no uso de comunicação livre para o desenvolvimento de comportamentos mais cooperantes e coordenados entre agentes no contexto de problemas de equipas em modelos homogéneos, em oposição ao uso de comunicação predefinida ou nenhuma comunicação de todo. Estas abordagens demonstraram maior taxas de sucesso na resolução dos problemas aqui explorados, como alternativas às abordagens mais comuns.

5.2 Trabalho futuro

Importante melhorar o código usado para a execução, definição destas experiências para as tornar mais extensíveis, com mais documentação para facilitar leitura e compreensão e tornar mais modular para permitir mais fáceis e menos trabalhosas alterações às experiências e facilitar a execução das várias e pequenas variantes destas experiências como o modelo neuro-evolucionário usado, o número de ensaios por avaliação do modelo, número de gerações e população, método de cálculo de novidade, definição dos comportamentos dos genomas, *et cetera*. Acredito ser possível atingir a solução ótima na segunda experiência 4.2, que já estive próximo de atingir com novidade, com um aumento na pormenorização da descrição do comportamento dos predadores não para 2 momentos (distâncias de cada predador à presa nos momentos intermédio e final) de cada ensaio, mas 4 ou mais, garantindo uma maior diversificação do espaço comportamental no processo evolutivo do modelos neuronais. O problema aqui proposto de captura em cerca 4.3, deve ser prosseguido, aumentado substancialmente o número de gerações do processo evolutivo e o tempo de treino do modelo, com melhorias também na descrição dos comportamentos dos agentes para um mais preciso cálculo da novidade para incentivar uma maior diversidade de comportamentos diferentes ao longo do processo evolutivo. Deste modo é garantida uma constante renovação dos genomas para evitar uma estagnação na evolução (máximo local), aumentando a probabilidade de encontrar a solução. Se não der fruto é então altura de recorrer a uma evolução mais incremental e por isso mais supervisionada dos modelos propostos o que vai contra a intenção original deste estudo. Também seria interessante experimentar este mesmo problema usando outras bibliotecas e metodologias de neuro-evolução[36] [37], assim como variantes do NEAT, para além da extensivamente usada neste estudo 3.2. Seria também muito interessante a complexificação desta experiência para um cenário de ambiente contínuo com agentes a usar sensores com perceção limitada do ambiente e descrição de movimento vetorial com valores de velocidade e aceleração, para eventual integração num cenário real com robôs reais.

Bibliografia

[1] Chern Han Yong and Risto Miikkulainen. Coevolution of role-based cooperation in multiagent systems. *IEEE Transactions on Autonomous Mental Development*, 1(3):170–186, 2009.

[2] Alhanoof Althnian and Arvin Agah. Evolving goal-driven multi-agent communication: what, when, and to whom. *Evolutionary Intelligence*, 9(4):181–202, 2016.

[3] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317–342, 1997.

[4] David T Cli, Inman Harvey, and Philip Husbands. Incremental evolution of neural network architectures for adaptive behaviour. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'93)*, pages 39–44, 1992.

[5] Nick Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive behavior*, 6(2):325–368, 1997.

[6] Geoff Nitschke. Emergence of cooperation: State of the art. *Artificial Life*, 11(3):367–396, 2005.

[7] Dario Floreano, Phil Husbands, and Stefano Nolfi. Evolutionary robotics. *Handbook of robotics*, 2008.

[8] Stephane Doncieux, Nicolas Bredeche, Jean-Baptiste Mouret, and Agoston E Eiben. Evolutionary robotics: what, why, and where to. *Frontiers in Robotics and AI*, 2:4, 2015.

[9] Kenneth O Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of artificial intelligence research*, 21:63–100, 2004.

[10] Yiming Peng, Gang Chen, Harman Singh, and Mengjie Zhang. Neat for large-scale reinforcement learning through evolutionary feature learning and policy gradient search. In *Proceedings of the genetic and evolutionary computation conference*, pages 490–497, 2018.

[11] Iaroslav Omelianenko. *Hands-On Neuroevolution with Python: Build high-performing artificial neural network architectures using neuroevolution-based algorithms*. Packt Publishing Ltd, 2019.

[12] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[13] Donald O Hebb. The first stage of perception: growth of the assembly. *The Organization of Behavior*, 4(60):78–60, 1949.

[14] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[15] Philip J Drew and John RT Monson. Artificial neural networks. *Surgery*, 127(1):3–11, 2000.

[16] Myeong-Wuk Jang, Amr Ahmed, and Gul Agha. Efficient agent communication in multi-agent systems. In *Software Engineering for Multi-Agent Systems III: Research Issues and Practical Applications 3*, pages 236–253. Springer, 2005.

[17] C Lee Giles and Kam-Chuen Jim. Learning communication for multi-agent systems. In *Innovative Concepts for Agent-Based Systems: First International Workshop on Radical Agent Concepts, WRAC 2002, McLean, VA, USA, January 16-18, 2002. Revised Papers 1*, pages 377–390. Springer, 2003.

[18] Kyle Wagner. Cooperative strategies and the evolution of communication. *Artificial Life*, 6(2):149–179, 2000.

[19] Marcus Hutter and Shane Legg. Fitness uniform optimization. *IEEE Transactions on Evolutionary Computation*, 10(5):568–589, 2006.

[20] Paulo Urbano and Henrique Vaz. Using fitness as minimal criteria for novelty search in the maze navigation domain.

