UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ESTENDENDO O LOGTELL: SUPORTE A AÇÕES ENTRE PERSONAGENS FIGURANTES

TRABALHO DE GRADUAÇÃO

Afonso Rodrigo de Figueiredo Martins Filho

Santa Maria, RS, Brasil
2007

ESTENDENDO O LOGTELL: SUPORTE A AÇÕES ENTRE PERSONAGENS FIGURANTES

por

Afonso Rodrigo de Figueiredo Martins Filho

Trabalho apresentado ao Curso de Ciência da Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

Orientador: Dr. Cesar Tadeu Pozzer

Trabalho de Graduação n. 215

Santa Maria, RS, Brasil

2007

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada, aprova o Trabalho de Graduação

ESTENDENDO O LOGTELL: SUPORTE A AÇÕES ENTRE PERSONAGENS FIGURANTES

elaborado por
Afonso Rodrigo de Figueiredo Martins Filho

como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Comissão Examinadora

_____
Cesar Tadeu Pozzer, Dr.
(Presidente/Orientador)

_____
Benhur de Oliveira Stein, Dr.

_____
Marcos Cordeiro d'Ornellas, PhD.

Santa Maria, 05 de Março de 2007.

Dedicatória

Dedico este trabalho à memória do grande amigo Rodrigo Copês. Uma carreira brilhante, um futuro brilhante interrompidos a vinte dias de sua formatura. Jamais será esquecido.

Agradecimentos

Agradeço a Universidade Federal de Santa Maria, ao Centro de Tecnologia e ao Curso de Ciência da Computação. Agradeço aos professores que tive até hoje, começando por meus pais. Agradeço ao Prof. Cesar Pozzer pela participação importante nesses últimos semestres da minha graduação.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

Estendendo o LOGTELL: Suporte a ações entre personagens figurantes

AUTOR: AFONSO RODRIGO DE FIGUEIREDO MARTINS FILHO
ORIENTADOR: PROF. DR. CESAR TADEU POZZER
Data e Local da Defesa: Santa Maria, 05 de Março de 2007.

Ao longo deste documento são discutidos, estudados e aplicados métodos para criação de batalhas consistentes entre personagens figurantes em um ambiente de narração de histórias interativas, o Logtell. Foram feitas pesquisas em diversas áreas como agentes autônomos, inteligência artificial para jogos e computação gráfica. Para realizar a simulação de batalhas foi necessário compreender profundamente e fazer modificações no Logtell. A possibilidade de execução de ações entre personagens figurantes foi desenvolvida para tornar possível a batalha entre personagens figurantes. Foram incluídos esquemas de movimentação alternativos combinando o comportamento de direcionamento *seek* com o comportamento de grupo *separation.* Também é abordada a utilização do método *nonPenetrationConstraint*, buscando tornar a movimentação mais realista. São criados mecanismos para permitir que a câmera virtual, que capta as cenas da história, consiga acompanhar os vários personagens envolvidos nas batalhas.

Palavras-chave: *Interactive Storytelling*; Logtell; Inteligência artificial; *steering behaviors,*câmera virtual.

LISTA DE TABELAS

LISTA DE FIGURAS

SUMÁRIO

Capítulo 1 - Introdução

Jogos 3D, animação gráfica de alto realismo, filmes, consoles (vi*deo games*) cada vez mais poderosos e avanços no *hardware* relacionado são apenas algumas facetas de uma indústria gigantesca, a do entretenimento. Para alimentar a crescente demanda por entretenimento, a cada dia surgem novas tecnologias e novos paradigmas de entretenimento valorizando cada vez mais a interatividade com o usuário.

A maneira como o usuário interage com a forma de entretenimento em questão varia, podendo ser uma exposição de conteúdo como nos filmes, ou uma exploração de conteúdo no caso dos jogos. Independente da forma de entretenimento um fator é comum, a necessidade de despertar e manter o interesse do usuário ou expectador. Um filme ou um jogo só são bem sucedidos se cumprirem essa premissa básica, para tanto precisam constantemente de inovações.

Nesse contexto surge a *Interactive Storytelling* (Narração Interativa de Histórias) que é uma alternativa aos meios tradicionais de entretenimento, puramente lineares, como livros e filmes. A *Interactive Storytelling* engloba geração, interação e visualização de histórias por meio do computador (GLASSNER, 2004). O diferencial está na maneira com que se faz a interação com o usuário que pode assumir uma postura mais ativa em relação à história dramatizada.

Diversos sistemas disponíveis na literatura como os propostos por Cavazza (2002), Mateas (2002) e Spierling (2002), que tratem com histórias interativas e as dramatizem por meio de recursos gráficos, fazem uso de personagens animados, cenários, enredos, entre outros elementos que criam o universo da história. O diferencial entre eles está na forma como as histórias são geradas, como são dramatizadas e como ocorre a interação com o usuário.

A geração de enredos interessantes de forma dinâmica e "automática" é sem dúvida a etapa mais difícil do processo, pois envolve processos cognitivos que são complexos até mesmo para seres humanos. Para a visualização, geralmente faz-se uso de *engines* utilizados em jogos, visto que personagens, ações e cenários têm muito em comum em ambas as aplicações.

Neste trabalho faz-se uso de duas ferramentas ligadas à *Interactive Storytelling*: o IPG e o Logtell. O IPG (*Interactive Plot Genrator*) (CIARLINI, 1999) é usado como ferramenta

para geração das histórias. Ele foi desenvolvido sob a teoria do pesquisador russo Wladimir Propp que, mediante observação de contos de fadas russos, observou que é muito comum a ocorrência de eventos típicos e de padrões de encadeamento entre os eventos. Propp (1968) sugeriu a caracterização dos textos de um determinado gênero pela associação de funções a pequenos trechos das narrativas. O IPG usa um método formal baseado em operações lógicas para a especificação dessas funções. O IPG, fazendo uso de inferência de objetivos e planejamento, gera seqüências de eventos que definem o rumo da historia. Deve-se observar que o IPG não gera uma história a partir do nada. O autor da história deve especificar o conjunto de objetivos a serem perseguidos pelos personagens, um conjunto de funções que descrevem eventos e descrição dos personagens e seus atributos (ANEXO A e ANEXO B).

Pozzer (2005) propôs um sistema de geração, interação e visualização de histórias interativas chamado Logtell. O Logtell faz uso do IPG como ferramenta de geração das histórias. Ele também incorpora uma interface amigável por onde o usuário interage com a geração da história e sua dramatização.

O IPG pode facilmente trabalhar transparentemente com qualquer contexto de história, que pode ir desde contos de fadas às simulações empresariais. Já o Logtell, por incorporar um módulo de visualização gráfico 3D das histórias, limita-se à fazer a dramatização de cenas de ação, visto que representações gráficas deste tipo de cena são mais simples do que representação de cenas que envolvam aspectos mais sutis, como diálogos ou cenas de cunho emotivo. Neste contexto, contos de fadas, com suas ações características (funções Propianas) que envolvem raptos, lutas, libertação e casamentos são melhores e mais facilmente representadas somente com o uso de recursos gráficos.

Foi criada em Pozzer (2005) uma história exemplo, fundamentada no trabalho de Propp. A história exemplo pode ser sintetizada no seguinte enunciado:

"Um vilão, no caso um dragão, rapta uma princesa desprotegida. O rei, em desespero, requisita ajuda a um herói que parte em busca da princesa, mata o dragão e, como recompensa, casa-se com ela."

Os personagens principais dessa história são Marian (a princesa), Draco (o dragão) Brian e Hoel (os heróis). Como figurantes aparecem os guardas do castelo da princesa que são atacados pelo dragão (Figura 1.1-a), os guardas dos castelos dos heróis e os guardas do castelo do dragão que são atacados pelo(s) herói(s) (Figura 1.1-b).

Durante a dramatização ocorrem diferentes ações como as demonstradas na Figura 1.1. Ocorrem também outras ações como: o seqüestro da princesa, o fortalecimento do herói após perceber que é mais fraco que o vilão, a luta final entre o(s) herói(s) e o vilão, a derrota do vilão, a libertação da princesa e o casamento da princesa que ao conhecer o seu libertador se apaixona por ele.

Figura 1.1 – a) Dragão ataca guardas do castelo da princesa e b) herói ataca guardas do castelo do dragão, respectivamente (POZZER, 2005, p.124).

Um problema freqüente em sistemas como o Logtell é o curto tempo de dramatização das histórias. Este problema provém da dificuldade de criação de histórias com enredo complexo para narração interativa. Portanto, buscando enriquecer a curta dramatização do enredo da história exemplo, são propostos mecanismos para que no Logtell exista a possibilidade de criação de batalhas entre personagens figurantes.

A batalha se dá entre os personagens figurantes que compõe os exércitos que defendem o castelo da princesa e aqueles que defendem o castelo do dragão. Dessa maneira, ao invés de o próprio dragão lutar contra os soldados do castelo da princesa, são enviados seus soldados para atacar o local[1].

[1] Na história exemplo é essencial que o dragão ataque o castelo da princesa para que o castelo fique desprotegido o suficiente para que ocorra o seqüestro da princesa.

De uma batalha são inferidas duas premissas:

- É uma ação coletiva[2], envolve vários personagens, no caso figurantes;
- Envolve movimentação de grupos de personagens.

Destas premissas emergem vários problemas que precisam ser tratados:

- No Logtell, originalmente, apenas os personagens principais entram em confronto entre si ou com os figurantes, o que impede a execução de grandes batalhas entre exércitos (figurantes);
- Efetuar o controle da movimentação dos vários personagens envolvidos na batalha é uma tarefa difícil;
- Vários personagens interagindo em uma ação coletiva ocasionam colisões e sobreposicionamento;
- No Logtell, originalmente, ações coletivas não são previstas;
- É necessária a criação de um gerenciador de batalhas.
- São necessários novos recursos de câmera virtual para capturar as novas ações coletivas.

É estudada a possibilidade da utilização de s*teering behaviors* (comportamentos de direcionamento) ou métodos alternativos para garantir o *zero overlap* (sobreposicionamento zero) dos personagens. Comportamentos de direcionamento são métodos aplicados para calcular as trajetórias desejadas para satisfazer os objetivos, por exemplo, ir do ponto A até o ponto B do cenário.

Os comportamentos de direcionamento geram uma força de direcionamento que descreve em que direção e em que velocidade o personagem deve se mover para chegar ao destino. Permitindo assim, melhorar e tornar mais realista a maneira como os personagens se locomovem, através da utilização de atributos como força, massa e aceleração.

Essas técnicas contribuem para a solução de outro problema que não fica evidente devido à inexistência de ações coletivas no sistema original. A movimentação de um menor

[2] Entende-se por ação coletiva uma ação delegada a mais de um personagem sendo que, o resultado dessa ação é um só oriundo dos resultados das ações individuais de cada personagem envolvido na ação.

número de personagens torna menos perceptível as colisões e o sobreposicionamento entre os personagens. Esses problemas podem ser vistos na Figura 1.2 em duas diferentes situações.

Figura 1.2 – a) Dragão passando por guardas do castelo do herói e b) Dragão passando por guardas do seu castelo, respectivamente.

