

Dax Barreto Bogo

Comportamento Emergente em Jogos

Florianópolis, 2007

Dax Barreto Bogo

Comportamento Emergente em Jogos

Orientador:
Prof. Mauro Roisenberg

Universidade Federal de Santa Catarina
Centro Tecnológico

Florianópolis, 2007

SUMÁRIO

1	INTRODUÇÃO	01
1.1	MOTIVAÇÃO.....	02
1.2	OBJETIVOS.....	02
1.3	ORGANIZAÇÃO DO TRABALHO.....	03
2	CONTEXTUALIZAÇÃO	05
2.1	EMERGÊNCIA.....	05
2.2	FLOCKING.....	06
2.3	INTELIGÊNCIA ARTIFICIAL EM JOGOS.....	07
2.3.1	CONCEITOS DE IA EM JOGOS.....	08
2.3.2	O JOGO PREYS AND PREDATORS.....	10
2.4	DESENVOLVIMENTO DE NPCs.....	12
2.4.1	NPCS EM JOGOS DE RPG.....	12
2.4.2	NPCS EM JOGOS DE ESTRATÉGIA.....	14
2.4.3	NPCS EM OUTROS GÊNEROS.....	17
2.4.4	NOVAS TÉCNICAS.....	17
2.5	EMERGÊNCIA EM JOGOS.....	18
2.5.1	SIMCITY.....	19
2.6	NPCS EMERGENTES.....	20
3	PROJETO	22
3.1	COMO.....	22
3.2	ATUALMENTE.....	22
3.3	O JOGO.....	23
3.3.1	MANUAL.....	24
3.4	O MODELO.....	33

3.5	BASES TEÓRICAS.....	34
3.6	COMPORTAMENTOS IMPLEMENTADOS.....	35
3.6.1	CONSCIÊNCIA DO NINHO.....	35
3.6.2	MOVIMENTO CONSTANTE.....	36
3.6.3	NÍVEIS DE AGRESSIVIDADE.....	36
3.6.4	ESTÁGIOS DA DEFESA.....	37
3.6.5	DANÇA DAS ABELHAS.....	37
3.6.6	GRUPOS DE ABELHAS.....	38
3.6.7	RELAÇÕES COM AFÍDIOS.....	38
4	IMPLEMENTAÇÃO.....	40
4.1	MÉTRICAS.....	40
4.1.1	DIFICULDADE APROPRIADA.....	40
4.1.2	DIVERSIDADE DE COMPORTAMENTOS.....	40
4.1.2	AGRESSIVIDADE.....	40
4.1.4	MOVIMENTAÇÃO.....	40
4.2	IMPLEMENTAÇÃO E CONCLUSÕES.....	41
4.2.1	CONSCIÊNCIA DO NINHO.....	41
4.2.2	MOVIMENTO CONSTANTE.....	42
4.2.3	NÍVEIS DE AGRESSIVIDADE.....	43
4.2.4	ESTÁGIOS DA DEFESA.....	45
4.2.5	DANÇA DAS ABELHAS.....	48
4.2.6	GRUPOS DE ABELHAS.....	48
4.2.7	RELAÇÕES COM AFÍDIOS.....	50
	CONCLUSÃO.....	51
	REFERÊNCIAS.....	53

Lista de Abreviaturas e Siglas

Abs: Absolut

AG: Algoritmo Genético

AI: Artificial Intelligence

CPU: Central Processing Unit

ESA: Entertainment Software Association

FSM: Finite State Machine

IA: Inteligência Artificial

IDE: Integrated Development Environment

J2ME: Java 2 Micro Edition

MIDP: Mobile Information Device Profile

NPC: Non-Player Character

PC: Personal Computer

RPG: Role Playing Game

WW: Whatever Works

Lista de Figuras

2.1	Regras para a movimentação dos <i>Boids</i>	6
2.2	Screenshot de um jogo estilo Preys & Predators.....	10
2.3	Age of Empires 3: tipos de unidades.....	15
3.1	Castelo Defendido por Soldados Vermelhos.....	28
3.2	Recursos Sendo Atacados por Soldados Azuis.....	29
3.3	Tela de Posicionamento dos Soldados.....	31
3.4	Exemplo de Execução de Ordem.....	32
3.5	Campo de Visão de um Agente.....	33
3.6	FSM de um soldado.....	34
4.1	Soldados se movem, mas não se afastam do ninho.....	41
4.2	Soldados vermelhos mortos, quando não fogem.....	44
4.3	Soldados vermelhos mortos, quando não fogem.....	45
4.4	Soldados vermelhos mobilizados para a defesa.....	47
4.5	Soldados vermelhos mais afastados alheios ao ataque.....	47

1 Introdução

Jogos eletrônicos são uma parte importante da indústria do entretenimento e têm tido grandes avanços recentemente, conforme as vendas totais aumentam e os orçamentos para o desenvolvimento idem (forbes.com). Contudo, os avanços, que foram principalmente na parte de criar um mundo com gráficos e sons convincentes, têm diminuído de ritmo. Para continuar a tornar os jogos cada vez mais interessantes e atraentes existem outras características que podem ser desenvolvidas.

A jogabilidade (maneira como o jogo é jogado, funções de botões e outros controles e responsividade do jogo aos comandos) foi radicalmente alterada pela Nintendo em 2006, com seu console Wii, que têm controles sensíveis ao movimento (nintendo.com). Outra característica importante nos jogos é a inteligência artificial. Ela é usada, entre outras coisas, para ajustar dinamicamente o nível de dificuldade de um jogo, tornando-o mais divertido, e para controlar personagens do jogo. Praticamente todos os jogos atuais possuem personagens controlados pelo computador, com os quais o jogador interage. Esses personagens são chamados de NPCs (Non-Player Characters).

Diversas técnicas de IA vêm sendo aplicadas para desenvolver NPCs adequados aos jogos. Atualmente, as mais estudadas e testadas consistem no uso de Algoritmos Genéticos e Redes Neurais, ou uma combinação de ambos, como estudado por Yannakakis (Yannakakis, 2005).

Este trabalho visa o estudo e a aplicação de técnicas de Inteligência Artificial em NPCs. Mais especificamente Sistemas Emergentes, que consistem em agentes simples que interagem gerando um comportamento complexo e muitas vezes imprevisível.

1.1 Motivação

Apenas dentro dos Estados Unidos, as vendas de jogos para computadores e consoles no ano de 2006 foram de \$7.4 bilhões de dólares segundo a *ESA (Entertainment Software Association, www.theESA.com)*. Os NPCs são uma parte muito importante dos jogos, de várias maneiras: eles podem definir o grau de dificuldade de um jogo, caso o jogador tenha que enfrentar algum NPC (por exemplo, jogos de futebol), podem ser o tema principal do jogo, no qual o jogador interage muito com NPCs (série *Creatures*), para o caso de haverem muitos NPCs para serem processados, e uma IA complexa, podem influenciar negativamente o desempenho de um jogo.

Contudo, o desempenho das técnicas de inteligência artificial usadas em jogos é freqüentemente criticado em análises de jogos. Essas análises são escritas há anos, por profissionais experientes, e são lidas e apreciadas em todo o mundo, ou seja, desenvolver IA para jogos é ao mesmo tempo importantíssimo e desafiador.

Com a intenção de melhorar o desempenho da IA nos jogos, a inspiração para este trabalho surgiu durante a leitura de um livro chamado *Emergência*, de Steven Johnson. Nele, o autor apresenta o conceito de emergência bem como vários exemplos onde ela ocorre (sistemas emergentes).

O emprego de sistemas emergentes em jogos com vários NPCs deve fazer surgir comportamentos complexos e possivelmente imprevisíveis desses NPCs, isso deve tornar os jogos mais atraentes e divertidos, pois o jogador gosta de ser surpreendido e de ver novidades em jogos. Posteriormente, análises mais positivas que gerem maiores vendas de jogos e lucro para os desenvolvedores.

1.2 Objetivos

Os objetivos deste trabalho são criar um modelo teórico de agente que componha um sistema emergente e a implementação de um jogo no estilo *Preys and Predators* utilizando esses agentes.

Em jogos no estilo *Preys and Predators* existem dois grupos de agentes: os caçadores e as presas. Geralmente os caçadores têm um comportamento fixo e se restringem a perseguir e eventualmente comer as presas. As presas são grupos de agentes que são implementados com o objetivo de testar e avaliar técnicas de IA. Por exemplo, algoritmos de *flocking* e algoritmos genéticos.

Aqui, o estilo foi um pouco alterado, no sentido de que os inimigos, no papel de presas, conseguem atacar os personagens do jogador, mas sua função principal é a defesa do território. Os agentes do jogador, no papel de predadores, devem atacar o território do inimigo, que compreende os agentes, construções com recursos e o castelo (ou ninho), que deve ser defendido a qualquer custo.

O modelo teórico de agentes deverá ser bastante flexível e adaptável para gerar diferentes sistemas emergentes, como por exemplo: formigas em um formigueiro virtual, aves que se deslocam em bando e aldeões em uma vila de jogos de RPG (*Role-Playing Game*).

No jogo deste trabalho, as presas formarão um sistema emergente, onde, a partir de seres relativamente simples e limitados interagindo uns com os outros, um comportamento mais complexo deverá emergir do grupo. Já os predadores serão controlados pelo jogador, mas também terão um comportamento emergente.

1.3 Organização do Trabalho

O trabalho será dividido em cinco capítulos onde o primeiro apresenta a introdução ao trabalho, objetivos e motivação.

Em um segundo momento, um capítulo chamado de Contextualização irá abranger uma pesquisa bibliográfica e desenvolvimento de idéias e técnicas ligadas ao tema do trabalho.

O terceiro capítulo, batizado de Projeto, será composto pelo modelo de agente para sistemas emergentes, os comportamentos que serão implementados pelos agentes e a definição do jogo estilo Preys and Predators.

Na seqüência, o capítulo Implementação apresenta algumas métricas que nortearão as conclusões sobre os comportamentos implementados. Este capítulo também descreverá o desenvolvimento do jogo e os resultados obtidos com as técnicas aplicadas.

Por fim, uma conclusão geral, com uma avaliação dos objetivos propostos e resultado prático obtido a partir da técnica empregada, e sugestão de trabalhos futuros.