[21] Joel Lehman and Risto Miikkulainen. Neuroevolution. *Scholarpedia*, 8(6):30977, 2013.

[22] Lishuang Wang, Mengfei Zhao, Enyu Liu, Kebin Sun, and Ran Cheng. Tensorized neuroevolution of augmenting topologies for gpu acceleration. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1156–1164, 2024.

[23] Michael Merry, Patricia Riddle, and James Warren. Propneat-efficient gpu-compatible backpropagation over neuroevolutionary augmenting topology networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 425–433, 2025.

[24] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[25] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[26] David Filliat, Jérôme Kodjabachian, Jean-Arcady Meyer, et al. Incremental evolution of neural controllers for navigation in a 6-legged robot. In *Proceedings of the Fourth International Symposium on Artificial Life and Robots*, pages 753–760. Oita University Press Oita, Japan, 1999.

[27] Thomas Haynes and Sandip Sen. The evolution of multiagent coordination strategies. *Adaptive Behavior*, 315, 1997.

[28] Miroslav Benda. On optimal cooperation of knowledge sources: an empirical investigation. *Technical Report, Boeing Advanced Technology Center*, 1986.

[29] Thomas Haynes, Sandip Sen, et al. Crossover operators for evolving a team. *Genetic programming*, 199, 1997.

[30] Jörg Denzinger and Matthias Fuchs. *Experiments in learning prototypical situations for variants of the pursuit game*. Technische Universität Kaiserslautern, Fachbereich Informatik, 1999.

[31] Shin I Nishimura and Takashi Ikegami. Emergence of collective strategies in a prey-predator game model. *Artificial Life*, 3(4):243–260, 1997.

[32] Joel Lehman and Kenneth O Stanley. Novelty search and the problem with objectives. *Genetic programming theory and practice IX*, pages 37–56, 2011.

[33] Joel Lehman and Kenneth O Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218, 2011.

[34] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. Novelty-driven cooperative coevolution. *Evolutionary computation*, 25(2):275–307, 2017.

[35] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[36] Sander Koelstra, Christian Muhl, Mohammad Soleymani, Jong-Seok Lee, Ashkan Yazdani, Touradj Ebrahimi, Thierry Pun, Anton Nijholt, and Ioannis Patras. Deap: A database for emotion analysis; using physiological signals. *IEEE transactions on affective computing*, 3(1):18–31, 2011.

[37] Mark A Coletti, Eric O Scott, and Jeffrey K Bassett. Library for evolutionary algorithms in python (leap). In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 1571–1579, 2020.

