

Agentes Autónomos e Sistemas Multiagente

2o Semestre - 2015/2016

A Comunidade de Agentes Wolf Pack



Grupo 17 - <Taguspark>

*<André Lima>* - <70572>

*<Samuel Gomes>* - <76415>

*<André Fonseca>* - <84977>

# Abstrato

**A solução de problemas multiagentes, principalmente quando envolvem aprendizagem por reforço tem enorme potencial mas são de difícil implementação**. Este projeto visa criar uma solução a um problema multiagente do género de perseguição. Implementámos 3 tipos de soluções: reativas, deliberativas e de aprendizagem. **Os resultados foram satisfatórios principalmente com a solução deliberativa e com a aprendizagem**. Com estas técnicas comprovamos como é possível resolver de forma rápida e eficiente este problema multiagente.

**Keywords**: *multiagentes, arquiteturas reativas, arquiteturas deliberativas, arquitetura de aprendizagem, Q-learning modular*

# Índice

1. Introdução 4

2. O Cenário 4

3. Arquiteturas de agentes / algoritmos 5

3.1 Arquitetura Reativa 5

3.2 Arquitetura Deliberativa 5

3.3 Coordenação, Cooperação e Negociação 6

3.4 Componente de Aprendizagem 7

4. Estudo Comparativo 8

5. Conclusão 10

6. Referências 11

# Introdução

No âmbito da Unidade Curricular de *Agentes Autónomos e Sistemas Multi-Agente*, o grupo 17 escolheu realizar o desafio denominado por *Wolf Pack*. **Este desafio consistiu na elaboração de três sistemas: (1) um sistema reativo, (2) um sistema deliberativo e (3) um sistema de aprendizagem.**

Um agente reativo é um agente que, tomando em conta apenas as suas percepções do ambiente, reage ao que acontece à sua volta. Neste projeto, o nosso agente reativo possui um número variável no seu estado interno**. Este número define o seu campo de visão e é usado como se definisse o alcance dos olhos do lobo.**

Um agente deliberativo possui um estado interno, que memoriza certas propriedades do ambiente que o rodeia e permite guardar estados do ambiente dentro da memória do agente. Os nossos agentes deliberativos possuem um plano de pathfinding e dois vetores. Cada vetor no seu estado interno é referente às suas comunicações com outros lobos.

**A parte mais importante deste desafio é um problema de *Q-learning* modular**. Este sistema de *Q-learning* trata cada lobo como um *Independent Learner* e cada lobo tem um módulo em que toma por referência um parceiro e a presa. Esta implementação permitiu dar a volta ao problema do gigante número de estados que um problema de *Q-learning* monolítico criaria. Assim conseguimos segmentar um problema grande cheio de estados e melhorá-lo aproveitando-nos de indireção.

**O desafio do *Wolf Pack* descreve o problema no qual quatro lobos têm de caçar uma presa, encurralando-a entre eles**. Esta presa pode ter vários movimentos e o lobo também. Os vários diferentes tipos de lobos vão nos permitir observar comportamentos emergentes, de cooperação e de alcateia em relação á caçada da presa.

**Neste documento estão descrições detalhadas das arquiteturas e implementações dos três tipos de agentes, bem como do cenário e também dos resultados das implementações, as suas conclusões e comparações com base em valores parametrizáveis.**

# O Cenário

**Este projeto é composto por 2 tipos de agentes num total de 5 agentes: 4 agentes do tipo lobo e um agente do tipo ovelha**. **Ambos os agentes partilham as mesmas ações: mover para cima, mover para baixo, mover para a esquerda, mover para a direita**. Só é possível mover para tais posições se estiverem desocupadas. Os agentes com aprendizagem têm uma quinta ação que é ficarem parados.

**A ovelha é extremamente simples, capaz apenas de se mover de forma aleatória pelo mapa.** Acrescentámos também uma opção para uma versão reativa da ovelha capaz de fugir aos lobos próximos. Nesta situação a ovelha tem como sensor um campo de visão capaz de distância d capaz de identificar os lobos. E faz como que o contrário do que os lobos fariam ao vê-la.

**Os lobos são muito mais complexos**. Na sua versão reativa têm apenas uma variável associada, o seu *field of view* (não confundir com memória visto que não guarda informação apresentada ao longo do tempo. A única coisa que o lobo “memoriza” é o seu alcance máximo de visão). Com ela, o lobo pode identificar e saber (naquele momento) onde estão os outros lobos e a ovelha.