2 Contextualização

2.1 Emergência

Emergência é o surgimento de comportamentos de um nível mais alto a partir de massas de elementos relativamente simplórios de um nível mais baixo (Johnson, 2001). Um excelente exemplo, que consta na literatura do tema, é a organização dos formigueiros. Ao contrário do que podemos imaginar a princípio, uma colônia de formigas não é governada pela formiga rainha. A rainha apenas põe ovos. São as próprias formigas que se organizam em colônias que duram até quinze anos, ao contrário de cada formiga, que vive no máximo doze meses (Johnson, 2001).

As características mais comuns nesses sistemas emergentes são: novidades (características não observadas anteriormente em sistemas); coerência ou correlação (significando estruturas e relações que se mantêm durante um período de tempo); são resultado de um processo dinâmico, que evolui; possuem um nível que representa o “todo”, ou seja, o resultado da emergência no sistema; e, por fim, a emergência é perceptível, i.e. pode ser notada (Corning, 2002).

É fácil, para qualquer programador com conhecimentos no campo da inteligência artificial, perceber as várias possíveis utilizações de sistemas emergentes em jogos. Uma ambiciosa, seria uma cidade virtual habitada por NPCs que se auto-organizasse e evoluísse, onde um jogador pudesse visitar e interagir. No outro extremo, um exemplo muito simples é um cardume de peixes. Os peixes são independentes, nadam sozinhos, porém, eventualmente se organizam e um padrão é notado, com os peixes nadando todos próximos uns dos outros e na mesma direção. Para desenvolver este comportamento, uma técnica conhecida como *Flocking* é empregada.

2.2 Flocking

Flocking é uma técnica que consegue reproduzir com impressionante realismo o comportamento de bandos de pássaros, miríades de insetos e cardumes de peixes. A primeira simulação foi desenvolvida por Craig Reynolds em 1986 e chamava-se *Boids*. Consistia de agentes simples que se movimentavam com a ajuda de algumas regras simples:

- Separação: evitar vizinhanças muito lotadas
- Alinhamento: seguir na mesma direção que os vizinhos
- Coesão: se dirigir à posição média dos vizinhos

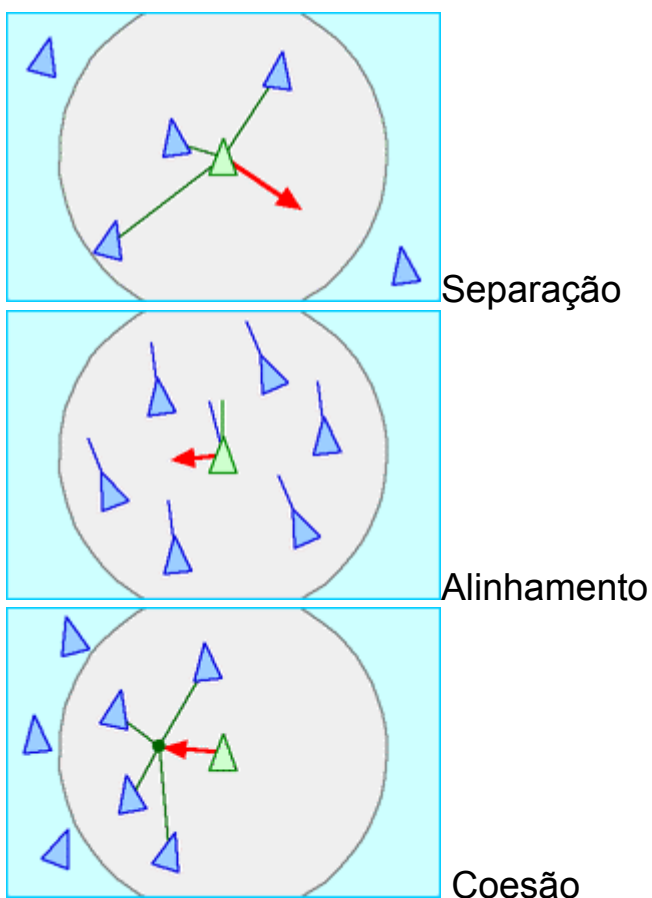


Figura 2.1: Regras para a movimentação dos *Boids*.

Essas três regras são baseadas no conceito de vizinhos. Vizinhos são os outros agentes que se encontram na área próxima ao agente definida por um raio, chamado de raio de vizinhança.

A técnica tem vasta aplicação em jogos, tendo, como exemplo notável, os cardumes de peixes no jogo Zelda para o Nintendo 64.

Diversos fatores podem aumentar a complexidade da simulação, como por exemplo, a inserção de objetivos, obstáculos a serem desviados e inserção de novas regras para a movimentação. Uma nova regra poderia instruir quanto ao comportamento do agente para o caso de ele não ter nenhum vizinho para se orientar.

2.3 Inteligência Artificial em Jogos

Atualmente, segundo os números da ESA, os jogos de estratégia, tiro e esportes somam por volta de 50% das vendas de jogos para PCs. Nesses gêneros de jogos, na grande maioria das vezes, o jogador joga contra e/ou junto com NPCs. O desempenho desses personagens virtuais vem se tornando tão importante quanto os quesitos jogabilidade e gráficos, entre outros.

O objetivo do emprego da Inteligência Artificial dentro destes jogos é tornar os personagens e oponentes mais inteligentes, capazes, criativos e parecidos com humanos. Uma cilada comum ao desenvolver um software é tentar criar algoritmos que joguem da maneira mais eficiente possível, um lutador que ganhe sempre, um jogador de futebol que sempre faça gol... Tudo isso é realizável, mas o mais importante é uma IA que torne o jogo uma experiência divertida e gratificante.

A IA como ciência é muito recente, tendo surgido em meados do século XX (Russel, Norvig, 2003, p. 17) , e sua aplicação em jogos mais recente ainda. Nas décadas passadas, quando os computadores pessoais estavam ainda engatinhando em termos de hardware e processamento e os consoles de vídeo games estavam na geração

8-bits, não havia como desenvolver IA como conhecemos hoje, os jogos se limitavam a implementar padrões de comportamento e em muitos casos os programadores deixavam a IA “roubar”, ou seja, a IA tinha acesso a informações que um jogador normal não teria, concebendo assim uma performance muito melhor, com pouco custo de processamento, ou o agente controlado pelo computador tinha acesso a bônus (ex. dinheiro extra, vidas, velocidade...).

Esses métodos paliativos, com o tempo, deixaram de ser usados em virtude do desenvolvimento de hardware mais avançado e para aumentar o grau de satisfação dos jogadores, que se sentem melhor jogando contra IAs mais realistas. Acompanhando a evolução do hardware, temos também o avanço das técnicas de IA aplicadas nos jogos.

2.3.1 Conceitos de IA em Jogos

Brian Schwab, em seu livro *AI Game Engine Programming* chama a atenção do leitor para o modelo *Whatever Works* (qualquer coisa que funcione), *WW*. Isso significa dizer que IA aplicada a jogos não se preocupa tanto com a excelência do código ou com a otimização máxima dos algoritmos, mas sim com a tarefa de criar um agente realista, isto é, que tome decisões consideradas humanas pelo jogador. Isto não significa dizer que devemos relaxar na parte de codificação ou não nos atentar para a eficiência do código, significa apenas que devemos focar no grau de satisfação do jogador.

A estrutura básica do módulo de IA do software contém o carregamento de dados, a instanciação e o laço principal. O laço principal é dividido em três fases: tomada de decisões, percepção e navegação. Na tomada de decisões é onde o agente, ao receber estímulos, avalia as opções e age. Várias técnicas podem ser utilizadas, incluindo Máquinas de Estados Finitos e Redes Neurais. A percepção é o conjunto de todos os dados que o agente tem acesso para tomar uma decisão. Sugere-se que esses dados estejam reunidos em uma estrutura central, que pode ser acessada por todos

os agentes. Navegação é o modo que um agente encontra para ir de um ponto A até um ponto B. Sendo a maneira mais simples é transformar o terreno todo em uma malha de quadrados ou hexágonos, atribuir a cada setor um valor de facilidade de travessia e utilizar o algoritmo A* para encontrar o menor caminho.

A* é um algoritmo de busca em árvores que encontra o caminho de menor custo com o auxílio de eurísticas admissíveis, ou seja, que nunca superestimam o custo de um caminho.

A máquina de estados finitos (FSM) é, provavelmente, a estrutura mais utilizada na programação de IA em jogos. Uma FSM consiste basicamente de três partes: os estados, entradas e uma função de transição. Devido à facilidade da etapa de projeto, implementação e correção de erros e aplicabilidade em quase todos os problemas de IA, é uma ferramenta que qualquer desenvolvedor deve dominar.

Quando o problema é muito complexo para ser resolvido em tempo hábil durante um jogo, e o espaço de soluções é muito grande, podemos nos contentar com uma boa solução, não necessariamente uma solução ótima. Nessa busca de uma solução, Algoritmos Genéticos têm sido usados. Eles podem ser usados, por exemplo, para calibrar as técnicas de esquiva de uma nave espacial desviando de asteróides. Quando desviar? A qual distância? Para qual direção? Cria-se uma população inicial, a cada geração vai-se avaliando a aptidão dos indivíduos em desviar de asteróides e utiliza-se o princípio biológico da sobrevivência do mais apto. Isso será feito antes de o jogo ser lançado, durante o desenvolvimento, mas também é possível utilizar AGs em tempo real, durante o jogo, principalmente em jogos que incluem aprendizado.

Onde os AGs utilizam o princípio da sobrevivência do mais apto para encontrar evoluir soluções de dentro de um grupo de possibilidades, Redes Neurais simulam o funcionamento do cérebro. Em jogos, podemos treinar uma rede com casos de uso, ou com dados de jogadores humanos reais, e ela poderá fazer um bom trabalho utilizando esses padrões, mesmo quando as entradas são diferentes das contidas no treino.

2.3.2 O Jogo Preys and Predators

Como mencionado no capítulo anterior, jogos no estilo Preys and Predators são jogos onde existem dois grupos de agentes: os caçadores e as presas. Geralmente os caçadores têm um comportamento fixo e se restringem a perseguir e eventualmente comer as presas. As presas são grupos de agentes que são implementados com o objetivo de testar e avaliar técnicas de IA. Por exemplo, algoritmos de *flocking* e algoritmos genéticos.