Apêndice A

Funções utilizadas nas experiências efetuadas

```
1
2  #File containing all functions used by the capture experience
3
4  #DIST is The dimensions of map, onlyapplicable if map is square
5  DIST = 200
6  #width of map
7  WIDTH = 200
8  #height of map
9  HEIGHT = 200
10 #Step how much the agents(preds and prey) move per turn. Should
       allways be DIST/50
11 STEP = DIST / 50
12 #DIRECTIONS
13 DIRECTIONS = [(STEP,0), (-STEP, 0), (0, STEP), (0,-STEP)]
14
15 def createpredators_bottom(height, width, n, step):
16     return [Agent((-width+ predx * step, -height)) for predx in
           range(n)]
17
18 def createpredators_right(height, width, n, step):
19     return [Agent((width - predx * step, height)) for predx in
           range(n)]
20
21 def createpredators_edges(height, width, n, step):
22     pass
23
24 def createpreds_parasimularchoque(step):
25     return [Agent((step, 0)), Agent((step*2, 0)), Agent((step*3, 0)),
           Agent((step*4, 0))]
26
27 def prey_parasimularchoque():
28     return Agent((0, 0))
29
30 def createPrey(height, width, preds, step):
31     encontrei = False
32     while True:
33         presa = Agent((random.randint(-width//step, width//step) *
               step, random.randint(-height//step, height//step) * step))
```

```
34          if presa.get_coords() not in [pred.get_coords() for pred in
                preds]:
35              return presa
36
37  def createPreys(height, width, preds, step, n):
38      return [createPrey(height, width, preds, step) for _ in range(n)]
39
40  def createPreys9(height, width, preds, step):
41      segmentw = width*2/3
42      segmenth = height*2/3
43      prey1 = Agent((random.randint(-width//step, (-width+
            segmentw)//step) * step, random.randint(-height//step,
            (-height+segmenth)//step) * step))
44      prey2 = Agent((random.randint((-width+ segmentw)//step, (width -
            segmentw)//step) * step, random.randint((-height//step),
            (-height+segmenth)//step) * step))
45      prey3 = Agent((random.randint((width - segmentw)//step,
            width//step) * step, random.randint(-height//step,
            (-height+segmenth)//step) * step))
46      prey4 = Agent((random.randint(-width//step, (-width+
            segmentw)//step) * step,
            random.randint((-height+segmenth)//step,
            (height-segmenth)//step) * step))
47      prey5 = Agent((random.randint((-width+ segmentw)//step, (width -
            segmentw)//step) * step,
            random.randint((-height+segmenth)//step,
            (height-segmenth)//step) * step))
48      prey6 = Agent((random.randint((width - segmentw)//step,
            width//step) * step, random.randint((-height+segmenth)//step,
            (height-segmenth)//step) * step))
49      prey7 = Agent((random.randint(-width//step, (-width+
            segmentw)//step) * step,
            random.randint((height-segmenth)//step, height//step) * step))
50      prey8 = Agent((random.randint((-width+ segmentw)//step, (width -
            segmentw)//step) * step,
            random.randint((height-segmenth)//step, height//step) * step))
51      prey9 = Agent((random.randint((width - segmentw)//step,
            width//step) * step, random.randint((height-segmenth)//step,
            height//step) * step))
52      return [prey1, prey2, prey3, prey4, prey5, prey6, prey7, prey8,
            prey9]
53
54  def turtle_agent(agent_coords, color= "blue", forma = "circle"):
55      ag = turtle.Turtle(shape = forma, visible= False)       # create a
            turtle named tpred
56      ag.color(color)          # make tess blue
57      ag.pensize(1)             # set the WIDTH of her pen
58
59      ag.penup()
60      ag.goto(agent_coords[0], agent_coords[1])
61
62      ag.showturtle()
63      ag.pendown()
64
65      ag.old_pos = ag.position()
66      ag.initial_pos = ag.position()
```

```
67      return ag
68
69  #change this function bellow it isnt optimal follow example from
        toroidalDistance_coords
70  #normal distance calculations
71
72  #distance between to coords, offsets are negative se a presa estiver
        em cima ou à direita do predador, e positivos se em baixo ou à
        esquerda do predador
73  def toroidalDistance1coord(coord1, coord2, dim):
74      dif = coord1 - coord2
75      if dif < 0:
76          if abs(dif) <= dim:
77              return dif
78          return  2*dim - abs(dif)
79      if dif <= dim:
80          return dif
81      return -(2*dim - dif)
82
83  #calculates toroidal distance given 2 positions in 2d space
84  def toroidalDistance_coords( pos1, pos2, width, height):
85      x1,y1 = pos1
86      x2,y2 = pos2
87      dx = abs(x1 - x2)
88      dy = abs(y1 - y2)
89
90      # Consider toroidal wrapping
91      toroidal_dx = min(dx, ((width*2)-dx +1))
92      toroidal_dy = min(dy, ((height*2)-dy+1))
93      #euclidian distance
94      return (toroidal_dx**2 + toroidal_dy**2)**0.5
95      #manhatan distance for ortogonal env
96      #return abs(toroidal_dx) + abs(toroidal_dy)
97
98  #only to check and update position of agents according to toroidal
        environment, no visualization
99  def toroidal_pos(pos, max_width, max_height):
100     '''
101     pega em pos e se sair fora dos limites volta a por dentro dos
            limites do outro lado(Toroide)
102     '''
103     x,y = pos
104     if abs(x) > max_width:
105         x = math.copysign(max_width-(abs(x) - max_width), x * -1)
106     if abs(y) > max_height:
107         y = math.copysign(max_height-(abs(y) - max_height), y * -1)
108     return x,y
109     #other way
110     #if x > max_width:
111     #    dif = x - max_width
112     #    x = -max_width + dif
113     #elif x < -max_width:
114     #    x = max_width + (x + max_width)
115     #if y > max_height:
116     #    dif = y - max_height
117     #    y = -max_height + dif
```

```
118     #elif y < -max_height:
119     #    y = max_height + (y + max_height)
120     #return x,y
121 
122 #to use only in simulation with visualization
123 def toroidalcheck(turtle, max_width, max_height, speed):
124     if abs(turtle.xcor()) > max_width:
125         turtle.hideturtle()
126         turtle.penup()
127         turtle.speed(max_width)
128         turtle.setx(math.copysign(max_width-(abs(turtle.xcor()) -
                max_width), turtle.xcor() * -1))
129         turtle.speed(speed)
