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

TÉCNICAS DE GERENCIAMENTO DE CENÁRIOS PARA AUMENTO DO TEMPO DE DRAMATIZAÇÃO EM STORYTELLING

TRABALHO DE GRADUAÇÃO

Eduardo Ceretta Dalla Favera

Santa Maria, RS, Brasil

2009

TÉCNICAS DE GERENCIAMENTO DE CENÁRIOS PARA AUMENTO DO TEMPO DE DRAMATIZAÇÃO EM STORYTELLING

por

Eduardo Ceretta Dalla Favera

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

Orientador: Prof. Cesar Tadeu Pozzer

Trabalho de Graduação N. 288

Santa Maria, RS, Brasil

2009

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

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

TÉCNICAS DE GERENCIAMENTO DE CENÁRIOS PARA AUMENTO DO TEMPO DE DRAMATIZAÇÃO EM STORYTELLING

elaborado por
Eduardo Ceretta Dalla Favera

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

COMISSÃO EXAMINADORA:

Prof. Cesar Tadeu Pozzer
(Presidente/Orientador)

Prof. Benhur de Oliveira Stein (UFSM)

Prof[a]. Marcia Pasin (UFSM)

Santa Maria, 05 de Janeiro de 2009.

“Ciência da Computação não é mais sobre computadores do que astronomia é sobre telescópios”

— E. W. DIJKSTRA

AGRADECIMENTOS

Agradeço, acima de tudo, a Deus, por ter me dado a oportunidade de estar aqui, por ter me dado a segurança de nunca estar sozinho nos momentos difíceis permitindo que pudesse me superar a cada dia.

Agradeço aos meus Pais Aluisio e Elena, por terem dado suporte em todos os momentos que precisei, por nunca me negarem uma palavra de afeto e por me colocarem no caminho certo para que pudesse crescer e amadurecer. Ao meu irmão Alexandre, por sempre ser um amigo e companheiro de todas as horas.

Agradeço a minha Namorada Karine, por estar ao meu lado sempre que precisei, pelos sorrisos sinceros e por tornar esta jornada tão mais agradável.

Agradeço ao meu Orientador Cesar Pozzer por ter me dado a oportunidade de realizar este trabalho, por confiar no meu potencial e por ter se tornado um grande amigo.

Meus agradecimentos a todos os amigos que me impediram de passar a faculdade inteira programando compartilhando alegrias e sempre estando presente nas dificuldades.

Agradeço aos professores por terem fornecido seu conhecimento e por me incentivarem a buscar cada vez mais.

Meus sinceros agradecimentos a todos que me apoiaram e tornaram este momento possível.

RESUMO

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

TÉCNICAS DE GERENCIAMENTO DE CENÁRIOS PARA AUMENTO DO TEMPO DE DRAMATIZAÇÃO EM STORYTELLING

Autor: Eduardo Ceretta Dalla Favera
Orientador: Prof. Cesar Tadeu Pozzer
Local e data da defesa: Santa Maria, 05 de Janeiro de 2009.

Muitas aplicações computacionais ligadas ao entretenimento digital utilizam histórias geradas automaticamente. Boa parte delas estão baseadas na dramatização de um roteiro previamente definido. Frequentemente, essas aplicações sofrem com um baixo tempo de dramatização, uma consequência direta do baixo nível de detalhes das histórias geradas por processos automatizados. Este trabalho propõe o aumento do tempo de dramatização ao estender o evento que realiza a movimentação dos personagens (evento do tipo *Go*()). Para permitir isto, o cenário onde a narrativa é interpretada foi estendido e mais detalhado e uma estrutura de dados que possa manter um grande cenário foi desenvolvida. Ao mesmo tempo, o usuário pode fornecer dicas que irão influenciar a dramatização. Uma abordagem de intervenção fraca do usuário foi utilizada durante a dramatização.

Palavras-chave: Storytelling; dramatização; busca de caminhos; computação gráfica; grandes cenários; TV digital.

ABSTRACT

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

SCENARIO MANAGEMENT TECHNIQUES TO ENLARGE DRAMATIZATION TIME IN STORYTELLING

Author: Eduardo Ceretta Dalla Favera
Advisor: Prof. Cesar Tadeu Pozzer

Several computer applications make use of automatically generated stories in digital entertainment. Some of those are based on a dramatization of a previously defined plot. Often, these applications suffer from short dramatization time, a consequence of the shallowness of the plot. This work proposes the enlargement of the dramatization time by enhancing the event that moves the characters. To support this, the scenario where the narrative takes place was enlarged and more detailed, and a data-structure that can handle a large scenario was developed. At the same time, the user can give tips defining preferences to be taken into account during the dramatization. A weak user intervention approach was adopted during dramatization.

Keywords: storytelling, dramatization, path search, computer graphics, large scenarios, digital TV.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IPG	Interactive Plot Generator
SVM	Support Vector Machine
LaCA	Laboratório de Computação Aplicada
GUI	Graphical User Interface

SUMÁRIO

1 INTRODUÇÃO

Uma das áreas da computação atual que tem apresentado um forte crescimento é o mercado do entretenimento digital. Aplicativos gráficos que têm por objetivo distrair e entreter um determinado usuário por um período de tempo têm começado a utilizar narrativas para tornar seu conteúdo mais atrativo. Jogos de computador e aplicativos experimentais para TV digital interativa já fazem uso de enredos gerados de forma dinâmica e com participação do usuário. Estas narrativas podem ser contadas automaticamente pelo computador, introduzindo conteúdo personalizado para quem está interagindo.

O *software* necessário para tais aplicações está sendo intensamente pesquisado atualmente. Tais sistemas utilizam técnicas de *storytelling* (POZZER, 2005; CAVAZZA; CHARLES; MEAD, 2002) e incluem, não somente o processo de criação de histórias, mas também formas de dramatizar essa narrativa tornando o seu conteúdo mais atrativo para os usuários.

A aplicabilidade desses sistemas já vem sendo explorada pela indústria de jogos, onde esta tecnologia é utilizada para substituir *scripts* pré-estabelecidos e codificados à mão, com uma melhor chance de introduzir variações surpreendentes e interessantes na história do jogo. Deste modo o usuário deixa o papel de um simples ator na narrativa do jogo para se tornar um ator e co-produtor da história. Outras oportunidades de pesquisa surgem ao considerar aplicativos que utilizam interação para criar e modificar conteúdo de entretenimento casual. A televisão digital é considerada uma destas oportunidades, tendo em vista que ela fornece ferramentas para que um telespectador possa customizar o conteúdo que está sendo apresentado através da interação com a sua *set-top box* (*hardware* de suporte à interação e qualidade digital na TV digital). Na TV digital, o usuário teria, a principio, um papel mais passivo do que em jogos, mas ainda assim poderia interagir com a história de diversas maneiras.

A literatura apresenta alguns sistemas interativos de *storytelling* que tentam atingir as expectativas de um sistema de geração automática de histórias (CAVAZZA; CHARLES; MEAD, 2002; MATEAS; STERN, 2000; YOUNG, 2000; SPIERLING et al., 2002; GRASBON; BRAUN, 2001; CIARLINI et al., 2005, 2008). Estes trabalhos utilizam uma sequência de algoritmos de planejamento para produzir roteiros não lineares, os quais mantêm a coerência com um gênero de narrativa. A partir disto pode-se utilizar um motor de dramatização em tempo real para exibir a história gerada. O algoritmo de planejamento apresenta para o usuário algumas possibilidades para a história e, baseada na resposta dada por ele, gera a próxima sequência do roteiro. No final, um sistema interativo de *storytelling* produz um roteiro que pode ser utilizado como entrada para um sistema de dramatização para mídia digital.

Diversos conceitos de *storytelling* foram abordados visando prover aplicativos multimídia específicos (CHARLES et al., 2002; CIARLINI et al., 2005; DORIA; CIARLINI; ANDREATTA, 2008). A maioria das abordagens é baseada na visualização tridimensional de um roteiro pré-definido. Frequentemente, estas visualizações apresentam um tempo de dramatização bastante reduzido. Isto ocorre devido à construção de enredos muito simplificados por parte dos geradores de história. Estes roteiros possuem somente alguns eventos de alto nível que representam a essência da narrativa, mas não detalham como ela deve ser dramatizada.

O processo inverso ocorre na mídia de entretenimento atual. Quando um filme, um seriado ou uma peça de teatro é proposta, o diretor recebe um *script* contendo a história a ser dramatizada nos mínimos detalhes. Este *script* contém ações, posicionamento, locais e emoções dos personagens; minúcias descritas pelo redator da história visando que a dramatização final seja o mais próximo de sua imaginação. A figura 1.1 mostra um comparativo entre *scripts* gerados por um roteirista[1] e por um sistema automático.

Considerando uma famosa trilogia do cinema: O Senhor dos Anéis. A série é composta por três filmes adaptados dos livros escritos por John Ronald Reuel Tolkien. Juntos, os três filmes possuem mais de 9 horas de duração. Contudo, se somente observarmos a perspectiva do personagem principal da história, Frodo Bolseiro, o roteiro principal se torna bastante simples. Ao retira-se os personagens e objetivos secundários, a narrativa pode ser escrita como:

[1] Hitchhiker's Guide to the Galaxy - By Douglas Adams and Karey Kirkpatrick. Obtido em www.movie-page.com

He moves to a different rack of instruments, consults a check list, starts flipping some switches.	Go(Ford, Rack);
FORD I mean, why do you want to know the Ultimate Question?	Talk(Ford, "bla bla bla");
ZAPHOD Oh. Well -- partly the curiosity, partly a sense of adventure, but mostly I think it's for the fame and the money.	Talk(Zaphod, "bla bla bla");
He continues preparing the ship. Trillian consults the huge MANUAL, turning dials, flipping switches...	Read(Trillian, Manual);
ARTHUR But -- you're President of the Galaxy, aren't you?	Talk(Arthur, "bla bla bla");
ZAPHOD Yes, Arman.	Talk(Zaphod, "bla bla bla");
ARTHUR Arthur.	Talk(Arthur, "bla bla bla");
ZAPHOD Whatever. Presidential fame is temporary, I find the Question, that's permanent. It sticks. Plus everyone thinks you're deep. Win-win.	Talk(Zaphod, "bla bla bla");
(CONTINUED)	

Figura 1.1: Comparação entre um *script* real e uma história que poderia ser criada por um gerador automático

Frodo recebe o anel de Bilbo;

Gandalf fala com Frodo sobre o anel;

Frodo decide levar o anel a Montanha da Perdição;

Frodo vai até à Montanha da Perdição;

Frodo destrói o anel;

Sauron morre;

Frodo volta para casa.

Pode-se notar que este roteiro possui estrutura semelhante àqueles gerados por algoritmos de planejamento tal como em Ciarlini(2002).

Estender a dramatização da jornada do herói até o seu destino final é uma abordagem comumente utilizada em várias formas de entretenimento. O Senhor dos Anéis é um exemplo de narrativa que se encaixa nesta descrição. Essencialmente, após Frodo aceitar

a missão ele inicia uma viagem através de um mundo muito grande, cheio de perigos e desafios, até atingir o local onde o anel pode ser destruído. Geralmente, a viagem de volta para o ponto de partida não é tão importante e o personagem parece chegar ao seu destino sem esforço. Um dos fatores que tornam esta história interessante é o fato do mundo no qual Frodo viaja ser bastante grande, rico em detalhes e possuir diversos personagens, desafios e sub-objetivos pelo caminho.

Uma abordagem semelhante a esta pode ser utilizada para aumentar o tempo e a qualidade da dramatização de um sistema de *storytelling*. Visando aumentar o apelo de um sistema de *storytelling* propõe-se, neste trabalho, estender o caminho de um personagem por um mundo tridimensional de grandes proporções, possibilitando que novas formas de dramatização sejam desenvolvidas. Com isto, almeja-se tornar os sistemas de *storytelling* cada vez mais próximos do conteúdo de entretenimento tradicional. Este trabalho foi focado especialmente em alternativas de baixa intervenção durante a dramatização, ao mesmo tempo em que a duração da dramatização é estendida. Estas alternativas foram aplicadas na nova versão do sistema interativo de *storytelling* Logtell, o qual tem sido desenvolvido para ser executado em um ambiente de TV digital interativa.

1.1 Objetivos do Trabalho

O principal objetivo deste trabalho é obter uma solução para o problema de baixo tempo de dramatização em sistemas de *storytelling* interativo. Tendo por base que o enredo da história é estabelecido previamente à dramatização e que este é extremamente resumido, anseia-se obter pontos possíveis de estender o tempo e qualidade da história. Ao mesmo tempo, deve-se manter o contexto inicial da história, permitindo que esta seja capaz de entreter um telespectador por um período de tempo considerável.

Para atingir o objetivo principal os seguintes objetivos secundários devem ser realizados:

- Propõe-se uma técnica que estende a dramatização de um evento especifico, o qual é responsável pela movimentação dos personagens pelo cenário (evento *Go*). Tal evento está presente na maioria dos sistemas de *storytelling* e é usualmente utilizado para fazer os personagens se moverem pelo mundo virtual. Para possibilitar o aumento do caminho de um personagem, o cenário onde a narrativa é realizada deve ser adequado. É necessário que este cenário seja grande, forneça recursos à

dramatização, seja rico em detalhes e possua um bom sistema de navegação para os atores.

- Ao lidar-se com mundos amplos, o problema de busca de caminhos se torna evidente. Quando um personagem precisa ir de uma localidade para outra deve existir um caminho que interconecte estes locais. Um dos fatos que motivou a escolha do Senhor dos Anéis como um exemplo para este trabalho é o nível de detalhe do mundo criado por Tolkien. Este nível ajuda a visualizar as essências de um bom cenário tais como: longas distâncias, variedade geológica e diversidade de caminhos possíveis.

- Desenvolver uma estrutura de dados que possa manter um cenário grande para as propostas citadas anteriormente. Além de gerenciar e manter um grande cenário, esta estrutura visa permitir que um usuário possa influenciar o modo como a dramatização ocorre e disponibilizar informações úteis ao sistema de dramatização, possibilitando que câmeras sintéticas se posicionem de forma correta e produzam tomadas que dêem a idéia de transições entre diferentes regiões.

Com a técnica proposta não se pretende resolver unicamente a questão total do baixo tempo de dramatização. Entretanto, acredita-se fortemente que esta pode ser considerada como um passo inicial, tendo em vista que ela permite o desenvolvimento de novas técnicas que possam fazer uso da estrutura criada para gerar novos métodos de aumentar o tempo e qualidade da dramatização.

Visando implementar o sistema de dramatização, foi utilizada a Game Engine Unity3D (UNITY3D, 2009). Ela permitiu testar os algoritmos em um grande cenário e produzir material de apresentação com qualidade. Outra vantagem da utilização da Unity é o fato dela possibilitar a renderização do seu conteúdo junto a navegadores de internet, uma solução que se encaixa perfeitamente em aplicações do tipo *WebTV* e jogos para internet.

