Universidade Federal do Maranhão

Engenharia da Computação

Curso de Inteligência Artificial

Prof. Dr.: Thales Levi Azevedo Valente

Alunos e matrículas:

Gabriel Felipe Carvalho Silva - 2023098664

Judson Rodrigues Ciribelli Filho - 2019038973

Giordano Bruno de Araujo Mochel - 2019004080

MODELAGEM DE FORMIGUEIRO VERSÃO HxH EM NETLOGO

São Luís - MA

2024

Gabriel Felipe Carvalho Silva

Judson Rodrigues Ciribelli Filho

Giordano Bruno de Araujo Mochel

MODELAGEM DE FORMIGUEIRO VERSÃO HxH EM NETLOGO

Este relatório tem como objetivo detalhar as implementações realizadas na modelagem de formigueiro em netlogo baseadas na temática do arco das quimeras de Hunter x Hunter, bem como seu comportamento e impacto.

São Luís – MA

2024

Sumário

1 Introdução

A Modelagem Orientada a Agentes vem suprir a demanda pelo desenvolvimento de novas aplicações, que atendam aos requisitos e características das organizações sociais e de seus relacionamentos, de forma autônoma e integrada (VERA *ET AL*, 2008). Os Sistemas Multiagente (SMA) são formados por agentes que interagem entre si e desenvolvem comportamento autônomo e cooperativo (LIMA, GUSTAVO L., 2024). Segundo esse autor, os tipos de sistemas são utilizados para resolver problemas nos quais as entidades são descentralizadas.

Neste projeto de modelagem, baseado no modelo biológico de formigas do site da NetLogo, busca-se simular o comportamento de formigas e caçadores utilizando os princípios de sistemas multiagentes, por meio da modelagem orientada a agentes. A ideia central é criar um ambiente no qual formigas (comportando-se como agentes reativos baseados em modelos) realizam ações baseadas no seu estado atual, como buscar alimentos, atacar caçadores e gerar novas formigas (no caso da rainha) com diferentes status.

Os caçadores, por sua vez, têm como objetivo defender as aldeias humanas dos ataques das formigas, movendo-se e atacando as formigas que, eventualmente, entram em seu campo de visão, o que estabelece uma dinâmica de predador e presa.

O ambiente de tarefa deste modelo é projetado para fornecer as condições nas quais os agentes, tanto formigas quanto caçadores, executam suas ações e interagem entre si, influenciando diretamente seus comportamentos. A seguir, estão as medidas de desempenho, atuadores, sensores e propriedades do ambiente:

a) **Medidas de desempenho:**
   - **Formiga**: Sobrevivência por mais tempo ou até que a comida no mapa se acabe.
   - **Caçador**: Proteger as aldeias, eliminando o máximo de formigas; eliminar o rei e/ou rainha das formigas.

b) **Ambiente**: Formigueiro, aldeias e pontos de alimento.

c) **Atuadores:**
   - **Formigas**: Mover-se no mapa, seguir rastros de feromônio, atacar alvos.
   - **Caçadores**: Mover-se no mapa com foco nas aldeias, atacar alvos

d) **Sensores:**
   - **Formigas**: Percepção de feromônio, detecção de alvos, detecção de comida, status próprio (vida, ataque, tipo), detecção de estado do alvo (vida).
   - **Caçadores**: Detecção de alvos, detecção de áreas para proteção, detecção do estado do alvo (vida).

Propriedades do ambiente**:**

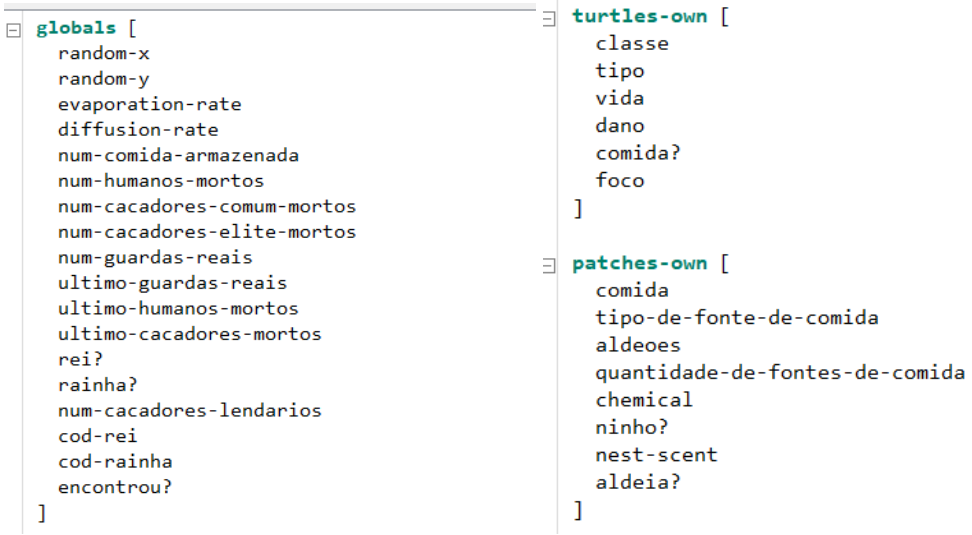
- **Parcialmente observável**: Os agentes percebem apenas os que estão próximos.
- **Discreto**: O ambiente é dividido em células ou pixels.
- **Dinâmico**: Os agentes se movem e interagem constantemente.
- **Estocástico**: O comportamento dos agentes pode ser aleatório, como a movimentação ou a seleção de alvos, e o ambiente também pode ser aleatório, com o posicionamento dos recursos.
- **Sistema Multiagente**: Existem dois tipos de agentes globais — formigas e caçadores — divididos em classes que desempenham diferentes papéis dentro da modelagem.
    - **Autônomos**: Tomam decisões independentes com base em suas percepções e objetivos.
    - **Percepção limitada**: Os agentes têm percepção parcial do ambiente e tomam decisões com base nessa percepção.
    - **Comunicação**: Compartilham informações, como rastros de feromônio, posição e estado dos patches.
    - **Ambiente competitivo**: Os agentes competem entre si para atingir seus objetivos, com as formigas atacando os caçadores e os caçadores protegendo as aldeias.