130         turtle.showturtle()
131         turtle.pendown()
132     if abs(turtle.ycor()) > max_height:
133         turtle.hideturtle()
134         turtle.penup()
135         turtle.speed(max_height)
136         turtle.sety(math.copysign(max_height-(abs(turtle.ycor()) -
                max_height), turtle.ycor() * -1))
137         turtle.speed(speed)
138         turtle.showturtle()
139         turtle.pendown()
140 
141 def agent_go(agent, to_move, speed):
142     x,y = agent.pos()
143     x_togo,y_togo = to_move
144     if abs(x- x_togo) > WIDTH or abs(y- y_togo) > HEIGHT:
145         agent.hideturtle()
146         agent.penup()
147         agent.speed(WIDTH*2)
148         agent.goto(x_togo, y_togo)
149         agent.speed(speed)
150         agent.showturtle()
151         agent.pendown()
152     else:
153         agent.goto(x_togo, y_togo)
154 
155 # to move turtle object pred
156 def tpred_newpos(pred, output, step):
157     pred.old_pos = pred.pos()
158     if 0.2 > output >= 0:
159         to_move = (0,0)
160     elif 0.4 > output >= 0.2:
161         #print("predador move-se para este!")
162         to_move= (step,0)
163     elif 0.6 > output >= 0.4:
164         #print("predador move-se para norte!")
165         to_move= (0,step)
166     elif 0.8 > output >= 0.6:
167         #print("predador move-se para oeste!")
168         to_move= (-step,0)
169     else:
170         #print("predador move-se para sul!")
171         to_move= (0,-step)
```

```
172     newpos = tuple(a + b for a, b in zip(pred.pos(), to_move))
173     x,y = toroidal_pos(newpos, WIDTH, HEIGHT)
174     return x,y
175
176
177 # to calculate the new predator position not turtle object pred
178 def pred_newpos(pred, output, step):
179     x,y = pred.get_coords()
180     pred.old_coords = pred.get_coords()
181     if 0.2 > output >= 0:
182         #print("predador não se moveu!")
183         to_move = (0,0)
184     elif 0.4 > output >= 0.2:
185         #print("predador move-se para este!")
186         to_move= (step,0)
187     elif 0.6 > output >= 0.4:
188         #print("predador move-se para norte!")
189         to_move= (0,step)
190     elif 0.8 > output >= 0.6:
191         #print("predador move-se para oeste!")
192         to_move= (-step,0)
193     else:
194         #print("predador move-se para sul!")
195         to_move= (0,-step)
196     new_coords = tuple(a + b for a, b in zip(pred.get_coords(),
           to_move))
197     x,y =toroidal_pos(new_coords, WIDTH, HEIGHT)
198     return x,y
199
200 #to move not turtle object pred
201 #def pred_move(pred, output, step):
202 #     x,y = pred_newpos(pred, output, step)
203 #     pred.set_coords(x,y)
204
205
206 #to move turtle object prey
207 def tprey_newpos(prey, tpreds, step):
208     #calculating closest pred
209     closestdistance = toroidalDistance_coords(tpreds[0].old_pos,
           prey.pos(), WIDTH, HEIGHT)
210     closestpred = tpreds[0]
211     for tp in tpreds[1:]:
212         distance = toroidalDistance_coords(tp.old_pos, prey.pos(),
               WIDTH, HEIGHT)
213         if distance < closestdistance:
214             closestdistance = distance
215             closestpred = tp
216     #to make the prey move only when a predator is close enough to it
217     #if closestdistance > 10*STEP:
218     #     return prey.pos()
219
220     #para manter um registo da posição anterior
221     prey.old_pos = prey.pos()
222     xcoor,ycoor = prey.pos()
223
224     farthest = -1
```

```
225     farthestheading = None
226     directions = [((STEP,0), 0), ((-STEP, 0), 180), ((0, STEP), 90),
            ((0,-STEP), 270)]
227     random.shuffle(directions)
228     for ((ax, ay), a) in directions:
229         newpos = ax+xcoor , ay+ ycoor
230         newpos =toroidal_pos(newpos, WIDTH, HEIGHT)
231         dist =toroidalDistance_coords(closestpred.old_pos, newpos,
                WIDTH, HEIGHT)
232 
233         if dist > farthest:
234             farthest = dist
235             farthestheading = a
236             x_tomove, y_tomove = newpos
237     return x_tomove, y_tomove
238 
239 #to move not turtle object prey
240 def prey_newpos(prey, predadores, step):
241     #calculating closest pred
242     preds=copy.copy(predadores)
243     random.shuffle(preds)
244     closestdistance = toroidalDistance_coords(preds[0].old_coords,
            prey.get_coords(), WIDTH, HEIGHT)
245     closestpred = preds[0]
246     for tp in preds[1:]:
247         distance = toroidalDistance_coords(tp.old_coords,
                prey.get_coords(), WIDTH, HEIGHT)
248         if distance < closestdistance:
249             closestdistance = distance
250             closestpred = tp
251     #to make the prey move only when a predator is close enough to it
252     #if closestdistance > 10*STEP:
253     #    return prey.get_coords()
254 
255     #para manter um registo da posição anterior
256     prey.old_coords = prey.get_coords()
257     xcoor,ycoor = prey.get_coords()
258 
259     farthest = -1
260     farthestheading = None
261     random.shuffle(DIRECTIONS)
262     for (ax, ay) in DIRECTIONS:
263         newpos = ax+ xcoor , ay+ ycoor
264         newpos = toroidal_pos(newpos, WIDTH, HEIGHT)
265         dist =toroidalDistance_coords(closestpred.old_coords, newpos,
                WIDTH, HEIGHT)
266 
267         if dist > farthest:
268             x_tomove, y_tomove = newpos
269             farthest = dist
270     return x_tomove, y_tomove
271 
272 # to move turtle object pred with 5 outputs
273 def tpred_newpos5(tpred, outputs, step):
274     tpred.old_pos = tpred.pos()
275     biggerOutput = outputs[0]
```

```
276     biggerOutputN = 0
277     for n in range(1, len(outputs)):
278         if outputs[n] > biggerOutput:
279             biggerOutput = outputs[n]
280             biggerOutputN = n
281     if biggerOutputN == 0:
282         #print("predador não se moveu!")
283         to_move = (0,0)
284     elif biggerOutputN == 1:
285         #print("predador move-se para este!")
286         to_move = (step,0)
287     elif biggerOutputN == 2:
288         #print("predador move-se para norte!")
289         to_move = (0,step)
290     elif biggerOutputN == 3:
291         #print("predador move-se para oeste!")
292         to_move = (-step,0)
293     else:
294         #print("predador move-se para sul!")
295         to_move = (0,-step)
296     newpos = tuple(a + b for a, b in zip(tpred.pos(), to_move))