1.2 Estrutura do Trabalho

Este trabalho está organizado como segue. O capítulo 2 apresenta trabalhos relacionados e revisão bibliográfica sobre *storytelling* e técnicas de cinematografia. Informações referentes a *GameEngine* utilizada na implementação estão presentes no capítulo 3. No capítulo 4 são tratadas as proposições do trabalho e maneiras de solucionar a questão do

baixo tempo de dramatização. O capítulo 5 discorre sobre a implementação do trabalho e as técnicas utilizadas. Resultados são apresentados no capítulo 6. O capítulo 7 contém as considerações finais e trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Existem diversos trabalhos que propõem técnicas que suportam *storytelling* interativo, as quais utilizam conceitos de diversas áreas tais como: Computação Gráfica, Inteligência Artificial, Ciência Cognitiva, Literatura e Psicologia. A usabilidade de cada alternativa depende do objetivo de cada aplicação. A maioria destas pesquisas pode ser classificada em duas categorias: Baseada em personagem (CAVAZZA; CHARLES; MEAD, 2002; YOUNG, 2000) e baseada em roteiro (SPIERLING et al., 2002; GRASBON; BRAUN, 2001; CIARLINI et al., 2005). Ainda assim, foram propostas abordagens que combinam as duas abordagens (MATEAS; STERN, 2000).

Em um modelo baseado em personagem, a narrativa resulta da interação em tempo real entre agentes virtuais autônomos que incorporam um comportamento deliberativo. Cada agente tem a sua própria conduta e motivações, e, determinados por estas, eles devem interagir com o mundo virtual e outros agentes para atingir seus objetivos. Com esta abordagem, o usuário pode interferir a qualquer momento com a história, modificando objetos no cenário e interagindo com os atores virtuais, possivelmente motivando-os a selecionar alternativas para atingir seus objetivos. Entretanto, como resultado de uma intervenção tão forte fica muito difícil estimar as decisões ou ações que serão tomadas pelos atores virtuais. Logo, o sistema de dramatização não tem o mesmo controle do processo como ocorre no processo de filmagem normal feito por um diretor.

Outra abordagem é o modelo baseado em roteiro, o qual obtém a narrativa em um passo prévio a dramatização. Este passo é baseado em um modelo lógico de um gênero de narrativa e, juntamente com a interação do usuário, gera o roteiro final. Após esse processo inicial, a dramatização do roteiro pode ser realizada. Os personagens devem seguir regras restritas definidas no roteiro. Esta abordagem garante que os atores sigam um script pré-definido de ações as quais são conhecidas a priori, o que limita a intervenção

do usuário.

Sistemas de *storytelling* geralmente possuem duas partes distintas, a geração do roteiro e a dramatização deste. Tendo em vista que este trabalho se baseou na reconstrução do módulo gráfico do sistema de *storytelling* Logtell (POZZER, 2005; CIARLINI et al., 2005), para melhor compreender o processo de dramatização, o processo de criação é descrito na seção 2.1, enquanto processos de câmera e técnicas de cinematografia necessárias para um bom aspecto final da dramatização são apresentados na seção 2.2.

2.1 Geração do Enredo no Logtell

O Logtell é um sistema que encapsula a geração e visualização da história. A geração é baseada na especificação lógica de um modelo de um gênero de história escolhido, onde possíveis ações e objetivos de personagens são descritos. Ele é composto por diversos módulos os quais são apresentados na figura 2.1. Estes módulos são responsáveis pela geração do roteiro, interação e dramatização.

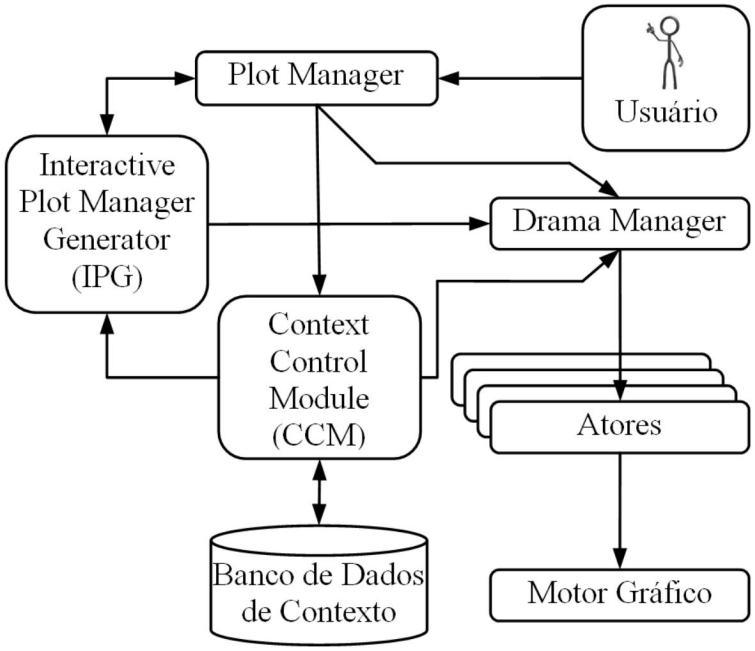


Figura 2.1: Arquitetura do Logtell
Fonte: Ciarlini(2005)

A geração do roteiro é realizada pelo módulo chamado IPG (Interactive Plot Generator) (CIARLINI; VELOSO; FURTADO, 2000). Roteiros são formados por múltiplos ciclos de inferência-objetivo, planejamento e interação do usuário. A interação do usuário é sempre indireta. Durante a simulação, o usuário pode intervir passivamente, somente

deixando os roteiros gerados parcialmente que parecem interessantes serem continuados, ou, em uma maneira mais ativa, tentar forçar a ocorrência de eventos e situações. Estes são rejeitados pelo sistema sempre que não existir uma maneira de reajustar a história para acomodar a intervenção. No contexto de contos de fadas, o qual foi o exemplo utilizado para validar o Logtell, eventos possíveis de serem gerados pelo IPG são: *ReduceProtection, Go, GetStronger, Kidnap, Attack, Fight, Kill, Free e Marry*. Personagens definidos são: os cavaleiros Brian e Hoel (os heróis), a Princesa Marian (a vítima) e Draco (o vilão).

A seguinte história clássica pode ser gerada pelo IPG: "A proteção do castelo de Marian está reduzida. Draco vê isso como uma oportunidade de sequestrá-la. Draco, então, vai ao castelo da princesa o ataca e sequestra a princesa. Brian, sendo um nobre cavaleiro, se sente compelido a salvá-la. Ele vai ao castelo de Draco, o ataca duas vezes e então luta com Draco. Finalmente, Brian mata Draco e liberta Marian, a qual se apaixona por Brian. Motivados pelo seu afeto mútuo, Brian e Marian estabelecem matrimônio."

O Plot Manager se comunica com o IPG para executar a geração do roteiro e manter a coerência, e, juntamente com o Drama Manager controla a dramatização. O Plot Manager contém a interface com o usuário, pela qual o usuário pode participar na escolha dos eventos que irão aparecer no roteiro e decidir a sequência final (figura 2.2).

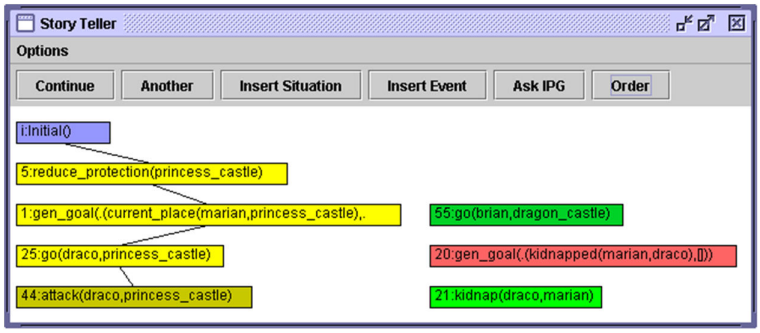


Figura 2.2: Interface Gráfica do Plot Manager

Ao final de cada passo durante a criação da história, a dramatização do roteiro pode ser ativada para exibir o roteiro tanto final quanto parcial. A dramatização é realizada através da utilização de atores 3D, os quais têm suas ações determinadas por planejamento de baixo nível, onde as tarefas envolvendo cada evento são definidas. As figuras 2.3 demonstram o sistema de dramatização do Logtell tal criado por POZZER (2005).

Figura 2.3: Sistema de dramatização do Logtell

2.2 Câmera e Cinematografia

Ao se utilizar conceitos de cinematografia em aplicações de *storytelling* existem duas alternativas mais comuns. A primeira é a utilização de idiomas (CHARLES et al., 2002), os quais representam o modo mais usual de apresentar o tipo de uma cena. A segunda abordagem é a divisão do sistema em diferentes módulos ou agentes que representam vários papéis que as pessoas normalmente desempenham em um estúdio de cinema, tal

como HAWKINS (2004). Existem trabalhos que utilizam as duas abordagens (COURTY et al., 2003). Entretanto, estes trabalhos incorporam apenas superficialmente as regras de cinematografia.

Um trabalho que vem sendo desenvolvido no laboratório LaCA propõe a criação de um diretor para sistemas de *storytelling* baseados em roteiro. O diretor utiliza uma coleção de Support Vector Machines (SVM) (LIMA et al., 2009), treinadas com conhecimento cinematográfico para selecionar, em tempo-real, o melhor aspecto para a dramatização da cena.

Em pesquisas envolvendo cinematografia aplicada a sistemas de *storytelling*, existe uma clara distinção entre técnicas que podem ser aplicadas a modelos baseados em personagem e modelos baseados em roteiro. Modelos baseados em roteiro dão acesso a todas as ações antes do planejamento de câmera, permitindo que o sistema tenha um maior controle das cenas baseados em cinematografia pura. Modelos baseados em personagens não permitem o mesmo nível de controle sobre as cenas, tornando o planejamento de câmera mais complexo devido a dinamicidade do sistema.

O termo cinematografia foi criado na indústria de filmes há muito tempo atrás para descrever o processo de criação de cenas em filmes. Com o avanço da indústria e a emergência de novas tecnologias em vídeo digital com novos formatos de alta definição, o termo está expandido. Atualmente ele é entendido como um termo genérico cobrindo todos os aspectos de configuração de câmera, incluindo aspectos criativos envolvidos em criar imagens esteticamente agradáveis e aspectos técnicos envolvidos com o uso de câmeras, luzes, e outros equipamentos (NEWMAN, 2008).

Mesmo que um filme possa ser avaliado como uma sequência linear de quadros, é bom considerar um filme como possuindo uma estrutura. Em um nível mais elevado um filme é uma sequência de cenas. Cada cena é composta de um número de tomadas, sendo uma tomada a filmagem feita por uma câmera sem interrupções. A transição de uma tomada para outra é conhecida como corte.

O tamanho de uma imagem em um filme é determinado pela distância da câmera e do assunto a ser filmado. Quanto mais perto a câmera está maior é a imagem. Esta distância determina o tipo de tomada. Supondo que o objeto é um personagem, um possível tipo de tomada é uma tomada média, a qual representa o personagem do quadril até a cabeça. Outro exemplo é o *close-up*, o qual apresenta o personagem do peito até acima da cabeça.

O tipo de ângulo de câmera influencia fortemente a forma como a cena é percebida pelos telespectadores. Ele também define como eles irão participar da ação. Quando se escolhe um ângulo mais objetivo, o espectador visualiza o evento como se fosse um observador invisível (MASCELLI, 1998). Um ângulo de câmera subjetivo faz com que o espectador faça parte da cena.

Outro aspecto importante das filmagens corresponde aos movimentos de câmera. Eles afetam as propriedades estéticas e psicológicas da cena. Um exemplo de movimento de câmera é o *tracking*, quando a câmera se move ao lado do personagem enquanto realiza a filmagem, dando aos espectadores o sentimento de estarem andando ao lado do personagem (MASCELLI, 1998). Movimentos devem ser executados de forma a não desorientar os telespectadores.

Cinematografia é um processo complexo, e muitas regras demandam interpretação humana das cenas para serem corretamente aplicadas. Contudo, cinematografistas vêm definindo algumas heurísticas para boas tomadas (ARIJON, 1976). Alguns exemplos são:

- Criar uma linha de interesse: esta linha conecta dois pontos importantes da cena (em geral dois atores interagindo);
- Edição Paralela: Cenas devem alternar entre diferentes contextos, locais e tempos;
- Mostrar somente momentos importantes da história. Movimentos repetidos devem ser eliminados;
- Não cruzar a linha: Uma vez definida uma linha de interesse, a câmera deve, em princípio, se manter deste lado, sem fazer movimentos inesperados. A câmera pode trocar de lado, mas somente com uma tomada que mostra esta transição.
- Deixe o ator guiar: O ator deve iniciar todos os movimentos. A câmera deve iniciar e terminar o seu movimento um pouco após o ator.
- Quebre o movimento: Uma cena ilustrando um movimento deve ser quebrada em no mínimo duas tomadas.

3 UNITY3D

A Unity3D (UNITY3D, 2009) é uma ferramenta de desenvolvimento multi-plataforma criada pra facilitar a criação de jogos. Ela possui diversas características interessantes que facilitam o seu uso para a concepção de diversas aplicações. Ela tem sido utilizada para desenvolver o novo sistema de dramatização do Logtell e testar os conceitos criados sobre expansão do tempo de dramatização. A principal ferramenta da Unity3D e o meio para utilizar todo o seu potencial é o Editor. Através desse editor pode-se:

- arrastar e soltar *Assets* e objetos dentro de uma cena;
- adicionar texturas, som, componentes e *scripts* de variáveis a *Assets* ;
- criar hierarquias de *GameObjects* lógicos;
- customizar o editor através de *scripts*;
- criar *Prefabs*;
- desenhar e editar terrenos.

A Unity3D oferece suporte a uma grande gama de configurações de *hardware/software* tanto em DirectX quanto em OpenGL. Além disso, pode-se utilizar o *Unity3D Web Player Plug-in* para renderizar aplicações feitas em Unity3D em um navegador, recurso este muito interessante para transmissão de conteúdo interativo para TV interativa baseado em *storytelling*. O *plug-in* tem um tamanho bastante reduzido (aproximadamente 3 MB) e funciona na maioria dos navegadores modernos. A aplicação é enviada por *streaming* facilitando a utilização dele em diversas plataformas.

Outra característica bastante interessante da Unity3D é o pipeline gráfico otimizado tanto para DirectX como para OpenGL, o qual gera imagens de alta qualidade gráfica com

rapidez mesmo em *hardware* mais antigo. Deste modo pode-se criar e utilizar malhas 3D animadas, sistemas de partículas, luzes e sombras de alta definição. Luzes otimizadas e sombras em tempo real são funcionalidades que foram exploradas visando estender a qualidade gráfica da aplicação sem necessidade de configurações detalhadas.