Dessa forma, a relação entre formigas e caçadores no modelo propõe uma interação complexa e dinâmica, onde ambos os agentes buscam otimizar suas ações para atingir seus objetivos, seja para caçar ou para sobreviver. Além disso, a implementação de conceitos inspirados nas Quimeras do Hunter x Hunter traz uma camada extra de complexidade, onde as formigas não são entidades fixas, mas sim mutáveis, adaptando-se ao seu ambiente e aos desafios impostos pelos caçadores e pelo ambiente apresentado.

2 Desenvolvimento

As implementações realizadas no código visam a inclusão de um novo tipo de agente (caçador), além da aleatoriedade do mapa, aldeias, percepção de estado e classes. Na Figura 1 que segue, são apresentadas as variáveis globais e de status dos agentes:

Figura 1 - Variaveis globais e de status dos agentes



Fonte: Autores (2024)

Para as condições de estado, as variáveis como **num-cacadores-comum-mortos** registram o estado atual para o quantitativo de caçadores mortos e **ultimo-cacadores-mortos** simula a memória para o estado anterior, atribuindo então à simulação (e ao agente) a habilidade ou percepção de variações no ambiente. As variáveis booleanas como **rei?** e **ninho?** são utilizadas também para a percepção do cenário atual quanto a existência de agentes chave ou condição para patch.

A função setup, além de inicializar variáveis globais e setups secundários, é responsável por definir a posição do ninho (Figura 2).

Figura 2 - Parte da função padrão setup.

```
to setup
  clear-all
  print "Bem-vindo à Simulação HxH Quimera."
  set random-x random-xcor
  set random-y random-ycor
```

Fonte: Autores (2024)

No nível dos patches, o seu setup pode ser dividido em: formigueiro, comida padrão e aldeias. Para o formigueiro, foram modificadas as ações de criação e destaque, incluindo também a formiga rainha e as formigas soldados iniciais (Figura 3).

Figura 3 - Funções para criação de formigueiro com rainha com soldados.

```
to criar-formigueiro
  let propriedades (propriedades-formiga "amarelo")

    create-turtles 1 [
    set classe "formiga"
    set tipo item 0 propriedades
    set vida item 1 propriedades
    set dano item 2 propriedades
    set color item 3 propriedades
    setxy random-x random-y
  ]

  set cod-rainha one-of turtles with [color = yellow]
  set rainha? true

  ; Cria formigas vermelhas (soldados)
  criar-formigas-como "vermelho" 10
end

to destacar-formigueiro
  if random-x > (max-pxcor - 1) [ set random-x (max-pxcor - 1) ]
  if random-x < (min-pxcor + 1) [ set random-x (min-pxcor + 1) ]
  if random-y > (max-pycor - 1) [ set random-y (max-pycor - 1) ]
  if random-y < (min-pycor + 1) [ set random-y (min-pycor + 1) ]
  ask patches [
    if (distancexy random-x random-y) < 2
    [
    set pcolor violet
    ]
  ]
end
```

Fonte: Autores (2024)

O mesmo é feito para a comida padrão, como segue na Figura 4, com sua criação e destaque randômicos. Uma diferença é na quantidade de comida por patch que também é selecionada de forma randômica.

Figura 4 - Criação de comida padrão.

```
to criar-comida-padrao
  set quantidade-de-fontes-de-comida one-of [0 1 2]

  repeat quantidade-de-fontes-de-comida [
    let float random-float 10 + 1
    let new-x random-xcor * float
    let new-y random-ycor * float

    if (distancexy (new-x * float) (new-y * float)) < (distancexy random-x random-y)
    [
    set tipo-de-fonte-de-comida one-of [1 2 3]
    ]
    if tipo-de-fonte-de-comida > 0
    [
      set comida one-of [10 20]
    ]
  ]
end

to destacar-patch-de-comida
  if comida > 0 [
    if tipo-de-fonte-de-comida = 1 [ set pcolor cyan ]
    if tipo-de-fonte-de-comida = 2 [ set pcolor sky ]
    if tipo-de-fonte-de-comida = 3 [ set pcolor blue ]
  ]
end
```

Fonte: Autores (2024)

As aldeias também são geradas de forma aleatória no mapa com o adendo de incluir feromônio ao seu redor (objetivo de atrair as formigas para o local) e shapes de casas. Segue o código na Figura 5.

Figura 5 - Criação de aldeias

```
to criar-aldeia [quantidade]
 repeat quantidade [
    let x random-xcor
    let y random-ycor

    ask patches with [abs (pxcor - x) <= 2 and abs (pycor - y) <= 2] [
      set aldeoes random 20 + 10
      set pcolor lime
      set chemical 100
      set aldeia? true
    ]
  ]

 ask patches with [aldeia?] [
    sprout 1 [
      set shape "house"
      set color gray
      set size 1
    ]
  ]
end
```

Fonte: Autores (2024)

Subindo um nível, no caso das turtles ou agentes, incluindo também a ação de criar do observador, tem-se dois tipos: formigas e caçadores. No âmbito das formigas, é utilizado um report para definir as diferentes classes e carregar os detalhes do seu status (tipo, vida, dano e cor), como segue na Figura 6.

Figura 6 - Classes de formigas

```
to-report propriedades-formiga [formiga-cor]
  if formiga-cor = "rosa" [
    report ["movel" 130 6 magenta]
  ]
  if formiga-cor = "laranja" [
    report ["movel" 160 10 lime]
  ]
  if formiga-cor = "amarelo" [
    report ["imovel" 500 25 yellow]
  ]
  if formiga-cor = "fucsia" [
    report ["movel" 200 20 pink]
  ]
  report ["movel" 100 5 red]
end
```

Fonte: Autores (2024)

Esse report é chamado sempre que um novo agente da classe formiga é preciso ser criado. Para ser criado, esse também possui uma ação modulada (Figura 7) com local de nascimento no centro do ninho.