Quando deliberativos, os lobos têm uma ação extra (o uivar) que transmite uma mensagem aos outros lobos indicando o lado pelo qual planeiam flanquear a ovelha, comunicando assim o seu plano e os planos de outros lobos com quem já comunicaram. Para poderem saber o plano de outros lobos e a quem já enviaram mensagem, têm agora memória sobre estes vários valores.

Por último, na sua versão com aprendizagem, os lobos perdem novamente a capacidade de enviar mensagens mas têm agora conhecimento dos múltiplos *Q-values* possíveis para cada par “outro lobo-presa” e, além disso regem-se por receber recompensas no ambiente à sua volta.

# Arquiteturas de agentes / algoritmos

## Arquitetura Reativa

Para o problema proposto, os nossos agentes, os lobos, possuem um estado interno. Um campo de visão que é, no máximo e em cada dimensão, metade do tamanho do mundo nessa dimensão. Este campo de visão é feito como um quadrado de dimensão d x d à volta do agente. Usando este estado interno, os agentes conseguem detetar se a presa está dentro do seu campo de visão. Usámos um número que representa d como estado interno, esse número foi usado como estado interno porque é uma característica dos agentes.

Em relação às regras perceção -> ação, estas foram desenvolvidas igualmente para todos os lobos.

Neste caso, cada lobo verifica se a presa está no campo de visão, e tenta ir na sua direção. No entanto em caso de empate entre distâncias em x e y é escolhida a direção aleatoriamente

Em último caso, quando algum lobo não consegue ver a presa, apenas escolhe aleatoriamente uma de quatro ações: andar em frente, para trás, para os lados.

## Arquitetura Deliberativa

No que diz respeito a esta parte do projeto, chegámos à conclusão que uma arquitetura BDI seria demasiado complexa para o tipo de agentes que considerámos.

Assim não seguimos a estrutura BDI dado que a implementação desta técnica se revelou opcional.

A nossa implementação resume-se a três módulos:

* **A comunicação entre os agentes** de modo a obterem um consenso sobre o seu subobjetivo;
* Procura de caminhos: **recorremos ao algoritmo A\*** tal como na aula de laboratório 3, para procurar caminhos mais curtos até cada uma das posições adjacentes à ovelha (norte, sul, este e oeste). **Assim, usámos como heurística para atualizar os custos, a distância euclidiana entre pontos dado que mapeamos a solução ótima ao caminho mais curto.** Os nós da procura são implementados como uma lista com 3 campos que representam o f, o g e uma posição na grelha (representada por uma lista com coordenadas);
* Estado wander: quando um lobo não tem informação necessária para poder cumprir o seu objetivo, este entra num estado que consiste em vaguear pelo mundo em zigzag de modo a voltar a obter informação o mais rapidamente possível. Esse movimento é gerado a partir de operações locais que usam os limites do mapa para poderem mudar de direção quando conveniente. Esta operação têm em conta o *field of view* do agente, dado que quanto maior o *fov* do agente **menor a frequência com que precisa de passar pelo centro do mapa** (dado que adquire mais dados sobre o mundo a cada momento).

**O agente deliberativo utiliza todos estes módulos de forma a minimizar o tempo demorado a encontrar a presa**. A prioridade para o agente neste sistema é formar um plano em conjunto com os outros lobos. Isto é, obter uma posição adjacente à ovelha para a qual ele e só ele é responsável de ir e depois manter-se nessa posição. Se o agente não tem plano, assim que ele vê a ovelha, este é formado, e assim que possível tenta comunica-lo com os outros lobos. **Se não consegue encontrar a ovelha, usa o movimento definido no estado *wander***. Por último, se vê outros lobos e não lhes enviou mensagem sobre o seu objetivo atual, realiza esta tarefa nesse “tick”.

## Coordenação, Cooperação e Negociação

**No sistema deliberativo, os lobos são capazes de comunicação de forma a coordenarem entre eles um plano de ataque. Cada lobo tem conhecimento sobre o flanco pelo qual pretende atacar a presa, os flancos escolhidos pelos outros lobos atacar e a informação sobre se os outros lobos ouviram ou não a sua mensagem.**

Quando um lobo encontra pela primeira vez a ovelha, este decide encurralá-la pelo lado que lhe está mais próximo. No entanto, se tem conhecimento que outro lobo já escolheu esse plano de ataque, decide por o segundo melhor, terceiro ou até mesmo quarto, dependendo dos planos já usados.