A física é processada nativamente pela Unity3D com a utilização da *engine* física Ageia PhysX. Esta engine é a mais avançada atualmente e foi utilizada em jogos como Unreal Tournament 2007 e Ghost Recon3. O tratamento de colisões entre objetos é realizado ao definir se um objeto vai possuir ou não movimentações frequentes e determinar algum volume delimitante. O volume limitante podem ser modelos simples como cubos e esferas ou malhas 3D complexas.

O conteúdo presente na Unity3D é apresentado na forma de *Assets* . Estes incluem modelos 3D, animações, texturas, scripts, shaders, sons, etc. Cada Asset corresponde a um arquivo, o qual é atualizado automaticamente e apresentado no editor a cada modificação.

Um dos recursos mais poderosos da Unity3D é a possibilidade de transformar grupos de *Assets* em *Prefabs*. *Prefabs* são *GameObjects* reusáveis. Eles podem ser facilmente instanciados via editor ou *script*. Mudanças na configuração do *Prefabs* original são propagadas para todos os seus dependentes tornando o seu uso extremamente eficiente e versátil para atualizar um grande número de *GameObjects* de uma só vez.

Juntamente com o editor a Unity3D traz um gerador de terrenos bastante robusto, o qual permite a criação, edição e texturização da malha 3D do terreno. Além disso, ele automaticamente pode povoar o terreno com árvores, pedras, e *billboards* que dão a impressão de grama e arbustos. A qualidade gráfica de um terreno pode ser estendida ainda mais ao se utilizar diversos pincéis para evitar repetições de padrões e calculando mapas de iluminação de todas as luzes direcionas, obtendo, assim, sombras de acordo com a geografia do terreno.

3.1 Desenvolvimento de código

A programação dentro da Unity3D é feita através de *scripts*. As linguagens suportadas são: Java*script*, C#, e um dialeto de Python chamado Boo. Todas as três são igualmente rápidas e operam entre si. *scripts* usualmente são considerados lentos e limitados, mas, segundo a documentação da *GameEngine*, estes são compilados para código nativo e exe-

cutam com desempenho semelhante a C++. Além disso, os *scripts* possuem um bom casamento com o editor possibilitando que variáveis públicas sejam modificadas via interface gráfica do editor o que aumenta consideravelmente o tempo destinado a testes e personalização de objetos.

O código produzido durante a produção deste trabalho foi realizado totalmente em C# através do uso de dois paradigmas diferentes, a Orientação a Objetos, e a Orientação a componentes. A orientação a objetos é um paradigma de programação comum e bastante eficiente, inclusive é o paradigma padrão da linguagem C#. Entretanto, devido à estrutura disponibilizada pela Unity3D uma abordagem voltada à programação orientada a componentes pode ser utilizada quando tratamos cada *script* como um componente independente o qual pode ser adicionado a um *GameObject* e se comunicar com outros componentes deste de forma simples. A próxima seção comenta sucintamente sobre a programação orientada a componentes dentro da Unity3D e como ela foi utilizada durante este trabalho.

3.2 Programação Orientada a Componentes

Componentes podem ser descritos como objetos que se comunicam com outros componentes de uma forma estruturada (ALDRICH; CHAMBERS; NOTKIN, 2001). Em uma visão mais geral, componentes são primeiramente entendidos como unidades de reuso, contudo, na sua forma moderna, são considerados como unidades de extensão. O paradigma de orientação a componentes foi uma evolução dos paradigmas convencionais: estruturados, modulares e orientado a objetos (FROHLICH, 2003). A seguir são apresentadas as formas que os diversos paradigmas definem componentes.

- Em programação estruturada, componentes são operações individuais (processos e funções);
- Em programação modular, componentes são módulos que encapsulam uma coleção de operações relacionadas;
- Em programação orientada a objetos, componentes são unidades binárias, que, novamente, encapsulam uma coleção de operações relacionadas;
- Em programação orientada a componentes, componentes são módulos que encapsulam uma coleção de operações e coleções de tipos.

O paradigma modular e orientado a objetos adotam certos conceitos de programação estruturada e abandonam outros (mantém-se o número limitado de estruturas de controle, mas substituem-se processos por mecanismos de abstração mais avançados). Similarmente, o paradigma orientado a componentes adota conceitos de paradigmas anteriores ao mesmo tempo em que deixam outros para trás. A figura 3.1 ilustra a evolução de paradigmas de programação e clarifica a visão acima.

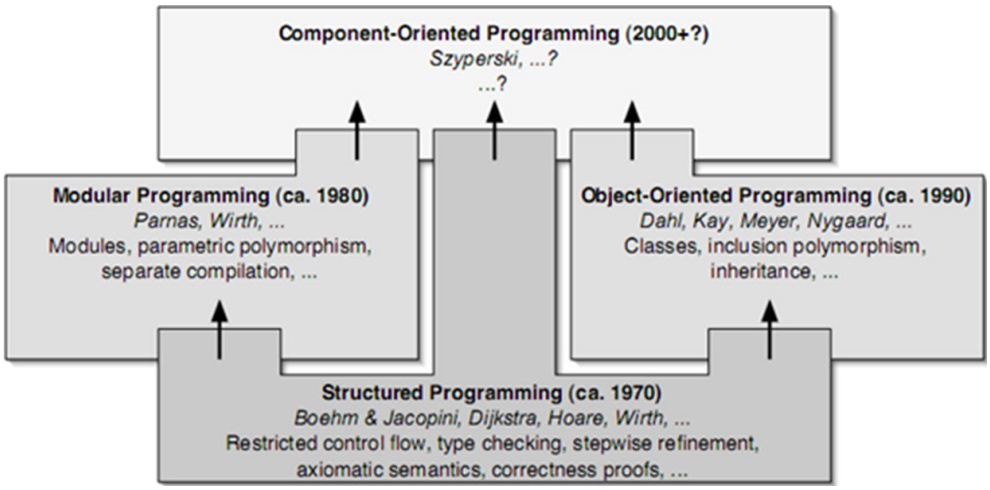


Figura 3.1: Uma visão da evolução dos paradigmas de desenvolvimento. As flechas indicam o fluxo de certos conceitos, datas são aproximadas. (FROHLICH, 2003).

A programação orientada a componentes difere da programação orientada a objetos. Entretanto, elas possuem pontos em comum. Orientação a objetos se foca no relacionamento entre classes que são combinadas em um único binário executável. Orientação a componentes se foca na troca de módulos de código que trabalham independentemente e não requerem familiaridade com a sua forma interna de uso (LOWY, 2003).

Atualmente, nenhuma linguagem de programação foi desenvolvida para este paradigma, (FROHLICH, 2003). Ao invés disso, algumas linguagens que combinam conceitos de programação modular e orientação a objetos são utilizadas na orientação a componentes. Linguagens que se caracterizam neste padrão são: Java, Modula-3, Component Pascal, por exemplo.

3.2.1 Unity3D e componentes

GameObjects são os objetos mais importantes na Unity3D. Todo objeto em uma aplicação Unity3D é um *GameObject*. Eles são basicamente contêineres, que podem manter diferentes componentes. A figura 3.2 apresenta um *GameObject* que possui diversos com-

ponentes, os quais o caracterizam como uma câmera.

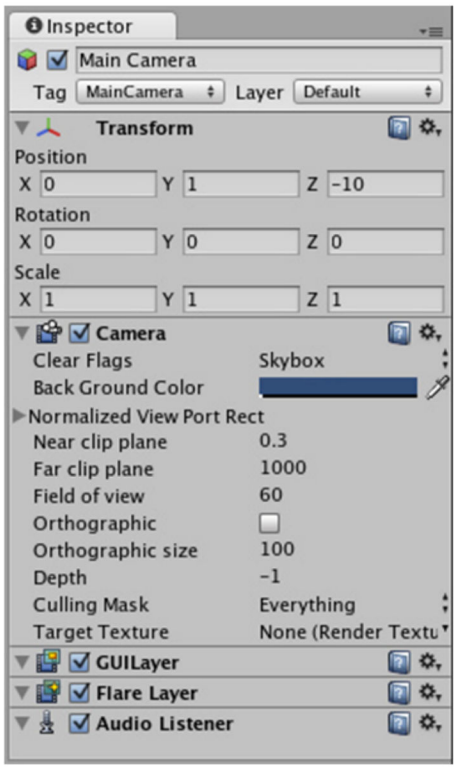


Figura 3.2: Estrutura de um *GameObject* que possui diversos componentes.

Componentes definem objetos e comportamentos, e podem ser adicionados a *GameObjects* em diversas combinações. Alguns componentes, inclusive, funcionam melhor em combinação com outros. Por exemplo, um componente Rigidbody, o qual realiza cálculos físicos de um objeto que se move em uma cena, necessita de um componente do tipo Collider, o qual define um volume limitante que será utilizado na verificação de colisões. Um componente pode incluir referências à outro componente, arquivo, ou *GameObject*. O processo de referência é feito via editor, simplesmente arrastando e soltando o objeto a ser referenciado. Este processo pode ser visto na figura 3.3.

Scripts são considerados componentes. Ao se criar um *script* e adicioná-lo a um *GameObject*, o *script* aparece junto a este *GameObject* no editor. Em termos técnicos, um *script* é compilado como sendo do tipo componente, e é tratado como qualquer outro componente presente na engine. Qualquer variável pública declarada no *script* poderá ser editada via GUI do editor e qualquer outro componente que também estiver conectado ao mesmo *GameObject* do *script* pode ser acessado via *script*.

Deste modo o modelo utilizado pela Unity3D produz resultados rápidos que podem ser reutilizados com muita facilidade. A configuração de *GameObjects* através de compo-

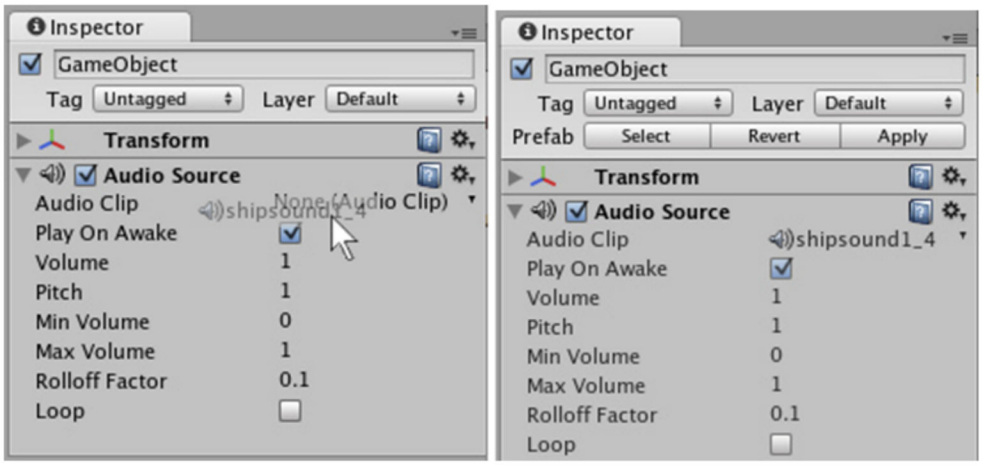


Figura 3.3: Adição de referência a *Assets* via editor.

nentes permite uma expansão rápida em funcionalidade de uma maneira bastante simples. Por mais que não seja um modelo puramente orientado a componentes a Unity3D pode ser considerada um *framework* para a criação, edição e manipulação de componentes.

4 PROPOSIÇÃO DO TRABALHO

Storytelling tem sido bastante estudado atualmente e pode contribuir com diversos conceitos para a geração de conteúdo para entretenimento digital. Um dos pontos mais importantes para que aplicações realmente interessantes possam ser desenvolvidas utilizando esta técnica reside no tempo de duração da dramatização da história gerada.

Considerando o modelo baseado em roteiro (descrito no capítulo 2), a história gerada possui um encadeamento lógico e mantém um contexto único e adequado ao gênero de história, contudo, ao dramatizá-la o resultado final não dura nem uma dezena de minutos (POZZER, 2005). Para solucionar esta questão, neste capítulo, serão propostos métodos que objetivam estender o tempo e qualidade da dramatização de roteiros gerados pelo IPG.

As próximas seções irão tratar da expansão do cenário e como este fato contribui para o melhoramento da dramatização, da estrutura desenvolvida para manipular um cenário com características definidas, do método concebido para realizar a navegação dos personagens e de possibilidades de interação por parte dos usuários.

4.1 Expansão do Cenário

Não é difícil encontrar em filmes e jogos um roteiro que se encaixa na descrição a seguir: *"Algum personagem está vivendo a sua vida normal quando algo estranho ocorre ao seu mundo. Impelido por sua determinação e coragem ele decide salvar o mundo. Ele, então, inicia uma jornada de perigos incontáveis através de um mundo devastado e descansa somente quando chega ao seu objetivo. Uma vez lá ele luta contra o mal e vence. Tendo salvado o mundo ele retorna a sua terra natal e o mundo se mantém em paz novamente"*. Este roteiro se assemelha a, por exemplo: *O Senhor dos Anéis, O Código Da Vinci, The Legend of Zelda, Super Mario World* e *Full Throttle*.

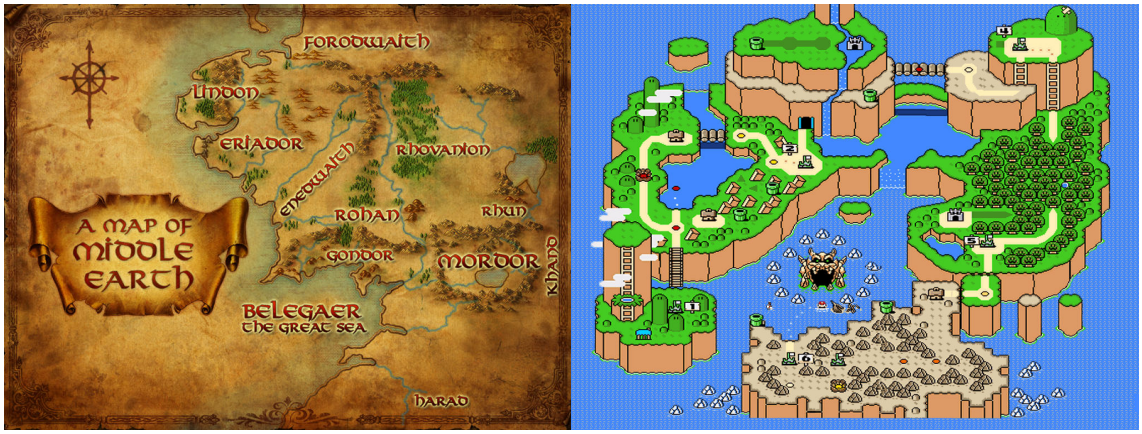
Além deste roteiro especifico muitas outras narrativas podem adaptar este tipo de movimento em um cenário como parte de seu roteiro. A característica mencionada acima é a nossa motivação para produzir uma técnica que possa ser usada em sistemas de *storytelling* para produzir um conteúdo mais apreciável.

Usualmente, sistemas de *storytelling* possuem um evento que representa a viagem de um personagem em direção a um local específico. Generalizando, este evento irá ser referido como o evento *Go*. Este evento representa a jornada de um personagem de sua posição atual até o seu destino final; ele recebe dois parâmetros: um personagem e um local.