Figura 7 - Criação de agentes de classe fomiga.

```
to criar-formigas-como [formiga-cor quantidade]
  let propriedades (propriedades-formiga formiga-cor)

  create-turtles quantidade [
    set classe "formiga"
    set tipo item 0 propriedades
    set vida item 1 propriedades
    set dano item 2 propriedades
    set color item 3 propriedades
    set size 1
    setxy random-x random-y
  ]
end
```

Fonte: Autores (2024)

Para gerar novas formigas existe um modelo ou contrato com condições que vão definir em que estado um tipo específico de formiga pode nascer, salvo no início da simulação (onde 10 formigas soldados são criadas junto com a rainha) e no nascimento de um rei (que força o surgimento de uma nova rainha também). A ação responsável por esse modelo é a **gerar-novas-formigas,** Figura 8.

Figura 8 - Modelo para criação de formigas.

```
to gerar-novas-formigas
  if num-comida-armazenada > 0 and num-comida-armazenada mod 5 = 0 and rei? = false [
    criar-formigas-como "vermelho" 1
  ]
  if num-humanos-mortos > 0 and num-humanos-mortos mod 10 = 0 and rei? = false [
    criar-formigas-como "rosa" 1
  ]
  if num-cacadores-comum-mortos > 0 and num-cacadores-comum-mortos mod 10 = 0 and rei? = false [
    criar-formigas-como "fucsia" 1
    print "General nasceu!"
  ]
  if num-cacadores-elite-mortos > 0 and num-cacadores-elite-mortos mod 5 = 0 and num-guardas-reais < 3 and rei? = false [
    criar-formigas-como "laranja" 1
    set num-guardas-reais num-guardas-reais + 1
    print "Guarda-real nasceu!"
  ]
  if num-guardas-reais > ultimo-guardas-reais and num-guardas-reais mod 3 = 0 and rei? = false [
    ask cod-rainha [
      print "A rainha está morta! Longa vida ao rei!"
      wait 1
      die
    ]
    criar-formigas-como "amarelo" 1
    set cod-rei one-of turtles with [color = yellow]
    set rei? true

    criar-rainha
    set rainha? true

    set ultimo-guardas-reais num-guardas-reais
  ]
```

Fonte: Autores (2024)

No modelo, todas as formigas, exceto as de tipo amarelo, que representam o rei e a rainha, são criadas apenas quando o rei ainda não existe. Essa condição é satisfeita quando uma rainha está alojada em um ninho (há outra situação que será discutida posteriormente). As demais condições para a criação de formigas dependem do estado do número de alimentos disponíveis e da quantidade de humanos mortos, sejam aldeões ou caçadores.

Para a formiga vermelha, a condição é que haja comida no ninho e que a quantidade seja um múltiplo de 5. Já no caso da formiga de elite (rosa), a condição é que o número de aldeões mortos seja um múltiplo de 10. Por fim, para as formigas guardas-reais, a condição é que o número de caçadores de elite mortos seja um múltiplo de 5 e que não existam mais do que três formigas guarda-reais no modelo. Essa última restrição é uma referência direta à história do anime, onde o rei possui exatamente três guardas-reais ao seu lado.

Por fim, a condição para a morte natural da rainha e o nascimento do rei é que, durante sua vida, três formigas guardas-reais tenham sido criadas, replicando a narrativa apresentada no anime.

Além disso, ações complementares, como o comportamento relacionado ao feromônio, a criação da rainha em casos específicos (detalhados posteriormente) e o

movimento das formigas, são implementadas. Essas ações fazem parte do código original ou seguem a mesma lógica das já descritas neste relatório.

Por fim, uma última ação exclusiva do âmbito dos agentes de classe formiga é a busca do rei por uma nova rainha, denominada **procurar-conjuge**. o rei (representado pela variável **cod-rei**) realiza uma busca em um raio de 1 patch ao seu redor para localizar outro agente identificado como **cod-rainha**. Caso a busca seja bem-sucedida, o rei morre, e a nova rainha torna-se o centro do novo ninho. A partir disso, ela será alimentada pelas formigas existentes e pelas novas geradas nesse ninho

Figura 9 - Ação procurar-conjugue do rei

```
to procurar-conjugue
    let alvo one-of turtles in-radius 1 with [self = cod-rainha]
    if alvo != nobody [
      set random-x [xcor] of alvo
      set random-y [ycor] of alvo

      wait 1
      set rei? false
      set encontrou? true
      print "O rei está morto! Longa vida à rainha!"
      die
    ]
end
```

Fonte: Autores (2024)

No âmbito dos caçadores, uma lógica similar foi utilizada para criação de novos agentes, considerando um report e uma ação, como segue na Figura 10. O termo caçador também é uma referência ao anime, na prática ele serve como um protetor das aldeias, já que seu movimento é restrito ao redor delas.

Figura 10 - Classe e criação de caçadores

```
to criar-novo-cacador [tipo-cacador quantidade]
  let propriedades (propriedades-cacadores tipo-cacador)

  create-turtles quantidade [
    set shape "wolf"
    set size 1
    set classe "cacador"
    set tipo item 0 propriedades
    set dano item 2 propriedades
    set vida item 1 propriedades
    set color item 3 propriedades
    if (item 4 propriedades) = true [
    set foco one-of patches with [aldeia?]
    ]
    setxy random-xcor random-ycor
  ]
end



to-report propriedades-cacadores [tipo-cacador]
  if tipo-cacador = "cacador-elite" [
    report ["cacador-elite" 300 20 orange true]
  ]
  if tipo-cacador = "cacador-lendario" [
    report ["cacador-lendario" 750 30 pink true]
  ]
  report ["cacador-comum" 200 15 blue true]
end
```

Fonte: Autores (2024)

O modelo de criação dos caçadores é apresentado na Figura 11.