O capítulo seguinte provê detalhes sobre *Interactive storytelling*, discorrendo sobre as diferentes abordagens que podem ser adotadas. Também é feita uma descrição das diferentes fases do processo de geração da história. No Capítulo 3 os comportamentos de direcionamento têm seu funcionamento demonstrado. Também são abordados comportamentos de grupo, em especial o *separation* que é utilizado em conjunto com o comportamento de direcionamento *seek* no mecanismo de movimentação alternativo implementado. No Capítulo 4 é apresentado o Logtell, sua estrutura interna, funcionamento e particularidades. Neste capítulo o módulo de Visualização Gráfica é descrito minuciosamente a fim de expor onde são feitas as modificações propostas neste trabalho para enriquecer o Logtell. No quinto capítulo são descritas as modificações efetuadas no Logtell, suas implicações, bem como os problemas enfrentados durante o trabalho. No sexto e último capítulo são apresentadas as conclusões e sugestão de trabalhos futuros.

Capítulo 2 - Interactive Storytelling

Interactive Storytelling (narração interativa de histórias) é um paradigma recente de entretenimento que engloba geração, interação e visualização de histórias (GLASSNER, 2004). Para tanto utiliza conhecimentos de diversas áreas como comunicação entre agentes autônomos (CUNHA, 2001), Inteligência Artificial e Computação Gráfica.

Os personagens em Interactive Storytelling podem ser agentes reativos ou cognitivos (RUSSEL, 1995), dependendo do contexto onde estão inseridos. Agentes cognitivos têm a capacidade de raciocinarem sob o estado atual do mundo para tomar decisões. Agentes reativos têm como característica somente reagirem a estímulos externos. Esta categoria de agentes é utilizada no módulo gráfico do Logtell, visto que os personagens são meros atores que devem interpretar seqüências de ações definidas pelo IPG. Para a implementação de agentes reativos geralmente faz-se uso de máquinas finitas de estados (*finite state machines)*, o padrão de software mais utilizado na implementação de agentes reativos (RABIN, 2002).

A geração da história é a parte responsável por criar a estrutura básica da narrativa determinando quais personagens, ações e relacionamentos existem no contexto criado. Segundo Grasbon (2001), o processo de criação da história é um ciclo iterativo de definições e testes, executado pelo autor.

Quanto à interação e direcionamento da história existem duas abordagens principais: uma orientada a enredos *(plot based*) e outra orientada a personagens (*character based*). Estas abordagens são vistas em (GRASBON, 2001) e (CAVAZZA, 2002), respectivamente.

Em histórias orientadas a enredos o nível de interação é mais restrito e a história é pontual, geralmente representada através de uma sucessão de eventos em alto nível. A história pode ser representada como um grafo orientado onde os vértices são as ações a serem cumpridas e as arestas os diferentes caminhos possíveis para a realização da ação. Mateas (2003) constatou que algumas arestas, ao serem percorridas, definem o formato da narrativa.

No paradigma das histórias orientadas a enredo o usuário interage somente com a história criada, sem poder alterar os aspectos mais globais da mesma, ficando a cargo somente do autor a definição desses aspectos, tornando somente o "*telling*" (contar) interativo. Neste caso a variação entre várias visualizações ocorre no caminho entre os vértices que representam os eventos que compõem os aspectos mais globais da história.

Na abordagem orientada a personagens o enredo é gerado dinamicamente com base nas interações entre os agentes autônomos do ambiente. Segundo Cavazza (2002), nessa

abordagem geração e interação são unificadas. Dessa maneira em histórias orientadas a personagens a geração das histórias pode ser considerada interativa.

A autonomia e capacidade de interação entre agentes são utilizadas para buscar soluções e gerar o enredo em abordagens orientadas a personagens, porém em abordagens orientadas a enredo essa autonomia se não controlada, pode ser considerada um problema, pois pode levar o agente a fugir do enredo previsto.

Em ambas as abordagens a inteligência artificial dos agentes (D'AMICO, 1995), que compõe a lógica dos personagens, é importante. A tecnologia atual não permite garantir que as atividades exercidas sejam coerentes abrindo caminho para muitas pesquisas nesta área juntamente com áreas relacionadas com a cognição, modo de pensar e agir dos seres humanos (TORRES, 2002).

Cavazza (2002), em seus experimentos em narração interativa orientada a personagens lança mão de recursos para que, dado o enredo, as ações sejam coerentes. O controle do comportamento dos personagens é descrito com base em regras que associam ações a objetivos correspondentes. Um conjunto ordenado de regras constitui um plano. Para que existam diversos planos diferentes, mas coerentes do ponto de vista do usuário vários fatores contribuem para tornar a ordem das ações imprevisível, como:

- A distribuição inicial dos personagens, que tem forte influência na determinação da duração de cada ação, o que conseqüentemente conduz a situações diferentes;
- Em sistemas orientados a personagens as regras de vários personagens podem ser dinamicamente combinadas, ou seja, pode haver interações entre os planos dos personagens;
- Resultados randômicos gerados por ações terminais[3];
- Estado emotivo do personagem (*mood)*;
- Interação do usuário.

A representação gráfica (geralmente 3D) é a última fase do processo, que é realizada em tempo real à medida que os motores de enredo fornecem as informações relativas à cena a ser dramatizada a cada momento. É nessa parte onde é adicionada a percepção temporal e espacial aos eventos da história.

[3]Ações terminais são ações genéricas que podem gerar outras ações intermediárias até a sua finalização. Por exemplo, uma ação lutar implica em aproximar-se do adversário (ação intermediária) antes que a luta ocorra.

A Figura 2.1 mostra imagens do sistema proposto por Cavazza (2002) e a Figura 2.2 mostra imagens do sistema proposto por Mateas (2002), ambos orientados a personagens. O sistema proposto por Cavazza (2002) utiliza o motor gráfico 3D do jogo *Unreal* (*Unreal Technology*), a interação ocorre por meio de elementos da cena. No sistema proposto por Mateas (2002), com visualização gráfica 3D em primeira pessoa, a interação ocorre por meio de diálogos. O usuário digita a mensagem mostrada na tela, a mensagem é analisada pelo sistema que reage conforme necessário, inclusive respondendo à perguntas.

Figura 2.1 – Exemplos de cenas de diferentes histórias (CAVAZZA, 2002, p. 20 e 23).

Figura 2.2 – Cenas de interação com o casal (MATEAS, 2002, p. 9).

Levando em consideração que é uma área de pesquisa recente, faltam formalismos de planejamento no que tange a aspectos gerais da narração. Existem opiniões divergentes quanto aos aspectos da narração. Os trabalhos, até o presente momento, através de experimentos e pesquisas buscam formar padrões e criar conceitos e definições na área.

Entre os muitos problemas a serem discutidos e tratados pode-se destacar o papel do usuário no sistema que pode variar conforme o nível de interatividade suportado pela aplicação. Pode ser um mero espectador, passando por controlador do desenrolar da história intervindo nos diferentes caminhos pelos quais a história pode ocorrer e em alguns casos participando como o próprio autor. Correlatos ao problema em questão surgem outros como: o nível de representação e controle da narrativa, modelos de interação, relações entre personagens e enredo (CHARLES, 2001).

Alguns destes problemas provenientes do paradoxo centralizado no papel do usuário são frutos do conflito entre narração e interação. Onde a narração é mais forte tende-se para um papel mais discreto do usuário como nas abordagens orientadas a enredos, enquanto que, quando a interação é mais valorizada abordagens orientadas a personagens naturalmente tendem a apresentar melhores resultados. Lembrando que, a implementação de sistemas orientados a personagens é muito mais complexa e os resultados nem sempre são previsíveis.

Independente da abordagem adotada, de alguma maneira o usuário deve estar envolvido com o controle e direcionamento da história, caso contrário voltaríamos aos conteúdos lineares existentes como filmes, livros, entre outros. Esse envolvimento é o elemento fundamental para caracterizar os sistemas que utilizam *Interactive Storytelling*.

Capítulo 3 - Steering Behaviors

Comportamentos de direcionamento são cálculos aplicados em um modelo físico, como o modelo de veículo simples baseado em massa pontual (Figura 3.1), que se mostra bastante apropriado para uso no módulo de Visualização Gráfica do Logtell. Em jogos, por exemplo, são utilizados modelos bem mais complexos conforme a necessidade[4]. O resultado da aplicação dos comportamentos de direcionamento é um vetor força que ajusta a velocidade do personagem e conseqüentemente a sua posição e direção. Para a aplicação desses comportamentos são utilizados conceitos físicos (vistos em POZZER, 2006).

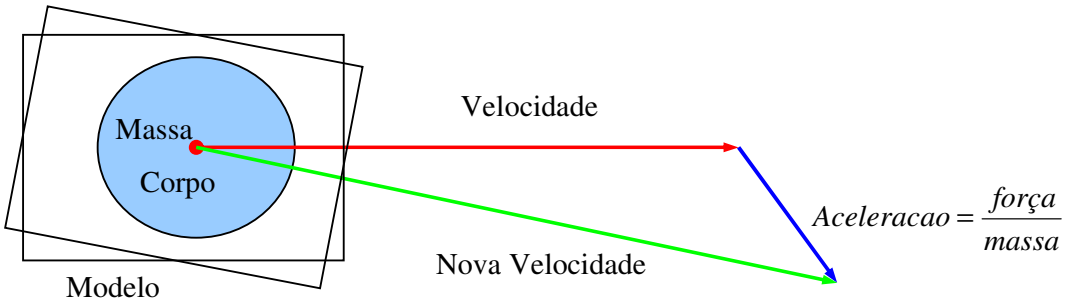


Figura 3.1 – Modelo físico baseado em massa pontual.

Reynolds (2001) separou as manobras em três camadas distintas: seleção da ação, direcionamento e locomoção (Figura 3.2). A camada de seleção da ação é a responsável pela decisão de ir de um ponto A até um ponto B.

Ao usar comportamentos de direcionamento, juntamente com um modelo físico, a camada de direcionamento permanece independente, compatível com qualquer tipo de locomoção. É nessa camada onde ocorre cálculo do vetor força de direcionamento resultante da aplicação dos comportamentos de direcionamento ativos.

Na camada de locomoção a força calculada é transformada em movimento. Por exemplo, um cavaleiro e um motoqueiro têm mecanismos de locomoção distintos. No

[4] Quanto mais complexo o modelo utilizado, maior será o custo de processamento associado à atualização dos comportamentos de direcionamento ativados.

primeiro a locomoção se dá através de músculos, articulações, ossos, enquanto que, no segundo é feita por meio do movimento das rodas impulsionadas pela tração gerada pelo motor. No caso do Logtell a camada de locomoção é basicamente locomoção envolvendo movimentação de caminhada, exceto pelo dragão que utiliza as asas para se locomover.

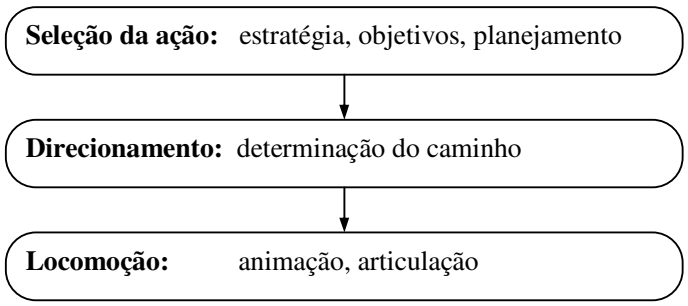


Figura 3.2 – Camadas de manobra.

Existem diversos comportamentos de direcionamento como *seek, flee*, *pursuit*, *evasion*, *offsetpursuit*, *arrival*, *wander*, entre outros (REYNOLDS, 1999). Também descreve os três comportamentos de grupos: *separation, alignment* e *cohesion.* Serão vistos em detalhes o comportamento de direcionamento *seek* (seção 3.1) e o comportamento de grupo *separation* (seção 3.2).