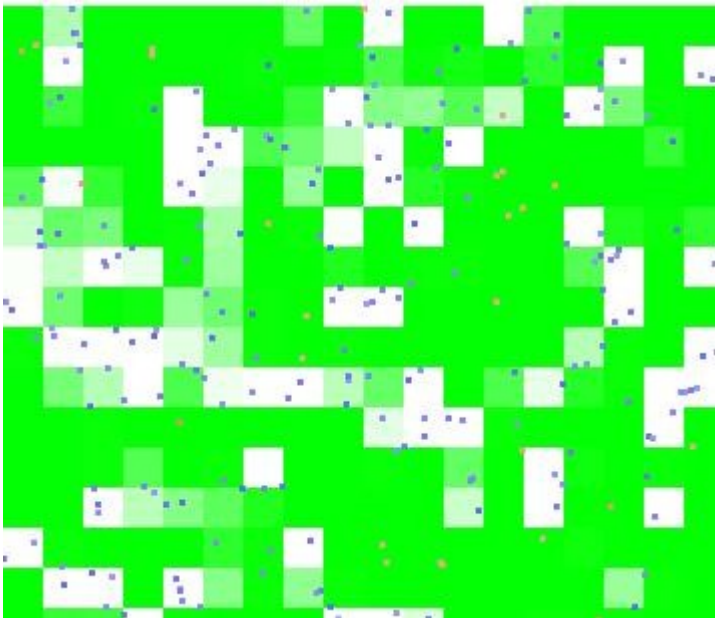


Figura 2.2: Screenshot de um jogo estilo Preys & Predators.

Como podemos conferir na Figura 2.2, caçadores estão representados por pontos vermelhos, as presas por pontos azuis e a comida da presa (grama) por quadrados verdes. Onde não há comida o chão está branco. Nessa implementação, especificamente, as presas só sobrevivem onde há grama. Caso várias presas fiquem no mesmo local, não haverá grama para todas, e elas morrerão de fome. Já os predadores se alimentam exclusivamente de presas.

Existem variantes dessa implementação. Podemos considerar que as presas têm comida infinita, e só precisam se preocupar em evitar os

predadores, o que simplifica comportamento da presa. Outra opção é inserir barreiras no mapa, isso dificulta a movimentação dos agentes, o que pode privilegiar os que tiverem melhores habilidades de locomoção. Podemos explicitamente desenvolver um sistema que entre em equilíbrio após algumas épocas, para demonstrar um sistema mais próximo da natureza. Nesse sistema, ao se elevar o número de presas, os predadores, com comida abundante acabam se multiplicando rapidamente. Isso causará uma diminuição drástica na quantidade de presas, e os predadores, eventualmente morrerão de fome. Com menos predadores, as presas se multiplicam. E assim sucessivamente, para o caso de nenhuma espécie ser extinta. Com a extinção dos predadores, as presas se multiplicam livremente. Com a extinção das presas, a extinção dos predadores é só uma questão de tempo.

Esse estilo de jogo é muito prático para testar e avaliar técnicas de IA, pois apresenta uma interface gráfica muito simples de ser desenvolvida, e dá ao observador um retorno bem intuitivo. Até uma criança pode entender, por exemplo, que os predadores estão comendo todas as presas.

Eric Obermuhler, em seu programa Blobs, simula evolução genética tanto para os predadores quanto para as presas. Ele avisa aos observadores que, como a evolução é bastante randômica, tanto as presas quanto os predadores podem evoluir rápido demais e desbalancearem o meio. O código genético de cada agente é extremamente simples, contendo estados e direções. Os estados são mudados com eventos simples como esbarrar em uma parede ou encontrar algum outro agente. Para cada estado uma direção de movimentação é associada. As direções são norte, sul, leste, oeste, nordeste e etc. Além destas, ainda existem as direções relativas a quem o agente encontrou, e ficar parado, ou se mover aleatoriamente.

Nas primeiras gerações podemos observar indivíduos extremamente ineficientes, que se movem aleatoriamente para cada direção, a cada momento. No fim, eles acabam mal saindo do lugar. Com o tempo, os predadores que preferem andar mais tempo na mesma direção acabam desbancando os outros, exceto quando eles ficam presos em

um canto do cenário, nesse caso, a linhagem acaba perdida. As presas também se dão melhor quando se movem em uma mesma direção, exceto se quando elas se movem demais, então eventualmente encontrarão um predador.

2.4 Desenvolvimento de NPCs

“Inserir NPCs em seu jogo será uma das tarefas mais difíceis que você irá enfrentar.” André LaMothe, CEO da Xtreme Games LLC.

NPC (Non-Player Character em inglês), significa exatamente Personagem que não é um jogador. NPCs são todos os personagens controlados pelo computador. Além de serem uma parte muito importante do jogo, são também complexos e cheios de particularidades. Cada gênero de jogo demanda um comportamento específico.

Como o tema deste trabalho é a emergência, precisamos de um número mínimo de NPCs em um jogo para que algo nesse sentido aconteça. Dos gêneros mais comuns de jogos, RPGs e Estratégia possuem, geralmente, o maior número deles, por isso vamos nos concentrar nos mesmos. Os outros gêneros vão ser abordados resumidamente, além de observações sobre as novas técnicas de desenvolvimento de NPCs abordadas em artigos recentes.

2.4.1 NPCs em Jogos de RPG

Jogos de RPG surgiram fora do mundo dos computadores, onde grupos jogavam apenas com livros e dados, ou nem isso. O apelo deste gênero, inclusive nos videogames, é a possibilidade de o jogador se transportar para outro mundo e “encarnar” no personagem com quem se joga. Para tornar isso tudo mais divertido, os universos

nos jogos de RPG devem ser completos e detalhados, o que aumenta o senso de imersão nos jogos.

Para tornar o mundo mais crível e o jogo mais divertido, diversas classes de NPCs podem estar presentes. Existem os habitantes normais das cidades e vilas, que apenas passeiam por aí e transmitem algumas informações ao jogador. Alguns desses habitantes são mercadores, que além do padrão, ainda interagem com o jogador na forma de compra e venda de mercadorias. Outros NPCs são membros da equipe do jogador, ou seja, lutam junto com ele.

Esses membros da equipe foram muito desenvolvidos nos últimos anos, em parte por causa dos combates em tempo real nos jogos de RPG. Em uma batalha em tempo real, o jogador não consegue controlar todos ao mesmo tempo, esses membros devem agir sozinhos, e de maneira inteligente. Vários jogos disponibilizam ao jogador alguma maneira simplificada de programação. Nesse sentido, os membros podem ficar mais covardes ou mais agressivos, podem focar na cura ou na defesa do jogador. O mais notável exemplo é o japonês Final Fantasy 12, lançado em 2006. Nele, o jogador podia combinar centenas de comportamentos (atacar inimigo mais forte, curar líder quando estiver com menos de 50% de energia, ...), gerando milhares de possibilidades.

Uma das técnicas de IA mais utilizadas nesses jogos é a criação de roteiros (*scripts*). Por exemplo, no trecho de código abaixo, o jogo avisa ao jogador o objetivo atual através de um NPC. Note que, através do uso de *flags*, o rei dirá a primeira frase apenas uma vez, depois mudará para a segunda, e após o herói salvar a princesa, ele mudará novamente para a terceira frase.

```
Se (player.falaCom(rei_caspio)){
    Se flag_rei_caspio = 0{
        Mensagem(script.mensagens[0]);
        flag_rei_caspio = 1;
    }
    Se flag_rei_caspio = 1{
        Mensagem(script.mensagens[1]);
    }
}
```

```

    }
    Se flag_rei_caspio = 2{
        Mensagem(script.mensagens[19]);
    }
}

```

Mensagens:

//ao falar primeira vez com o rei

Mensagens[0] = "Você deve salvar a princesa, ela está na masmorra!"

//depois da primeira vez, sem ter salvo a princesa

Mensagens[1] = "Você ainda não salvou a princesa? Corra!"

...

//quando a princesa já foi salva

Mensagens[19] = "Muito obrigado, bravo herói!"

Nesse exemplo rústico de roteiro, o código é escrito e compilado apenas uma vez, em outro arquivo ficam as mensagens. Essas mensagens podem ser escritas, alteradas e traduzidas por alguém sem conhecimento algum de programação.

Ainda sobre o código acima podemos observar uma FSM que controla o rei, ela tem três estados. FSMs são amplamente utilizadas no controle de NPCs, tanto em NPCs que apenas conversam com o jogador, quanto em NPCs que são membros da equipe.

2.4.2 NPCs em Jogos de Estratégia

Jogos de estratégia permitem ao jogador a possibilidade de ser o general supremo de todo um exército e de controlar a economia que o reabastecerá. Por economia entende-se o conjunto de trabalhadores, construções e recursos naturais que o universo do jogo provê ao jogador para que ele possa desenvolver seu exército. O mais complexo tipo de NPC em um jogo desse estilo são os generais supremos de outros exércitos. Além de envolver vários objetivos conflitantes (coletar recursos, atacar adversário...) e o controle específico de cada unidade, o general ainda tem que lidar com

planejamento de longo prazo: por ex. o que fazer para ter um exército forte daqui a 30 minutos. Neste trabalho o foco serão os outros NPCs, as unidades básicas, que podem estar no exército do jogador, em outros exércitos e até mesmo serem unidades livres ou fazerem parte da fauna do cenário.

Sob o controle dos generais estão as unidades básicas, mesmo sendo controladas em parte pelo general, precisam ter algum tipo de comportamento autônomo. Elas se dividem em dois grupos: de combate e econômicas. As unidades de combate precisam, principalmente, decidir sobre a própria navegação, desvio de obstáculos, concentração de ataques e a hora de fugir. As unidades econômicas raramente lutam, elas servem para coletar recursos e construir estruturas. Geralmente elas não decidem sozinhas o que coletar, isso é decidido pelo general, o comportamento autônomo está na navegação, desvio de obstáculos e fuga.

Existe ainda um terceiro tipo de unidade básica, que não está sob o controle de nenhum general, apenas compõe o cenário. Essas unidades podem ser índios, animais de caça ou predadores que atacam unidades econômicas se tiverem chance.



Figura 2.3: Age of Empires 3: tipos de unidades.