Ao considerar exemplos de filmes e jogos como citados acima se pode perceber que, além de seus personagens principais realizarem jornadas por um cenário grande, todos possuem mundos divididos em áreas com características distintas. A figura 4.1 apresenta alguns cenários que possuem características que vão ao encontro dos conceitos citados. Observando-se a figura notam-se as características distintas, por exemplo, o cenário do Super Mario World para Super Nintendo apresentado na figura 4.1b vai ficando mais assustador e desolado à medida que vai se chegando próximo ao castelo final. No mapa de Hyrule do jogo The Legend of Zelda: Ocarina of Time para Nintendo 64, figura 4.1c, pode-se notar como existem diversas áreas geograficamente distintas, entre elas desertos, lagos, florestas etc. Visando organizar o nosso cenário adotamos uma forma de divisão semelhante. O cenário foi dividido em áreas distintas, as quais são chamadas de regiões; a partir do refinamento destas regiões de acordo com níveis de detalhe mais específicos obtêm-se as sub-regiões. Este conceito é semelhante à organização de mapas políticos, nos quais o mundo é dividido em continentes, estes em países, os quais são compostos por estados ou províncias e assim sucessivamente. A subseção 4.1.1 provê maiores detalhes sobre regiões e sub-regiões.

4.1.1 Regiões

Considerando os exemplos citados na seção anterior pode-se notar que esta diversidade de regiões influencia diretamente a dramatização. Se em O Senhor dos Anéis Frodo tivesse simplesmente andado alguns metros para destruir o anel a maior parte da trama não se desenrolaria. De posse de tamanha variedade de regiões, a narrativa pode incluir várias cenas em cada uma dessas regiões. Como ocorre nos filmes, a maior parte das



(a) (b)

(c)

Figura 4.1: (a) Mapa de O Senhor dos Anéis. Fonte:*In game Screenshot* do jogo The Lord of the Rings Online: Shadows of Angmar. (b) Mapa de Super Mario World. Fonte:*In game Screenshot*. (c) Mapa de The Legend of Zelda: Ocarina of Time. Fonte:*In game Screenshot*.

cenas significativas ocorre em regiões diferentes.

Ao considerar-se a dramatização em *storytelling* utilizando grandes cenários com diversas regiões, uma questão se torna mandatória: transição de cenas entre regiões. Cada vez que um personagem entra ou sai de uma região, ou ainda, quando um corte de cena entre diferentes regiões inicia, atenção deve ser tomada para produzir um resultado coerente. Existe uma diversidade de conceitos cinematográficos por trás das técnicas de câmera que podem ser utilizadas para tornar essas transições suaves (HE; COHEN; SALESIN, 1996; HALPER; HELBING; STROTHOTTE, 2001). Conceitos de câmeras não são o foco deste trabalho, mas devemos considerar o seu resultado, já que o modo como

a dramatização é apresentada para a audiência tem impacto na sua aceitação. Consequentemente, regiões não são apenas um bom modo de se referenciar uma localidade, mas também uma estrutura lógica que provê informações úteis à dramatização, como apresentado por AZEVEDO et al. (2008).

Uma característica que pode ser observada quando se considera cenários gigantescos é a diversidade de caminhos possíveis. Sempre que um personagem necessita ir de um local a outro o número de caminhos possíveis cresce exponencialmente de acordo com o tamanho do cenário. Este aspecto revela quão importante é a busca de caminhos para atingir uma dramatização interessante. Em geral a busca por um caminho entre dois pontos recai na teoria de grafos para a obtenção do menor caminho, contudo, neste caso o menor caminho não é o fator essencial. Aspirando aumentar a qualidade da dramatização e introduzindo um nível de interação foi desenvolvido um método de busca de caminhos que leva em conta não somente a distância ou a dificuldade de se chegar a um determinado ponto, mas sim características específicas dos locais que os atores virtuais irão percorrer de acordo com critérios selecionados por um usuário interativo. Este assunto será expandido na seção 4.3.

Visando permitir uma maior interação do usuário propõe-se explicitar algumas características de cada região. Desta forma uma região deve possuir atributos que quantificam algumas características específicas. Isso permite que um usuário possa influenciar o processo de dramatização de acordo com a seleção desses atributos. O processo de interação com o usuário é tratado na seção 4.4.

Cada região possui alguns atributos associados a ela que a caracterizam e são utilizados para selecionar caminhos de acordo com as preferências do usuário, tal como apresentado nos parágrafos anteriores.

O grupo de atributos das regiões é fixo, mas pode ser expandido ou modificado com pequenos ajustes na interface do usuário. Na implementação corrente foram definidos quatro atributos:

- Beleza: O quão atrativa visualmente a região é. Uma queda d'água pode ser considerara uma região bonita, ao contrário de uma cidade abandonada.

- Terror: O quão assustador a região parece. Uma caverna sombria deve possuir valores baixos.

- Segurança: O quão segura uma região é. O castelo da princesa pode ser considerado um local seguro, contrastando com regiões habitadas por ladrões e assassinos.

- Densidade Populacional: O quão populosa é a região. O mercado central de uma cidade receberia um valor baixo para representar uma alta densidade.

Considerando o *script* que resume a narrativa do Senhor dos Anéis apresentada no capítulo 1. A dramatização do evento *"Frodo vai à montanha da Perdição"* fosse estendida ao considerar um grande cenário, pode-se perceber que a história ainda manteria o seu significado e possuiria duração maior. A duração total, comparada com o filme original, ainda seria reduzida, mas superior a dramatizada em um cenário pequeno.

Para por em prática estes conceitos, algum tipo de estrutura deve ser utilizada para manter os dados necessários para a aplicação. Devido ao grande número de caminhos possíveis, a estrutura deve possuir características de um grafo orientado. Além disso, a estrutura deve representar corretamente a relação hierárquica entre regiões e sub-regiões. Dentre as estruturas de dados presentes na literatura não se encontrou uma que atendesse todos os requisitos acima, então se concebeu uma estrutura que recebeu o nome de Treeph e será tratada na seção 4.2

4.2 Treeph

Um grande cenário traz uma grande malha 3D a ser considerada, por conseguinte uma expansão exponencial no número de caminhos possíveis e também a possibilidade de se enriquecer a dramatização. A estrutura Treeph (Tree + Graph) almeja lidar com todas estas questões, ao tomar vantagens de características de árvores e grafos.

A Treeph é um grafo direcional multi-nível definido por $G = (V,A)$ onde V é um conjundo de vértices. Cada vértice de V pode ser definido recursivamente por um sub-grafo direcional multí-nivel. A é um conjunto de pares ordenados de vértices que correspondem aos arcos de adjacência.

A estrutura de dados Treeph foi utilizada para representar um cenário em seus diferentes níveis de detalhe, tal como apresentado na seção 4.1.1. A conexão entre os diferentes níveis é dada pela hierarquia da árvore, enquanto a adjacência entre nós de uma mesma região em um mesmo nível de detalhe é dada pelos arcos de uma estrutura de grafo.

Com a árvore, explora-se a característica hierárquica e o modo simples de inserir e remover nós. Além disso, ela permite o uso de diversos algoritmos de busca e possui

diferentes métodos de percorrê-la, os quais podem ser utilizados para encontrar a relação parental entre nós de diferentes níveis.

As características de grafo dessa estrutura provêm uma organização que mantém a adjacência entre nós de um mesmo nível. Essa propriedade é utilizada para relacionar regiões que estão conectadas no cenário.

A figura 4.2 demonstra uma representação esquemática de um mapa político aplicada a uma Treeph. Pode-se notar que a estrutura tem uma grande similaridade com uma árvore, mas as características de um grafo são facilmente notadas. As caixas representam os nós da Treeph, os quais são rotulados de acordo com as regiões correspondentes. Nós Pretos representam as regiões de mais alto nível presentes no globo terrestre tais como: América Latina, Ásia e Europa. Nós azuis denotam as subdivisões dessas regiões em países. Brasil, Colômbia, México e Panamá são sub-regiões da América Latina. Nós verdes são mais um nível de detalhamento da região do Brasil e correspondem aos estados do Amazonas, Pará e Tocantins. Linhas tracejadas indicam relação parental de nós de diferentes níveis. Linhas sólidas representam arcos de adjacência entre nós de um mesmo nível. A figura 4.3 apresenta uma melhor visualização do formato esquemático em três dimensões da estrutura.

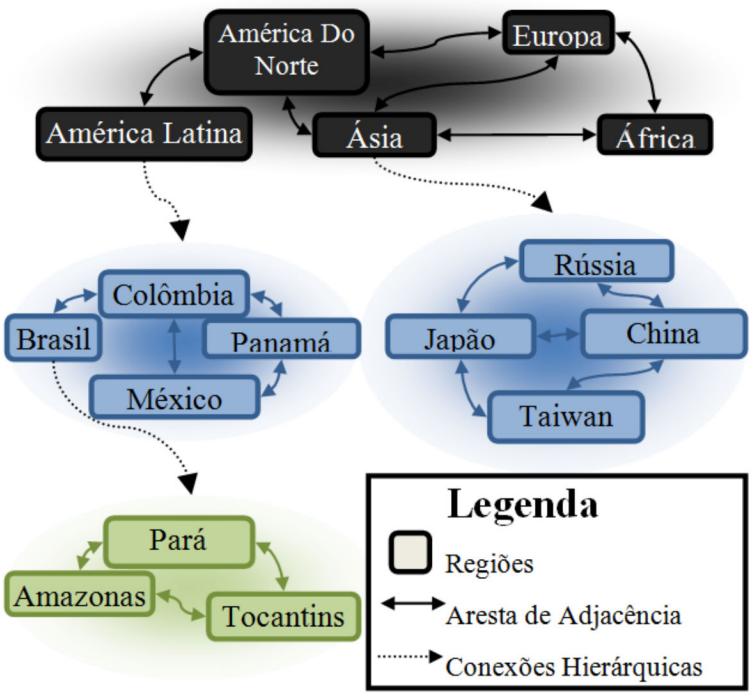


Figura 4.2: Representação de uma Treeph contendo as regiões do globo terrestre. Obs: Representação ilustrativa. Conceitos geográficos reais não se aplicam.

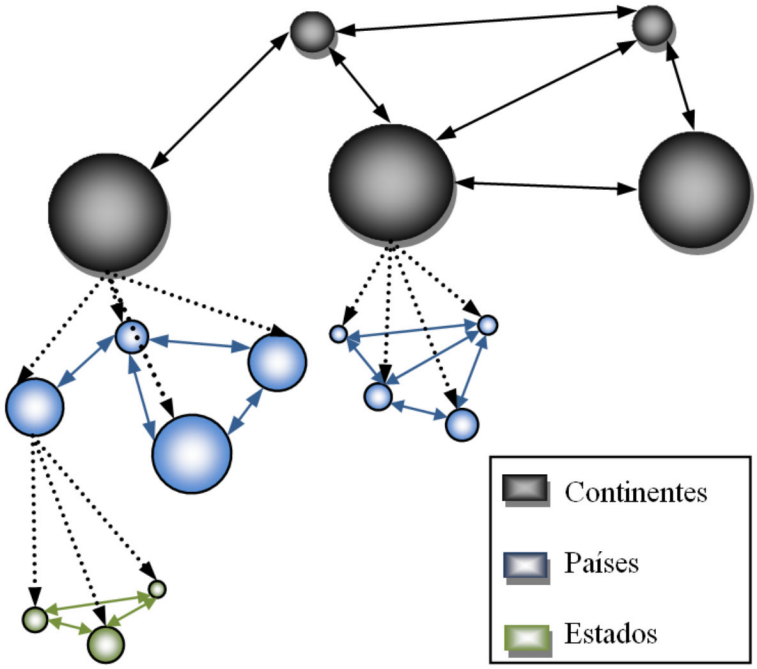


Figura 4.3: Representação tridimensional da Treeph da figura 4.2

A adjacência entre dois nós em um mesmo nível representa regiões no cenário que estão conectadas por um caminho. A associação entre nós e seus filhos representa regiões subdivididas em regiões menores.

Considerar regiões em diferentes níveis de abstração permite a criação de regiões com características distintas. O Japão é uma região de clima temperado e apresenta uma vegetação com predominância de florestas temperadas, contrastando com a Rússia, a qual possui um clima subpolar com vegetação predominante de taiga e florestas boreais. Entretanto, elas pertencem à mesma área de alto nível, a Ásia. É importante mencionar que na implementação corrente, cada região de alto nível consiste em uma malha 3D específica, a qual é carregada em memória a medida que o personagem se move pelo cenário.

Nós-folha representam as regiões mais especializadas, e denotam áreas que mantêm características distintas. Elas são ligadas às malhas 3D dos cenários onde a dramatização ocorre. Consequentemente, elas devem manter informações para ajudar os atores nas suas andanças por ele. Essa informação é mantida na forma de *waypoints* - objetos associados a coordenadas espaciais do cenário que mantém alguma informação relacionada com esta localização.

Um grafo de *waypoints* conectado representa os caminhos possíveis, e é utilizado para guiar os atores através do cenário, percorrendo regiões e sub-regiões. Se os atores fossem simplesmente posicionados no mundo e providos da direção do seu alvo, eles pos-

sivelmente nunca atingiriam seu destino; eles poderiam acabar impedidos de prosseguir por alguma barreira geográfica ou simplesmente vagar sem propósito. Para resolver isto, é necessário um sistema de navegação que possa mapear as regiões que um ator pode se mover, e prover um caminho contínuo de um ponto a outro. Logo, ao posicionar-se grande quantidade de *waypoints* em um cenário cria-se uma malha que ajuda o sistema de navegação a dirigir atores pelo cenário. Portanto, a região que mantém os *waypoints* carrega um grafo orientado destes.

Além de serem marcadores no cenário, *waypoints* podem denotar alguns atributos especiais em suas regiões. Existem alguns casos onde um caminho através de duas regiões diferentes é contínuo, e um *waypoint* pertencente a um nó-folha pode se conectar a outro de uma diferente região. Contudo, existem algumas regiões vizinhas que possuem uma conexão descontínua (duas malhas 3D diferentes). Deste modo, *waypoints* podem incluir atributos que fornecem esse tipo de informação ao sistema de dramatização. Quando uma situação de descontinuidade é detectada o ator é reposicionado na sua próxima posição ao invés de filmar a sua viagem através de uma área descontínua.

Visando melhor organizar a informação detalhada sobre um cenário, os *waypoints* foram especializados em três tipos:

1. O primeiro tipo de *waypoint* somente mantém informação sobre uma posição no cenário e sobre suas adjacências. Este é o tipo mais comum e é utilizado para conectar localidades comuns no cenário. Qualquer caminho que percorra somente dentro de uma mesma região folha irá possuir somente este tipo de *waypoint*.