297     x,y = toroidal_pos(newpos, WIDTH, HEIGHT)
298     return x,y
299
300 # to move not turtle object pred with 5 outputs
301 def pred_newpos5(pred, outputs, step):
302     x,y = pred.get_coords()
303     pred.old_coords = pred.get_coords()
304     biggerOutput = outputs[0]
305     biggerOutputN = 0
306     for n in range(1, len(outputs)):
307         if outputs[n] > biggerOutput:
308             biggerOutput = outputs[n]
309             biggerOutputN = n
310     if biggerOutputN == 0:
311         #print("predador não se moveu!")
312         to_move = (0,0)
313     elif biggerOutputN == 1:
314         #print("predador move-se para este!")
315         #pred.set_coords(x+step, y)
316         to_move = (step,0)
317     elif biggerOutputN == 2:
318         #print("predador move-se para norte!")
319         #pred.set_coords(x, y+step)
320         to_move = (0,step)
321     elif biggerOutputN == 3:
322         #print("predador move-se para oeste!")
323         #pred.set_coords(x-step, y)
324         to_move = (-step,0)
325     else:
326         #print("predador move-se para sul!")
327         #pred.set_coords(x, y-step)
328         to_move = (0,-step)
329     new_coords = tuple(a + b for a, b in zip(pred.get_coords(),
        to_move))
330     x,y =toroidal_pos(new_coords, WIDTH, HEIGHT)
```

```
331     return x,y
332
333 #captura: a presa é capturada quando estiver cercada a norte sul este
        e oeste por predadores
334 def captura(preds, prey):
335     x,y = prey.get_coords()
336     vizinhas = [toroidal_pos((x+ dx, y+dy), WIDTH, HEIGHT) for dx,dy
            in DIRECTIONS]
337     for p in preds:
338         if p.get_coords() not in vizinhas:
339             return False
340     return True
341
342 def captura_t(preds, prey):
343     x,y = prey.pos()
344     vizinhas = [toroidal_pos((x+ dx, y+dy), WIDTH, HEIGHT) for dx,dy
            in DIRECTIONS]
345     for p in preds:
346         if p.pos() not in vizinhas:
347             return False
348     return True
349
350 #um agente choca com outro se os agentes forem diferentes um do outro
        e posições iguais entre eles
351 #ToDo otimizar para por de fora aqueles que já se sabe que não choca
352 def choca(agent, agents):
353     ag1, ag_pos1 = agent
354     for ag, ag_pos in agents:
355         if ag1 != ag and ag_pos1 == ag_pos:
356             return True
357     return False
358
359 # verificar se dois agentes se atravessaram
360 def atravessaram(agent, agents):
361     ag1, ag1_pos = agent
362     for ag, ag_pos in agents:
363         if ag1 != ag and ag1.get_coords() == ag_pos and
                ag.get_coords() == ag1_pos:
364             return True
365     return False
366
367 def atravessaram_t(agent, agents):
368     ag1, ag1_pos = agent
369     for ag, ag_pos in agents:
370         if ag1 != ag and ag1.pos() == ag_pos and ag.pos() == ag1_pos:
371             return True
372     return False
373
374 def limpamovimentos(newagentpos):
375     #ciclo que se houver choque não vai para a posição que quer e fica
            na mesma posição(posição velha) e corre o ciclo até que não
            haja choques nem atravessamentos
376     problemas = True
377     while problemas:
378         #problemas = False
379         newagentpos2 = []
```

```
380         for ag, ag_pos in newagentpos:
381             if choca((ag, ag_pos), newagentpos) or atravessaram((ag,
                    ag_pos), newagentpos):
382                 problemas = True
383                 newagentpos2.append((ag, ag.get_old_coords()))
384             else:
385                 newagentpos2.append((ag, ag_pos))
386                 problemas = False
387         newagentpos = newagentpos2
388     return newagentpos
389
390 def limpamovimentos_t(newagentpos):
391     #ciclo que se houver choque não vai para a posiçâo que quer e fica
            na mesma posição(posição velha) e corre o ciclo até que não
            haja choques nem atravessamentos
392     problemas = True
393     while problemas:
394
395         #problemas = False
396         newagentpos2 = []
397         for ag, ag_pos in newagentpos:
398             if choca((ag, ag_pos), newagentpos) or atravessaram_t((ag,
                    ag_pos), newagentpos):
399                 problemas = True
400                 newagentpos2.append((ag, ag.old_pos))
401             else:
402                 newagentpos2.append((ag, ag_pos))
403                 problemas = False
404         newagentpos = newagentpos2
405     return newagentpos
406
407 #functions used when the neural networks uses no comunication and is
        heterogeneous
408 def ann_inputs_outputs_T(preds, prey, net):
409     preds_coords = [p.get_coords() for p in preds]
410     preyx, preyy = prey.get_coords()
411     input_data = []
412     for x,y in preds_coords:
413         offsetx = toroidalDistance1coord(x, preyx, WIDTH)
414         offsety = toroidalDistance1coord(y, preyy, HEIGHT)
415         norm_offsetx = ((offsetx/ WIDTH) +1 )/2
416         norm_offsety = ((offsety/ HEIGHT) +1 )/2
417         input_data.append(norm_offsetx)
418         input_data.append(norm_offsety)
419     return net.activate(input_data)
420
421 def ann_inputs_outputs_t_T(tpreds, tprey, net):
422     preds_coords = [p.position() for p in tpreds]
423     preyx, preyy = tprey.position()
424     input_data = []
425     for x,y in preds_coords:
426         offsetx = toroidalDistance1coord(x, preyx, WIDTH)
427         offsety = toroidalDistance1coord(y, preyy, HEIGHT)
428         norm_offsetx = ((offsetx/ WIDTH) +1 )/2
429         norm_offsety = ((offsety/ HEIGHT) +1 )/2
430         input_data.append(norm_offsetx)
```

```
431         input_data.append(norm_offsety)
432     return net.activate(input_data)
433 
434 #functions used when the neural networks uses no comunication and is
    homogeneous
435 #preds_n to garantee different order of inputs for each predator
    neural network
436 def ann_inputs_outputs_I(preds, preds_n, prey, net):
437     preds_ordered = copy.deepcopy(preds)
438     pred_to_order = preds_ordered[preds_n]
439     del preds_ordered[preds_n]
440     preds_ordered.insert(0, pred_to_order)
441 
442     preds_coords = [p.get_coords() for p in preds_ordered]
443     preyx, preyy = prey.get_coords()
444     input_data = []
445     outputs= []
446     for x,y in preds_coords:
447         offsetx = toroidalDistance1coord(x, preyx, WIDTH)