Uma técnica de IA comum em jogos desse tipo é o general controlar as unidades de forma individual e impossível para humanos. Um exemplo é o comportamento dos arqueiros no jogo Age of Empires. Eles atiram uma flecha, recuam, atiram outra flecha, recuam... Ao mesmo tempo em que controlam todos os outros aspectos do jogo. Para um humano isso é impossível, supondo que ele consiga apenas emitir 2 ou 3 ordens por segundo, um computador pode emitir milhares. Ao observar a CPU utilizar esse recurso, os jogadores se sentem traídos.

Para compensar essa limitação humana, os desenvolvedores poderiam restringir o número de ordens que o general controlado pelo computador pode emitir. Além disso, seria possível criar NPCs que agissem melhor em grupos, sem ordens de superiores. Ao desenvolver maneiras de um NPC se comunicar com os outros e buscar a cooperação, por ex. um arqueiro poderia ser facilmente cercado após atacar um soldado, e este pedir ajuda aos seus companheiros. Os comportamentos emergentes desses grupos de NPCs não só resolveriam esse tipo de limitação dos jogadores humanos, como também aumentariam a diversão do jogador.

Para que este comportamento aconteça, os NPCs devem ser capazes de se comunicar. Isso pode ser implementado com o uso de mensagens. Por ex. caso um soldado esteja sendo atacado, ele pode comunicar a todas as outras unidades dentro de seu raio de comunicação. Em termos de performance, mensagens são muito mais eficientes do que outras soluções mais simples. Imagine um agente que precisa saber se tem alguém sendo atacado, ele percorre um vetor com todos os agentes e verifica se algum está sendo atacado. Muito mais eficiente seria um agente não precisar verificar, mas sim receber uma mensagem de outro agente, dizendo “estou sendo atacado!”.

2.4.3 NPCs em outros Gêneros

Outros gêneros que dependem enormemente de NPCs são os jogos de corrida, esportes e tiro em primeira ou terceira pessoa (FTPS). Nos jogos de corrida temos os corredores e os outros humanos que compõem o cenário, como nos jogos de RPG. Jogos de esportes possuem NPCs extremamente complexos e detalhados, tendo em vista que o jogador desses jogos conhece bem o esporte em questão. NPCs de jogos de esporte costumam ter seus comportamentos ditados por uma base de dados, onde constam várias jogadas e estilos de jogo, entre outras informações. Por ex. Um atacante em um jogo de futebol pode ser modelado para representar um jogador real. Nesse caso, a base de dados conterá informações sobre o estilo de jogo desse jogador, jogadas mais comuns e graus de habilidade em cada quesito.

Jogos do estilo FTPS costumam ter NPCs bastante variados. Existem jogos que a diversão está em atirar em quantidades enormes de monstros que apenas correm na direção do jogador e atiram. Outros jogos apresentam um modo de jogo onde o objetivo é vencer apenas um inimigo, nesse caso, o NPC é modelado de maneira a jogar tão humanamente quanto possível.

2.4.4 Novas Técnicas

Ultimamente, graças à necessidade de melhorias na parte de IA de NPCs, novas técnicas vem sendo desenvolvidas para complementar as tradicionais máquinas de estados finitos, máquinas de estados finitos com lógica difusa, trocas de mensagens e roteiros.

Um dos problemas mais famosos da IA é o do Caixeiro Viajante. Como encontrar o menor caminho para passar por várias cidades. Enquanto é relativamente simples para poucas cidades, a dificuldade cresce exponencialmente e se torna computacionalmente inviável encontrar

o menor caminho caso o número de cidades seja muito grande. Algoritmos genéticos vêm sendo usados com sucesso para encontrar bons caminhos. Vários problemas de IA para jogos têm grandes semelhanças com o problema do caixeiro viajante. Como por exemplo a orientação (pathfinding). AGs podem ser usados em NPCs para que eles encontrem um bom caminho entre o ponto A e ponto B.

Em jogos que simulam aprendizagem, por ex. um NPC que seja um animal doméstico que leva choque sempre que come a ração azul, então começa a comer a ração verde, Redes Neurais vêm sendo aplicadas. Sempre que o cão come a ração azul e leva um choque, os pesos da rede neural são modificados, considerando comer a ração azul uma decisão ruim. Ao comer a ração verde, os pesos também são alterados, mas positivamente.

2.5 Emergência em Jogos

Como visto anteriormente neste capítulo, a emergência é um fenômeno interessantíssimo e com grande potencial no mundo dos jogos eletrônicos. Revisando: Emergência é o termo que designa o fenômeno no qual agentes simples, seguindo regras simples, são capazes de gerar, através da auto-organização, estruturas surpreendentemente complexas.

Transportando esse conceito para os jogos, os agentes representam os NPCs (em um sentido mais amplo), as regras simples são a programação simples que o desenvolvedor cria para o NPC e a partir de alguma forma de auto-organização, ou seja, interação entre os NPCs, surge um comportamento mais complexo do que poderia se esperar individualmente desses NPCs.

Em 1996 foi lançado um jogo chamado *Gearheads*. No jogo existiam 12 tipos diferentes de brinquedos, cada um com um comportamento exclusivo. No jogo, o jogador deveria escolher alguns tipos desses brinquedos e lançá-los em um tabuleiro, onde competiriam contra os bonecos adversários, quem atravessasse o tabuleiro com 21 deles

primeiro venceria. Uma vez dentro do tabuleiro, os brinquedos já não poderiam ser controlados pelo jogador, estavam independentes. Dependendo de como esses NPCs interagissem entre si, o jogo poderia ser ganhado ou perdido. Graças a essas interações, comportamentos inusitados, e não previstos, emergiam, como os grupos de 3 brinquedos específicos que se juntavam e percorriam juntos o tabuleiro.

Quando um mundo de um jogo é construído de maneira a permitir interações entre todos os indivíduos que nele habitam, um jogador humano, que faz parte desse mundo, perceberá que o que ele faz influencia o estado geral desse mundo. Imagine um jogador que gosta de caçar os lobos que habitam a região próxima a uma cidade. Se esse jogador matá-los todos, e outro caçador, um NPC, não conseguir peles de lobo para revender ao comerciante da cidade, o jogador, surpreendentemente verá os preços dos casacos de pele subirem

Em um universo totalmente realista esse efeito não seria surpreendente, mas os jogadores maduros se acostumam com as várias abstrações dos jogos, dentre elas a simulação incompleta das interações entre os diversos fatores envolvidos neste exemplo. Assim sendo, o jogo simula um caçador vendendo peles de lobo e um comerciante revendendo-as na cidade, mas essas peles não são as mesmas dentro da simulação, e a falha em um dos elos dessa cadeia produtiva acaba não repercutindo para o jogador.

Temos então, até aqui, duas maneiras de o jogador se deparar com a emergência em seu jogo. Caso ele esteja na posição de um Deus ou general, ele espera controlar indiretamente bandos, agrupamentos de NPCs que estarão no cenário do jogo, por exemplo soldados em um campo de batalha. Há também a possibilidade de o jogador fazer parte desse cenário, interagir com os NPCs como um semelhante, observar suas ações e efeitos nas relações entre os outros personagens.

2.5.1 **SimCity**

SimCity 2000 foi um jogo inicialmente lançado para computadores em 1993 e foi um grande sucesso de público e crítica.

No jogo, o jogador constrói uma cidade partindo do nada, ou quase nada. O segredo está na organização da cidade, na forma imprevisível como ela evolui. A falta de delegacias numa região pode aumentar a criminalidade e afugentar moradores, tornando as residências meras favelas, e conseqüentemente minar o comércio ao redor. O jogador, mesmo querendo ter uma cidade saudável e próspera, não tem como influenciar diretamente nesses casos, cabe a ele detectar a falta de delegacias e construir alguma na região.

Cada setor do mapa foi programado para influenciar os setores adjacentes, como no caso das favelas que acabam por prejudicar o comércio, gerando assim um comportamento emergente, que no final dá à simulação o aspecto de um ser vivo, que cresce e reage a estímulos do jogador.

2.6 **NPCs Emergentes**

Um dos primeiros, senão o primeiro modelo de NPC que apresenta emergência quando em grupo, é o Boid, de Burt Reynolds. Cada Boid tem apenas três regras simples (separação, alinhamento e coesão), e no entanto, padrões aparentemente complexos e realistas surgem. Atualmente Reynolds trabalha para a Sony desenvolvendo para o Playstation 3, em 2006 já havia conseguido que quinze mil NPCs fossem simulados e exibidos em tempo real.

Muito já se foi estudado sobre a emergência de agentes egoístas, como por exemplo no dilema dos prisioneiros. Teoria dos jogos também é um tema recorrente, e a emergência estudada em outros

campos do conhecimento, como a economia. Neste trabalho estamos focando em jogos, e no caso dos NPCs, fazemos uma restrição, os NPCs se esforçam para cooperar uns com os outros, seja um grupo de soldados atacando uma base militar, seja um grupo de aldeões defendendo sua vila de lobos ferozes.

Para essa cooperação acontecer, os agentes precisam se comunicar. Em trocas de mensagens eles dizem para os outros o que estão fazendo e o que estão vendo, pois cada agente tem um campo de visão restrito, e diferente. Ao fazer uma decisão, um agente leva em consideração a situação dos outros, como já acontecia com os Boids. Uma lista possível de passos para cada agente seria:

1. Observar
2. Processar intenções de outros agentes
3. Processar observações de outros agentes
4. Avaliar benefício de cada ação possível
5. Executar a melhor ação

Obs. Podemos dizer que, no passo 5, ele nem sempre executa a melhor ação. Isso por várias razões: pode não ter tido tempo de encontrar a melhor ação, pode não contar com informações completas sobre o sistema e ter q tentar “adivinha”, pode deliberadamente escolher outra opção, para parecer mais humano.

3 Projeto

3.1 Como

A plataforma adotada para o desenvolvimento é um computador tipo PC com o Eclipse IDE. Em cima do eclipse é instalado um *Development Toolkit* específico para a plataforma-alvo, que são aparelhos móveis com implementações de máquinas virtuais Java J2ME (Java 2 Micro Edition).

Dentro da especificação do J2ME, vários subsistemas foram previstos. Eles são chamados de Mobile Information Device Profile (MIDP). Para um aparelho se encaixar em um desses perfis, ele deve ser capaz de atender aos pré-requisitos de hardware e implementar todas as funções das bibliotecas dos perfis.