2. O Segundo tipo de *waypoint* é necessário nas bordas entre regiões adjacentes. Além de manter a posição absoluta dessa localidade, ele também marca a borda entre regiões dentro de uma mesma malha 3D. Duas regiões com uma aresta de adjacência possuem este tipo de *waypoint* marcando-a.

3. O terceiro tipo é utilizado para denotar o limite entre duas regiões que consistem de duas malhas 3D diferentes. Este tipo de *waypoint* é muito importante, pois ele sinaliza uma aresta entre regiões descontínuas, onde um ator deve ser reposicionado e um cenário diferente deve ser carregado em memória.

A figura 4.4 mostra um cenário onde os três tipos de *waypoints* estão posicionados. Iremos identificá-los como: *waypoints* Regulares, *waypoints* IntraMalha, e *waypoints* In-

terMalha. Na figura 4.4 os dois círculos vermelhos delimitados por linhas contínuas representam duas sub-regiões do Brasil, a Amazônia, mais a esquerda, e Pará, círculo central. Cada região é composta por algumas sub-regiões, delimitadas pelas linhas tracejadas. A região central possui, inclusive, mais um nível de detalhamento, o qual está apresentado no círculo mais a direita. Os círculos brancos, pretos e cinza ligados entre si são *waypoints* e só estão dispostos somente nos nós-folhas. Pode-se visualizar este fato pela área sem *waypoints* na região central. Os *waypoints* brancos correspondem ao tipo Regular, os pretos ao tipo IntraMalha e sempre aparecem em duplas pois todos os caminhos são de mão dupla. Os nós cinza estão presentes somente nas bordas da malha tridimensional e representam os *waypoints* InterMalha, sempre que um personagem chegar a um *waypoint* deste tipo ele deve ser reposicionado em outra malha.

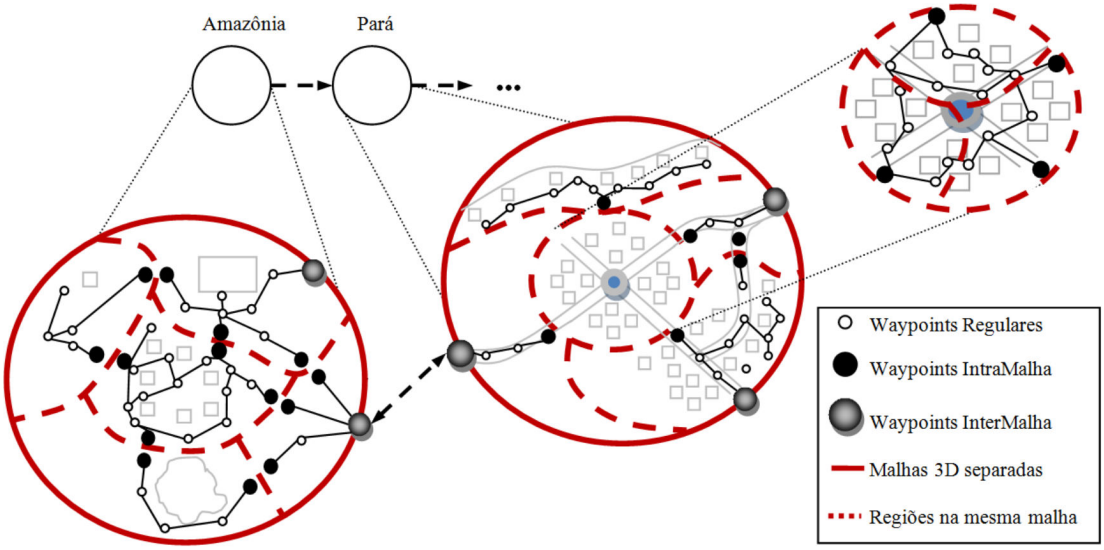


Figura 4.4: Ilustração dos tipos de *waypoints*.

Ao mesmo tempo em que alimenta o sistema de busca de caminhos com informações sobre possíveis localizações e conexões entre regiões, *waypoints* mantêm informação útil para um agente de câmera. Este tipo de conduta permite que um diretor de filmagem sugira movimentos para um ator, o que é comumente encontrado na indústria de filmes. Quando um ator está preste a sair de um determinado local, como por exemplo, uma vila, a câmera faz uma tomada do tipo fading-shot do ator e seu caminho à frente. Por outro lado, se um ator está chegando a uma localidade, a câmera filma a entrada na Vila e segue o ator no seu caminho. Tais decisões podem utilizar informações presentes nos *waypoints* para tomar as decisões entre as escolhas possíveis.

4.3 Planejamento de Caminhos

Um dos desafios de se estender o tempo de dramatização em *storytelling* interativo ao se utilizar grandes cenários é o desenvolvimento de algoritmos de busca de caminhos. Quando agentes autônomos são colocados em um mundo virtual e possuem um roteiro para seguir, deve-se notar que estes agentes necessitam ser alimentados com informação suficiente sobre o ambiente que os cerca visando atingir seus objetivos. Do contrário, eles poderiam vagar pelo cenário sem pistas de como atingir seus alvos. Essa é a razão pela qual um sistema de busca de caminhos é fundamental pra o sistema de dramatização.

A Treeph é uma estrutura multi-nível bastante complexa. Consequentemente, projetar algoritmos de busca de caminhos para ela não é trivial. Foi desenvolvido um método que percorre a estrutura e extrai um caminho, o qual é composto por *waypoints* que um ator deve seguir para chegar ao seu destino.

O problema do planejamento de caminho tratado aqui é similar ao problema do par simples (TARJAN, 1983), o qual consiste em obter o menor caminho a partir de uma determinada fonte *s* para um dado destino *t*. O caminho mais curto pode ser definido como um caminho *p* do nó *s* para o nó *t* onde a soma do comprimento das arestas em *p* é mínimo. Para realizar a busca de caminhos serão utilizados uma composição de dois algoritmos de busca de caminhos em grafos: Dijkstra(seção 4.3.1) e A*(seção 4.3.3).

4.3.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra foi concebido pelo cientista da computação Edsger Dijkstra em 1959 (DIJKSTRA, 1959). Ele é um algoritmo de busca em grafos que resolve o problema do par simples para um grafo que não possua valores negativos como pesos para as arestas, produzindo o a árvore de menor caminho.

A busca pelo menor caminho é conduzida pela Treeph e percorre a estrutura em seus diversos níveis. Ela inicia no nível mais alto e continua recursivamente até atingir os nós-folha, onde estão localizados os *waypoints*. A cada nível, ela considera a propriedade de grafos presente nela e utiliza o algoritmo de Dijkstra para achar um caminho através das regiões presentes nesse nível. O algoritmo de Dijkstra produz sequências de nós que representam o curso para guiar o ator por estas regiões.

Uma vantagem deste algoritmo é que, mesmo com a sua complexidade original de O(n^2), (DIJKSTRA, 1959), se usarmos uma d-heap para manter os vértices rotulados,

onde:

$$d = (2 + \frac{m}{n})$$

e *m* é o número de arestas e *n* é o número de nós, o tempo de execução desta implementação, como definida por (JOHNSON, 1977), é:

$$O(m \log_d n)$$

Uma questão importante ligada à utilização do algoritmo de Dijkstra é o fato dele exigir um grafo com valores não negativos para trabalhar. Tendo em vista que as regiões não folhas não possuem ligação com o cenário, sendo apenas estruturas lógicas que denotam um nível hierárquico, a questão dos pesos das arestas é definido na próxima seção.

4.3.2 Atributos e Peso das Arestas

Regiões são estruturas que possuem características distintas distribuídas de maneira hierárquica pela Treeph. Estes atributos são de suma importância para que a dramatização possa se adequar às preferências dos usuários. Cada região possui valores que determinam quão bela, aterrorizante, segura e povoada ela é e estes valores devem ser usados, juntamente com alguma noção de distâncias, para obter o peso das arestas que interconectam dois nós adjacentes.

Para tal buscou-se um modo de calcular o peso de uma aresta de acordo com os atributos que os nós possuem. Caso um usuário tenha preferência por regiões belas, o caminho obtido pelo algoritmo de Dijkstra deve possuir uma quantidade grande de nós com esta característica. Inicialmente a técnica utilizada para obter o custo para ir de uma região a outra era a utilização do valor do atributo do nó que ser quer ir. A figura 4.5a apresenta esta abordagem. Existem três regiões no grafo, sendo o nó azul o nó inicial e o nó verde o nó final. Considerando a preferência do usuário por regiões belas o nó azul seria uma região de beleza mediana, o nó verde já estaria mais próximo a uma região feia enquanto o nó preto determinaria uma região de extremo apelo visual. Se o peso da aresta for considerado como o valor do próximo nó o caminho escolhido seria azul=>verde com peso total de 0,6 ao invés de azul=>preto=>verde com peso total de 0,7.

A técnica descrita acima determina um caminho com certa eficiência, entretanto o resultado ideal para este caso seria a passagem pelo nó preto, o qual possui atributos que

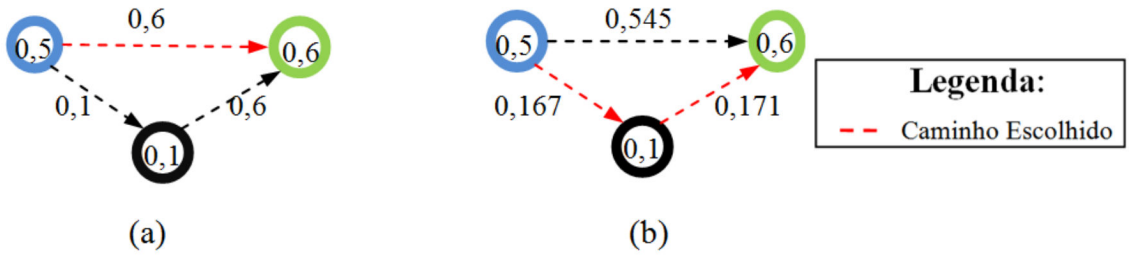


Figura 4.5: Abordagens diferentes para a seleção de caminhos considerando o nó azul como inicial e verde como final. (a) Peso da aresta equivale ao atributo do nó destino. (b) Peso da aresta equivale a média harmônica entre os atributos de ambos os nós

melhor qualificam as preferências do usuário. Deste modo outra abordagem foi necessária. A partir de testes de caso chegou-se a conclusão que a utilização da média harmônica para o cálculo do peso da aresta melhor se adequava para as necessidades da aplicação. A média harmônica foi escolhida ao invés da media aritmética devido a valores pequenos terem uma maior influência sobre valores maiores. A figura 4.5b apresenta os valores da média harmônica para o caso descrito acima. Percebe-se que o caminho escolhido pelo algoritmo de Dijkstra foi um caminho que levava em conta as opções do usuário.

Apesar do algoritmo de Dijkstra encontrar um caminho satisfatório entre as regiões, um ator deve seguir *waypoints*, os quais estão presentes somente nos nós-folhas da Treeph. Consequentemente, o algoritmo deve ser executado recursivamente para cada região obtida pela avaliação prévia do algoritmo de Dijkstra. Este ciclo de ações continua até um nó-folha ser atingido. Quando um nó-folha é encontrado, não existem mais regiões filhas para uma chamada recursiva do algoritmo de busca de Dijkstra. Contudo, o nó-folha ainda possui um grafo de *waypoints*. Neste ponto, o algoritmo de Dijkstra não é mais aplicado, sendo substituído pelo algoritmo A* (A estrela) para buscar um caminho entre os *waypoints*. A troca para o algoritmo A* é realizada devido aos *waypoints* representarem uma posição mais precisa, o que é benéfico para a heurística de distância-mais-custo do algoritmo A*. A heurística foi considerada como:

$$f(x) = g(x) + h(x)$$

onde $g(x)$ é a função custo-do-caminho e $h(x)$ é a heurística-admissível a qual representa um segmento de reta onde o seu comprimento equivale à distância em linha reta de um *waypoint* até outro, tendo em vista que esta é a menor distância possível entre dois pontos.

O algoritmo A* se adapta muito bem ao grafo de *waypoints*, contudo, visando adicionar certa taxa de não determinismo na obtenção de caminhos, foi feita uma pequena

modificação no algoritmo. O não determinismo é um ponto favorável sempre que se lida com questões subjetivas e visa possibilitar que um espectador assista à dramatização de uma mesma história, mas com pequenas diferenças no processo de dramatização. A próxima subseção detalha o processo de adição do não determinismo ao algoritmo A*.

4.3.3 A* Estocástico

O A* é um algoritmo rápido que encontra o menor caminho entre dois nós, sempre que possível. Ele é um algoritmo determinístico: dados duas rotas com o mesmo peso, ele sempre escolhe o mesmo caminho. Neste trabalho, objetiva-se explorar o cenário para dar ao usuário um maior número de possibilidades de visualização para a mesma operação. Portanto, desenvolveu-se um A* estocástico.

O algoritmo consiste na adição de um valor aleatório para o peso das arestas de um grafo de *waypoints* visando quebrar o seu determinismo. Adicionando-se um valor de $\pm 20\%$ de seu peso à aresta obtemos um caminho que cruza uma região de forma consistente e ao mesmo tempo, não determinística. A função heurística do A* foi mantida sem modificações.

4.3.4 Busca Recursiva

O processo de busca pelas diversas regiões é recursivo, e é iniciado no nível mais alto e vai descendo até as folhas. Um ponto importante desta abordagem é o modo de manter a coerência entre as sub-regiões que serão utilizadas no próximo passo recursivo do algoritmo. Por exemplo, ao se realizar uma busca de um caminho que leve um personagem, o qual está no continente (região de nível mais alto) Ásia para o continente da América do Sul, o resultado obtido é: Ásia => América do Norte => América do Sul (de acordo com a análise da figura 4.2). Após este passo o processo de recursão é iniciado e uma busca entre as sub-regiões da Ásia é iniciada. Ao se observar a figura 4.2, percebe-se que a Ásia possui quatro sub-regiões, mas não existe informação que diga o nó inicial e o nó final utilizados na busca do caminho para estas sub-regiões. Neste caso o nó inicial se refere à região em que o personagem em questão está posicionado e o nó final corresponde à região que possui a aresta que conecta a Ásia com a América do Norte. Estas informações podem ser obtidas respectivamente pela Tabela de Referências (subseção 4.3.5) e Tabela de Arestas (subseção 4.3.6).

4.3.5 Tabela de Referências

O sistema de geração de roteiro (IPG) gera uma série de nomes que se referem a locais e personagens. Para que o sistema de dramatização possa atuar de acordo com estes nomes é necessária uma conexão entre o nome e o objeto presente no mundo virtual. Neste caso os personagens e locais devem estar associados a uma posição específica do cenário. As posições são associadas através de *waypoints*. Tendo por base que personagens somente irão se mover por caminhos preestabelecidos, os quais são formados por *waypoints*, pode-se afirmar que um personagem sempre vai estar em um *waypoint* ou indo em direção a um. Para criar a relação entre nomes e *waypoints* foi concebida a tabela de referências.