448         offsety = toroidalDistance1coord(y, preyy, HEIGHT)
449         norm_offsetx = ((offsetx/ WIDTH) +1 )/2
450         norm_offsety = ((offsety/ HEIGHT) +1 )/2
451         input_data.append(norm_offsetx)
452         input_data.append(norm_offsety)
453     output = net.activate(input_data)
454     return output
455     #for n in range(5):
456     #
        outputs.append(net.activate(input_data))#random.shuffle(input_data)
457     #return outputs
458 
459 def ann_inputs_outputs_t_I(tpreds, tpred_n, tprey, net):
460     tpreds_ordered = copy.copy(tpreds)
461     tpred_to_order = tpreds_ordered[tpred_n]
462     del tpreds_ordered[tpred_n]
463     tpreds_ordered.insert(0, tpred_to_order)
464 
465     preds_coords = [p.position() for p in tpreds_ordered]
466     preyx, preyy = tprey.position()
467     input_data = []
468     outputS = []
469     for x,y in preds_coords:
470         offsetx = toroidalDistance1coord(x, preyx, WIDTH)
471         offsety = toroidalDistance1coord(y, preyy, HEIGHT)
472         norm_offsetx = ((offsetx/ WIDTH) +1 )/2
473         norm_offsety = ((offsety/ HEIGHT) +1 )/2
474         input_data.append(norm_offsetx)
475         input_data.append(norm_offsety)
476     return net.activate(input_data)
477 
478 #new functions to handle communication in individual(homogeneous)
    network
479 def ann_inputs_outputs_signal_I(pred, signals, prey, net):
480     #signal1, signal2 = signals #mudar porque o numero de sinais
        depende do numero de predadores e não serão sempre 2
481     x,y = pred.get_coords()
```

```
482     preyx, preyy = prey.get_coords()
483     offsetx = toroidalDistance1coord(x, preyx, WIDTH)
484     offsety = toroidalDistance1coord(y, preyy, HEIGHT)
485     norm_offsetx = ((offsetx/ WIDTH) +1 )/2
486     norm_offsety = ((offsety/ HEIGHT) +1 )/2
487     input_data = [norm_offsetx, norm_offsety] + signals
488     output, signal = tuple(net.activate(input_data))
489     return output, signal
490 
491 def ann_inputs_outputs_signal_t_I(tpred, signals, tprey, net):
492     x,y = tpred.pos()
493     preyx, preyy = tprey.pos()
494     offsetx = toroidalDistance1coord(x, preyx, WIDTH)
495     offsety = toroidalDistance1coord(y, preyy, HEIGHT)
496     norm_offsetx = ((offsetx/ WIDTH) +1 )/2
497     norm_offsety = ((offsety/ HEIGHT) +1 )/2
498     input_data = [norm_offsetx, norm_offsety] + signals
499     output, signal = tuple(net.activate(input_data))
500     return output, signal
501 
502 #new functions to handle communication in team(heterogeneous) neural
    network
503 def ann_inputs_outputs_signal_T(preds, signals, prey, net):
504     #signal1, signal2 = signals #mudar porque o numero de sinais
        depende do numero de predadores e não serão sempre 2
505     preds_coords = [p.get_coords() for p in preds]
506     preyx, preyy = prey.get_coords()
507     input_data = []
508     outputs= []
509     contpred= 0
510     for x,y in preds_coords:
511         offsetx = toroidalDistance1coord(x, preyx, WIDTH)
512         offsety = toroidalDistance1coord(y, preyy, HEIGHT)
513         norm_offsetx = ((offsetx/ WIDTH) +1 )/2
514         norm_offsety = ((offsety/ HEIGHT) +1 )/2
515         input_data = input_data + [norm_offsetx, norm_offsety]
516     input_data = input_data + signals
517     outputs, signals = tuple(net.activate(input_data))
518     return outputs, signals
519 
520 def ann_inputs_outputs_signal_t_T(tpreds, signals, tprey, net):
521     tpreds_coords = [p.pos() for p in tpreds]
522     tpreyx, tpreyy = tprey.pos()
523     input_data = []
524     outputs= []
525     contpred= 0
526     for x,y in tpreds_coords:
527         offsetx = toroidalDistance1coord(x, tpreyx, WIDTH)
528         offsety = toroidalDistance1coord(y, tpreyy, HEIGHT)
529         norm_offsetx = ((offsetx/ WIDTH) +1 )/2
530         norm_offsety = ((offsety/ HEIGHT) +1 )/2
531         input_data = input_data + [norm_offsetx, norm_offsety]
532     input_data = input_data + signals
533     outputs, signals = tuple(net.activate(input_data))
534     return outputs, signals
535 
```

```
536 #novelty
537 def calculate_novelty(behavior, archive, dim, threshold):
538     # Calculate novelty as the average distance to the k-nearest
        neighbors in the archive
539     k = min(10, len(archive))  # Use up to 10 neighbors
540     if k == 0:
541         return float(threshold)  # 0.1 novelty if archive is empty
542 
543     distances = sorted([np.linalg.norm(np.array(behavior) -
        np.array(b)) for b in archive])
544     novelty_score = np.mean(distances[:k])/(dim*2)
545     return novelty_score
546 
547 #novelty
548 def calculate_novelty(behavior, dim, K, c_gen_behaviours= None,
    archive= None):#novo argumento a indicar o K em vez do numero
    definido em bainxo
549     if archive != None:
550         all_behaviours = archive + c_gen_behaviours
551         k = K+1
552         distances = sorted([np.linalg.norm(np.array(behavior) -
            np.array(b)) for b in all_behaviours])
553         novelty_score = np.mean(distances[:k])/(dim*2)
554         return novelty_score
555     elif archive == None:
556         # Calculate novelty as the average distance to the k-nearest
            neighbors in the current gen_behaviours
557         k = K+1
558         distances = sorted([np.linalg.norm(np.array(behavior) -
            np.array(b)) for b in c_gen_behaviours])
559         novelty_score = np.mean(distances[:k])/(dim*2)
560         return novelty_score
561     #for the case where only archive is used(random behaviour to
        archive)
562     #elif c_gen_behaviours == None:
563     #    pass # duvida quanto ao calculo de novelty no caso da
        inserção ao acaso de comportamentos no arquivo
564 
565 def load(file):
566     """
567     The function to load records list from the specied file into this
        class.
568     Arguments:
569         file: The path to the file to read agents records from.
570     """
571     with open(file, 'rb') as dump_file:
572         return pickle.load(dump_file)
573 
574 def dump(record, file):
575     """
576     The function to dump records list to the specified file from this
        class.
577     Arguments:
578         file: The path to the file to hold data dump.
579     """
580     with open(file, 'wb') as dump_file:
```

```
581         pickle.dump(record, dump_file)
```