**Assim que o lobo encontra um outro lobo, se ainda não lhe enviou a mensagem do seu plano de ataque, então envia essa mensagem para ter a certeza que o outro lobo sabe**. Enviar a informação é algo penoso ao lobo pelo que ele tem de “parar e uivar” gastando um turno nessa ação. Decora também que a mensagem foi enviada àquele colega para não a enviar novamente sem necessidade.

**Existem situações em que ocorrem conflitos e dois lobos escolhem atacar o mesmo flanco.** Para resolver esse impasse, quando um lobo recebe uma mensagem que indica que já alguém escolheu o seu plano**, o lobo desiste da sua ideia em prol da do colega e tem de escolher um flanco novo dos ainda disponíveis**. Naturalmente, isso significa que terá novamente de avisar todos os outros lobos do seu plano novamente.

Esta última ação por parte dos lobos é bastante interessante visto que leva a um novo comportamento emergente em que, quando próximos de apanhar a ovelha, os lobos trocam algumas mensagens entre si até terem planeado completamente quem ataca por que lado.

## Componente de Aprendizagem

Devido ao aumento exponencial do espaço de estados de *Q-learning* quando aumenta o número de agentes, por causa das condições de ações dinâmicas implementámos uma técnica diferente: *Q-learning modular*. A técnica de *Q-learning modular* divide o problema do *Q-learning monolítico*, para cada lobo, em três problemas de *Q-learning monolítico* para cada lobo. Como cada lobo avalia o seu comportamento tendo como base o de um parceiro e da presa ao mesmo tempo, cada lobo vai ter três *Q-learners* diferentes, sendo que um deles atualiza o valor da sua aprendizagem relativamente a um determinado parceiro e a presa. **Existem três -learners pois cada lobo pode ter até três parceiros no seu campo de visão**. Ainda podemos acrescentar que cada um desses *Q-learners* é na verdade uma lista de quatro estruturas: **uma estrutura de dimensão *(2d+1)^4\*nA*, duas matrizes de dimensões *(2d+1)\*(2d+1) \*nA* e um vetor de dimensão *nA*** (*nA* é o número de ações possíveis para o agente).

A primeira estrutura preenche-se com os *Q-values* quando se escolhe uma ação tendo em conta a *prey* e um lobo parceiro. A primeira matriz preenche-se com os *Q-values* quando se escolhe uma ação tendo em conta um parceiro mas não a presa: **caso ela não esteja no campo de visão**; A segunda matriz preenche-se com os *Q-values* quando se escolhe a ação tendo em conta a presa e não o parceiro: **caso não haja um parceiro no campo de visão** e o vetor preenche-se com os *Q-values* **caso o lobo em causa não veja nem um lobo nem a presa**.

**Cada agente tem um loop: selecionar ação, executar ação, receber uma recompensa e dar update do seu Q-value**.

**A seleção da ação acontece chamando a função de avaliação *soft-max*** que tem inicializações diferentes consoante o tipo de matriz que se vai preencher com *Q-value*. Como existem apenas 3 ações que podem ser escolhidas (devido a cada referencial de acordo com um parceiro diferente), **o lobo em causa age como mediador e escolhe a ação que maximiza o Q-value dessas três que recebeu** (para maximizar o valor do par estado, ação).

Muitas vezes tivemos de adicionar métodos diferentes com a mesma funcionalidade, apenas para que funcionassem para as diferentes estruturas. Ou seja, **subdividimos o problema de um *Q-learning monolítico* em 12 (4 estruturas \* 3 companheiros) *Q-learnings* mais pequenos**.

Depois de selecionar a ação, o agente vai ter de a executar, que vai alterar as suas posições no mundo (e as posições relativas dos outros a si no seu referencial em que calcula as posições relativas da ovelha e do seu parceiro a si mesmo). Esta função vai dar valores a uma auxiliar *qIndex* que são os valores passados à respetiva estrutura de *Q-learning*.