Figura 11 - Modelo de criação de caçadores.

```
to criar-cacadores
  if num-humanos-mortos > 0 and num-humanos-mortos mod 5 = 0 and num-humanos-mortos > ultimo-humanos-mortos [
    criar-novo-cacador "cacador-comum" 1
    set ultimo-humanos-mortos num-humanos-mortos
  ]

  if num-cacadores-comum-mortos > 0 and num-cacadores-comum-mortos mod 5 = 0 and num-cacadores-comum-mortos > ultimo-cacadores-mortos [
    criar-novo-cacador "cacador-comum" 1
    criar-novo-cacador "cacador-elite" 1
    set ultimo-cacadores-mortos num-cacadores-comum-mortos
  ]

  if rei? = true and num-cacadores-lendarios <= 4 [
    criar-novo-cacador "cacador-lendario" 2
    set num-cacadores-lendarios num-cacadores-lendarios + 2
    print "Caçador lendario nasceu!"
  ]

end
```

Fonte: Autores (2024).

No caso dos caçadores comuns, a condição para gerar novos é que o número de humanos mortos seja um múltiplo de 5 e que esse valor tenha aumentado em relação ao último estado registrado. A mesma lógica serve para os caçadores de elite. Para o caçador

lendário, a condição é que o rei esteja vivo no ambiente e que a quantidade de caçadores lendários não ultrapasse 4.

Para a interação entre agentes de diferentes classes (formigas e caçadores) foi implementada a ação **verificar-alvos,** utilizada por ambos. Na Figura 12 é apresentada a parte do caçador.

Figura 12 - Ação de interação entre agentes (verificar-alvos) para caçadores.

```
to verificar-alvos [classe-agente]
  ask turtles with [classe = classe-agente] [
    if classe = "cacador" [
    let alvo one-of turtles in-radius 1 with [classe = "formiga"]
    if alvo != nobody [
      ask alvo [
         set vida vida - [dano] of myself
        if vida <= 0 [
            if color = orange [set num-guardas-reais num-guardas-reais - 1]
            if self = cod-rei [
              print "O rei foi morto!"
              set rei? false
            ]
            if self = cod-rainha [
              print "A rainha foi morta!"
              set rainha? false
            ]
            die
          ]
        ]
        ;set vida vida - 10
      ]
    ]
```

Fonte: Autores (2024)

Para o caçador, a lógica de ataque consiste em buscar um agente da classe formiga dentro de um raio de 1 *patch* ao seu redor. Caso encontre um alvo, ele causará uma redução na vida desse agente, equivalente ao dano atribuído ao caçador. Se, após o ataque, a vida do alvo for igual ou inferior a zero, o agente formiga será eliminado.

Se o alvo morto for o rei ou a rainha, será exibido no console um aviso de sua morte, e a variável que indica sua existência no ambiente será definida como falsa. No caso de o alvo ser um guarda real, a variável que contabiliza o número total de guardas no mapa será decrementada.

Uma lógica similar é utilizada para o agente formiga, considerando agora as variáveis de caçadores comuns e de elite mortos (Figura 13).

Figura 13 - Ação de interação entre agentes (verificar-alvos) para formigas.

```
  if classe = "formiga" [
    let alvo one-of turtles in-radius 1 with [classe = "cacador"]
    if alvo != nobody [
      ask alvo [
       set vida vida - [dano] of myself
       if vida <= 0 [
          if tipo = "cacador-comum" [set num-cacadores-comum-mortos num-cacadores-comum-mortos + 1]
          if tipo = "cacador-elite" [set num-cacadores-elite-mortos num-cacadores-elite-mortos + 1]
         ; print "Um caçador foi eliminado!"
          die
        ]
      ]
      set vida vida - 10
    ]
  ]
]
```

Fonte: Autores (2024)

O procedimento **tempestade** implementa um evento climático visual na simulação, limitado a ocorrer no máximo duas vezes. Ele utiliza um contador global para rastrear o número de execuções, interrompendo novas ocorrências após o limite definido. Durante a execução, todos os patches assumem uma coloração cinza, simulando a presença de nuvens, enquanto a interface é atualizada continuamente para exibir os efeitos visuais. A tempestade tem duração total de cinco segundos, com pausas controladas para garantir que as mudanças sejam visíveis. Após o término, as cores dos patches são restauradas ao estado original, e uma mensagem indica o fim do evento.

Figura 14 - Modelo de criação de uma catástrofe

```
; == Catástrofes (Tempestade) ===

to tempestade
  ; Verifica se ainda pode ocorrer uma tempestade
  if contador-tempestades >= 2 [
    stop ; Impede que a tempestade ocorra mais de 2 vezes
  ]
  ; Incrementa o contador de tempestades
  set contador-tempestades contador-tempestades + 1
  print "A tempestade começou! Nuvens e raios estão no céu."
  ; A tempestade dura 5 segundos (50 ciclos de 5s)
  repeat 50 [
    ask patches[
      set pcolor gray
    ]
    display ; Atualiza a interface para mostrar os efeitos
    wait 3 ; Aguarda 3.0 segundo antes do próximo ciclo
  ]
  display ; Atualiza a interface novamente

  print "A tempestade acabou. O céu está limpo novamente."
end
```

Fonte: Autores (2024)

A função ‘Check-catastrophes’ (Figura 15) é responsável por verificar a ocorrência de eventos catastróficos na simulação, com foco específico na tempestade. Inicialmente, ela avalia se o número total de tempestades registradas, armazenado na variável global ‘Contador-de-tempestades’ , é menor que 1, garantindo que o evento seja limitado a, no máximo, uma ocorrência.

Em seguida, utiliza uma probabilidade de 10% (definida pela condição **random 100 < 10**) para determinar aleatoriamente se uma tempestade ocorrerá naquele instante. Caso as condições sejam atendidas, a função chama o procedimento tempestade, disparando o evento e executando seus efeitos na simulação.