Se o algoritmo de planejamento produz *"Frodo vai à montanha da Perdição"* (vide capítulo 1) o sistema de planejamento de caminhos deve possuir uma referência para a *Montanha da Perdição*; tal referência é criada ao se adicionar uma entrada na tabela de referência durante o processo de design do cenário. Esta referência irá associar o nome *Montanha da Perdição* com o *waypoint* que marca a sua posição no mundo. Esta tabela também mantém a posição atual do personagem *Frodo*. Consequentemente, cada vez que ele se mover a sua entrada na tabela deve ser atualizada.

A partir desta tabela pode-se obter o *waypoint* de cada personagem ou local necessário pelo sistema de *storytelling*. Entretanto, ao se realizar uma busca é necessário se obter qual região possui o *waypoint* em questão. Para tal pode-se escalar a Treeph de baixo para cima, a partir do *waypoint*, e obter a região no nível desejado. Toda região e *waypoint* possui uma referência para a sua região pai, deste modo pode-se obter a região que possui o *waypoint* fornecido pela verificação da Tabela de Referências.

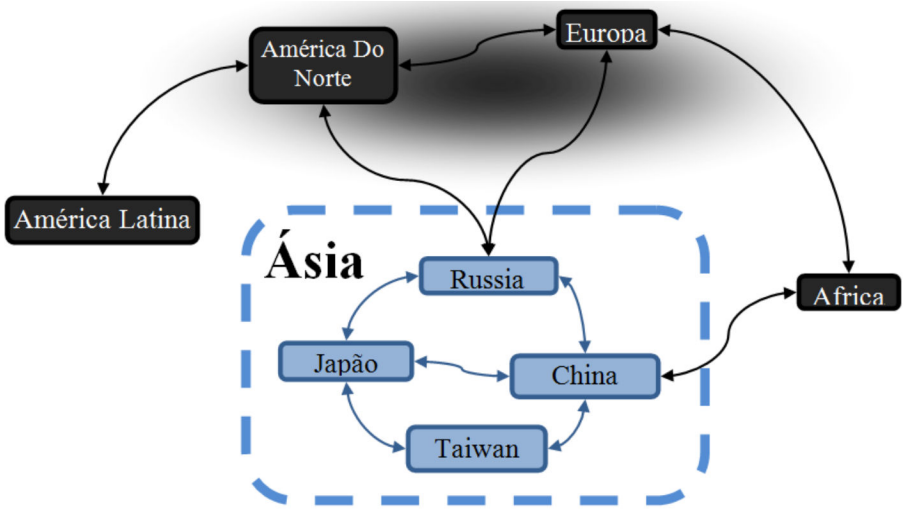
4.3.6 Tabela de Arestas

Ao se realizar uma busca o processo recursivo é iniciado e necessita das sub-regiões filhas que possuam a aresta que conecta a região pai com outra região do mesmo nível. Para obter estas regiões uma sub-região deve possuir informações sobre a adjacência de seus pais.

A figura 4.6a ilustra um único nível da Treeph (um grafo). Este grafo está expandido em 4.6b, onde a região Ásia está detalhada mostrando seus nós filhos e as arestas que estão rearranjadas em cada filho, demonstrando qual sub-região está conectada com qual região de mesmo nível da região pai. No caso citado, a Rússia (nível 1 na Treeph) está

conectada com a América do Norte e a Europa (níveis 0 na Treeph), o mesmo ocorre com a China possuindo adjacência paterna com a África [1].

(a)



(b)

Figura 4.6: Ilustração da adjacência parental nos filhos da região Ásia. (a) Grafo de nível mais alto. (b) Demonstração dos filhos da região Ásia e a sua relação de adjacência com o nível superior.

Considerando que cada região possui informações sobre a adjacência de sua região pai é necessário criar-se uma estrutura que permita o fácil acesso a estas informações em tempo de execução. Para solucionar esta questão foi desenvolvida a Tabela de Arestas, a qual consiste em uma tabela para cada sub-grafo presente na Treeph. A tabela possui dimensão *n*x*n*, onde *n* é o número de nós presentes no grafo em questão. A tabela é semelhante à matriz de adjacências utilizada em teoria de grafos. Considerando a Figura 4.2 esta possuiria quatro tabelas de arestas, uma representando os continentes (apresentada na Tabela 4.1), uma os países da América Latina, uma os países da Ásia e uma os estados do Brasil.

Ao considerar a tabela 4.1[2] pode-se verificar qual região filha se conecta com uma

[1]OBS: Conceitos geográficos reais não se aplicam a este exemplo

[2]OBS: A tabela está mais completa do que a figura 4.2 por motivos estéticos. A Treeph completa produziria uma estrutura muito grande, o que dificultaria a compreensão da figura 4.2

região da adjacência paterna. Na tabela percebe-se que, por exemplo, a Ásia se conecta a América do Norte através da Rússia, enquanto que a América do Norte se conecta a Ásia por meio do Canadá. Desta forma com um acesso simples a esta tabela na forma de

$$tabelaArestas[”Asia”][”AmericadoNorte”]$$

obtêm-se a região filha da Ásia que corresponde a Rússia.

Tabela 4.1: Tabela de arestas correspondendo ao primeiro nível de Treeph.

	África	**América Latina**	**América do Norte**	**Ásia**	**Europa**
África	-	-	-	Egito	Argélia
América Latina	-	-	México	-	-
América do Norte	-	EUA	-	Canadá	EUA
Ásia	China	-	Rússia	-	Rússia
Europa	Itália	-	Espanha	Ucrânia	-

4.3.7 Ligando os Fatos

A figura 4.10 apresenta uma busca sendo realizada na Treeph. A busca corresponde a um evento Go que move um personagem X até o Coliseu. A seguir cada passo que levou a formação desta figura será detalhado.

- Passo 1: A primeira etapa corresponde à obtenção das regiões de nível mais alto para que a busca seja iniciada. Inicialmente possuem-se os seguintes nomes "Personagem X" e "Coliseu"(respectivamente (1) e (2) na figura 4.7). Com um acesso a tabela de referências ((3) na figura 4.7) obtém-se os *waypoints* correspondentes às posições iniciais do "Personagem X" e do "Coliseu". Através do método de escalada da Treeph (descrito na seção 4.3.5) obtemos as Regiões de mais alto nível como sendo respectivamente: América Latina e Europa. A figura 4.7 ilustra esta etapa.

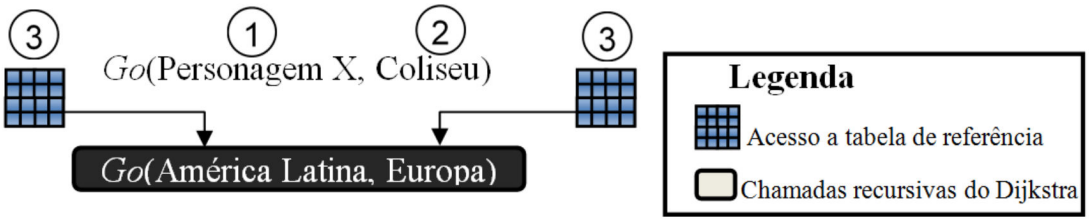


Figura 4.7: Primeira etapa na realização da busca pelo caminho que leva o Personagem X ao Coliseu.

- Passo 2: Aplica-se o Algoritmo de Dijkstra nas regiões obtidos pelo Passo 1 obtendo-se o caminho que liga estas regiões. A figura 4.8 demonstra esta etapa.

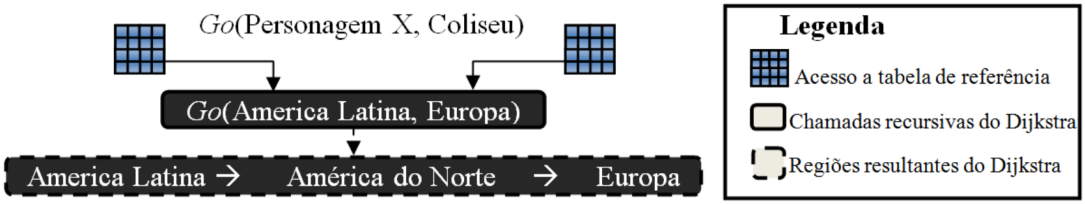


Figura 4.8: Segunda etapa na realização da busca pelo caminho que leva o Personagem X ao Coliseu.

- Passo 3: O processo recursivo é então iniciado com os nós filhos de cada região obtida pelo algoritmo de Dijkstra. Para permitir esta recursão, o nó inicial e o nó final para este nível devem ser obtidos. Isto cria três diferentes instâncias para executar: o primeiro nó, América Latina ((1) na figura 4.9); o nó interno, América do Norte ((2) na figura 4.9); e o último nó, Europa((3) na figura 4.9). Cada um representa um caso diferente e com seus próprios nós iniciais e finais. Os nós internos recaem sobre o caso regular onde o nó inicial é aquele onde a aresta conectando seu pai está localizada. A referência para este nó é obtida através da tabela de arestas((4) na figura 4.9). No caso da América do Norte o nó inicial é a região filha EUA, a qual é a região que está conectada à região América Latina. O nó final é obtido seguindo-se a mesma lógica.

  O primeiro e último nó do caminho são similares, já que ambos possuem somente uma adjacência. No primeiro caso, o nó inicial não existe. O nó final pode ser obtido pelo método regular; o mesmo se aplica ao ultimo nó, mas referindo-se ao nó inicial. Os nós ainda não encontrados são obtidos ao realizar-se a verificação na tabela de referências, ao invés de buscar na tabela de arestas. A figura 4.9 apresenta os conceitos citados.

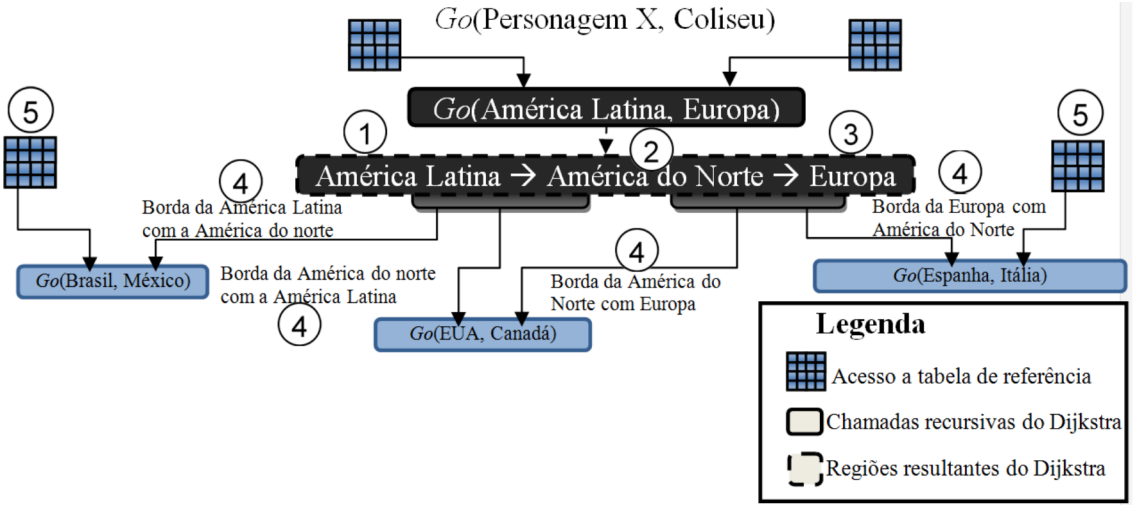


Figura 4.9: Terceira etapa na realização da busca pelo caminho que leva o Personagem X ao Coliseu.

- Passo 4: Para finalizar a busca na árvore o passo 3 é repetido recursivamente até que sejam atingidos os nós-folha ((1) na figura 4.10). Encerra-se assim o processo de busca ao retornar-se um caminho composto por *waypoints*, os quais serão seguidos pelos atores na suas viagens pelo mundo. A figura 4.10 apresenta a busca completa realizada em uma Treeph que corresponde ao arranjo do globo terrestre de regiões.

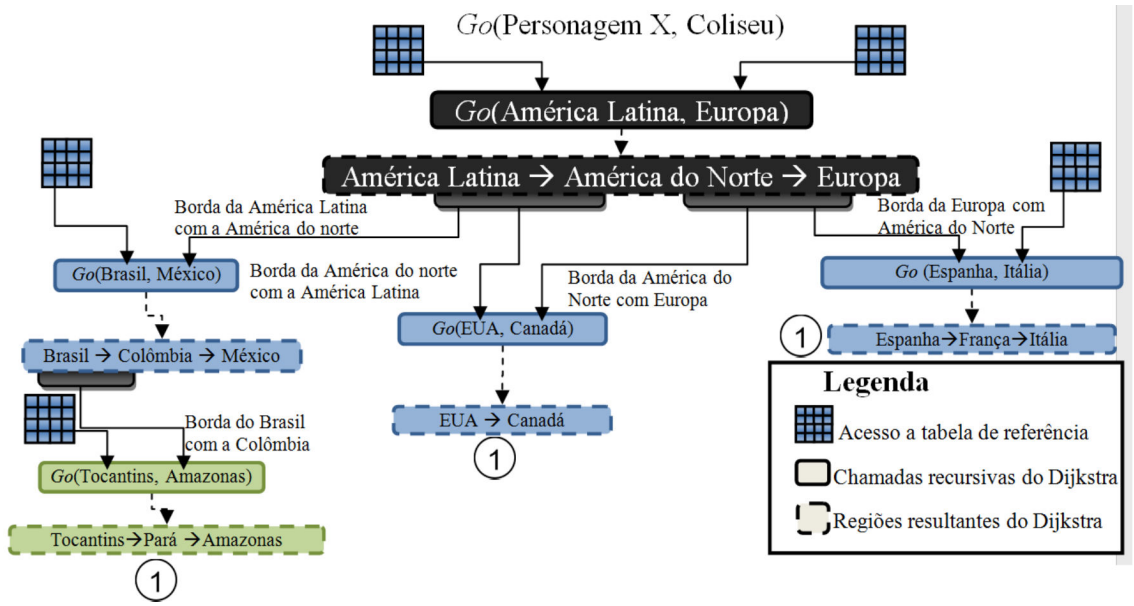


Figura 4.10: Etapa final na realização da busca pelo caminho que leva o Personagem X ao Coliseu.

Com esta abordagem um sistema de *storytelling* pode obter caminhos que cruzem um mundo extenso provendo jornadas interessantes aos atores. Isto pode ajudar tais sistemas

a melhorar a qualidade e duração da dramatização.

4.4 Interações do Usuário

Uma vez que a rota que navega pelas regiões pode ser selecionada de acordo com os seus atributos a geração de conteúdo personalizado é possível. Com apenas algumas interações, o usuário pode modificar completamente a dinâmica do roteiro a ser apresentado.

Com esta abordagem, o mesmo roteiro pode ser exibido de diversas formas. O usuário poderia, por exemplo, testemunhar o drama vivenciado pelo herói para resgatar a princesa sendo apresentado em cenários bonitos em que o protagonista irá passar. Alternativamente, os planos maléficos do vilão poderiam ser demonstrados com uma percepção aterradora.