A aplicação será desenvolvida focando um aparelho que tenha uma resolução de tela de pelo menos 176 por 220 pixels e MIDP 2.0. Dentre os aparelhos que se encaixam nesse requisito estão os populares Sony Ericsson W800 e K750 e todos os aparelhos Nokia da série 60.

3.2 Atualmente

Não há, para a plataforma-alvo escolhida, nenhum jogo que tenha uma jogabilidade baseada em comportamento emergente. Devido às limitações de hardware e orçamentos

para o desenvolvimento dos jogos, IA é, assim como muitos anos atrás acontecia com as demais plataformas, relegadas ao segundo plano. Fazem o mínimo suficiente para não chamar atenção negativamente, ou nem isso. Um exemplo de jogo de estratégia que tem personagens burros é *Strategy Wars* de 2004. Nele, os personagens, tanto controlados pelo jogador quanto os inimigos se restringem a atacar um adversário que esteja dentro de seu campo de visão.

Ao mudar para plataformas mais avançadas, como os PCs, encontramos alguns exemplos do emprego de comportamento emergente em jogos em maior ou menor grau. SimCity, comentado anteriormente, aplica a emergência no crescimento dinâmico das cidades. Ou jogo, abordado por Steven Johnson em seu livro *Emergência*, é o *Evolva*. Nele, uma criatura inimiga, em um primeiro momento solitária, pode atacar o jogador, se ela estiver morrendo, foge. Ao fugir, pode ser que ela encontre outras criaturas similares, comunique a elas que existe um jogador por perto e esse grupo recém-formado ataca novamente o jogador. Esse tipo de comportamento é muito popular em jogos de ação onde os inimigos controlados por computador devem agir em grupo. Esses grupos geralmente são pequenos, de 8 até poucas dezenas, e geralmente possuem algum tipo de controle superior.

3.3 O Jogo

Ao propositadamente criar uma simulação onde os NPCs são simples e conseguem um comportamento emergente, conseguimos algo inédito, no sentido de explorar

especificamente o potencial da emergência em jogos, contribuindo diretamente para a jogabilidade, além de um grande avanço no que já existe atualmente nos jogos para a plataforma-alvo.

3.3.1 Manual

Sinopse:

Um jogo de estratégia onde se planeja uma batalha e depois se executa uma simulação, visando destruir o castelo inimigo.

Objetivo:

Configurar um exército que vença o exército adversário destruindo seu castelo, em cada um dos 3 estágios, com crescente dificuldade.

Vitória ou Derrota:

O jogador vence uma batalha quando todos os soldados adversários são mortos ou o castelo é destruído.

O jogador perde uma batalha quando todos os seus soldados são mortos.

Ambiente:

O ambiente do jogo será uma planície, sem obstáculos, com os membros dos dois exércitos.

A planície terá 172x110 setores lógicos (cada setor com 8x8 pixels) totalizando 1408x880 pixels.

A planície será desenhada na tela com o uso de tilesets.

Os soldados de cada exército serão de três tipos básicos: Infantaria, Arqueiros e Cavaleiros.

Cada soldado será representado por um ícone animado de sua classe. Uma espada para a Infantaria, um arco-e-flecha para os arqueiros e uma ferradura para os cavaleiros.

Cada soldado terá 8x8 pixels de área.

Além dos soldados, os inimigos terão construções representando um castelo e fontes de recursos. Cada uma dessas construções terá 16x16 pixels.

Soldados:

Atributos:

Não são exibidos ao jogador.

Velocidade: Número de pixels que ele percorre por segundo.

Freqüência de ataque: Intervalo, em milisecs, que o soldado leva entre um ataque e outro.

Poder de Ataque: Valor que define quanto dano ele irá causar sobre quem está sendo atacado.

Defesa/Armadura: Valor que diminuirá a perda de vida, quando for atacado.

Pontos de vida. Tem um valor inicial maior que zero. Ao chegar a zero, o soldado morre.

Distância de ataque. Distância, em pixels, que o ponto de referência do soldado deve estar do o ponto de referência do inimigo para que possa atacar.

Infantaria:

Velocidade: 10

Freqüência de ataque: 1000

Poder de Ataque: 20

Defesa /Armadura: 6

Pontos de vida: 60

Distância de Ataque: 12

Arqueiros:

Velocidade: 10
Frequência de ataque: 1000
Poder de Ataque: 16
Defesa /Armadura: 4
Pontos de vida: 50
Distância de Ataque: 28

Cavaleiros:

Velocidade: 10
Frequência de ataque: 1000
Poder de Ataque: 26
Defesa /Armadura: 10
Pontos de vida: 90
Distância de Ataque: 12

Campo de Visão:

Sempre que o comportamento de um NPC for simulado, ele levará em conta todas as unidades em seu campo de visão, que será de 84 setores. Alcance de 6 setores (48 pixels, em linha reta). Como na figura (Cada número um setor e X o soldado):

```
      6 5 6
     6 5.4. 5.6
    6 5.4. 3.4.5 6
   6 5.4.3. 2.3.4.5 6
  6 5.4.3.2. 1.2.3.4.5 6
65.4.3.2.1.X.1.2.3.4.5 6
  6 5.4.3.2. 1.2.3.4.5 6
   6 5.4.3. 2.3.4.5 6
    6 5.4. 3.4.5 6
     6 5.4.5 6
      6 5 6
       6
```


Movimentação

Um soldado pode se movimentar em qualquer direção, respeitando as áreas ocupadas por outros soldados (o tamanho do ícone) e os limites da tela.

Ao tocar o limite da tela ele terá que mudar de direção ou parar.

A velocidade pode ser qualquer valor entre 0 e o atributo Velocidade que é calculada somando-se as velocidades, em módulo, no eixo X e no eixo Y.

Velocidade = $Abs(velX) + Abs(velY)$

Ataque

Ao avistar inimigos em seu campo de visão, ele irá escolher um deles (IA) e atacá-lo.

Este ataque tem duas etapas:

Aproximação: Irá se mover em direção ao inimigo até que a distância entre eles seja menor ou igual à Distância de Ataque.

Dano: Quando estiver a uma distância suficiente, iniciará os ataques físicos. O dano reduzirá o total de pontos de vida do soldado atacado seguindo a fórmula:

Dano = Ataque – Defesa;

Dano: Dano causado ao soldado atacado.

Ataque: Poder de ataque do soldado atacante.

Defesa: Defesa/ Armadura do soldado atacado.

Quando os pontos de vida do soldado atacado chegarem a zero, ele morre.

Esse dano causado será calculado a cada intervalo de tempo especificado no atributo Freqüência de Ataque.

Ex (Para uma freqüência de ataque de 1000 milisecs).

Instante 23500 = Vê Inimigo

Instante 25000 = Chega próximo o suficiente e ataca

Instante 25500 = Não faz nada, pois ainda não pode atacar de novo.

Instante 26000 = Causa dano novamente.

Um soldado só pode atacar um inimigo de cada vez, mas vários inimigos podem atacá-lo ao mesmo tempo.

Construções:

Castelo

Em cada estágio os inimigos terão um castelo, fortemente protegido. Esse castelo só poderá ser atacado caso não houver nenhum inimigo dentro de seu perímetro (o mesmo do campo de visão dos soldados). O castelo, ao ser cercado por soldados do jogador, sem nenhum soldado vermelho para protegê-lo, será destruído e o jogo termina.



Figura 3.1 Castelo Defendido por Soldados Vermelhos.

Recursos

Em cada estágio os inimigos terão de zero a quatro construções com recursos. Essas construções serão defendidas por unidades inimigas.

Elas só poderão ser conquistada caso não houver nenhum inimigo dentro de seu perímetro (o mesmo do campo de visão dos soldados). A construção, ao ser cercada por soldados do jogador, sem nenhum soldado vermelho para protegê-la, será

destruída e os recursos serão liberados ao jogador. Os recursos serão na forma de unidades adicionais, de tipo e em quantidade pré-determinadas, que serão colocadas no cenário na região da construção destruída.



Figura 3.2 Recursos Sendo Atacados por Soldados Azuis.

Interatividade:

Etapas de Configuração:

Antes de iniciar uma batalha, o jogador deverá configurar seu exército.

Texto.

Através de um pequeno texto, o jogador será informado de como está o exército inimigo. O texto deve conter as informações: Quantidade aproximada de soldados adversários.

Se contém um número particularmente alto de algum tipo de soldado.

A região aproximada onde as unidades estão concentradas.

Exemplo:

“O inimigo tem um exército de aproximadamente 50 homens, apenas arqueiros e infantaria. Pegos de surpresa, eles estão bem separados, com a infantaria no canto superior direito e os arqueiros no canto superior esquerdo”.

Compras

Após exibir o texto, uma tela informará quantos pontos (dinheiro) ele tem para montar seu exército. Cada estágio terá um valor diferente.

Com os pontos ele poderá:

Comprar Soldados, custo unitário: 25.

Comprar Arqueiros, custo unitário: 25.

Comprar Cavaleiros, custo unitário: 50.

Posição

Após decidir como gastar o dinheiro disponível, uma tela com uma miniatura do cenário será mostrada.

A miniatura será uma matriz com onze linhas e dezoito colunas, formando 198 setores.

O jogador poderá selecionar um setor, usando os direcionais (ou 2, 4, 6 e 8).

Ao selecionar um setor e pressionar o botão “7” o jogador setará aquele setor como inicial para um soldado. Cada setor comporta até 10 soldados.

Os soldados serão colocados em cada setor seguindo uma ordem por tipo. Primeiro toda a infantaria, depois todos os arqueiros, por fim, todos os cavaleiros.

Não será possível colocar soldados em todos os setores, apenas os disponíveis em cada estágio. Os setores que não estiverem disponíveis terão estarão preenchidos pela cor preta.

Os setores disponíveis estarão preenchidos pela cor verde. Quando o jogador colocar 10 soldados no mesmo setor, a cor mudará para preto e não será possível colocar mais soldados.

Quando a batalha for iniciada, a posição exata de cada soldado será sorteada dentro do respectivo setor.