Figura 15 - Condição para catástrofe

```
; === Vereficando Catástrofes

to check-catastrophes
  if contador-tempestades < 2 [  ; Limita a tempestade a no máximo 2 ocorrências
    if random 100 < 10 [ ; 10% de chance de ocorrer uma tempestade
      tempestade
    ]
  ]
end
```

Fonte: Autores

A função “**exterminar-formigas”** (figura 16) denota um evento catastrófico na simulação (a Rosa dos Pobres). Ela ocorre como consequência direta da **morte do Rei**, simbolizando o colapso total da colônia. Esse evento extermina todas as formigas quimeras no ambiente, resetando suas quantidades a 0 e marcando o **fim da simulação**.

Figura 16 - Condição para Rosa dos Pobres

```
to exterminar-formigas
  ask turtles with [classe = "formiga"] [
    die
  ]
  print "Todas as formigas foram exterminadas pela Rosa dos Pobres!"
  user-message "Fim da Simulação: Rosa dos Pobres ativada!"
  stop
end
```

Fonte: Autores

Por fim, a ação padrão responsável por chamar as demais em cada tick da simulação, **go** , é apresentada na Figura 17.

Figura 17 - Ação **go** responsável pelas sequência de eventos.

```
to go
  check-catastrophes  ; Verifica se ocorre uma tempestade no início do tick
  ask turtles with [classe = "formiga" and tipo = "movel"] [
    if who >= ticks [ stop ]             ; sincroniza a saída das formigas do ninho com o tempo
    verificar-alvos "formiga"
    ifelse comida? = false and rei? = false [
      procurar-por-comida                ; procura por comida se não estiver carregando
    ] [
      retornar-ao-formigueiro            ; retorna ao ninho se estiver carregando comida
    ]
    wiggle                               ; movimento aleatório para simular procura
    fd 1                                 ; move-se para frente
  ]
  diffuse chemical (diffusion-rate / 100)  ; difusão do feromônio entre os patches
  ask patches [
    set chemical chemical * (100 - evaporation-rate) / 100  ; evaporação do feromônio
    recolor-patch                      ; atualiza a cor do patch após mudanças
  ]

  ask turtles with [classe = "cacador"] [
    verificar-alvos "cacador"

    mover-cacadores
    fd 1
  ]

  if rei? = true [
    set encontrou? false
    ask turtles with [ color = yellow ] [
      wiggle
    ]

    ask cod-rei [ procurar-conjuge ]

    if rei? = false and encontrou? = true [
      ask patches [
        set nest-scent 0
        setup-ninho
      ]
      destacar-formigueiro
    ]
```

Figura 17 (continuação) - Ação **go** responsável pelas sequência de eventos.

```
    if rei? = false or not any? turtles with [self = cod-rei] and encontrou? = false [
      print "Os caçadores mataram o rei! Fim da simulação."
      print "O rei foi morto! Ativando Rosa dos Pobres."
      exterminar-formigas
      user-message "Fim!"
      stop
    ]

    if rainha? = false [
      print "O rei não consegue mais seguir com a linhagem. Fim da simulação."
      user-message "Fim!"
      stop
    ]

    ;set encontrou? false
  ]

  if rainha? = true and rei? = false and encontrou? = false [
    print "A rainha não consegue mais seguir com a linhagem. Fim da simulação."
    user-message "Fim!"
    stop
  ]

  if rainha? = false and rei? = false [
    print "As formigas não conseguem mais procriar! Fim da simulação."
    user-message "Fim!"
    stop
  ]

  ;Verifica população de formigas
  let populacao-formiga count turtles  with [classe = "formiga" and color != yellow]
  if populacao-formiga = 0 [
    print "População de Formigas Erradicada. Fim da simulação."
    user-message "Fim!"
    stop
  ]

  ;ações nivel observador
  gerar-novas-formigas
  criar-cacadores
  tick
end
```

Fonte: Autores (2024)

A ação começa direcionando formigas móveis (soldado, capitão e guardas) a executar suas ações específicas. Para sincronizar sua saída do ninho com o tempo da simulação, verifica-se a condição **who >= ticks**, interrompendo a execução caso não seja satisfeita. As formigas são instruídas a procurar alvos e, dependendo do estado **comida?** e **rei?**, decidem entre buscar comida ou retornar ao formigueiro. Durante o movimento, aplicam um comportamento aleatório com a ação **wiggle** para simular uma busca e avançam uma unidade com **fd 1**.

Os patches participam do modelo com a difusão e evaporação do feromônio. A difusão é controlada pelo parâmetro **diffusion-rate**, enquanto a evaporação reduz a intensidade do feromônio proporcionalmente a uma taxa definida. A aparência visual do ambiente é atualizada com a ação **recolor-patch**, refletindo mudanças nas concentrações de feromônio representadas visualmente pelas cores.

O código também gerencia a dinâmica relacionada ao rei e à rainha. Se o rei está ativo (**rei? = true**), ele realiza a ação procurar-conjuge dentro de sua vizinhança. Caso o rei ou a rainha sejam eliminados, mensagens informam o término da simulação, destacando a dependência reprodutiva entre eles. Após a morte do rei, se **encontrou? = true**, inicia-se a configuração de um novo ninho e os patches são ajustados para receber o novo ambiente. Caso contrário, a simulação é encerrada com mensagens específicas sobre o desfecho.

A ativação do evento é gerenciada pelo procedimento "**exterminar-formigas**" e por uma condição específica no loop principal da simulação (**GO**), que verifica se o Rei está ausente, seja porque:

- A variável global **rei?** foi definida como false, ou
- Não existem mais formigas associadas ao identificador único do Rei (**cod-rei**).