4.5 O Diretor

Em um estúdio de cinema, o diretor traduz o mundo descrito no roteiro em imagens específicas. Ele visualiza o *script* ao dar forma concreta a conceitos abstratos. Além disso, ele estabelece um ponto de vista na ação, o que ajuda a determinar a seleção das tomadas, posições de câmeras e movimentos. O diretor é responsável pela estrutura dramática, fluxo direcional do filme e pelo processo geral de decisões. Logo, ele garante que a narrativa seja atuada e demonstrada corretamente ao utilizar técnicas de filmagem. Entretanto, a cinematografia automática encara uma dificuldade não encontrada no mundo real: as descrições informais das regras de filmagem não são explícitas o suficiente para serem codificadas em uma linguagem formal. A maioria dos diretores trabalham a partir de um *script* que foi lido anteriormente e têm a possibilidade de editar a filmagem crua em um processo posterior, sem ter que se preocupar com a interação do usuário como está sendo proposto aqui.

O diretor proposto neste trabalho é um agente responsável pelo gerenciamento da dramatização do roteiro. A estrutura deste é apresentada na figura 4.11. Ele se comunica com o Gerenciador de Roteiro para receber a lista de eventos que o usuário selecionou para dramatizar. Do usuário, ele obtém atributos desejados que devam ser considerados para guiar as escolhas durante a dramatização. O planejador de caminhos (seção 4.3) é invocado para gerar um conjunto de *waypoints* que forma uma rota que um ator deve seguir de acordo com o roteiro. Cada ator recebe uma lista de ações a serem executadas

e finalmente, o módulo de direção de câmera decide qual o melhor posicionamento de câmera para cada situação (ARIJON, 1976; HE; COHEN; SALESIN, 1996; HALPER; HELBING; STROTHOTTE, 2001; CHARLES et al., 2002).

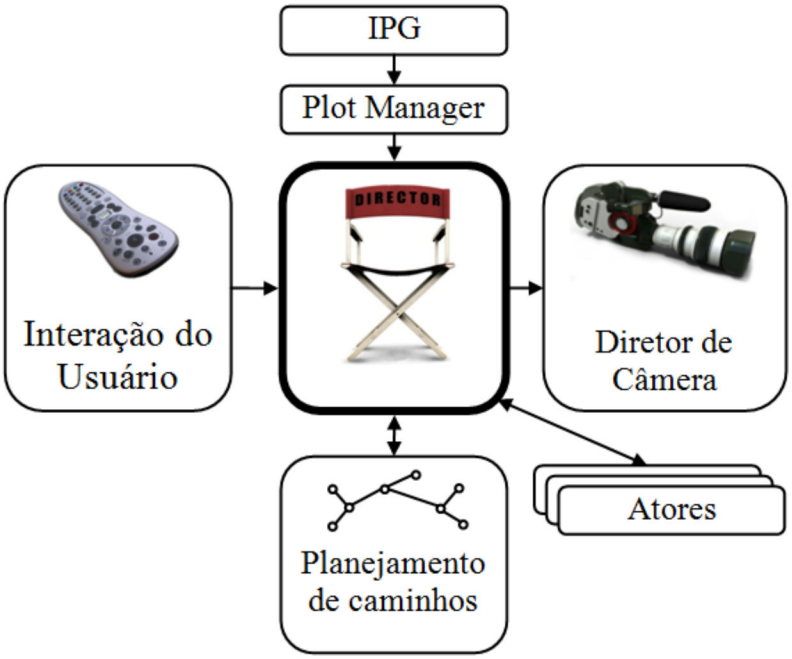


Figura 4.11: Esquematização dos módulos que compõem o sistema de dramatização.

Com o objetivo de produzir um efeito estético agradável, técnicas de cinematografia foram estudadas (vide seção 2.2) e adicionadas à versão atual do sistema na forma de um diretor de câmera. O diretor de câmera concentra os conhecimentos cinematográficos e decide em tempo real, a melhor forma de apresentar as cenas. O conhecimento é representado por diversas SVMs treinadas para resolver problemas cinematográficos envolvendo escolha de tomadas de câmeras. SVMs são um meio efetivo para reconhecimento padrão de propósito geral; elas são baseadas na teoria do aprendizado estatístico e são especializados para pequenos conjuntos de dados (TYAGI, 2008). *Support vector machines* têm uma generalização melhor que redes neurais e garantem soluções ótimas locais e globais similares às obtidas com redes neurais (GUNN, 1998).

O diretor de câmera desenvolvido (LIMA et al., 2009) extrai informações do cenário para realizar a escolha de posições de câmera. Estas informações provêm de:

- posição dos atores na cena, fornecido pelo *waypoint* em que o personagem está;
- estado emocional dos atores;

- ação atual do ator, estando andando, conversando ou realizando alguma outra ação.

De acordo com estes critérios o diretor de câmera utiliza as SVMs para selecionar uma das possíveis tomadas para aquela cena. Os tipos de tomadas de câmera possíveis para uma cena de diálogo estão apresentados na figura 4.12.

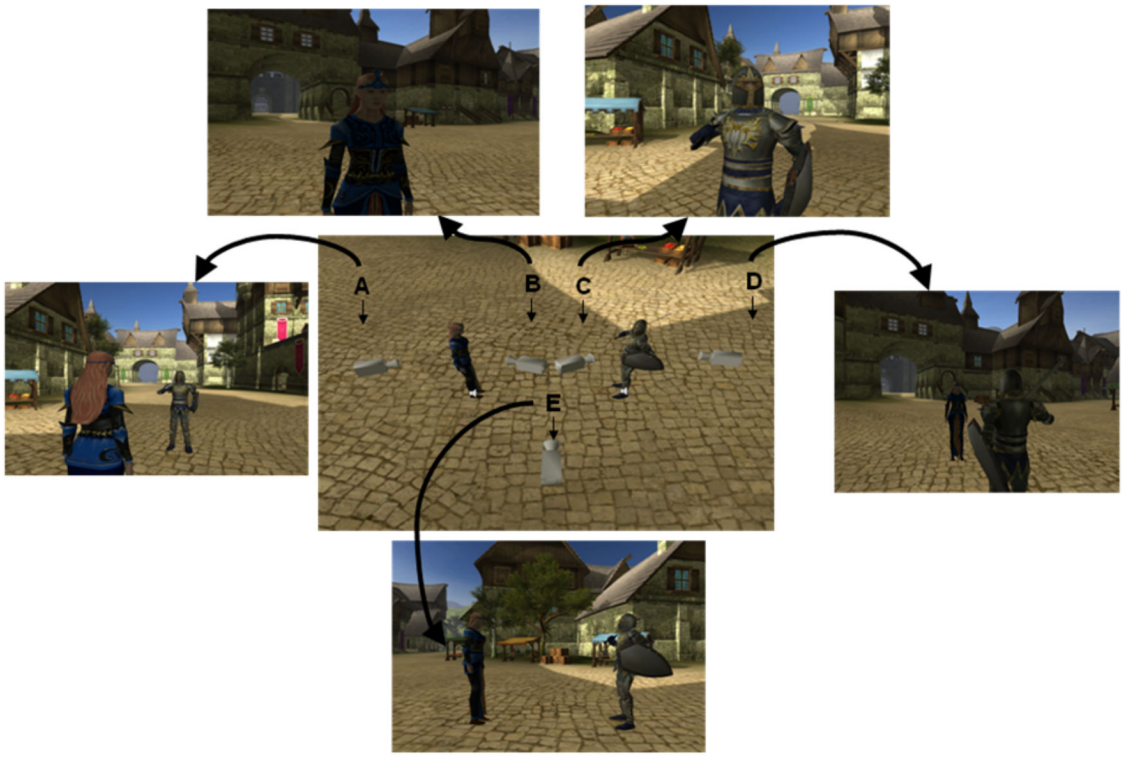


Figura 4.12: Tipos possíveis de tomadas de câmera para uma cena de diálogo.

5 IMPLEMENTAÇÃO

Os conceitos desenvolvidos no capítulo anterior para a criação de um sistema de dramatização para *storytelling* interativo foram implementados no novo módulo de dramatização do Logtell. O sistema foi desenvolvido com o uso da *GameEngine* Unity3D.

5.1 Logtell

O Logtell é um sistema de *storytelling* desenvolvido para criar e dramatizar histórias. A geração do enredo é realizada pelo IPG e a dramatização é feita pelo módulo de dramatização. Um novo módulo de dramatização está sendo desenvolvido no Laboratório de Computação Aplicada (LaCA) visando a utilização do Logtell em um ambiente de TV digital. Os conceitos desenvolvidos neste trabalho foram aplicados nesse novo módulo.

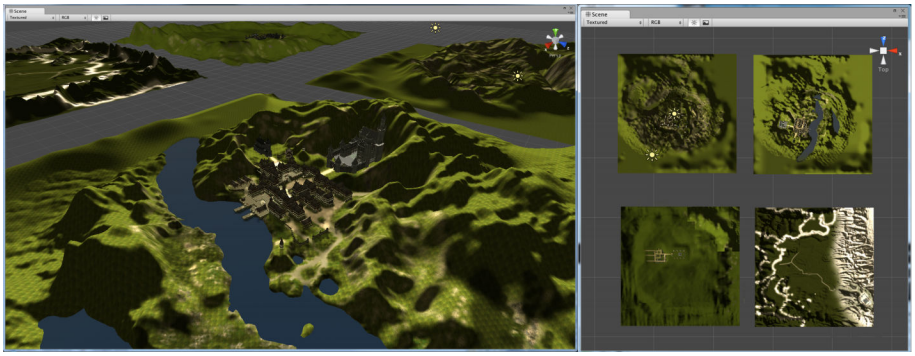
O módulo de dramatização se conecta com o sistema de geração de enredos (IPG) e recebe a história produzida pelo usuário. A história é composta por eventos que são dramatizados. Um destes eventos é o evento *Go*, abordado neste trabalho. Entretanto, outros eventos também são dramatizados, como por exemplo: *Fight, Kill, ReduceProtection, Free, Marry*.

Para que os conceitos descritos neste trabalho pudessem ser aplicados no Logtell as seguintes etapas foram desenvolvidas. Primeiramente, um grande cenário, composto por quatro malhas 3D de terreno foi montado. Este cenário apresenta castelos, vilas e personagens com características típicas do gênero medieval de contos de fadas, o qual é o modelo de histórias geradas pelo Logtell.

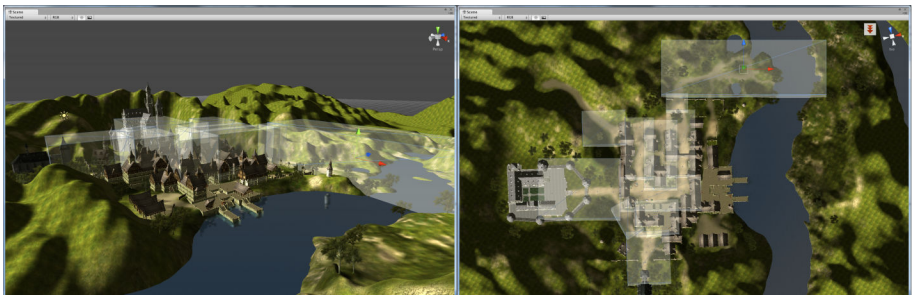
A segunda etapa consiste na separação de áreas com características distintas em regiões, e avaliação de cada região de acordo com os atributos definidos na seção 4.1.1. Subsequentemente posicionou-se os *waypoints* em cada região folha. O processo de construção do cenário é finalizado ao configurarem-se as relações de adjacências entre regiões

e *waypoints*, tal como descrito no apêndice A.

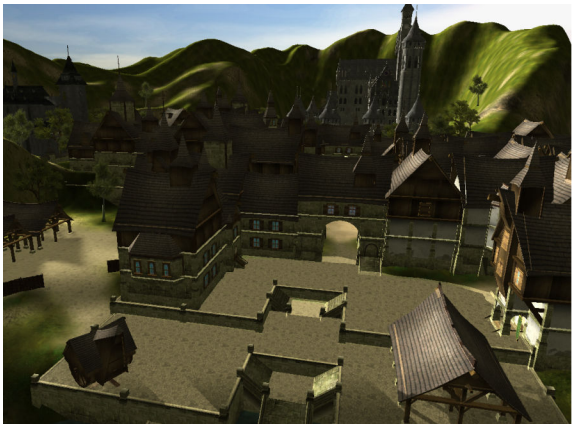
Na figura 5.1 pode-se observar o resultado obtido. Em (a) são apresentados os quatro cenários que compõem o mundo virtual utilizado. Em (b) estão presentes os *waypoints* e regiões que estão presentes nos cenários. Em (c) o aspecto visual do cenário é destacado.



(a)



(b)



(c)

Figura 5.1: Apresentação do Logtell. a) Terrenos; b) *Waypoints* e regiões; c) Exemplo de cenário.

5.2 Estruturas de Dados

Uma vez que a expansão da dramatização está baseada no evento Go, foi desenvolvido um algoritmo que navega pela Treeph e extrai um caminho composto por *waypoints* para

guiar os atores pelo cenário. O algoritmo foi implementado através de *scripts* desenvolvidos em C#. A seguir é apresentada uma breve descrição da implementação e estruturação de cada um dos scripts que compõem o algoritmo.

- Actor.cs: O componente Actor implementa o comportamento que um ator de *storytelling* deve seguir. Nele é descrito o procedimento que deve ser tomado quando um ator recebe um evento proveniente do Logtell. Em relação ao evento Go o ator é provido com uma lista de *waypoints* e realiza uma caminhada em linha reta entre um *waypoint* e outro. Ao chegar no *waypoint* final ele para e aguarda o recebimento de um novo evento.

- Attributes.cs: Classe que implementa o suporte a atributos para as regiões. Possui a definição de uma enumeração que determina o nome dos atributos. Visando a expansão do número de atributos de uma forma simples foram criados métodos para ler e escrever estas variáveis de forma independente. Toda manipulação de atributos é feita através dessas funções, impedindo que qualquer código fora desse script tenha que ser modificado ao se adicionar ou remover atributos.

- CameraDirector.cs: Componente que adiciona conceitos cinematográficos em uma câmera. A câmera recebe o ator a ser filmado e, de acordo com o tipo de *waypoint* no qual ele está posicionado, realiza uma determinada tomada;

- EdgeTable.cs: Classe que implementa os métodos da tabela de arestas. Cada região possui uma instância de uma EdgeTable para manter a relação de adjacências entre regiões e sub-regiões. As informações são mantidas através de uma tabela hash que relaciona duas regiões. Para implementar a tabela foi utilizada a classe *HashTable* do pacote *System.Collections.Generic* disponível para C#.