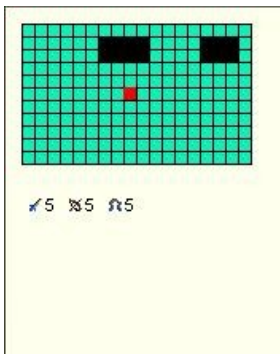


Figura 3.3 Tela de Posicionamento dos Soldados

Etapas de Simulação da Batalha:

Comandos do Jogador:

O jogador poderá, a qualquer momento da batalha, ordenar que seus soldados ataquem determinado setor do mapa, da mesma forma como posiciona as unidades no início da batalha, mas agora será uma matriz de 18 por 11, somando 198 áreas que dividem o mapa total.

Ao apertar o botão “5” o jogo entrará em pausa, e um menu será aberto. Ele poderá decidir se todos os soldados devem ir até a área indicada ou se todos menos os que estiverem atacando alguém.

O jogador seleciona a opção que desejar e pressiona “5” novamente para abrir o mini-mapa onde ele poderá escolher a área, entre as 198 possíveis.

Pressiona novamente o botão “5” e a ordem é confirmada, os soldados de seu exército se dirigirão ao local.

Durante a simulação da batalha, haverá a possibilidade de o mini-mapa ser exibido ou não. O jogador pode pressionar o botão “1” para alternar as opções.

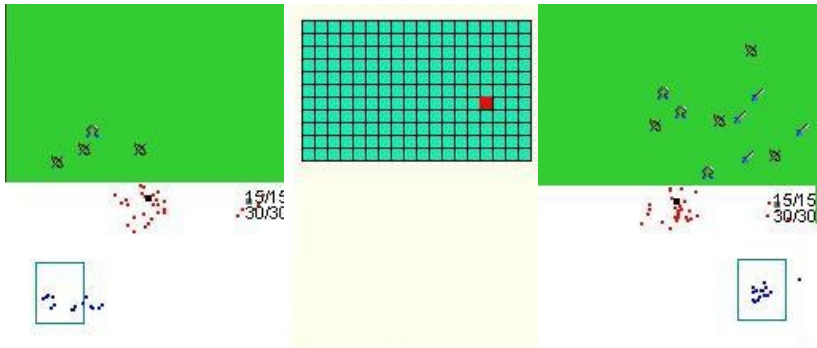


Figura 3.4 Exemplo de Execução de Ordem.

Interface Gráfica:

Na etapa de batalha a tela será ocupada pelo cenário do jogo, com os soldados movimentando-se pela planície e por um campo com informações, localizado na base da tela.

Como a tela não é grande o suficiente para exibir todo o cenário de uma vez (possui até 176 por 220 pixels), exibirá apenas um pedaço. Para conseguir visualizar as outras regiões do mapa, o jogador poderá, ao pressionar as teclas direcionais (além das teclas 2,4,6,8) movimentar a câmera, como nos jogos computador.

No campo inferior as seguintes informações serão expostas ao jogador:

Número de Soldados Vivos/ Total de Soldados.

Número de Soldados Adversários Vivos/ Total de Adversários.

Quando o mini-mapa estiver aberto ele será desenhado por cima do campo de batalha próximo ao fundo da tela, com 100x100 pixels de área. Pontos vermelhos e azuis (2x2 pixel) indicarão a posição dos soldados em toda a planície. Um ponto preto (4x4 pixels) indicará a posição do castelo e pontos cinza (4x4 pixels) indicarão as posições das construções com recursos.

3.4 O Modelo

O modelo parte da premissa que cada agente, sozinho, é simples e incapaz de atitudes complexas. Então ele possui uma capacidade limitada de perceber o ambiente ao redor. A cada etapa de tomada de decisões ele tem disponível apenas sua posição, distância do ninho e fontes de comida e uma lista com os aliados e oponentes dentro de seu campo de visão.

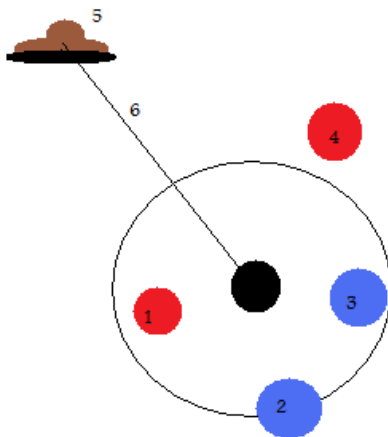


Figura 3.5: Campo de Visão de um Agente

Na figura acima, um agente representado pelo ponto preto tem um campo de visão representado pelo círculo preto, neste caso, a lista de agentes percebidos por ele são os agentes 1, 2 e 3. Ele está totalmente alheio ao agente 4. Quanto ao ninho(5) ele sabe a distância (6).

Flocking. Conceitos da técnica apresentada no capítulo anterior foram necessários para ajudar a modelar os agentes e as soluções para implementar os comportamentos.

FSMs. Máquinas de estados finitos constituem uma técnica largamente utilizada para vários problemas. No jogo o

comportamento individual e tomada de decisões serão implementados com uma FSM. Abaixo segue um esboço da máquina:

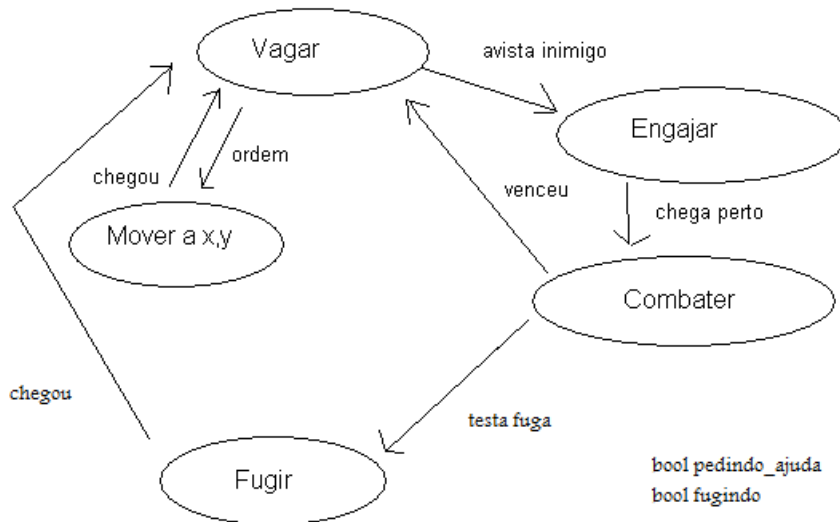


Figura 3.6: FSM de um soldado.

Trocas de mensagens (*messaging*), também chamadas de eventos, são uma solução para a comunicação entre os objetos. Ao invés de um soldado verificar frequentemente por uma mudança de estado de um outro soldado, ele aguarda uma mensagem que indica essa mudança.

3.5 Bases Teóricas

A idéia inicial para este trabalho surgiu durante a leitura de um livro chamado Emergência, de Steven Johnson. Nele, o autor apresenta o conceito de emergência bem como vários exemplos onde ela ocorre. Ele aborda a emergência em jogos

eletrônicos, mas de uma maneira superficial, o suficiente apenas para despertar o interesse nas suas enormes possibilidades. Ao explorar o tema, e aplicar aos jogos, espera-se conseguir um grande avanço nas técnicas de IA aplicadas a jogos.

Um algoritmo muito conhecido, testado e aprovado é o de Craig Reynolds, o Boids. Ao aplicá-lo e estendê-lo, poderemos ampliar seu uso nos jogos.

Técnicas clássicas também fazem parte do embasamento teórico, como FSMs e Messaging, elas podem ser encontradas em detalhes em muitos livros e artigos de jogos, como por exemplo, *AI Techniques for Game Programming*, de Mat Buckland.

3.6 Comportamentos Implementados

Comportamentos inspirados por insetos sociais implementados pelos agentes do jogo.

3.6.1. Consciência da posição do Ninho e da comida.

Os agentes encarregados de defender fontes de recursos e o ninho devem ter consciência de suas localizações e devem voltar aos próprios caso se afastem por motivos diversos.

Alternativa: Os agentes, uma vez que se afastem de suas posições iniciais de defesa, eles se “esquecem” dela e vagam livremente pelo cenário.

3.6.2. Guardas em movimento constante.

Os agentes encarregados de defender fontes de recursos e o ninho não ficam estacionários em suas posições iniciais, como as formigas, mantém movimentos aparentemente aleatórios que circulam ao redor da mesma região.

Alternativa: Os agentes permanecem estacionários até que algum evento os faça entrar em movimento.

3.6.3. Níveis de Agressividade [Knaden, Wehner - 2003].

Condições do ambiente que alteram o nível de agressividade dos agentes com relação a oponentes.

3.6.3.1 Proximidade do ninho. Os agentes próximos do ninho são muito mais agressivos e não fogem de combates. Já os agentes distantes do ninho tendem a ser menos agressivos, e às vezes recuam para o ninho.

3.6.3.2 Proximidade de outros agentes. O agente percebe outros agentes e sabe se está em superioridade ou inferioridade numérica. No primeiro caso, fica mais agressivo e tenta não fugir. No segundo caso as chances de fugir são maiores.

3.6.3.3 Proximidade de comida. Ao se localizar próximo a uma fonte de comida, o agente se porta como um guarda, e se torna mais agressivo contra inimigos.

Alternativas: O nível de agressividade dos agentes é sempre o mesmo.

3.6.4. Estágios da defesa de uma colônia [Wilson - 1976].

3.6.4.1 Defesa em perímetro exterior (fontes de comida). Uma parte dos agentes ataca os invasores. Outra parte percorre as adjacências da luta para avisar outros aliados e uma terceira parte retorna ao ninho para avisar os agentes de lá sobre a luta. Uma parte dos agentes avisados vai até o lugar da luta e ataca os invasores.

Alternativa: Os agentes simplesmente atacam os inimigos que enxergam no campo de visão.

3.6.4.2 Defesa no perímetro interior (ninho). Caso os agentes recuem ou sejam vencidos no perímetro exterior a luta ocorrerá na região do ninho. A maior parte defende as entradas do ninho, uma pequena parte continua buscando mais aliados.

Alternativa: Todos os agentes se concentram no ninho.

3.6.4.3 Fuga. Todos os agentes fogem do ninho, carregando o que conseguirem. Não será implementado, pois não adiciona nada à jogabilidade.