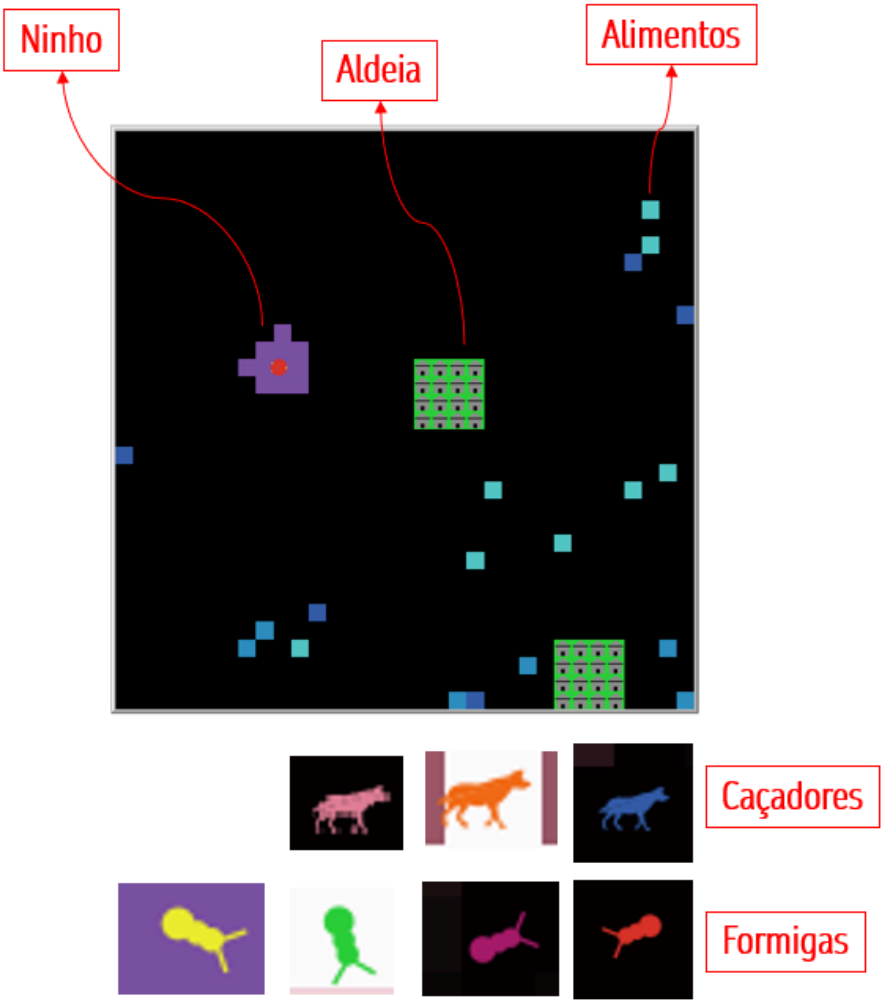
Caso essas condições sejam verdadeiras e o Rei não tenha encontrado uma nova Rainha (indicado por **encontrou? = false**), o evento é disparado.

Quanto aos caçadores, eles verificam alvos formigas, movem-se com **mover-cacadores** e avançam no espaço com **fd 1**. No nível do observador, ações globais como a geração de novas formigas e a criação de caçadores são realizadas para manter a simulação em andamento.

3 Resultados e Discussão

Como resultados obtidos, para o setup gerado randomicamente e as diferentes classes obteve-se o comportamento apresentado no exemplo da Figura 17 que segue,

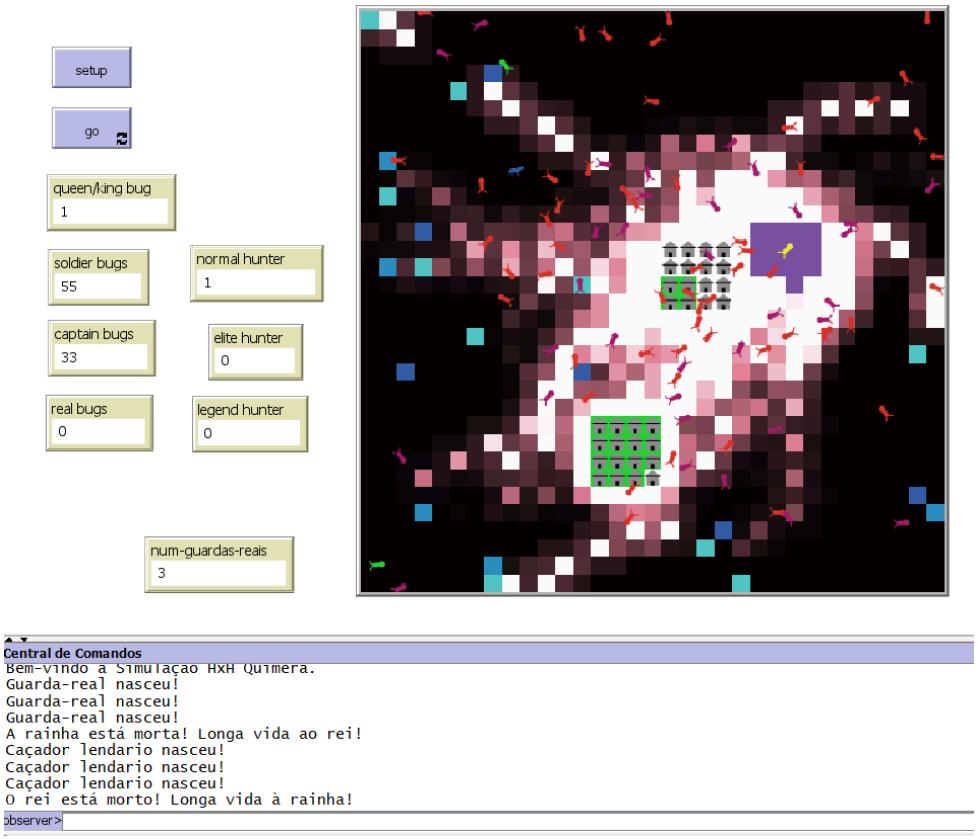
Figura 17 - Setup e classes de agentes.



Fonte: Autores (2024)

Nessa figura é apresentado o patch e os agentes que irão aparecer ao longo da simulação. Na interface é possível acompanhar a contagem de agentes por classe e entender os principais eventos pelo console (Figura 17).