3.1 Comportamento Seek

O comportamento *seek* faz o personagem se locomover em direção a uma posição fixa no cenário. A Figura 3.3 demonstra o cálculo do vetor de direcionamento (Velocidade Desejada – Velocidade Corrente). O vetor Velocidade Desejada é resultado da subtração da posição de destino (alvo) da posição corrente do personagem. O código que descreve o seu funcionamento pode ser visto na Figura 3.4.

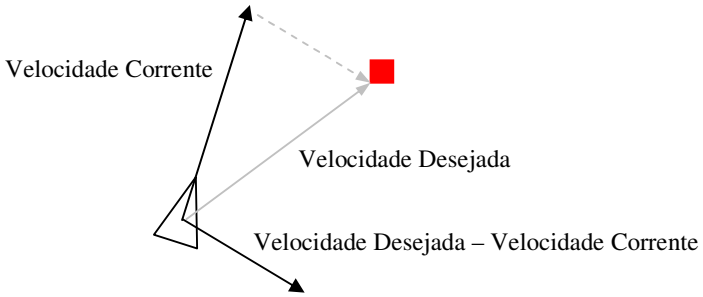


Figura 3.3 – Vetores no comportamento *seek*, reproduzido de (BUCKLAND, 2005).

```
Vector3 Steering::seek(Actor *actor, Vector3 targetPos)
{
   Vector3 desiredVelocity, actorpos;
   actorpos = actor->actorPos;
   actorpos.y = 0;
   desiredVelocity = targetPos - actor->actorPos;
   desiredVelocity.normalize2D("desiredVelocity - seek");
   desiredVelocity *= actor->maxVel;
   return (desiredVelocity - actor->getVel());
}
```

Figura 3.4 – Código referente ao algoritmo do *seek*.

3.2 Comportamentos de Grupos

Para a movimentação de batalhões uma boa abordagem é a utilização de *flocking* (BUCKLAND, 2005), que consiste na combinação de comportamentos de grupo gerando padrões organizados de movimentação através de elementos simples combinados. Esses padrões gerados podem ser caracterizados como comportamentos emergentes, ou seja, resultados finais complexos oriundos de elementos simples e coordenados (como por exemplo, o nado de cardumes de peixes, o vôo de bandos de aves ou o estouro de uma manada, como pode ser visto na Figura 3.8).

Nos comportamentos de grupos são utilizados três comportamentos básicos: separação, coesão e alinhamento. Na separação os personagens na vizinhança de um personagem são impulsionados para a direção oposta à posição do personagem gerador da força, o contrário acontece na coesão. O alinhamento é uma força que direciona os

personagens de um grupo. A Figura 3.5 demonstra o princípio do funcionamento desses comportamentos.

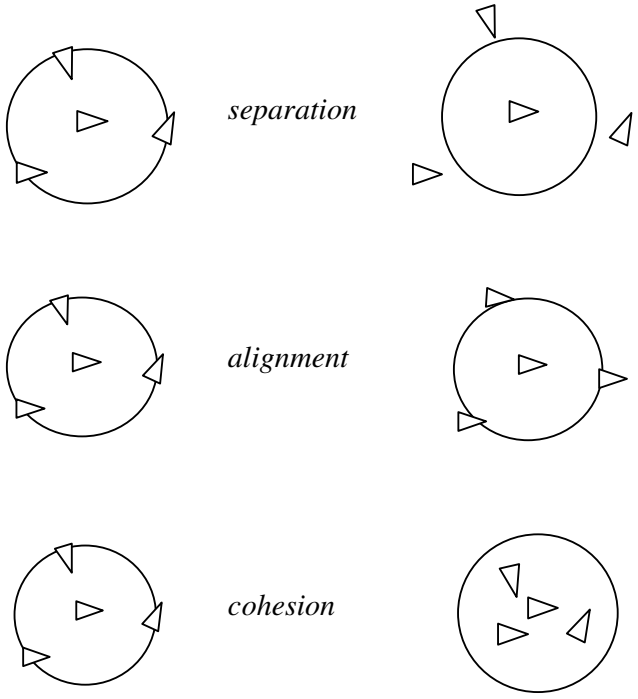


Figura 3.5 – *Separation*, *alignment* e *cohesion* (BUCKLAND, 2005, p. 115).

O círculo representado ao redor do personagem central na Figura 3.5 e na Figura 3.6 representa o raio de vizinhança do personagem sendo tratado no momento. Para determinar quais personagens estão dentro da sua vizinhança é calculada a distância entre a posição do personagem e a posição de todos os outros personagens. Aqueles que estiverem a uma distância menor ou igual ao raio de vizinhança devem ser marcados para que participem do cálculo do comportamento de grupo do personagem processado no instante.

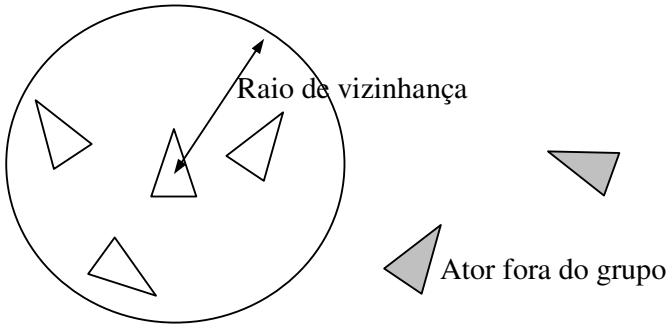


Figura 3.6 – Raio de vizinhança necessário para o cálculo dos comportamentos de grupo (BUCKLAND, 2005, p. 113).

O comportamento de grupo *separation* é utilizado no presente trabalho. A Figura 3.7 exibe o código do comportamento de grupo *separation*. Note que *actorRel->neighborTagged* representa se o personagem iterado no momento está no raio de vizinhança do ator que está sendo atualizado.

```
Vector3 SceneManager::separation(Actor* actor)
{
   Actor * actorRel;
   Vector3 SteeringForce;
   SteeringForce.set(0, 0, 0);
   vector<Actor*>::iterator iter;
   for (iter=vetActor.begin(); iter!=vetActor.end(); ++iter)
   {
      actorRel = *iter;
      if (actorRel->neighborTagged == 1)
      {
         if((actorRel != actor)&&(actorRel->neighborTagged == 1))
         {
            Vector3 ToAgent;
            ToAgent = actor->actorPos - actorRel->actorPos;
            //A força é inversamente proporcional a distância
            Vector3 aux =((ToAgent)/ToAgent.lenghtSqr());
            aux.normalize2D("Separation");
            SteeringForce += aux;
         }
      }
   }
   return SteeringForce;
}
```

Figura 3.7 – Método *separation*.

A atual concepção de comportamentos de direcionamento é resultante de quase duas décadas de pesquisas realizadas por Reynolds (2001). Esses conceitos têm seu funcionamento comprovado e são usados freqüentemente na indústria do entretenimento. A Figura 3.8 demonstra duas aplicações de *Steering Behaviors* em (a) a movimentação de um grupo de cavaleiros em um jogo da empresa *Eletronic Arts* e em (b) comportamento de grupos no consagrado filme O Rei Leão.

Figura 3.8 – (a) Cavaleiros em movimento, *Eletronic Arts*. (b) Estouro da manada em O Rei Leão, *Walt Disney Pictures.*

Capítulo 4 - Logtell

O sistema de geração, iteração e visualização de histórias interativas proposto por Pozzer (2005), o Logtell, é orientado a enredo (*plot based*), portanto a visualização é feita em relação a uma série de eventos previamente definidos. A geração dos enredos é realizada pelo IPG, que por um processo de simulação, gera seqüências parcialmente ordenadas de eventos. Cada seqüência de eventos corresponde ao suprimento de um objetivo definido pelo autor da história. Para o contexto de contos de fadas, foram definidos quatro objetivos a serem perseguidos pelos personagens:

- A vítima deve se tornar desprotegia. Isso irá despertar o interesse do vilão em raptá-la;
- Tendo uma vítima desprotegia, o vilão desejará raptá-la;
- Se a vítima for raptada, o herói desejará libertá-la. Isso tem como resultado o aumento da afeição da vítima pelo libertador;
- Se a afeição de dois personagens é alta, eles desejarão se casar.

Existem várias combinações de eventos que podem satisfazer estes objetivos. Por meio do Gerenciador de Enredos (Figura 4.1), o usuário, juntamente com o IPG, pode definir encadeamentos de eventos. Por meio desta interface o usuário pode inserir novos eventos, definir a ordenação total a partir de restrições temporais impostas pelo IPG, bem como solicitar ao IPG a geração de uma nova seqüência de eventos que seja de seu agrado.

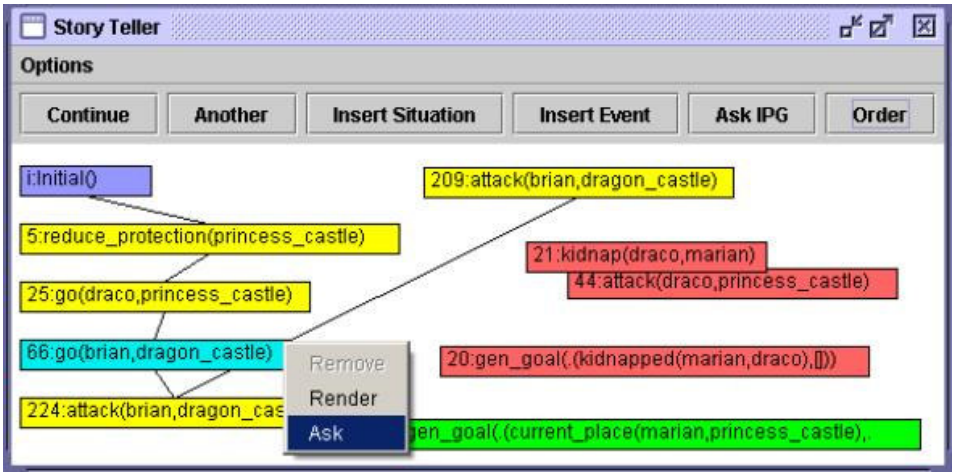


Figura 4.1 – Interface do gerenciador de enredos no Logtell.

Existem três grandes módulos que compõe o sistema, como ilustrado na Figura 4.2: O IPG (*Interactive Plot Generator*), a Interface do Gerenciador de Enredos e o módulo de Visualização Gráfica, sendo estes implementados em três linguagens distintas: Prolog, Java e C++, respectivamente.

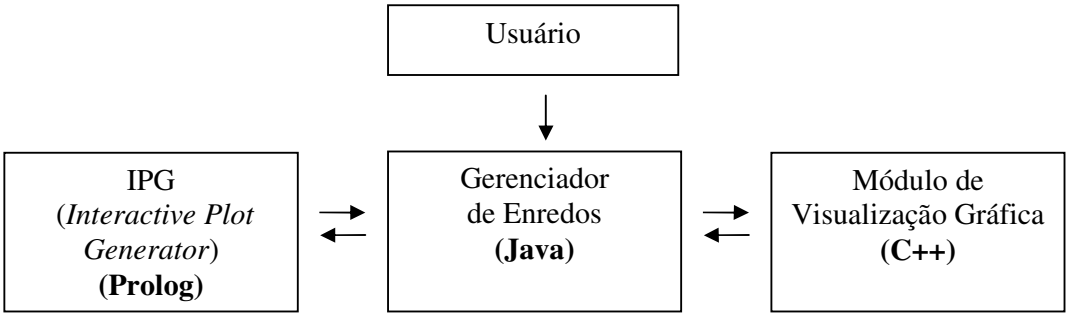


Figura 4.2 – Interação entre os módulos do Logtell.