3.6.5. Dança das abelhas [Crist - 2004].

3.6.5.1 Distância e direção. Com a dança, os agentes conseguem informar a distância e a direção de comida/inimigos.

3.6.5.2 Intensidade. Através do ímpeto na dança, os agentes conseguem comunicar se a comida é muito boa ou apenas razoável. No caso de inimigos, se há muitos ou poucos.

Alternativas: Nenhuma forma de comunicação entre os agentes.

3.6.6. Grupos de abelhas [Nieh et al - 2004]. Diferenças nas estratégias das abelhas quando lutando em grupos.

3.6.6.1 Um contra um. Ao lutarem dessa maneira, os agentes adotam uma postura mais defensiva enquanto aguardam outros aliados virem para ajudar.

Alternativa: Não implementar um modo “defensivo”, apenas um modo padrão de combate.

3.6.6.2 Dois contra um. Ao atacar um inimigo junto com mais um companheiro, os agentes se tornam mais agressivos e costumam atacar até aleijar ou matar o inimigo. É também tamanho de grupo de combate mais comuns para algumas espécies.

Alternativa: Não implementar um modo “agressivo”, apenas um modo padrão de combate. Não implementar a preferência por grupos de combate de dois contra um.

3.6.6.3 $N > 2$ contra um. Como no comportamento anterior, mas ocorre mais raramente por causa das limitações espaciais.

3.6.7 Relações entre formigas e afídios [Phillips, Willis -2005]. Ao serem surpreendidas por predadores de afídios e formigas inimigas ao mesmo tempo, as formigas atacam as inimigas em 90% dos casos. Mesmo deixando que os predadores devorem vários de seus afídios (fontes de comida), ela preferem atacar as inimigas, como se provando

que possuem boas defesas, para desencorajar futuros ataques.

3.6.7.1 Os agentes focam mais em adotar uma postura agressiva quando percebem inimigos do que em defender os recursos.

Alternativa: Os agentes não fazem distinção entre defender os recursos e atacar os inimigos.

4 Implementação

Cada comportamento descrito anteriormente será discutido acerca da implementação e conclusões sobre o acréscimo do comportamento para a jogabilidade. Para ajudar na conclusão, algumas métricas foram definidas:

4.1 Métricas

4.1.1. Dificuldade Apropriada.

Medida verificando a quantidade de soldados vivos no exército que venceu a batalha. Valor ideal: Apenas um soldado vivo.

4.1.2. Diversidade de comportamentos.

Medida avaliando variação da reação dos adversários ao se executar pequenas modificações na estratégia de ataque. Quanto maior a diversidade, melhor.

4.1.3. Agressividade.

Os inimigos não devem ficar estacionários quando próximos de soldados do jogador. Devem se mover com intenção de ataque ou de recuo. Quanto maior e mais rápida a movimentação, melhor.

4.1.4. Movimentação

A movimentação dos agentes não deve ser “errática”, com alterações bruscas e aparentemente incoerentes de direção. Os agentes devem se movimentar em harmonia com seus soldados do mesmo exército e evitar colisões, antecipadamente, além de terem um senso de propósito em cada movimento.

4.2 Implementação e Conclusões

4.2.1 Consciência da posição do Ninho e da comida.

Implementação:

Os agentes vermelhos mantêm uma variável armazenada que indica a posição na qual começaram a simulação. A cada intervalo de um segundo, eles fazem um teste comparando suas posições atuais com as iniciais, se estiverem fora do limite pré-estabelecido eles somam uma unidade a um contador. Quando este contador chega a 5, eles retornam à posição inicial.

Pos_Atual;

Pos_Inicial;

Contador;

Dist_Máx

A cada 1000 milissegundos faça:

Se ($\text{Dist}(\text{Pos_Atual}, \text{Pos_Inicial}) > \text{Dist_Máx}$) Então

Contador= Contador+1;

Se (Contador == 5) Então

RetornarPosInicial();

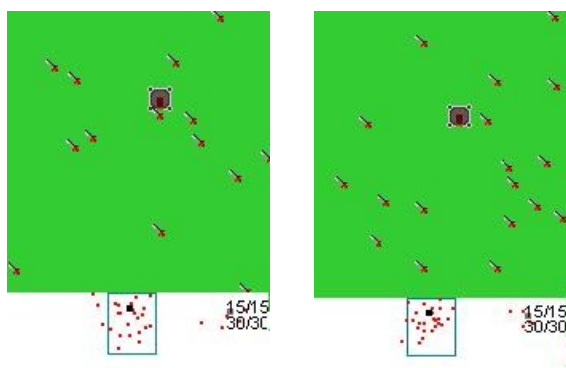


Figura 4.1 : Soldados se movem, mas não se afastam do ninho.

Conclusões:

Quando o jogo inicia, os soldados inimigos têm suas posições pré-definidas, elas são pensadas para que a defesa das fontes de comida e do ninho sejam eficientes e bem feitas. Vários eventos podem fazer com que o soldado deixe sua posição inicial, como enxergar um inimigo, por exemplo. Caso um, ou vários desses soldados deixem suas posições, a defesa ficará muito enfraquecida e eles perderão boa parte de sua utilidade, caso estejam em posições afastadas.

Baseado nas métricas previamente definidas (métrica 4, movimentação), este comportamento é, sem dúvida, bem-vindo.

4.2.2 Guardas em movimento constante.

Implementação:

Apenas inicializar os soldados vermelhos com uma velocidade aleatória para qualquer direção, o comportamento anterior fará com que se mantenham na área adequada.

Conclusões:

Quando os agentes defensores permanecem estacionários desde o começo da simulação, a defesa fica muito comprometida, eles ficam com o campo de visão muito restrito, e raramente percebem combates acontecendo ao redor deles. Caso estejam em movimento, a chance de perceberem um colega em combate ou um invasor aumenta drasticamente.

Baseado nas métricas definidas (em especial a agressividade) a movimentação enquanto se defende uma região é bem vinda e aumenta muito a satisfação do jogador.

4.2.3. Níveis de Agressividade [Knaden, Wehner - 2003].

Implementação:

A cada segundo um teste é feito em cada soldado vermelho. O teste leva em consideração a proximidade de comida e do ninho e a diferença numérica entre aliados e oponentes. Se ele está próximo do ninho ou em superioridade numérica ele nunca fugirá. Se estiver em igual número ou em inferioridade, as chances de fugir serão tão maiores quanto a grandeza da inferioridade. Se estiver perto de comida, terá uma chance pequena de fugir, se não estiver perto, terá uma chance maior, somada à inferioridade.

```
Vetor_Aliados; //aliados no campo de visão  
Vetor_Oponentes; //opponentes no campo de visão  
Pos_Atual;
```

```
Diferença_Numerica:  
    tamanho(Vetor_Aliados) + 1 – tamanho  
(Vetor_Oponentes);
```

```
Proximidade_Ninho:  
    Se ( dist(Pos_Atual, Pos_Ninho) < Dist_Prox_Ninho)  
Então:  
    return true;  
Senão  
    return false;
```

```
Proximidade_Comida:  
    resposta = false;  
    Para cada comida do cenário faça:  
        Se ( dist(Pos_Atual, Pos_Ninho) <  
Dist_Prox_Ninho) Então:
```

```

        return true;
    fim para-faça
return resposta;

//a cada 1000 milissegundos
Testar_Fuga:
    Se ( Proximidade_Ninho() ou Diferenca_Numerica() > 0)
Então: // Não foge
    Senão
        Chance_fuga = 0;
        Se (Proximidade_Comida()) Então
            Chance_fuga += Chance_fuga + 5;
        Senão:
            Chance_fuga += Chance_fuga + 10;
        Se (Diferenca_Numerica() == 0) Então:
            Chance_fuga += Chance_fuga + 5;
        Senão Se (Diferenca_Numerica() == -1 ) Então:
            Chance_fuga += Chance_fuga + 10;
        Senão
            // diferenca menor que -1
            Chance_fuga += Chance_fuga + 15;
        Se ( RAND(100) < Chance_Fuga) Então
            Fugir();
Fim;

```

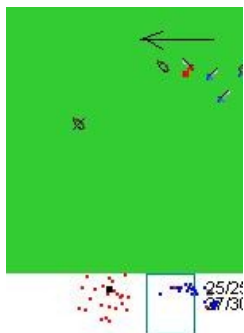


Figura 4.2 : Soldado Vermelho em fuga, em direção ao ninho.



Figura 4.3 : Soldados vermelhos mortos, quando não fogem.

Conclusões:

O efeito da fuga nos soldados é excelente, a fuga é visivelmente percebível e os padrões de fuga são entendidos como instintivos pelo jogador. Exemplo: Quando um grupo de inimigos longe de construções relevantes é atacado por soldados azuis em maioria, aproximadamente a metade deles deve fugir para o ninho.

Baseado em várias das métricas definidas (como a diversidade e a agressividade), este é um comportamento que adiciona muito à jogabilidade, aumentando a diversidade de comportamentos e incrementando a agressividade, no sentido de torná-la mais complexa e mais racional.

4.2.4. Estágios da defesa de uma colônia [Wilson - 1976].

Implementação:

Quando um agente vermelho percebe que um aliado está no estado Engajando (significando que aquele agente viu um inimigo e corre em direção a ele) existe uma probabilidade de aquele agente correr na mesma direção que o agente engajando, ou seja, buscando o combate também, ou que este corra em uma direção aleatória, avisando a todos os

aliados que ver no caminho que existe um combate acontecendo na região onde ele viu o soldado Engajando se dirigindo. Esses agentes alertados correrão para a área comunicada pelo agente. A terceira parte, que deveria voltar ao ninho buscando reforços apenas volta ao ninho, em um comportamento de fuga, mas não busca reforços.

```
Vetor_Aliados;  
Vetor_Oponentes;  
Buscar_Aliados_Engajando:  
    Para cada aliado em Vetor_Aliados faça:  
        Se (aliado.estado == Engajando) Então:  
            Se (RAND(100) < 50) Então  
                // ajudar no ataque  
            Senão  
                // correr para direção aleatória pedindo ajuda  
        Fim  
    Fim  
Fim  
  
Buscar_Aliados_Ajuda:  
    Para cada aliado em Vetor_Aliados faça:  
        Se (aliado.estado == Pedindo_Ajuda) Então:  
            // ajudar no ataque  
        Fim  
Fim
```

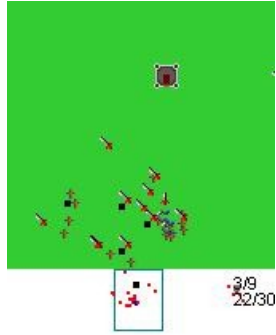


Figura 4.4 : Soldados vermelhos mobilizados para a defesa.