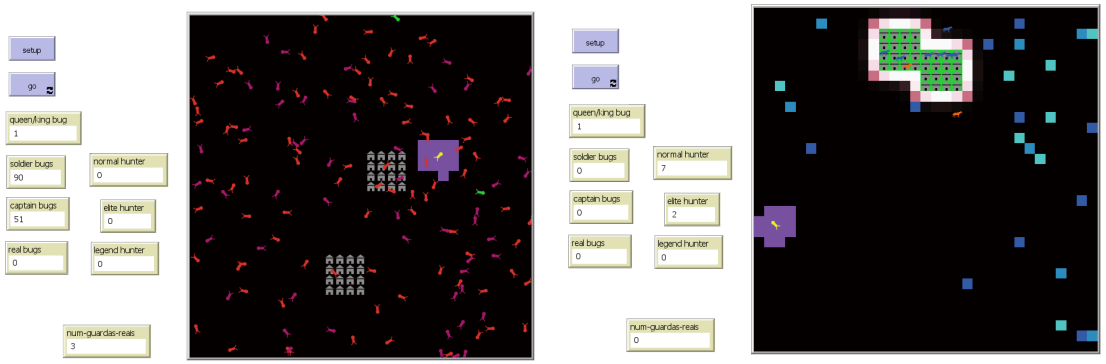
Figura 17 - Interface da simulação



Fonte: Autoes (2024)

As simulações demonstraram que o posicionamento das aldeias e do ninho exerce uma influência significativa nos resultados da interação entre formigas e caçadores. Quando esses elementos estão próximos, as formigas venceram na maioria dos testes realizados. Por outro lado, quando estão mais distantes, os caçadores obtiveram maior sucesso, devido à maior oportunidade de crescer numericamente (Figura 17)..

Figura 17 - Cenários enfatizando a distância entre ninho e aldeias

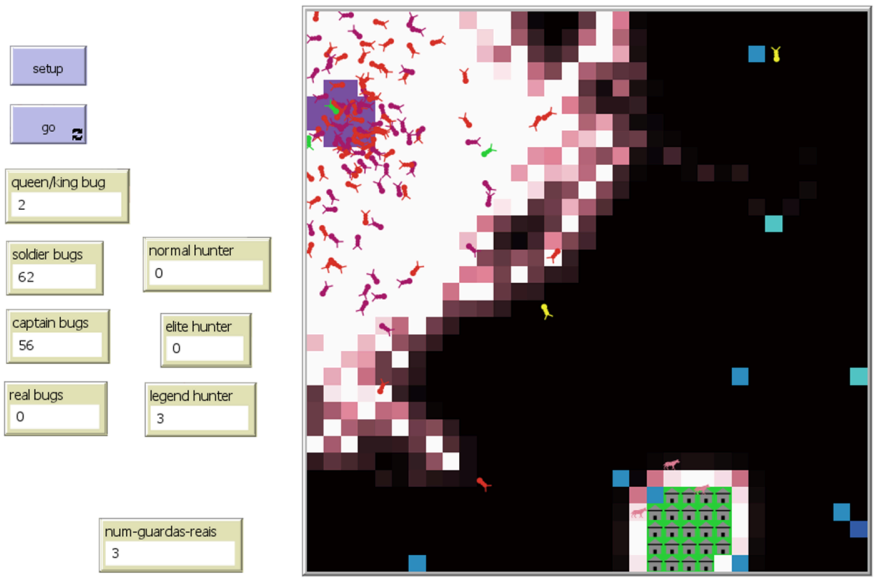


Fonte: Autores (2024)

O balanço entre dano e vida dos agentes também é um fator crítico nesta simulação, podendo as formigas vencerem pela quantidade ou os caçadores pelo dano muito alto.

Os cenários analisados revelam dois padrões distintos de movimentação coletiva das formigas, diretamente influenciados pela presença ou ausência da rainha no mapa. Quando a rainha está viva, as formigas se dedicam à busca de comida distribuída pelo mapa, como ilustrado na Figura 17. Contudo, após a morte da rainha e nascimento do rei, as formigas abandonam esse objetivo e passam a se concentrar no ponto de maior concentração de feromônio, que corresponde ao ninho (Figura 18).

Figura 18 - Comportamento das formigas na ausência da rainha.

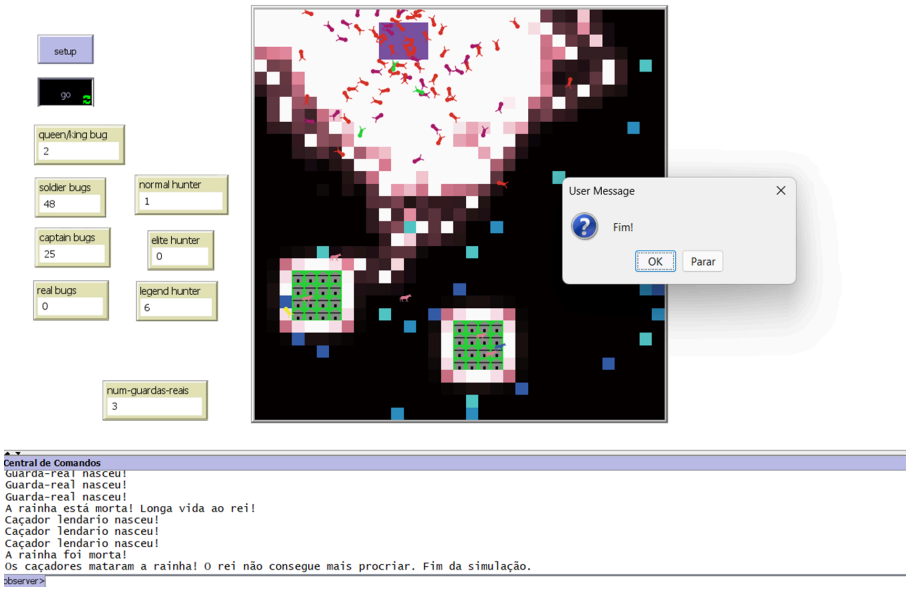


Fonte: Autores (2024)

Neste momento pode ser considerado o ponto mais crítico para a sobrevivência das formigas na simulação, pois, enquanto o rei e a rainha não se encontrarem, ambos estão indefesos e vulneráveis ao ataque dos caçadores, o que é esperado.