No IPG estão especificados os eventos e ações que podem ocorrer para gerar as cenas que compõe a dramatização do enredo proposto inicialmente, pelo autor. É com base na interação entre esses eventos, que estão ligados a determinados personagens, que o contexto da história contada é criado.

O Gerenciador de Enredos é responsável por fazer a comunicação do IPG com o módulo de Visualização Gráfica. Através da interface do Gerenciador de Enredos é possível

solicitar ao IPG uma seqüência de eventos, reordenar e inserir eventos, bem como controlar a visualização gráfica. A interface também garante que só possam ser dramatizadas histórias com ordenações consistentes[5].

O módulo de Visualização Gráfica, por sua vez, encapsula todas as funcionalidades necessárias à dramatização da história. É composto pelo gerenciador de ações, o cenário, os personagens, a câmara virtual e o motor gráfico.

Este módulo recebe do Gerenciador de Enredos eventos a serem dramatizados. Os eventos são passados ao módulo de Visualização Gráfica que determina a quais personagens o evento está associado informando-os do que devem fazer e onde se posicionar no cenário.

O posicionamento da câmara para captar a cena (HE, 1996) também fica a cargo do módulo de Visualização Gráfica. Como produto final é obtido uma malha poligonal que representa a dramatização de uma cena em um determinando momento. O módulo de Visualização Gráfica utiliza a biblioteca OpenGL (WOO, 1997).

4.1 Eventos e ações

Os eventos (operações para o IPG) que são dramatizados pelo módulo de Visualização Gráfica internamente são apresentados na Tabela 4.1. Esses eventos são interpretados pelo módulo de Visualização Gráfica que as transforma em ações que são executadas pelos personagens.

Tabela 4.1 – Exemplos de eventos e seus respectivos parâmetros

Eventos	Parâmetros
reduce_protection	{"*reduce_protection*","*NULL*","*princess_castle*"}
Go	{"*go*","draco","*princess_castle*"}
Attack	{"*attack*","draco","*princess_castle*"}
Kidnap	{"*kidnap*","draco","marian"}
Fight	{"*fight*","brian","draco"}
Kill	{"*kill*","brian","draco"}

[5] Uma ordenação de eventos é consistente quando não viola a lógica especificada pelo autor da história.

Pode-se ver na Tabela 4.1 que todo evento decomposto possui três parâmetros, o primeiro é sempre a ação a ser dramatizada. Os parâmetros podem ter funções diferentes conforme o evento em questão. O evento *reduce_protection* é relativo a um lugar, onde deve ser reduzida a proteção. No caso o local é *princess_castle* e o segundo parâmetro é nulo.

Eventos como *go* e *attack* utilizam o segundo parâmetro para determinar qual personagem executará a ação e o terceiro é relativo ao lugar. Outros eventos como *kidnap, fight* e *kill* possuem dois personagens como parâmetros: o primeiro sendo o executor da ação e o segundo o personagem com qual o primeiro deve interagir.

No momento em que o evento é decomposto em ações destinadas aos personagens em questão, esse passa a ser o evento corrente que permanece como tal até a realização das ações relativas a ele. Quando da finalização das ações relativas ao evento corrente, o módulo de Visualização Gráfica solicita ao Gerenciador de Enredos um novo evento.

4.2 Finalização de uma ação

O final de uma ação é detectado de maneiras diferentes dependendo da ação corrente. Existem basicamente dois tipos de ações no Logtell quanto ao término:

- As que terminam após um determinado tempo;
- As que terminam quando o personagem chega a um determinado lugar.

A ação *fight()* e a ação *stand()* são exemplos de ações finalizadas por um fator temporal. Como pode ser visto na Tabela 4.2, os protótipos dos construtores dessas ações possuem o parâmetro *duration*. A ação *walk()* utiliza o parâmetro *pos* como destino e só termina quando este é alcançado.

O evento *go* (Tabela 4.1) deriva em uma ação *walk()* e termina quando o personagem chega ao destino. Eventos mais complexos como o *kidnap()* que envolvem várias sub-ações *walk()* e *stand()* realizadas pelo dragão e pela princesa também acaba quando um personagem chega a um determinado local, no caso quando a princesa chega ao castelo do dragão onde é mantida em cativeiro.

Tabela 4.2 – Exemplos de ações e seus respectivos construtores

Ação	Protótipo
walk()	*Action(int type, Vector3 pos, Vector3 *relPos, Object* obj, float dist)*
kidnap()	*Action(int type, Actor * actor, Object* obj, float dist)*
fight()	*Action(int type, Actor *actor, float currTime, float duration, bool msg, float dist)*
stand()	*Action(int type, float currTime, float duration)*

4.3 O módulo de Visualização Gráfica

É no módulo de Visualização Gráfica onde as modificações propostas ocorrem. Na Figura 4.3 encontra-se o IPG, o Gerenciador de Enredos e o módulo de Visualização Gráfica, ampliado, mostrando em detalhes a integração entre as partes que compõem sua estrutura interna.

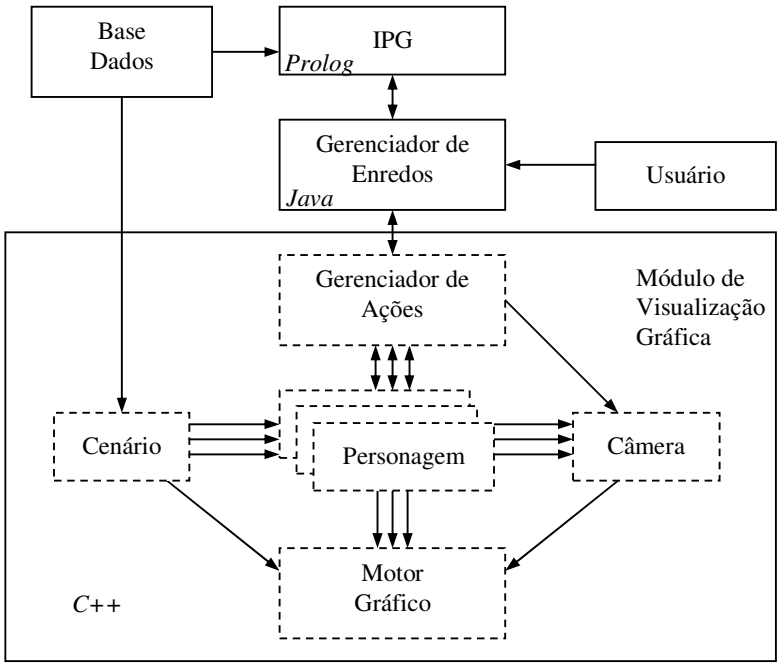


Figura 4.3 – Módulos do Logtell, módulo de Visualização Gráfica (C++) em detalhes (POZZER, 2005, p. 90).

O gerenciador de ações é o responsável por transformar os eventos oriundos do módulo Java (Gerenciador de Enredos) em ações a serem designadas aos personagens aos quais o evento compete. É nesse sub-módulo onde é executado o laço responsável pela execução da lógica e renderização gráfica de cada personagem (ator) da história.

Os personagens são agentes reativos (CUNHA, 2001), situados no cenário, que realizam ações conforme informado pelo gerenciador de ações do módulo de Visualização Gráfica. No momento oportuno cada um é atualizado, e sua lógica é processada (D'AMICO, 1995).

Os personagens principais são agentes relacionados ao IPG que interagem entre si gerando as condições necessárias para a execução do enredo proposto pelo autor. Os personagens figurantes são descritos apenas no módulo de Visualização Gráfica. Ambos são representados da mesma maneira, como um modelo 3D animado.

A câmara virtual é um importante sub-módulo responsável por capturar a ação que está acontecendo. A cada instante a câmera é atualizada e caso necessário (colisão com algum prédio, troca de ação, etc.) são estimadas pelo sistema três novas câmeras e a melhor é selecionada utilizando-se heurísticas de posicionamento baseadas em técnicas cinematográficas (TOMLINSON, 2000) (CHARLES, 2002).

O cenário é composto por: texturas, construções, terreno e o céu. Estes elementos que compõem o cenário fazem parte da base de dados. São carregados durante a inicialização do módulo de Visualização Gráfica e a cada atualização são renderizados.

4.3.1 O Gerenciador de Ações

Este sub-módulo é responsável pelo controle das ações de todos os personagens da história (principais e figurantes). Ele implementa um laço que itera todos os personagens e verifica o término do evento corrente. Em caso afirmativo, um novo evento é requisitado. Para que seja constatado o término de um evento é necessário que a ação corrente do personagem principal iterado no momento esteja relacionada com o evento corrente. Cabe também a este módulo delegar ações para personagens que não têm ação definida, o que ocorre na inicialização, por exemplo. Outro caso ocorre quando um personagem figurante termina uma ação específica. Por exemplo, quando a ação *fight()* é finalizada, o gerenciador de ações delega ao figurante a ação padrão *work()*.

O processamento da lógica dos personagens consiste na execução do método *run()* de cada personagem. O método *run()* seleciona o código específico da ação corrente e verifica o término das ações. As ações retornam um valor booleano que quando verdadeiro indica que a ação foi concluída. Ao ser detectado um final de ação, esta é encerrada. O trecho de código visto na Figura 4.4 sintetiza o funcionamento do método *run()*.

```
void Actor::run(float currTime, float frameSpeed)
{
   this->currTime   = currTime;
   this->frameSpeed = frameSpeed;
   (...)
   finishedAction = false;
   switch( currAction->actionType )
   {
      case ACTION_STAND:
         finishedAction = stand();
         break;
      case ACTION_WALK:
         finishedAction = walk();
         break;
      (...)
   }
   if( finishedAction == true )
     endCurrentAction();
}
```

Figura 4.4 – Funcionamento do método *run()* dos personagens.

Capítulo 5 - Implementação

5.1 A versão sem Prolog e Java

Na versão original do Logtell (POZZER, 2005), já foi comprovado que o IPG, juntamente com o Gerenciador de Enredos, são capazes de gerar enredos coerentes para a história proposta. Como aqui tratamos de uma extensão do mesmo, isolamos apenas o módulo de Visualização Gráfica, onde são efetuadas as modificações.

Originalmente o módulo de Visualização Gráfica (*C++*) é compilado como uma biblioteca dinâmica (*dll*), mas para permitir que fossem testadas as modificações sugeridas neste trabalho foi criada uma versão sem o IPG (*Prolog*) e sem o Gerenciador de Enredos (*Java*). Esta versão pôde ser compilada como executável, tornando possível a depuração das modificações no módulo de Visualização Gráfica.

Para simular eventos enviadas ao módulo de Visualização Gráfica pelo Gerenciador de Enredos criou-se uma matriz que contém um conjunto de eventos a serem dramatizadas (Figura 5.1). Os eventos são passados utilizando vetores de caracteres, da mesma forma como seria feito pelo Gerenciador de Enredos, mantendo assim a compatibilidade.

```
char operacoes[10][3][100] = {
      {"reduce_protection","NULL","princess_castle"},
      {"go","draco","princess_castle"},
      {"attack","draco","princess_castle"},
      {"kidnap","draco","marian"},
      {"go","brian","dragon_castle"},
      {"fight","brian","draco"},
      {"kill","brian","draco"},
      {"free","brian","marian"},
      {"marry","brian","marian"}
};
```

Figura 5.1 – Matriz de eventos a serem dramatizados.