Listing A.1: Código principal

Apêndice B

Classe que define os agentes usados nas experiências

```
    class Agent:
    def __init__(self, coords):
        """
        Creates new Agent with specified parameters.
        Arguments:
            location:               The agent initial position within
               maze
        """
        self.coords = coords
        self.old_coords = coords
        self.initial_coords = coords


    def get_coords(self):
        return self.coords

    def get_old_coords(self):
        return self.old_coords

    def get_initial_coords(self):
        return self.initial_coords

    def set_coords(self, x, y):
        self.coords = (x, y)

    def set_x():
        pass

    def set_old_coords(self, x, y):
        self.old_coords = (x,y)

class Predator:
    """
    This is the maze navigating agent
    """
    def __init__(self, coords):
        """
        Creates new Agent with specified parameters.
```

```
38          Arguments:
39              location:               The agent initial position within
                    maze
40          """
41          self.coords = coords
42          self.old_coords = coords
43          self.initial_coords = coords
44
45
46      def get_coords(self):
47          return self.coords
48
49      def set_coords(self, x, y):
50          self.coords = (x, y)
51
52      def move(self, output):
53          most_distant_coord = self.distance_to_prey_coords[0]
54          most_distant_coord_pos = 0
55          for i in range(1, self.distance_to_prey_coords):
56              if self.distance_to_prey_coords[i] >= most_distant_coord:
57                  most_distant_coord_pos = i
58          if self.most_distant_coord_pos[most_distant_coord_pos] >=
                output:
59              self.most_distant_coord_pos[most_distant_coord_pos]
                    -=output
```

Listing B.1: agents definition

Apêndice C

Outro código usado

C.1 Movimento presa segunda experiência

```
def prey_move(prey, preds, step):
    #calculating closest pred
    closestdistance = toroidalDistance_coords(preds[0].old_coords,
       prey.get_coords(), WIDTH, HEIGHT)
    closestpred = preds[0]
    for tp in preds[1:]:
        distance = toroidalDistance_coords(tp.old_coords,
           prey.get_coords(), WIDTH, HEIGHT)
        if distance < closestdistance:
            closestdistance = distance
            closestpred = tp
    #para manter um registo da posição anterior
    prey.old_coords = prey.get_coords()
    xcoor,ycoor = prey.get_coords()
    directions = [(step,0), (-step, 0), (0, step), (0,-step)]
    farthest = -1
    farthestheading = None
    random.shuffle(directions)
    for (ax, ay) in directions:
        newpos = ax+ xcoor , ay+ ycoor
        newpos = toroidal_pos(newpos, WIDTH, HEIGHT)
        dist =toroidalDistance_coords(closestpred.old_coords, newpos,
           WIDTH, HEIGHT)
        if dist > farthest:
            x_tomove, y_tomove = newpos
            farthest = dist

    prey.set_coords(x_tomove, y_tomove)
```