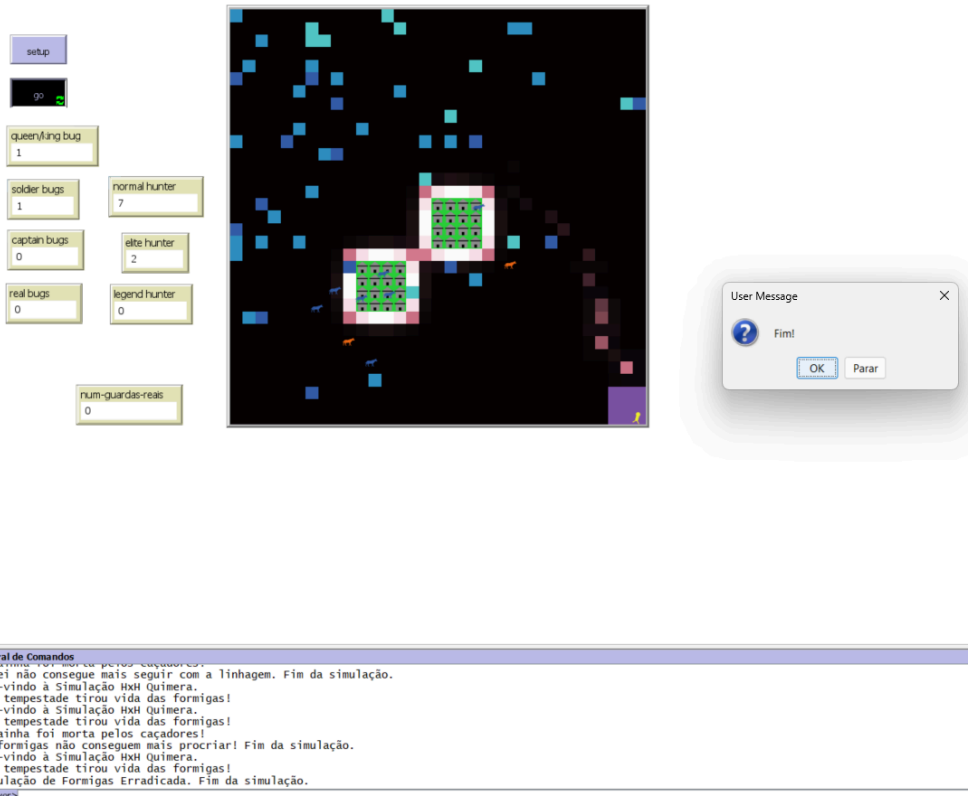
Como cenários de fim da simulação, foi modelado os casos da morte não natural do rei e rainha, o que finda a prosperidade das formigas (Figura 19) e para o caso de zerar a população de formigas (Figura 20).

Figura 19 - Cenário de morte não natural da rainha.



Fonte: Autores (2024)

Figura 20 - Formigas Erradicadas.



Fonte: Autores (2024)

4 Conclusão

A modelagem desenvolvida a partir da base no site do NetLogo e que foi inspirada no universo de Hunter x Hunter, proporcionou a análise sobre os comportamentos de agentes autônomos em um ambiente competitivo e dinâmico. A interação entre caçadores e formigas, com seus respectivos modelos, possibilitou testar as capacidades da simulação, onde a sobrevivência e o domínio de território são influenciados por diversos fatores.

Durante os testes, foi possível observar como a posição dos agentes e a dispersão dos recursos impactam diretamente no resultado das interações. A proximidade do ninho com as aldeias, por exemplo, mostrou-se determinante na eficiência das formigas, enquanto uma distância maior favoreceu os caçadores devido à sua capacidade de crescimento numérico. Além disso, as condições que envolvem a morte da rainha e o nascimento do rei introduziram um momento crítico para as formigas, deixando-as vulneráveis aos ataques dos caçadores, o que gerou um cenário de equilíbrio dinâmico e constante adaptação.

Na implementação, foi possível agregar mais parâmetros como a interação entre as variáveis de estado, como a quantidade de comida disponível e o número de caçadores mortos; modelos para os agentes e interação entre diferentes tipos de agentes. Com isso, foi possível verificar diferentes comportamentos dos agentes baseados em cenários pré-definidos e sua influência no resultado final.

Por fim, o trabalho contribuiu para a compreensão da dinâmica de sistemas complexos a partir da interação entre agentes e o desenvolvimento de cenários.

Referência bibliográfica

LIMA, Gustavo Lameirão de. Título da tese. 2024. Tese (Pós-Graduação em Computação) — Universidade Federal de Pelotas, Pelotas, 2024. Disponível em: https://guaiaca.ufpel.edu.br/handle/prefix/13591. Acesso em: 07 dez. 2024.

VERA, Maria B.; WERNECK, Luiz M. C.; KANO, Abrahão Y.; COPPETERS, Andrey Matheus; FASANDO, André Luiz; MARZULO, Leandro Augusto Justian; FURTADO, Leila de Oliveira; PEREIRA, Ligia Fortes; LOPEZ, Marco Antonio C.; PEREIRA, Rafael Barros; GRALHOZ, Ricardo Augusto; MARTINS, Ródnei Fernando de Andrade; SILVA, Thiago Schmidt da; SANTOS, Tiago Rocha M. Metodologias Orientada a Agentes. Cadernos do IME: Série Informática, v. 26, Rio de Janeiro, 2024. Disponível em: https://www.e-publicacoes.uerj.br/cadinf/article/download/6526/4645/23404. Acesso em: 07 dez. 2024.