Depois de se escolher a ação, calculamos a recompensa correspondente a ação que se acabou de executar. As recompensas foram mencionadas no *paper* que usámos para referência, no entanto achámos que **as recompensas desse *paper* não permitiam grande espaço de aprendizagem para os lobos** porque eles só iriam ter recompensas positivas no objetivo final e em todas as outras posições teriam a recompensa -0.1. Estas recompensas revelaram não se assentar bem na melhor solução para o problema proposto e como consequência, decidimos atribuir as nossas próprias recompensas:

* **A maior recompensa positiva vai continuar a ser na posição final, mas desta vez vai ter o valor de 5, ou seja todos os lobos receberão uma recompensa de 5.** Escolhemos esta como sendo a maior recompensa pois queremos que a posição mais desejada seja a posição final, por isso essa combinação de posições tem sempre maior recompensa;
* **Como no *paper*, damos uma recompensa de -0.1 a qualquer outra posição que não seja esta, no entanto este castigo pode ser amortizado ou agravado consoante outra condição** (e foi aí que nos diferenciámos do *paper* de apoio): **se o agente está numa posição na qual não tem a ovelha no seu campo de visão, o castigo vai agravar (recebendo uma recompensa de -1).**
* **No entanto, se um dos lobos estiver numa das posições finais mas os outros não, o lobo que está na posição correta vai receber uma recompensa de 2**. Este sistema de recompensas pareceu-nos que ajudaria mais a aprender os objetivos do que as recompensas sugeridas no *paper*. Além disso pensamos que estas recompensas sejam mais interessantes para incentivar um comportamento de cooperação e equilibrio interessantes. Por exemplo, se os lobos competirem pelo mesmo lugar vão ter uma recompensa mais baixa do que se forem cada um para um dos lados da ovelha.

A última parte do loop do agente com aprendizagem é atualizar o novo *Q-value* referente ao estado resultante de fazer a ação tendo em conta a recompensa recebida pela execução da ação.

Como estes agentes usam a função avaliadora *soft-max*, vão ter de repetir cada episódio várias vezes, mantendo os seus estados de aprendizagem entre episódios. **Isto vai ocorrer um número X de vezes, definido num *slider* na interface gráfica**. **Cada episódio termina quando os lobos caçaram a presa ou os lobos excederam o número de ações para caçar a presa**. A cada *reset*, mudamos o número de ações para caçar a presa de volta a zero (apesar disso, as estruturas de aprendizagem continuam preenchidas). Assim sendo, com o passar do tempo e a descida da temperatura da função avaliadora a maximização das recompensas à exploração passa a ser preferida e os lobos vão ser mais rápidos e mais eficientes a caçar. Assim, os valores mais altos de *Q-value* (entre outras coisas) vão constituir a política ótima para os lobos cumprirem o seu objetivo.

# Estudo Comparativo

Para podermos qualificar as várias implementações, conduzimos uma bateria de testes que nos permitiu obter um melhor entendimento sobre o funcionamento prático dessas implementações.