Listing C.1: movimentos presa

C.2 Função de cálculo da novidade

```
    def calculate_novelty(behavior, dim, K, c_gen_behaviours= None,
       archive= None):#novo argumento a indicar o K em vez do numero
       definido em baixo
```

```
    if archive != None:
        all_behaviours = archive + c_gen_behaviours
        k = K+1
        distances = sorted([np.linalg.norm(np.array(behavior) -
            np.array(b)) for b in all_behaviours])
        novelty_score = np.mean(distances[:k])/(dim*2)
        return novelty_score
    elif archive == None:
        # Calculate novelty as the average distance to the k-nearest
            neighbors in the current gen_behaviours
        k = K+1
        distances = sorted([np.linalg.norm(np.array(behavior) -
            np.array(b)) for b in c_gen_behaviours])
        novelty_score = np.mean(distances[:k])/(dim*2)#tirar /(dim*2)
        return novelty_score
```

Listing C.2: calculo novidade

C.3 Avaliação dos genomas

```
    for genome_id, genome in genomes:
        genome.fitness = 0.0
        net = neat.nn.FeedForwardNetwork.create(genome, config)
        #além do fitness result tenho de ter uma lista de nove
            boleanos a indicar se houve captura por ensaio
        # e se houve não cálculo o novelty desse individuo mas ponho
            um valor colossal (>= FITNESS_THRESHOLD) para sair com
            solução
        fitness_result, the_behaviour, capturas = eval_fitness(net,
            PREDS_DEF, PREYS_9, HEIGHT, WIDTH, TICKS)
        #this_generation_behaviours[genome_id] = the_behaviour
        #gen_behaviours.append((genome_id, genome, the_behaviour,
            capturas))
        gen_behaviours.append(the_behaviour)

        #keep record of best fitness result in current generation
        if fitness_result > best_generation_fitness[1]:
            best_generation_fitness = (genome_id, fitness_result,
                GEN_N, capturas, the_behaviour)
        if fitness_result > BEST_FITNESS_SCORE[2]:
            BEST_FITNESS_SCORE = (genome ,genome_id, fitness_result,
                GEN_N, capturas, the_behaviour)

        if all(capturas):
            genome.fitness = FITNESS_THRESHOLD+1
        #Tenho de associar capturas e behaviours ao genome_id
    for genome, behaviour in zip(genomes, gen_behaviours):
        novelty_score = calculate_novelty(behaviour, DIST, 10,
            gen_behaviours, NOVELTY_ARCHIVE)#em vez de none todos os
            comportamentos desta geração inclusive ele proprio(k+1)
```

Listing C.3: avaliação dos genomas

C.4 Anexos

C.4.1 Anexo A

Anexo A: Parâmetros de configuração 1ª experiência:

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1
pop_size              = 500
reset_on_extinction   = False

[DefaultGenome]
# node activation options
activation_default      = sigmoid
activation_mutate_rate  = 0.0
activation_options      = sigmoid

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
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_direct

# node add/remove rates
node_add_prob           = 0.2
node_delete_prob        = 0.2

# network parameters
num_hidden              = 0
num_inputs              = 2
num_outputs             = 1 *primeiro modelo
num_outputs             = 2 *segundo modelo

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 30
weight_min_value        = -30
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
```

```
max_stagnation       = 20
species_elitism      = 2

[DefaultReproduction]
elitism            = 2
survival_threshold = 0.2
min_species_size   = 2
```

C.4.2 Anexo B

Anexo B: Parâmetros de configuração 2ª experiência:

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 130
pop_size              = 1000
reset_on_extinction   = False

[DefaultGenome]
# node activation options
activation_default      = sigmoid
activation_mutate_rate  = 0.0
activation_options      = sigmoid

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
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_direct

# node add/remove rates
node_add_prob           = 0.2
node_delete_prob        = 0.2

# network parameters
num_hidden     = 0
num_inputs     = 6  *modelo CP homogéneo
num_outputs    = 1  *modelo CP homogéneo
num_inputs     = 6  *modelo CP heterogéneo A
num_outputs    = 3  *modelo CP heterogéneo A
num_inputs     = 6  *modelo CP heterogéneo B
num_outputs    = 15 *modelo CP heterogéneo B
num_inputs     = 4  *modelo CE homogéneo
num_outputs    = 2  *modelo CE homogéneo

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
```

```
weight_max_value        = 30
weight_min_value        = -30
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 20
species_elitism      = 2

[DefaultReproduction]
elitism            = 5
survival_threshold = 0.2
min_species_size   = 2
```

C.4.3 Anexo C

Anexo C: Parâmetros de configuração 3ª experiência:

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 18
pop_size              = 500
reset_on_extinction   = False

[DefaultGenome]
# node activation options
activation_default      = sigmoid
activation_mutate_rate  = 0.0
activation_options      = sigmoid

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
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_direct

# node add/remove rates
node_add_prob           = 0.2
node_delete_prob        = 0.2

# network parameters
num_hidden     = 0
num_inputs     = 8  *modelo CP homogéneo
num_outputs    = 1  *modelo CP homogéneo
num_inputs     = 8  *modelo CP heterogéneo A
num_outputs    = 4  *modelo CP heterogéneo A
num_inputs     = 8  *modelo CP heterogéneo B
num_outputs    = 20 *modelo CP heterogéneo B
num_inputs     = 5  *modelo CE homogéneo
num_outputs    = 2  *modelo CE homogéneo

# node response options
```

```
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 30
weight_min_value        = -30
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 20
species_elitism      = 2

[DefaultReproduction]
elitism            = 5
survival_threshold = 0.2
min_species_size   = 2
```