A dramatização da história exemplo é curta, mas a completa inicialização do sistema é demorada. Isto se deve ao fato de que nesse processo são carregados todos os modelos 3D dos personagens e o cenário. Estando o sistema inicializado a geração de uma ordenação parcial

de eventos consistente (história a ser dramatizada) pelo IPG demora aproximadamente dez segundos[6].

A solução adotada permite liberdade na escolha da operação a ser dramatizada, bem como na ordem da dramatização. Pode-se inclusive violar restrições lógicas especificadas pelo autor da história em benefício da geração de seqüências e eventos que venham a melhor testar os recursos adicionais sendo implementados ao Logtell neste trabalho. Com isso, é possível dramatizar apenas os eventos nos quais estão sendo feitas alterações, sem a necessidade de uma ordenação parcial consistente.

5.2 O Gerenciador de Batalhas

Para tornar possíveis batalhas que valorizam a participação dos personagens figurantes, no sistema original, seria necessário que na geração da história pelo IPG cada figurante fosse tratado como um personagem principal. Entretanto, isso ocasionaria uma sobrecarga de processamento que tornaria impraticável essa abordagem. Descrever os personagens no IPG é uma tarefa complexa. Quanto menor o número de personagens no IPG mais rápida e simples será a geração de uma ordenação consistente de eventos.

Batalhas entre dois exércitos diferentes formados por personagens figurantes podem ocorrer durante a dramatização sem alterar o enredo desde que nenhum personagem principal seja envolvido de maneira que contrarie o enredo proposto. Para evitar essa situação, o sistema impossibilita que um personagem figurante derrote um personagem principal em uma batalha.

Os soldados ficam na frente do castelo ao qual pertencem. Inicialmente todos eles realizam a ação *work()* que consiste em caminhar nas proximidades da parte frontal do castelo, patrulhando a área.

Uma batalha é uma ação coletiva. Várias ações *fight()* são delegadas aos personagens envolvidos na batalha. Essas ações são relativas a um mesmo evento *attack* e o resultado de todas essas ações determina o fim com sucesso, ou não dessa operação.

O *attack* original era delegado diretamente ao dragão (ou herói). O personagem principal partia de seu castelo com destino ao castelo onde iria atacar, ao chegar derrotava dois guardas e finalizava o ataque. O novo *attack* proposto é composto por quatro etapas, descritas na seqüência (seção 5.2.1).

[6] Em um AMD Athlon XP 2000+, 768 MB RAM.

O Gerenciador de Batalhas é um novo componente adicionado ao Logtell através do qual é feito o controle das várias ações envolvendo os vários personagens figurantes que compõe uma batalha coletiva que se deseja realizar. Quando o módulo de Visualização Gráfica recebe um evento *attack*, o Gerenciador de Batalhas é inicializado. Durante a incialização são selecionados soldados do personagem ao qual o evento é destinado. No evento *attack* da Figura 5.1 são selecionados soldados do personagem Draco e também guardas do local *princess_castle* que será atacado.

O fluxo de ações delegadas aos personagens envolvidos, bem como o cálculo dos efeitos da batalha, é realizado pelo Gerenciador de Batalhas. Quando é derrotado um número pré-estabelecido de guardas do castelo atacado o Gerenciador de Batalhas encerra sua atividade e é solicitada uma nova operação ao Gerenciador de Enredos (ou à matriz de eventos, na versão sem *Prolog* e *Java*).

5.2.1 Etapas de uma batalha

Inicialização da batalha

Nesta fase da batalha são selecionados soldados do personagem principal que irá atacar o castelo do adversário (inferidos dos parâmetros do evento corrente). À medida que vão sendo selecionados os soldados atacantes, também são selecionados os guardas do castelo que sofrerá o ataque. Ambos, atacante e guarda, são armazenados em uma matriz de atores que estão participando da batalha, para permitir a posterior contabilização dos efeitos da batalha. Então é delegada ao atacante uma ação *fight()* e o atacante parte em direção ao adversário.

Na inicialização cada guarda selecionado é marcado como já interagindo, impedindo que nesse momento inicial mais de um atacante selecione o mesmo guarda. Caso esta restrição não seja feita, vários atacantes quase sempre atacam o mesmo guarda. Isto ocorre devido ao fato de o guarda selecionado ser o mais próximo ao atacante. Os atacantes estando próximos uns dos outros existem grandes chances de um mesmo guarda ser o mais próximo a vários atacantes.

Quando são selecionados soldados suficientes conforme uma constante pré-estabelecida a inicialização acaba e tem início a contabilização dos efeitos da batalha. A Figura 5.2-a demonstra a seleção de um atacante e do adversário mais próximo em relação ao

atacante em questão, bem como a partida do atacante em direção ao adversário. Na Figura 5.2-b ocorre a situação onde é finalizada a inicialização de uma batalha com quatro atacantes e os atacantes partem rumo aos seus respectivos adversários.

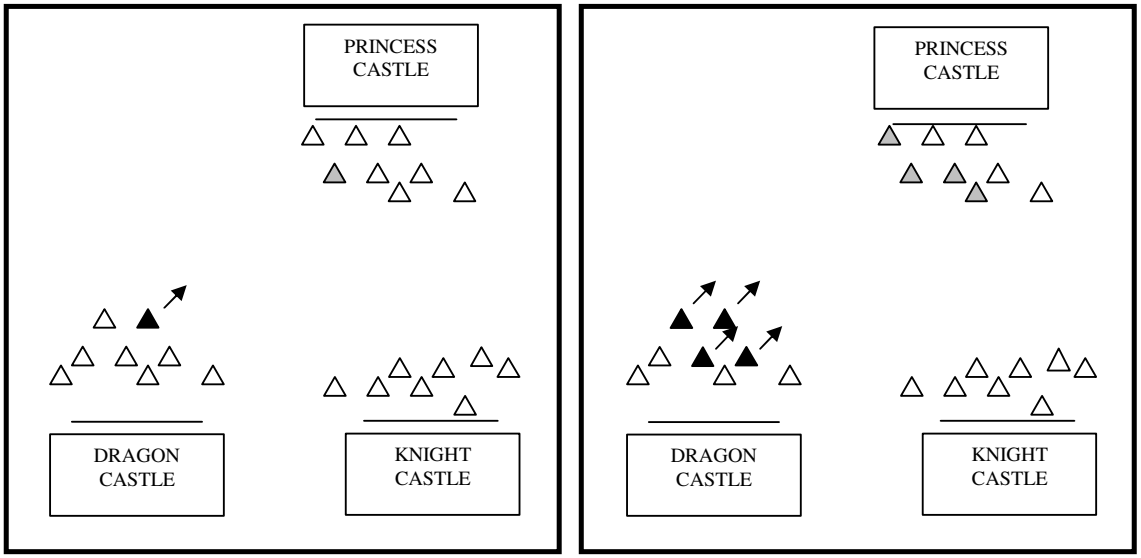


Figura 5.2 – (a) Soldado atacante (preto) partindo rumo ao seu alvo (cinza) (b) Quatro soldados partindo para o ataque.

Contabilização da batalha

Originalmente no Logtell a duração da ação *fight()* ocorre por um determinado tempo. O vencedor é aquele que primeiro detectar que a ação já excedeu o tempo de duração estabelecido. Então o personagem derrotado inicia uma ação *death()* e a ação *fight()* corrente é terminada.

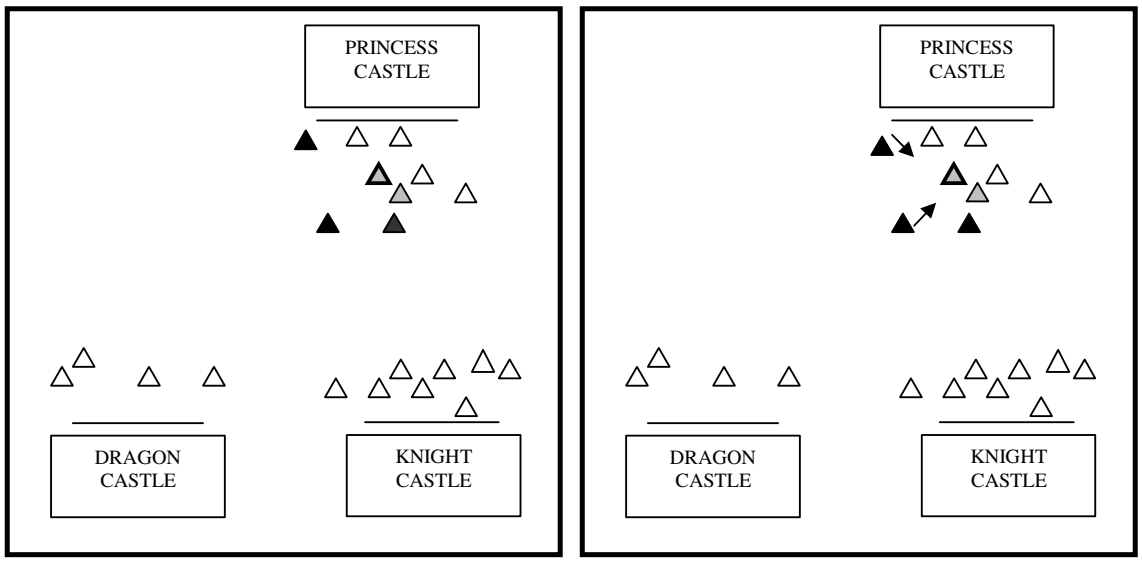
A ação *fight()* foi modificada para que, em caso de luta entre dois personagens figurantes, sejam utilizados indicadores de energia, que vão diminuindo ao longo da luta. Quando o indicador de energia de um personagem chegar a um valor menor ou igual a zero, de maneira análoga a implementação original, este personagem inicia uma ação *death()* e a ação *fight()* corrente é terminada.

Para que um personagem diminua a energia do seu adversário é necessária uma troca de mensagem onde o atacante informa ao adversário quanto de dano[7] lhe causou. Para tornar os confrontos mais realistas, cada personagem figurante recebe inicialmente um nível de energia aleatório, que varia de 1000 a 1500.

Inicialmente são contabilizados quantos e quais personagens (atacantes ou guardas) estão mortos. Caso o número de guardas mortos seja o suficiente para o término do evento *attack* (conforme constante pré-definida) a contabilização é encerrada e passa-se a próxima etapa da batalha.

Se ainda não foram derrotados adversários suficientes o Gerenciador de Batalhas continua atuando monitorando a batalha[8]. Os atacantes que derrotaram seus adversários encontram novos adversários, escolhendo preferencialmente adversários que já tenham se envolvido na batalha. Caso mais de um personagem ataque o mesmo alvo, este sofrerá uma redução de energia maior e tende a morrer mais rápido.

Na Figura 5.3-a pode ser visto um caso onde ocorrem várias situações distintas: dois atacantes vencem (pretos mais a esquerda), um é vencido pelo guarda (cinza com borda preta grossa) e outro continua a lutar. Os dois atacantes que já haviam vencido seus oponentes partem para atacar o guarda que derrotou o atacante (Figura 5.3-b).



[7] O dano é calculado somando-se um valor constante e um valor aleatório em um intervalo pré-determinado.

[8] Caso o primeiro grupo de atacantes fracasse é enviado um segundo grupo, na remota possibilidade do segundo grupo também fracassar é enviado o personagem principal, que certamente vencerá, pois o sistema impede que um personagem figurante derrote um principal.