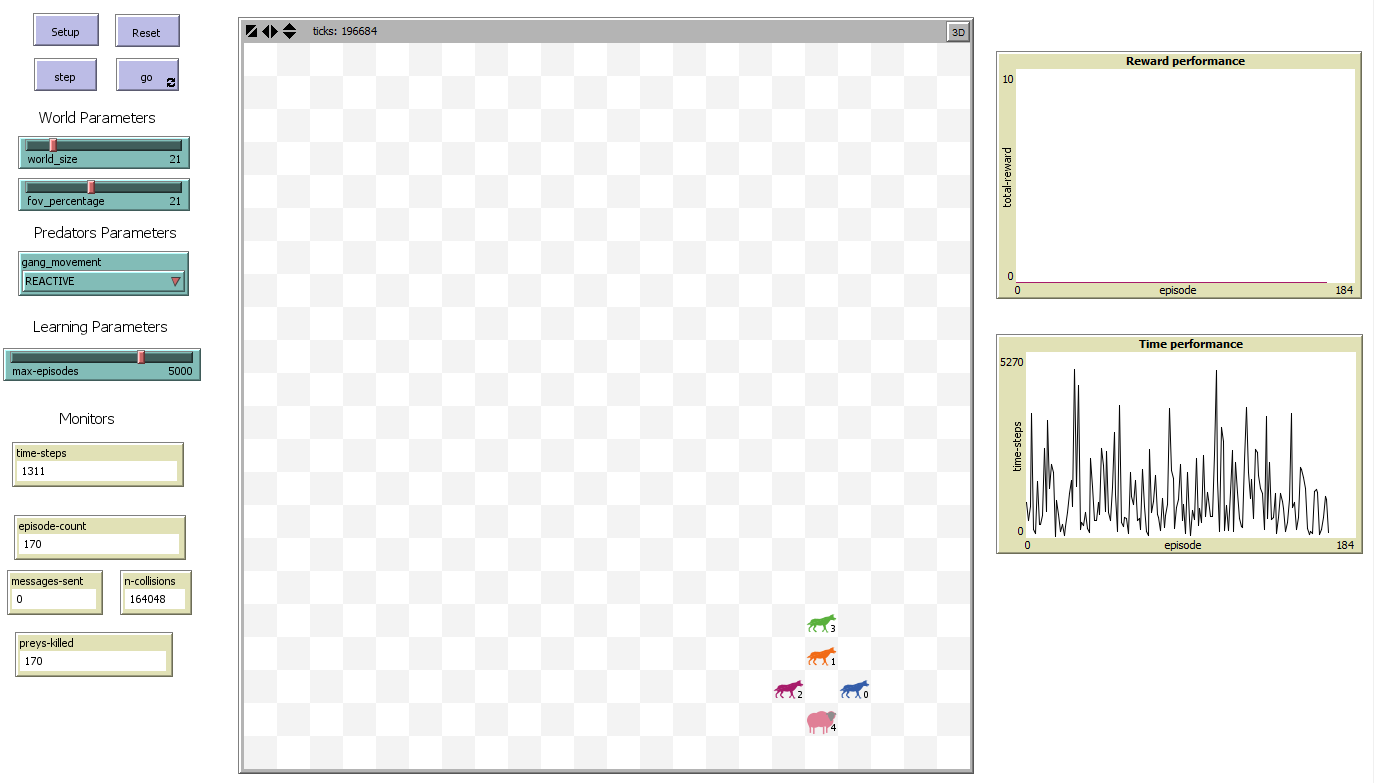
Assim para a arquitetura reativa considerámos os seguintes parâmetros:

* Field of View
* World Size

E os seguintes métodos de avaliação:

* O tempo que o sistema demora a encontrar uma solução

E como consequência, obtivemos os seguintes resultados:

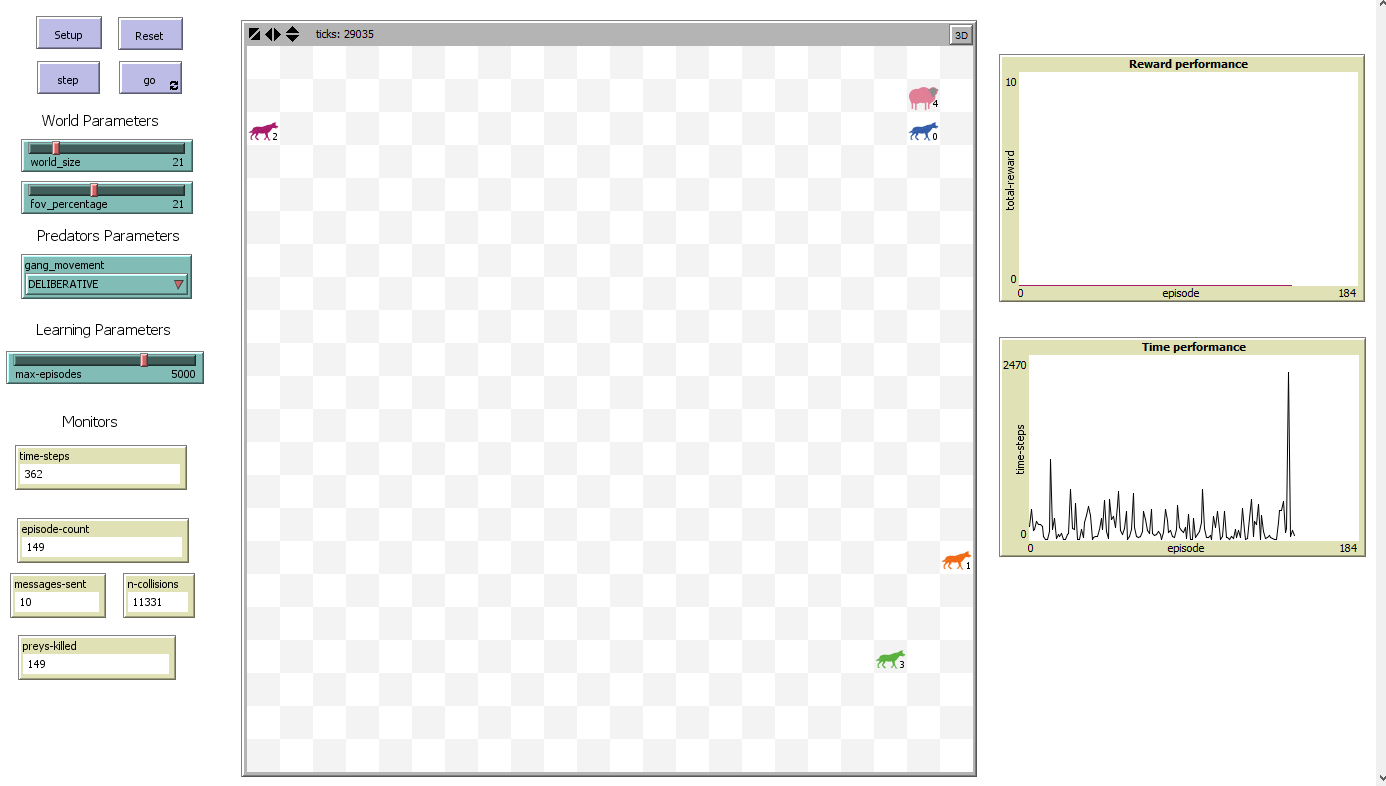


**Figura 1** – Arquitetura reativa em execução com field of view a 21% e world size 21

**A análise da imagem permite concluir o tempo demorado pelos agentes a encontrar a solução é bastante variável nesta arquitetura**. Com os parâmetros especificados na figura acima, tanto ocorrem episódios com valores baixos (próximos de 0), como se obtêm execução que demoram mais de 5000 ciclos.

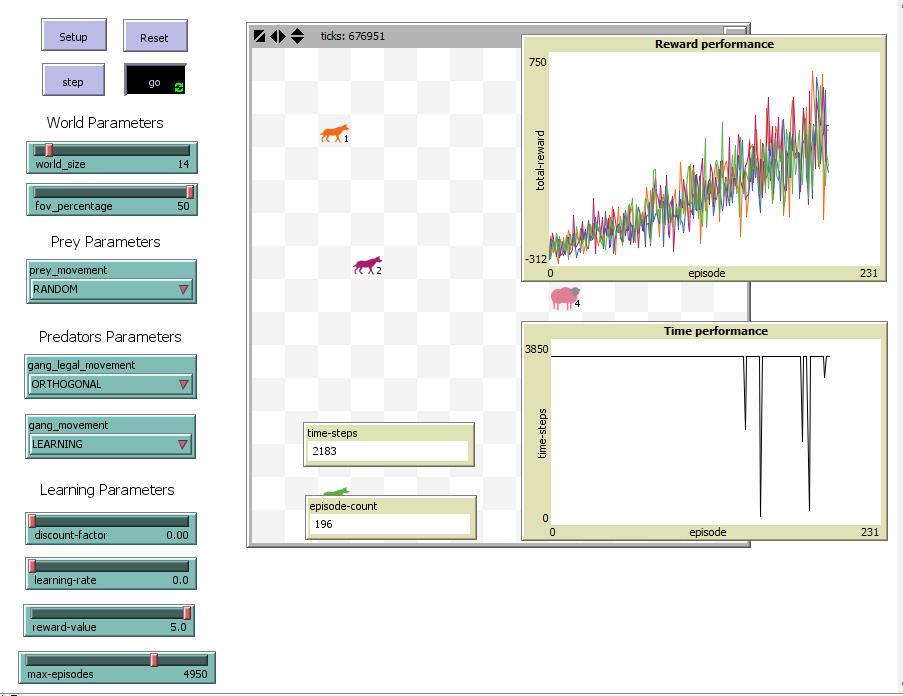
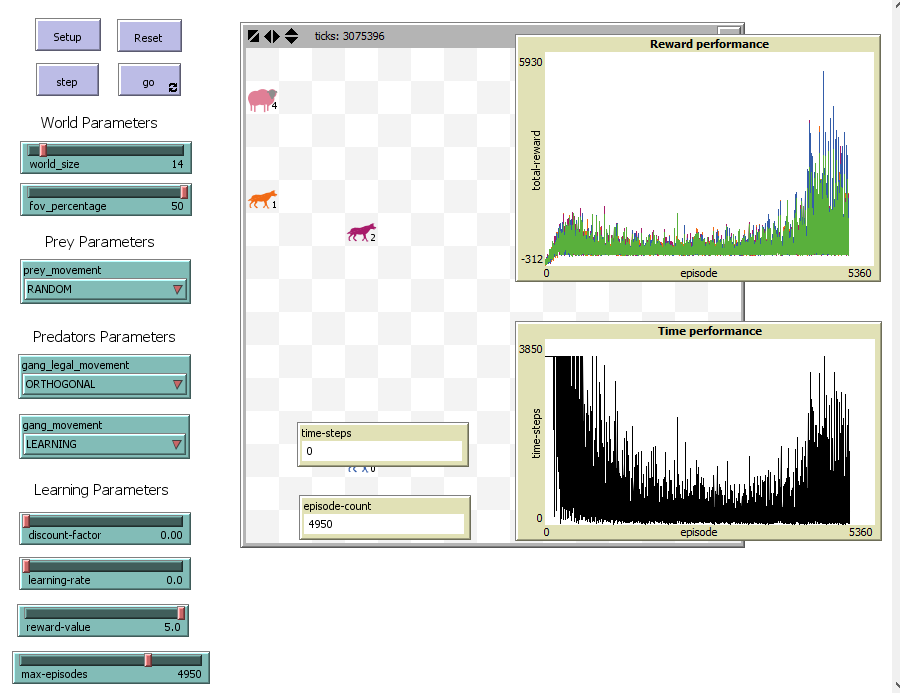
Para a arquitetura deliberativa foram considerados os mesmos parâmetros e métodos de avaliação que na arquitetura deliberativa. Para este caso, obtivemos os seguintes resultados:

**Figura 2** – Arquitetura deliberativa em execução com field of view a 21% e world size 21



Assim conclui-se que ao contrário da arquitetura reativa, os resultados são bastante consistentes para esta dimensão do mundo e *fov* (aproximadamente uma média de 200 iterações) tirando raros *outliers* que demoram bastante tempo acima do esperado.

Para a arquitetura com aprendizagem foi adicionado o método de avaliação desempenho de recompensa que indica o quanto os lobos recebem como recompensa por executar cada ação.



**Figura 3** – Arquitetura de aprendizagem em execução (à esquerda)

e após terminar (à direita) com field of view a 50% e world size 14

**Como se pode comprovar pelos gráficos acima representados, os agentes aprendem de facto a adaptar-se às situações inerentes ao ambiente.** Isto porque se observa desde logo que o valor de desempenho de recompensa vai subindo com o passar das iterações. Evidencia-se também que a partir de um certo ponto, embora se mantenha o desempenho da recompensa diminui em muito o tempo utilizado para capturar a presa. **Conclui-se portanto são capazes de explorar o mapa em seu redor durante uma fase inicial sem sacrificar o desempenho em fases mais avançadas da sua execução.**

# Conclusão

# 

Pela análise dos resultados, é de notar que, para problemas de menor dimensão e apesar da sua baixa fidelidade, a solução reativa teve bons resultados, **perdendo eficiência para problemas com uma complexidade mais elevada.**

Para problemas mais complexos, as outras soluções revelam resultados muito mais consistentes, sendo que a arquitetura **deliberativa revela ser fiável, não demorando muito tempo.**

**Também se conclui que a arquitetura que se baseia em aprendizagem será melhor para problemas onde a perceção do ambiente seja difícil**, dado que o objetivo deste tipo de agentes é, de facto, a adaptação dinâmica aos ambientes onde se encontra inserido.

# Referências

**Dissertation:**

1. *N. Ono and K. Fukumoto. “Multi-agent reinforcement learning: A modular approach”. In Second International Conference on Multi-agent Systems, pp. 252-258, Kyoto, 1996.*

**Music:**

1. *Duran Duran, “Hungry Like the Wolf”, 2003, Digital Remaster.*