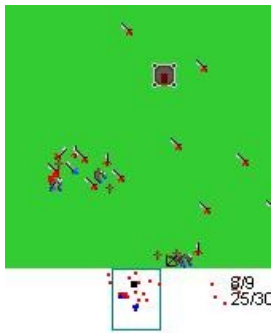


Figura 4.5 : Soldados vermelhos mais afastados alheios ao ataque.

Conclusões:

Com este comportamento a defesa do ninho e comida fica muito mais organizada e coerente. Os agentes conseguem mobilizar uma força muito maior para combater os inimigos e as chances de vitória aumentam enormemente. Quando esse comportamento não existe, muitas vezes agentes próximos do combate continuam alheios à invasão e nada fazem para ajudar os companheiros.

Com relação às métricas, essa capacidade de mobilização aumenta a satisfação do jogador ao ver os inimigos mais inteligentes e agressivos.

4.3.5. Dança das abelhas [Crist - 2004].

Implementação:

Este comportamento foi integrado ao comportamento anterior. Quando o agente vê um aliado engajando e decide correr para pedir ajuda, ele calcula a posição aproximada que o aliado estava correndo e guarda na memória. Quando ele encontra algum agente que não sabe do combate, a informação é comunicada e o agente se dirige para lá. O número de inimigos atacando não é guardado, pois não importando o número, o agente se dirigirá para o local.

Calculando_Posição_Aproximada:

$$\text{Posição_Aproximada} = \text{Posição_Atual_Soldado_Engajando} + \text{Velocidade} * \text{Um_Segundo}$$

Conclusões:

A informação da localização aproximada do combate é imprescindível para que um soldado pedindo ajuda tenha seu pedido atendido com êxito. Caso esse comportamento seja desligado, os agentes saberão que há um combate mas não saberão para onde se dirigir.

4.3.6. Grupos de abelhas [Nieh et al – 2004].

Implementação:

Quando um agente está atacando alguém, ele verifica se é o único aliado a atacar aquele alvo, se for, liga o modo defensivo. Se não for o único a atacar o alvo, desliga o modo defensivo. No modo defensivo, todo dano causado por e ao agente é dividido por dois.

Para a segunda parte do comportamento, a preferência por grupos de dois contra um, sempre que um agente está procurando um alvo ele verifica no seu campo de visão por um alvo que esteja sendo atacado por apenas um aliado, se encontrar, ele ataca aquele alvo, senão, atava o alvo mais próximo.

Modo_Defensivo:

```
Se ( atacando_alvo == 1) Então
    Modo_Defensivo = true;
Senão // atacando_alvo > 1
    Modo_Defensivo = false;
```

Calcular_Dano:

```
int Dano;
Se (Modo_Defensivo) Então
    Dano = Dano/2;
```

Conclusões:

O modo defensivo ajuda o grupo a ter uma menor taxa de mortalidade em uma batalha (-7% em média), mas apenas quando um único grupo a utiliza, se tivéssemos esse comportamento nos dois grupos, ele estaria sempre ligado, e nem um nem outro se beneficiaria. Então, do ponto de vista da criação de jogos, este comportamento não é utilizável caso desejemos implementar nos dois grupos.

A preferência por combates dois contra um apresenta um grande acréscimo ao desempenho dos soldados no jogo. Com esse comportamento desligado, os soldados tendem a atacar todos o mesmo inimigo, o mais próximo, e acabam deixando de lado outros inimigos, ou amigos em desvantagem. A adição desse comportamento representa

uma melhor distribuição dos soldados pelo ambiente e um aspecto gráfico mais agradável para o jogador.

4.3.7 Relações entre formigas e afídios [Phillips, Willis -2005].

Implementação:

Ao verem um inimigo, os soldados vermelhos partem para o ataque sem pensar na comida ou no ninho que estavam protegendo. Com isso eles podem se afastar muito.

Conclusões:

Salvo raras exceções nas quais o ninho e recursos ficam desprotegidos, atacar vigorosamente qualquer inimigo que se aproxime vai de acordo com as métricas definidas, tornando o jogo mais divertido e interessante.

Conclusão

Com o estudo realizado para este trabalho, podemos notar como é vasto e passível de aprimoramento o campo da inteligência artificial, principalmente quando aplicada a jogos. Como os jogos têm muitos requisitos não quantitativos, isso dificulta o processo de avaliação de uma boa IA para um jogo, o processo de desenvolver essa IA se torna também uma arte.

A Emergência surge como opção para os desenvolvedores de jogos, pois a partir de elementos simples consegue resultados complexos, ao passo o extremo oposto seria centralizar uma IA em um processo só, o que pode trazer inúmeras complicações, como a dificuldade na manutenção.

Como discutido no capítulo anterior, diversos comportamentos biológicos foram implementados em agentes simples, com percepções limitadas do ambiente simulado. O objetivo dos agentes é defender fontes de recursos e o ninho do jogador, que ataca-os com outro exército. Com a criação desses agentes cooperativos, a IA do jogo ficou desafiadora e divertida, como era o objetivo.

Comportamentos inesperados foram encontrados ao longo do desenvolvimento, como o fato dos agentes, quando se dirigindo em grandes quantidades para o mesmo ponto, trabalhando com um sistema de colisão onde o agente mais a frente segue adiante e o mais atrás recua um pouco, criam uma fila indiana que percorre distâncias sem choques e ainda permitem facilmente a entrada de novos agentes no meio da fila.

Outro comportamento que emergiu dos agentes, quando defendendo um ninho, foi uma concentração maior na região de invasores inimigos, mas também a existência de agentes por todo o perímetro do ninho, defendendo contra inimigos ainda não avistados.

O emprego da emergência em jogos parece muito promissor, e pode inspirar mais estudo e trabalhos. Na parte de trabalhos futuros, poderia haver a inserção de mais comportamentos biológicos, não só de insetos, mas de outros animais sociais. Outra opção seria testar os agentes em outros estilos de jogos, como jogos de ação ou RPG.

Referências

REYNOLDS, Craig. **Big Fast Crowds on PS3**. Disponível em: < www.research.scea.com/pscrowd >. 2006.

YANNAKAKIS, Georgios N.. **AI in Computer Games: Generating Interesting Interactive Opponents by the use of Evolutionary Computation**. 2005. 235 f. Tese (Doutorado) - Universidade de Edinburgh, Edinburgh, 2005.

PISAN, Yusuf. **Artificial Intelligence versus Clever Design for Creating Intelligent Game Characters**. Disponível em: < <http://www-staff.it.uts.edu.au/~ypisan/research/publications/pisan-jcis06.pdf> >. 2006.

HUSSAIN, Talib S. **Flexible and Purposeful NPC Behaviors using Real-Time Genetic Control**. 2006.

ORKIN, Jeff. **Symbolic Representation of Game World State: Toward Real-Time Planning in Games**. 2003.

KHOO, Aaron; DUNHAM, Greg; TRIENENS, Nick; SOOD, Sanjay. **Efficient, Realistic NPC Control Systems using Behavior-Based Techniques**. Disponível em: <AAAI.org>. 2002.

RAYNOLDS, Craig W. **Steering Behaviors for Autonomous Characters**. 1999.

CORNING, Peter A. (2002), **The Re-Emergence of “Emergence”: A Venerable Concept in Search of a Theory**, *Complexity* 7(6): 18-30 .

RABIN, Steve (2002), **AI Game Programming Wisdom**, Charles River Media, ISBN: 1584500778.

RUSSEL, Stuart J.; NORVIG, Peter (1995), **Artificial Intelligence A Modern Approach**, Prentice-Hall, ISBN 0-13-103805-2.

SEEMAN, Glenn; BOURG, David M. (2004), **AI For Game Developers**, O'Reilly, ISBN: 0-596-00555-5.

SPECTOR, L.; KLEIN, J.; PERRY, C. and M. Feinstein. 2003. **Emergence of Collective Behavior in Evolving Populations of Flying Agents**. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003), pp. 61–73. Berlin: Springer-Verlag.

*FROMM, Jochen (2004), **The Emergence of Complexity**, kassel university press GmbH, ISBN 3-89958-069-9.*

MCDERMOTT, Josh. **The Emergence of Orientation Selectivity in Self-Organizing Neural Networks**. The Harvard Brain, páginas 43 a 51, primavera de 1996.

CHANNON, Alastair. **The Evolutionary Emergence route to Artificial Intelligence**. 1996.

BUCKLAND, Mat (2005), **Programming Game AI by Example**, Wordware Publishing, ISBN 1-55622-078-2

BUCKLAND, Mat (2002), **AI Techniques for Game Programming**, Premier Press, ISBN: 1-931841-08-X.

ADAMS, Jim (2002), **Programming Role-Playing Games with Directx 8.0**, Premier Press, ISBN 1-931841-09-8.

SCHWAB, Brian (2004), **AI Game Engine Programming**, Charles River Media, ISBN 1-58450-344-0.

JOHNSON, Steven Berlin (2001), **Emergence: The Connected Lives of Ants, Brains, Cities, and Software**, Scribner's, ISBN 0-684-86876-8.

GOLDSTEIN, Jeffrey (1999), **Emergence as a Construct: History and Issues**, Emergence: Complexity and Organization 1: 49-72.

NITSCHE, Geoff. **Designing emergent cooperation: a pursuit-evasion game case study**. Disponível em:

<<http://www.springerling.com/content/f618815r53x3m25m/>>.

Acesso em: 26 nov. 2006.

EMERGENCE. 2007. Disponível em:

<http://en.wikipedia.org/wiki/Emergence>. Acesso em: 15 jan. 2007.

BLOBS. 2007. Disponível em:

<http://obermuhler.com/public/Projects/Applets/Blobs/index.html>.

Acesso em: 22 jun. 2007.

BOIDS. 2007. Disponível em: <http://en.wikipedia.org/wiki/Boids>.

Acesso em: 15 jan. 2007.

FLOCKING (BEHAVIOR). 2007. Disponível em:

http://en.wikipedia.org/wiki/Flocking_%28behavior%29. Acesso em: 15 jan. 2007.