Figura 5.3 – (a) Dois atacantes vencem, um é derrotado e outro continua a lutar. (b) Os dois atacantes parados partem para atacar o guarda que derrotou o atacante.

Envio dos sobreviventes para o castelo de origem

Quando o ataque é finalizado, atacantes sobreviventes são enviados para casa, independente da ação que estão executando. É delegada a cada um dos sobreviventes uma ação *walk()* com destino a posição em frente a porta do seu castelo de origem. Feito isso o Gerenciador de Batalhas avança para a próxima fase, a detecção da chegada dos sobreviventes em casa. A Figura 5.4-a mostra uma batalha ocorrendo, em seguida, na Figura 5.4-b, os quatro guardas derrotados[9]. Quanto todos os guardas atacados são derrotados, o Gerenciador de Batalhas atribui aos sobreviventes a ação *walk()* com destino ao castelo de origem.

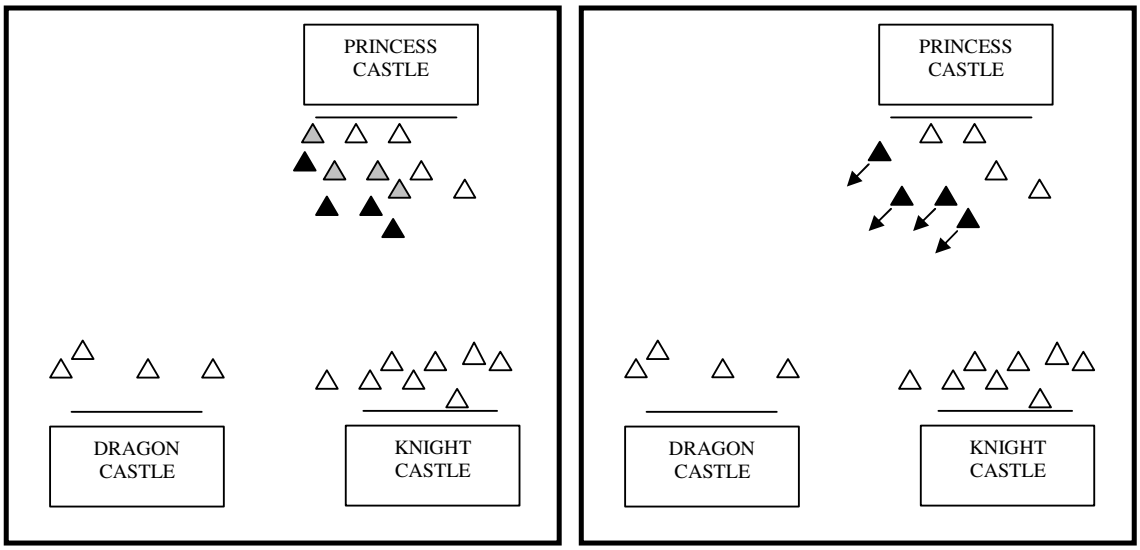


Figura 5.4 – (a) Batalha ocorrendo entre quatro atacantes e quatro guardas. (b) Ataque bem sucedido, guardas do castelo derrotados e o retorno para casa.

[9] Considerando que quatro seja o número de guardas que precisam ser derrotados para encerrar o ataque.

Chegada dos sobreviventes ao castelo de origem

Todo personagem tem uma distância mínima de interação. Essa distância representa, por exemplo, a distância mínima que um personagem deve estar para que possa começar uma luta. Todos os atacantes que estão retornando são testados. Quando todos os sobreviventes chegam a uma posição com distância menor ou igual à distância mínima de interação em relação à posição da porta do seu castelo de origem, o Gerenciador de Batalhas comuta para o estado final. A Figura 5.5 mostra o momento em que o primeiro dos sobreviventes chega a uma distância menor que a distância mínima de interação da porta do castelo.

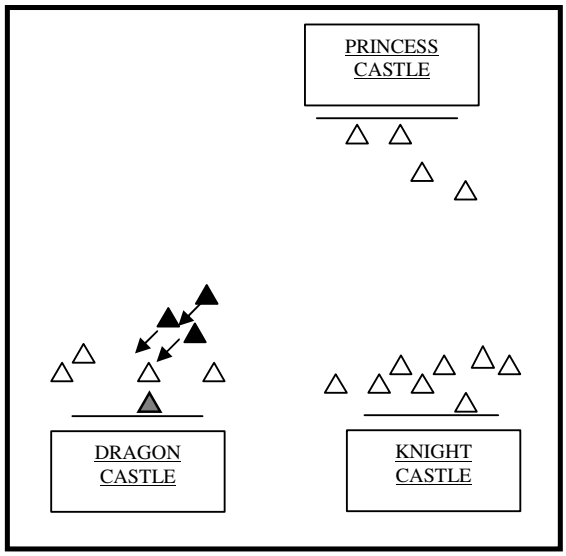


Figura 5.5 – Chegada do primeiro sobrevivente (cinza) ao seu castelo de origem.

No estado final, o evento *attack* é finalizado, juntamente com o Gerenciador de Batalhas. Neste momento o sistema sai do modo de ataque coletivo. Um novo evento é solicitado pelo gerenciador de ações ao Gerenciador de Enredos ou à matriz de eventos (Figura 5.1), e os personagens voltam a realizar ações convencionais que são delegadas pelo gerenciador de ações.

Pode ocorrer, dependendo da configuração inicial, que os soldados enviados para atacar um outro castelo fracassem em sua missão. Por exemplo, se forem enviados dois soldados para atacar o castelo do adversário quando para satisfazer o ataque coletivo é

necessário que cinco guardas sejam derrotados, muito provavelmente os atacantes serão derrotados[10]. Existem duas abordagens que podem ser seguidas para contornar o problema:

- Enviar o personagem principal ao qual o evento *attack* era originalmente delegado para que termine o ataque. Isso evita o caso em que eventualmente o novo pelotão enviado também não obtenha sucesso, o que torna a visualização repetitiva para o usuário. Como o sistema original, por padrão, não permite que um personagem figurante derrote um personagem principal, as lutas em que o personagem principal venha a se envolver certamente serão bem sucedidas. Porém, caso restem vários guardas a serem derrotados para satisfazer o ataque coletivo o personagem principal será obrigado a lutar com vários guardas até que possa ser encerrada a batalha. Esta estratégia impossibilita também a ocorrência da remota situação onde todos os guardas do castelo do personagem principal (que está atacando), que podem atacar o castelo adversário, sejam derrotados;
- Enviar um novo pelotão para concluir o ataque. Para isso basta reiniciar o Gerenciador de Batalhas sem solicitar novo evento. Neste caso os problemas citados anteriormente precisam ser tratados. Para solucionar a remota situação onde todos os guardas do castelo do personagem principal que está atacando são derrotados só há uma solução: enviar o próprio personagem principal. Esta acaba sendo uma terceira opção. Uma outra abordagem possível é a criação de novos personagens figurantes durante a execução da batalha. Porém, embora os personagens figurantes só estejam descritos no módulo de Visualização Gráfica optou-se por descartar essa hipótese, pois contraria completamente a concepção do Logtell onde todos os personagens são criados na inicialização do sistema.

Neste trabalho foi adotada uma solução híbrida: inicialmente é enviado um batalhão para atacar o castelo adversário. São enviados três soldados sendo que cada castelo tem sete soldados de guarda. Caso este batalhão falhe, o Gerenciador de Batalhas envia outro batalhão com o número de guardas sobreviventes ao primeiro ataque. O segundo batalhão ataca diretamente os guardas que sobreviveram ao ataque anterior, pois estes estão com nível de vida mais baixo. Caso esse segundo batalhão venha a falhar (o que é muito improvável), é enviado o personagem principal para realizar a tarefa. Variações no número de soldados

[10] Caso seja utilizado o *fight()* original, não baseado em nível de energia, é mais fácil para um atacante derrotar mais de um guarda.

enviados e no número de soldados que devem ser mortos para satisfazer o ataque coletivo podem tornar mais evidente o reenvio de tropas. Desde que não ocorra em excesso, o reenvio de tropas torna a representação gráfica mais duradoura e contagiante.

Como trabalho futuro pode-se incorporar ao Gerenciador de Enredos uma interface por onde o usuário possa selecionar como o evento *attack* deve ser executado: pelo personagem principal, por um batalhão de soldados ou solução híbrida em caso de falha do batalhão.

5.3 Garantindo o Sobreposicionamento Zero

Esperava-se inicialmente que uma combinação entre o comportamento de direcionamento *seek* e o comportamento de grupo de *separation* fosse suficiente para garantir que não houvesse intersecção entre os personagens no cenário. Segundo Buckland (2005), apenas o comportamento *separation* não é suficiente para garantir a ausência de intersecções entre os personagens. Buscando solucionar este problema foi aplicado o método conhecido por *nonPenetrationConstraint*, cujo algoritmo consta na Figura 5.6.

```
void SceneManager::nonPenetrationConstraint(Actor *actor)
{
   Actor *actorRel;
   vector<Actor*>::iterator iter;
   for (iter=vetActor.begin(); iter!=vetActor.end(); ++iter)
   {
      actorRel = *iter;
      if (actorRel == actor)
         continue;
         //distancia entre o ator em relação ao ator do iterador
      Vector3 toEntity = actor->actorPos - actorRel->actorPos;
      toEntity.y = 0;

      float distFromEachOther = toEntity.lenghtSqr();

      //Se essa distancia é menor que a soma dos raios deles então
      //este ator deve ser movido para longe
      //na direção paralela ao vetor toEntity.

      float amountOfOverLap = actor->actorRaio + actorRel->actorRaio -
                              distFromEachOther;
      if (amountOfOverLap >= 0)
      {
      //move o ator proporcionalmente ao overlap
      actor->actorPos = (actor->actorPos +
                        (toEntity/distFromEachOther) * amountOfOverLap);
      }
   }//Próximo ator
}
```

Figura 5.6 – Método *nonPenetrationConstraint.*

A ação *walk()* é fundamental em qualquer atividade que envolva movimentação, pois somente nesta ação a posição dos personagens é modificada. Por conseguinte, qualquer método que envolva movimentação opera na ação *walk()*, assim como o *nonPenetrationConstraint*.

No método *nonPenetrationConstraint* são iterados todos os personagens. A Figura 5.7 demonstra o funcionamento do método. Note que as atualizações da posição ocorrem antes da renderização do personagem. A operação do método *nonPenetrationConstraint* pode ser dividida em três fases distintas realizadas para cada personagem da iteração em relação ao personagem que está executando a ação *walk()*. O personagem que está executando é desconsiderado, para evitar que seja calculada uma distância zero incorretamente. As três fases são:

- Cálculo da distância entre o personagem iterado e o personagem que está executando (*distFromEachOther*).

- Subtração da soma dos raios dos personagens pela distância calculada (*amoutOfOverLap*).
- Caso essa subtração resulte em um valor positivo, indicando que a distância entre os personagens é menor que o seu raio, é somado à posição atual do personagem com um vetor normalizado (*toEntity/distFromEachOther*) multiplicado pelo comprimento da intersecção entre os personagens (*amoutOfOverLap*).

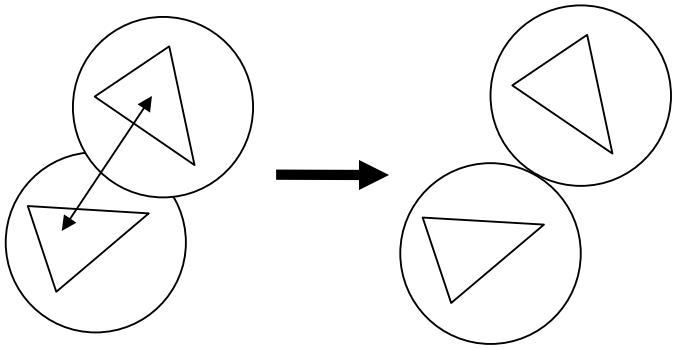


Figura 5.7 – Operação do método nonPenetrationConstraint (BUCKLAND, 2005, p. 125).

5.4 Comportamentos de Direcionamento e de Grupos

Visto que toda alteração na posição dos personagens ocorre na ação *walk()*, são criados mecanismos para que na ação *walk()* possa ser utilizado o comportamento de direcionamento *seek* e o comportamento de grupo *separation*. Combinados estes comportamentos criam uma maneira completamente alternativa de movimentação. Para sua utilização foram feitas modificações na ação *walk()*.

Originalmente a ação *walk()* modifica a posição dos personagens a cada execução com base em um multiplicador aplicado ao *framerate* (taxa de atualização da tela) corrente simplesmente somando esse valor resultante nas componentes x e z do vetor posição do personagem. Porém, com a utilização de comportamentos de direcionamento esse valor varia conforme a força de direcionamento e não mais por um valor baseado no *framerate*.

O vetor força de direcionamento resultante ao final do cálculo dos comportamentos de direcionamento ativos[11] é somado à posição atual do personagem. Este cálculo pode ser feito seguindo dois caminhos principais. No primeiro somam-se todas as forças ativas e a força resultante é truncada por um valor máximo, enquanto que, no segundo os comportamentos são avaliados por ordem de prioridade, adicionando-se a força de cada um dos comportamentos gradativamente conforme a sua relevância.

Optou-se pela segunda abordagem, pois a abordagem com prioridades detém recursos mais refinados de parametrização que permitem impedir casos onde forças contrárias, porém com relevâncias diferentes sejam interpretadas como de igual relevância. Por exemplo, se a força do comportamento de grupo *separation* for considerada como de igual relevância que a do comportamento de direcionamento *seek* pode ocorrer que quando vários personagens estejam próximos de um determinado personagem a força de separação seja tão grande que o personagem deixa de perseguir o alvo.

O método *calculate* (Figura 5.8) é responsável por ponderar os comportamentos ativos. No caso os comportamentos são calculados priorizando o comportamento *seek* em relação ao comportamento *separation*. Os parâmetros *MultSeek* e *MultSeparation* servem para sintonizar o peso de cada comportamento. A ordem também influencia no resultado, pois os comportamentos são avaliados na ordem em que são executados.

O método *accumulateForce* utilizado na Figura 5.8 executa os seguintes passos:

- A nova força pode ser adicionada à resultante se a soma não ultrapassar o limite máximo de força;
- Se não existe mais sobra de força resultante, a nova força e todas as forças ainda por serem calculadas são descartadas, visto que são de menor importância;
- Se ainda existir um pouco de força resultante, a força em questão é truncada para o valor que ainda pode ser adicionado à força resultante.

[11] Normalmente vários comportamentos de direcionamento são implementados em um sistema, mas são ativados apenas os necessários em cada situação.

```
Vector3 SceneManager::calculate(Actor *actor, Vector3 destiny)
{
	//reset the force.
	Vector3 SteeringForce;
	SteeringForce.set(0, 0, 0);

	Vector3 force;

	force = seek(actor, destiny) * MultSeek;
	if (!accumulateForce(actor, SteeringForce, force))
	{
		return SteeringForce;
	}

	tagNeighbors(actor);
	force = separation(actor) * MultSeparation;
	if (!accumulateForce(actor,SteeringForce, force))
	{
		return SteeringForce;
	}

	return SteeringForce;
}
```

Figura 5.8 – Método *calculate*, força acumulada com prioridades.

O método *tagNeighbors* chamado na Figura 5.8 opera de maneira semelhante a *nonPenetrationConstraint*. A diferença é que ao invés de modificar diretamente a posição do personagem este método apenas marca os personagens que estão dentro do raio de vizinhança do personagem que está tendo sua força de direcionamento calculada no momento. Determinar quais personagens estão dentro do raio de vizinhança é essencial para comportamentos de grupo como o *separation*.

5.5 Câmera

Com a inclusão de batalhas, ocorrem cenas que envolvem vários personagens. Por isso é preciso que a câmera possa estimar sua posição de modo a acompanhar da melhor forma possível o desenrolar das batalhas ocorridas. O método adotado utiliza uma estratégia que encontra um ponto médio entre os personagens envolvidos. Esse ponto será o centro a partir do qual a câmera se distanciará. Para permitir que todos os personagens envolvidos na ação apareçam, a câmera se distancia do ponto médio de maneira que enquadre esses personagens,

como ocorre, por exemplo, quando um fotógrafo se afasta do objeto que está fotografando para conseguir enquadrá-lo totalmente no retrato.

A posição central para a qual a câmera deve focar é calculada pelo método *calculateBareCentre* (Figura 5.9). O método consiste basicamente em encontrar um ponto médio entre os personagens atacantes que estão vivos.

```
Vector3 SceneManager::calculateBareCentre(void)
{
	int i, soldiersAlive =0;
	Vector3 bareCentre;
	bareCentre.set(0, 0, 0);
	for (i = 0; i < SOLDIERS; i++)
	{
		if(battle[AGGRESSOR][i]->getState() != STATE_DEAD)
		{
			bareCentre += battle[AGGRESSOR][i]->actorPos;
			soldiersAlive++;
		}
	}
	bareCentre = bareCentre/soldiersAlive;
	if (bareCentre.modulo() != 0)
		return bareCentre;
	printf("\nErro calculateBareCentre retornou nulo!    ");
	return bareCentre;
}
```

Figura 5.9 - Método *calculateBareCentre.*

Durante uma batalha ocorrem dois tipos de ações:

- *walk()*: Esta ação ocorre quando o grupo de atacantes está se deslocando até os guardas do castelo que vai ser atacado, quando os guardas se movem até os novos alvos após derrotarem os primeiros caso seja necessário e quando os sobreviventes retornam para o castelo de origem;
- *fight()*: Quando os atacantes estão efetivamente lutando contra os guardas.

Para determinar qual é a ação corrente é verificada qual a ação que um dos atacantes vivos está realizando. Com base nesta informação é selecionado o tipo de câmera utilizada no momento.

A detecção para que a câmera entre no modo de cobertura de ação coletiva é feita utilizando informações do gerenciador de ações que o sub-módulo da câmera pode acessar, como o tipo da operação corrente, por exemplo. Uma vez estando no modo de cobertura de ação coletiva, a câmara através da informação fornecida pelo método *calculateBareCentre* determina a posição para a qual a câmera escolhida pelo sistema deve ser direcionada.

Capítulo 6 - Conclusões

Compreender o Logtell para poder incorporar as funcionalidades propostas neste trabalho não foi tarefa fácil. É um sistema complexo, estruturado em várias classes, que se comunicam para troca de informações entre módulos sobre a situação corrente de ações, eventos, personagens e cenário. Toda modificação, por mais simples que fosse, exigiu mais tempo do que se esperava, não por falta de planejamento, mas sim por dificuldades que surgiram ao longo do caminho. Essas dificuldades ficaram mais evidentes com a necessidade de manter a compatibilidade com o sistema original. Por ser um trabalho mais prático do que teórico e por envolver assuntos de pesquisa novos, como *interactive storytelling*, exigiu pesquisas avançadas, tanto em nível conceitual como de estruturação de implementação.

No trabalho realizado foi criado o Gerenciador de Batalhas, ferramenta que permite a realização de batalhas consistentes entre grupos de personagens figurantes. Também foram criados mecanismos alternativos de locomoção (baseados em *steering behaviors* e no método *nonPenetrationConstraint*), bem como um novo modo de câmera virtual para captar cenas de batalhas.

O Gerenciador de Batalhas se mostrou como um recurso valioso para agregar realismo às representações gráficas no Logtell, além do que as batalhas criadas com ele conseguiram aumentar significativamente o pequeno tempo de dramatização da história. A batalha conseguiu demonstrar graficamente melhor idéia de enfraquecer a proteção do castelo adversário do que o próprio personagem principal simplesmente atacando alguns personagens.

O uso de comportamentos de direcionamento e de grupos traz consigo um problema, a dificuldade de sintonizar os parâmetros relacionados com o modelo físico empregado, bem como a dificuldade de ponderar a relevância de cada comportamento empregado no comportamento global. Toda a base para a utilização de comportamentos de direcionamento e de grupos foi implementada assim como o comportamento *seek* e o *separation*. Os testes realizados mostraram resultados indicando que os comportamentos estão funcionando corretamente, mas não se obteve uma boa parametrização.

O método *nonPenetrationConstraint* foi implementado e funcionou corretamente. Foram constatados alguns problemas em testes com muitos personagens, onde alguns personagens não conseguiam chegar ao seu destino. Tanto os comportamentos de direcionamento quanto o *nonPenetrationConstraint* podem ser ativados e desativados

dinamicamente durante a dramatização da história, permitindo assim mais uma configuração, uma forma de personalizar a dramatização da história.

Na proposta inicial modificações na câmera só seriam feitas caso fossem necessárias. Porém, ao longo do trabalho, foi possível perceber a necessidade da criação do modo de câmera para acompanhar os ataques coletivos. Durante a criação do Gerenciador de Batalhas a visualização da coerência das ações dos personagens só era possível por meio da câmera manual do Logtell. A câmera implementada é simples e funcionou de maneira satisfatória para demonstrar os eventos da história proposta.

6.1 Trabalhos Futuros

Nesta seção são apresentados algumas sugestões de trabalhos futuros:

- Reformular o sistema, prevendo novas extensões. No estado atual, adicionar novas funcionalidades implica em muitas adaptações no sistema para que funcionem da maneira desejada.
- Integrar funcionalidades à interface do Gerenciador de Enredos permitindo que o usuário escolha como deve ser realizado o ataque, bem como parametrizar o número de soldados que devem morrer para satisfazer o ataque coletivo.
- Aprimorar os mecanismos da câmera virtual, não somente referente a recursos para tratamento de ações coletivas, mas buscar tornar a câmera um diretor. Passando as ações primeiramente à câmera virtual, esta deve então se posicionar e contextualizar melhor a cena, para só então redirecionar a ação ao(s) personagem(s) específico(s).
- Aprimorar os esquemas de movimentação, realizando estudos mais detalhados acerca do uso de *steering behaviors*.
- Criar outros tipos personagens figurantes que realizem ações diferentes, enriquecendo o cenário, como por exemplo, vendedores ambulantes, comerciantes, criando assim um vilarejo que ajude a contextualizar a história.
- Criar mecanismos para que o usuário possa visualizar a história sobre a perspectiva de personagens específicos.

- Estudar a possibilidade de o usuário assumir o controle de um ou mais personagens, deixando a cargo do usuário a realização das ações delegadas ao(s) personagem(s) que está(ão) sob o comando do usuário.
- Incorporar efeitos sonoros no Logtell, através da criação de um componente que gerencie e sincronize os efeitos sonoros de acordo com o andamento da visualização gráfica.

Referências Bibliográficas

BUCKLAND, MATT. **Programming Game AI by Example.** Wordware publishing Inc, 2005.

CAVAZZA, M.; CHARLES, F.; MEAD, S. **Character-based interactive storytelling.** IEEE Inteligent Systems, special issue on AI in Interactive Entertainment, 17(4):17-24, July 2002.

CHARLES, F.; CAVAZZA, M.; MEAD, S. **Character-driven story generation in interactive storytelling.** Technical report, VSMM, Berkeley, 2001.

CHARLES, F.; LUGRIN, J.; CAVAZZA, M.; MEAD, S. **Real-time camera control for interactive storytelling.** In: GAME-ON, London, UK, 2002.

CIARLINI, A. **Geração interativa de enredos.** PhD thesis, Departamento de Informática, PUC-Rio, Rio de Janeiro, 1999.

CUNHA, L. S.; GIRAFFA, L. M. M. **Um estudo sobre o uso de agentes em jogos computadorizados interativos**. Technical Report Series, number 017, PUCRS, October 2001.

D'AMICO, C. B. **Inteligência artificial: uma abordagem de agentes.** Technical report, Porto Alegre: CPGCC da UFRGS, 1995.

GLASSNER, A. **Interactive Storytelling: Techniques for 21st Century Fiction.** AK Peters Ltd, 2004.

GRASBON, D.; BRAUN, N. **A morphological approach to interactive storytelling.** In: Fleischmann, M.; Strauss, W., editors, PROCEEDINGS: CAST01, LIVING IN MIXED REALITIES. SPECIAL ISSUE OF METZSPANNUNG.ORG/JOURNAL, THE MAGAZINE FOR MEDIA PRODUCTION AND INTER-MEDIA RESEARCH, p. 377-340, Sankt Augustin, Germany, 2001.

HE, L.; COHEN, M. F.; SALESIN, D. H. **The virtual cinematographer: a paradigm for automatic real-time camera control and directing.** In: PROCEEDINGS OF THE 23RD ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE THECNIQUES. ACM SIGGRAPH, volumen 30, p. 217-224, August 1996.

MATEAS, M.; STERN, A. **Façade: An experiment in building a fuly-realized interactive drama.** In: GAME DEVELOPERS CONFERENCE, GAME DESIGN TRACK, San Jose, CA, March 2003.

MATEAS, M.; STERN, A. **Architecture, authorial idioms and early observations of the interactive drama façade.** Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, December 2002.

POZZER, CESAR T. **Notas de aula – Programação de Jogos 3D – Física para Jogos.** UFSM, 2006.

POZZER, CESAR T. **Um Sistema para Geração, Interação e Visualização 3D de histórias para TV Interativa.** Rio de Janeiro, 2005. 156p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

PROPP, V. **Morphology of the Folktale (Laurence Scott, Trans.).** University of Texas Press, Austin, Texas 2nd edition, 1968.

RABIN, S. **Implementing a state machine language.** In: AI Game Programming Wisdon, Charles River Media, 2002.

REYNOLDS,CRAIG W. **Steering behaviors for autonomous characters,** Game Developers Conference, 1999.

REYNOLDS,CRAIG W. **Steering behaviors for autonomous characters,** Game Developers Conference, 2001.

RUSSEL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach.** Prentice-Hall, New Jersey, 1995.

SPIERLING, U.; BRAUN, N.; IURGEL, I.; GRASBON, D. **Setting the scene: playing digital director in interactive storytelling and creation.** Computers & Graphics, 26:31-44, 2002.

TOMLINSON, B.; BLUMBERG, B. ; NAIN, D. **Expressive autonomous cinematography for interactive virtual environments.** In: AGENTS 'OO: PROCEEDINGS OF THE FOURTH INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, p. 317-324. Press, 2000.

TORRES, V. S.; VICCARI, R. M.; VALENTINI C. B.; GRINGS E. S.; KRUM A. C. R.; LIMA I. G.; SAUER L. Z.; SEIXAS L. J.; BASSO M. V. A.; GELLER M.; FALKEMBACH G. A. M.. **Aspectos da Inteligência Artificial e sua relação com a educação**. In: Revista Roteiro, Joaçaba, v.27, p.61-79, 2002.

WOO, M.; NEIDER, J.; DAVIS, T.; SHREINER, D. **OpenGL Programming Guide**. Addison Wesley, Massachusetts, 3rd edition, 1997.

ANEXO A – Operações

```
operator(1,
       go(CH,PL1),
       [
         alive(CH),
         not(kidnapped(_,CH)),
         not(kidnapped(CH,_)),
         current_place(CH,PL0),
         place(PL1),
         dif(PL0,PL1)
       ],
       [
         not(current_place(CH,PL0)),
         current_place(CH,PL1)
       ],
       10,
       [current_place(CH,PL1)],
       [], []).

operator(2,
       reduce_protection(PL),
       [
         protection(PL,KIND,LPROT),
         { LPROT>0.0, LPROT1=LPROT-10.0}
       ],
       [
         not(protection(PL,KIND,LPROT)),
         protection(PL,KIND,LPROT1)],
       10,
       [protection(PL,KIND,LPROT1)],
       [], []).

operator(3,
       kidnap(VIL,VIC),
       [
         alive (VIC), alive (VIL), victim(VIC,VIC_L),
         character(VIC,KIND1),
         {VIC_L>80.0},
         not(kidnapped(VIC,_)),
         villian(VIL, VIL_L),
         {VIL_L>80.0},
         strength(VIC, VIC_S),
         current_place(VIC,PL),
         protection(PL,KIND2,LP),
         strenght(VIL,VIL_S),
         current_place(VIL,PL),
         home(VIL,PL1),
         dif(PL,PL1),
         {VIL_S>VIC_S+LP*KIND1*KIND2}
       ],
       [
         kidnapped(VIC,VIL),
         not(current_place(VIC,PL)),
         not(current_place(VIL,PL)),
         current_place(VIC,PL1),
         current_place(VIL,PL1)
       ],
```

```
       10,
       [kidnapped(VIC,VIL)],
       [],[]).

operator(4,
       attack(CH,PL),
       [
         alive(CH),
         not(kidnapped(CH,_)),
         character(CH,KIND1),
         current_place(CH,PL),
         protection(PL,KIND2,L_PROT),
         dif(KIND1, KIND2),
         {
           L_PROT>0.0,
           L_PROT1 = L_PROT-30.0
         }
       ],
       [
         not(protection(PL,KIND2,L_PROT1)),
         protection(PL,KIND2,L_PROT1)
       ],
       10,
       [protection(PL,KIND2,L_PROT1)],
       [], []).

operator(5,
       fight(CH1, CH2),
       [
         alive(CH1),
         alive(CH2),
         not(kidnapped(CH1,_)),
         not(kidnapped(CH2,_)),
         dif(CH1,CH2),
         strength(CH1,LS1), strength(CH2,LS2),
         character(CH1,KIND1),
         character(CH2,KIND2),
         dif(KIND1,KIND2),
         {
           LS1>=10.0,
           LS2>=10.0
         },
         current_place(CH1,PL), current_place(CH2,PL),
         protection(PL,KIND3,L_PROT),
         {
           L_PROT=<0.0,
           NEW_LS1=LS1-LS2,
           NEW_LS2=LS2-LS1
         }
       ],
       [
         not(strength(CH1,LS1)), not(strength(CH2,LS2)),
         strength(CH1,NEW_LS1), strength(CH2,NEW_LS2)
       ],
       10,
       [strength(CH1,NEW_LS1), strength(CH2,NEW_LS2)],
       [], []).

operator(6,
       kill(CH1,CH2),
       [
```

```
          victim(vic,_),
          alive(CH1), alive(CH2),
          not(kidnapped(CH1,_)),
          dif(CH1,CH2),
          character(CH1, KIND1),
          character(CH2, KIND2),
          dif(KIND1,KIND2),
          strength(CH1,LS1),strength(CH2,LS2),
          current_place(CH1,PL), current_place(CH2,PL),
          protection(PL,KIND3,L_PROT),
          {
            L_PROT=<0.0,
            LS2<0.0, LS1>0.0
          }
        ],
        [
          not(alive(CH2))/*,
          not(affection(VIC,CH1,LA)),
          affection(VIC,CH1,100.0) */
        ],
        10,
        [not(alive(CH2))],
        [], []).

operator(7,
        free(HERO,VIC),
        [
          hero(HERO,_),victim(VIC,_),
          alive(HERO), alive(VIC),
          kidnapped(VIC,VIL), not(alive(VIL)),
          current_place(VIC,PL), current_place(HERO,PL),
          affection(VIC,HERO,LA)
        ],
        [
          not(kidnapped(VIC,VIL)), not(affection(VIC,HERO,LA)),
          affection(VIC,HERO,100.0)
        ],
        10,
        [not(kidnapped(VIC,VIL))],
        [], []).

operator(8,
        marry(CH1, CH2),
        [
          hero(CH1,_), victim(CH2,_),
          alive(CH1), alive(CH2),
          affection(CH1,CH2,L1),
          {L1>80.0},
          affection(CH1,CH2,L2),
          {L2>80.0},
          current_place(CH1,church),
          current_place(CH2,church),
          not(married(CH1,_)),
          not(married(CH2,_))
        ],
        [
          married(CH1,CH2), married(CH2,CH1)
        ],
        10,
        [married(CH1,CH2),married(CH2,CH1)],
        [], []).
```

```
operator(9,
       get_stronger(CH1),
       [
         alive(CH1),
         strength(CH1,L1),
         {L2=L1+80}
       ],
       [
         not(strength(CH1,L1)),
         strength(CH1,L2)
       ],
       10,
       [strength(CH1,L2)],
       [], []).
```

ANEXO B – Regras que levam à geração dinâmica de objetivos

```
/*Se, no início da história, há uma vítima, ela vai realizar alguma ação
que a tornará vulnerável.*/

rule(
    [
       e(i,victim(VIC,LEVEL)),
       e(i,character(VIC,KIND0)),
       e(i,current_place(VIC,PLACE)),
       e(i,protection(PLACE, KIND1,PROT))
    ],
    (
      [T],
      [
         h(T,current_place(VIC,PLACE1)),
         h(T,protection(PLACE1,KIND2,PROT1)),
         h({(KIND2*KIND0*PROT1)<(KIND1*KIND0*PROT)}),
         h(T>i)
       ],
       true
      )
).

/*Se a vítima ficar desprotegida, o vilão desejará raptá-la.*/

rule(
    [
       e(i,victim(VIC,_)),
       e(i,character(VIC,KIND0)),
       e(i,current_place(VIC,PLACE1)),
       e(i,protection(PLACE1, KIND1,PROT1)),
       e(i,villian(VIL,_)),
       h(g,current_place(VIC,PLACE2)),
      h(g,protection(PLACE2, KIND2, PROT2)),
      h({(KIND2*KIND0*PROT2)<(KIND1*KIND0*PROT1)})
   ],
   (
     [T3],
     [
       h(T3,kidnapped(VIC,VIL))
     ],
     True
   )
).

/*Se a vítima foi raptada, o herói tentará salvá-la*/

Rule(
    [
      e(T1,kidnapped(VIC,VIL))
    ],
    (
      [T2],
      [
        h(T2,not(kidnapped(VIC,VIL))),
        h(T2>T1)
      ],
```

```
      True
    )
).

/*Se a afeição dos dois personagens é alta, eles desejarão casar*/

rule(
    [
      e(T,affection(CH1,CH2,L1)),
       h(T,affection(CH2,CH1,L2)),
       h(T,not(married(CH1,_))),
       h(T,not(married(CH2,_))),
       h({L2>95.0}), h({L1>95.0})
    ],
    (
      [T2],
      [
        h(T2,married(CH1,CH2)),
        h(T2>T)
      ],
      True
    )
).
```