- Graph.cs: Classe que implementa o suporte a grafos, disponibilizando métodos de inserção, remoção e busca utilizando o algoritmo de Dijkstra e A*. As informações de custo e adjacência entre regiões de um mesmo grafo são mantidas por tabelas hash. Para implementar as tabelas hash foi utilizada a classe *HashTable* do pacote *System.Collections.Generic*. Para manter todos os nós de um grafo utilizou-se a classe *List* do pacote *System.Collections*.

- PathPlannerDirector.cs: Componente que controla o sistema planejamento de caminhos. Contém uma instância da classe Treeph e realiza a chamada dos métodos que desempenham a busca do caminho. Este componente recebe o atributo selecionado pelo usuário e configura a Treeph para que ela faça a busca de forma adequada.

- ReferenceTable.cs: Classe que implementa os métodos da tabela de referências. Existe uma única instância dessa classe, a qual é criada e utilizada pelo *PathPlannerDirector*. As informações que relacionam nomes de locais e personagens a *waypoints* são mantidas por uma tabela hash. Para implementar a tabela foi utilizada a classe *HashTable* do pacote *System.Collections.Generic* disponível para C#.

- Region.cs: Componente que adiciona o comportamento a uma região. Cada região deve manter informações sobre seus nós filhos, nós adjacentes e se for uma região folha, os *waypoints*. A região possui uma instância de sua tabela de arestas, implementada pela classe EdgeTable; um grafo de seus nós filhos, através da classe Graph; e uma instância da estrutura de atributos, definida na classe Attributes.

- Treeph.cs: Classe que fornece o métodos de busca recursivo na estrutura. Existe uma única instância dessa classe, a qual é criada e utilizada pelo *PathPlannerDirector*. Ela necessita de uma referência para o nó raiz da estrutura de regiões presente no cenário e a partir deste realiza a busca pelo caminho considerando o atributo pelo usuário. Nesta classe é definido qual o atributo selecionado pelo usuário para obter os caminhos.

- Waypoint.cs: Componente que adiciona o comportamento a um *waypoint*. Cada *waypoints* deve manter informações sobre seus nós adjacentes, através de uma instância da classe *Graph*; e sobre o seu tipo, determinando se estes *waypoint* está na borda de uma região e se esta borda é o limite de uma malha3D. A diferenciação entre os diversos tipos é feita através da definição de uma enumeração.

5.3 Corte de Caminhos

Em alguns momentos é interessante que o caminho a ser apresentado pelo sistema de dramatização não seja mostrado inteiramente, como por exemplo, a ida de um herói para a sua casa após derrotar o vilão, ou a caminhada dos noivos a igreja. Visando possibilitar

esse tipo de decisão foi desenvolvido uma forma de se diminuir o número de *waypoints* percorridos por um personagem.

Sempre que o *PathPlannerDirector* realiza a busca por um caminho ele retorna uma lista contendo todos os *waypoints* que devem ser percorridos para que um dado personagem chegue a um local específico. Para que nem todo esse caminho seja apresentado um algoritmo de redução de caminho é executado na lista obtida para reduzir o número de *waypoints* a uma fração do seu tamanho. A escolha de quanto o caminho a ser exibido deve ser reduzida é realizada pelo usuário e é representada por uma porcentagem. Por exemplo, um usuário pode querer que a viagem de volta do ator para a sua casa após resgatar a princesa seja reduzida em 70%. Ele então move a barra apresentada na figura 5.2 para a posição equivalente a 70%.

Figura 5.2: Interface para configuração do corte de caminhos.

Por convenção, se o valor de redução do caminho for superior a 90% somente os *waypoints* do tipo *InterMalha* (equivalente a continentes nos exemplos citados neste trabalho) serão mostrados. Caso o valor de corte seja igual a 100% somente o primeiro e último *waypoint* serão exibidos.

O algoritmo de redução de *waypoints* considera a importância de cada *waypoint* da lista fornecida pelo *PathPlannerDirector* para determinar se ele irá fazer parte da lista final. Existem três níveis de importância para *waypoints*:

- Primeiro e último: O primeiro e o úlitmo *waypoint* são considerados os mais importantes e nunca são removidos da lista. A sua importância é devido à necessidade de que o local que o personagem estava e o local de seu destino sejam apresentados.

- *Waypoint InterMalha*: Os *waypoints* que pertencem à borda e de regiões de mais alto nível somente são removidos da lista caso o nível de corte seja definido como 100%.

- Fator de importância: Cada *waypoint* possui um atributo que representa o seu fator de importância. Quão maior o fator de importância maior a possibilidade desse *waypoint* estar contido na lista final.

O diagrama que representa o modo como o algoritmo funciona é apresentado na figura 5.3. Ele possui três etapas. Durante a primeira etapa (bloco 1 na 5.3) o primeiro e último são adicionados a lista de saída o fator de corte é verificado, se o fator de corte for igual a 100% a lista é retornada com o respectivo resultado. Na segunda etapa (bloco 2 na 5.3) a lista completa é percorrida e todos os *Waypoints InterMalha* são adicionados a lista de saída caso o fator de corte seja maior que 90% a lista de saída é retornada. A terceira (bloco 3 na 5.3) e última etapa consiste no sorteio de dois números aleatórios; o primeiro corresponde ao índice de um candidato presente no caminho inicial e o segundo será comparado com o fator de importância do candidato caso este seja menor o candidato é retirado da lista final. Esse passo é repetido até que a lista final possua o número correto de *waypoints*.

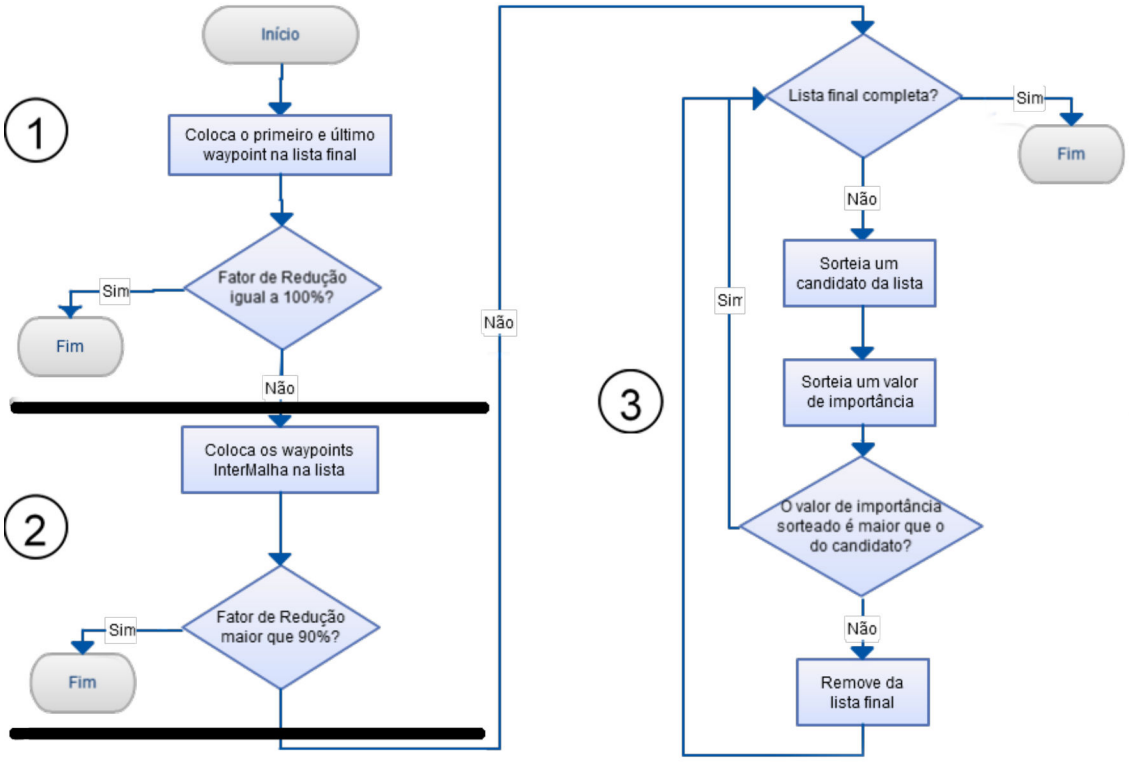


Figura 5.3: Algoritmo de corte de caminhos.

5.4 *Culling*

Um dos fatores a se considerar quando grandes malhas 3D são manipuladas é o desempenho obtido pela aplicação. O cenário atual do sistema de dramatização do Logtell conta com quatro malhas 3D complexas, cada uma esta povoada com casas, castelos,

torres e ornamentos medievais. Muitas dessas malhas e objetos não necessitam estar carregados em memória durante toda a dramatização. Para solucionar esta questão foram desenvolvidos alguns métodos de controle do carregamento de objetos.

O processo de remoção de objetos de um grupo baseado em um critério específico é denominado *Culling*. Esta técnica é amplamente utilizada em jogos e aplicações que manipulam grande quantidade de memória e esforço gráfico visando aumentar significativamente o desempenho da aplicação.

A Unity3D organiza os *GameObjects* hierarquicamente permitindo que objetos comuns sejam agrupados por características. Essa característica foi explorada para que o algoritmo de *Culling* removesse objetos de uma forma simples. Como o cenário atual está dividido em quatro grandes regiões de alto nível, cada uma delas foi disposta em um *Gameobject*. A figura 5.4 apresenta a organização do cenário e a hierarquia dos *GameObjects*. Ao analisar-se a figura nota-se que cada região possui dois filhos: um denominado *Objects* e o outro *Terrain*. O *GameObject Objects* possui todos os elementos estáticos presentes naquela região, incluindo casas, castelos, torres etc. No *GameObject Terrain* está contida a malha 3D daquela região.

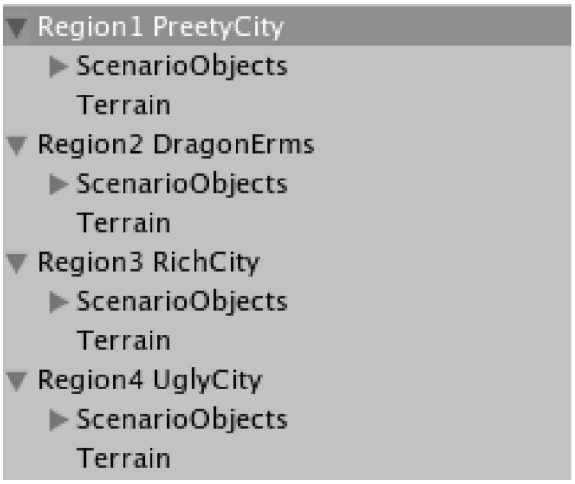


Figura 5.4: Organização do cenário e hierarquia de *GameObjects*.

A vantagem de se agrupar os objetos de uma mesma região de mais alto nível em um objeto advém do uso da função *SetActiveRecursively*, a qual torna todos os filhos de um determinado *GameObject* inativos. Deste modo pode-se utilizar esta função para que somente a região que está sendo filmada esteja ativa e consequentemente seja processada. Ao utilizar-se este método não é necessário adicionar-se comportamentos de *Culling* em cada objeto e sim somente nos *GameObjects* que servem de containeres para os objetos,

tornando, assim, a técnica bastante genérica e flexível. A separação entre *Terrains e Objects* é feita para possibilitar a escolha de realizar ou não o *Culling* da malha 3D. Devido à malha ser o objeto que mais necessita de memória, o seu carregamento pode levar algum tempo. Em favor da eficiência, pode ser vantajoso manter as estruturas de dados em memória do que ter uma pequena pausa em cortes de cena.

6 RESULTADOS

Um dos resultados desse trabalho consistiu na integração dos conceitos criados com o novo sistema de dramatização do Logtell. Para elucidar os resultados obtidos algumas comparações foram realizadas e estão apresentadas nas próximas seções.

6.1 Comparação de tempo

As tabelas a seguir representam os tempos coletados ao se realizar a comparação do sistema de dramatização antigo (POZZER, 2005) com o sistema que foi desenvolvido. A tabela 6.1 apresenta uma comparação dos dois sistemas considerando os seguintes enredos gerados pelo IPG:

- História Curta: *ReduceProtection(Marian, PrincessCastle); Go(Draco, PrincessCastle); Attack(Draco, PrincessCastle); Kidnap(Draco, Marian); Go(Brian, DracoCastle); Attack(Brian, DracoCastle); Fight(Brian, Draco); Kill(Brian, Draco); Free(Brian, Marian); Marry(Brian, Marian);*

- História Longa: *Go(Marian, Church); Go(Draco, Church); Kidnap(Draco, Marian); Go(Brian, DracoCastle); Attack(Brian, DracoCastle); Fight(Brian, Draco); Kill(Brian, Draco); Free(Brian, Marian); Go(Marian, Church); Go(Brian, Church); Marry(Brian, Marian);*

Tabela 6.1: Comparação do tempo total de dramatização de história nos diferentes sistemas.

História	**Aplicação**	**Tempo**
Curta	Sistema Antigo	2 min e 10 seg
Curta	Sistema Novo	5 min e 19 seg
Longa	Sistema Antigo	2 min e 27 seg
Longa	Sistema Novo	18 min e 2 seg

Na tabela 6.2 são apresentados os tempos de execução de alguns eventos Go em cada uma das versões do Logtell. Na nova versão os tempos presentes foram coletados com os respectivos valores de corte: 0%, 10%, 50%, 90% e 100%.

Tabela 6.2: Comparação do tempo de execução do evento *Go(Brian,Church)* em cada um dos sistemas e com vários fatores de corte(somente no novo).

Aplicação	Fator de Corte	Tempo
Sistema Antigo	-	18 seg
Sistema Novo	0%	2 min e 19 seg
Sistema Novo	10%	2 min
Sistema Novo	50%	1 min e 6 seg
Sistema Novo	90%	30 seg
Sistema Novo	100%	16 seg

6.2 Comparação de apresentação visual

Além de expandir o tempo de dramatização em um sistema de *storytelling*, o trabalho visa aumentar a qualidade gráfica do material a ser apresentado. Para tal foram adquiridos modelos 3D mais realistas, os quais foram inseridos no sistema atual. Visando apresentar uma comparação do acréscimo de qualidade adquirida no novo sistema as próximas figuras representam uma comparação entre o sistema antigo e o novo.

A figura 6.1 apresenta uma comparação entre os principais atores da história. A princesa é apresentada em 6.1a; o herói em 6.1b; e o vilão em 6.1c.

(a)

(b)

(c)

Figura 6.1: Comparação entre os principais atores da história nas diferentes versões do sistema de dramatização.

Na figura 6.2 é caracterizada uma comparação entre os principais locais da história. O castelo da princesa é apresentada em 6.2a; o castelo do vilão e em 6.2b; e a igreja em 6.2c. Pode-se notar o grande salto na qualidade gráfica e detalhes.

(a)

(b)

(c)

Figura 6.2: Comparação entre os principais locais da história nas diferentes versões do sistema de dramatização.

Uma comparação entre o mundo em que a história se desenrola é ilustrado na figura 6.3. Nota-se que o novo cenário está muito mais completo, apresentando maiores detalhes e inserindo um grande número de elementos.

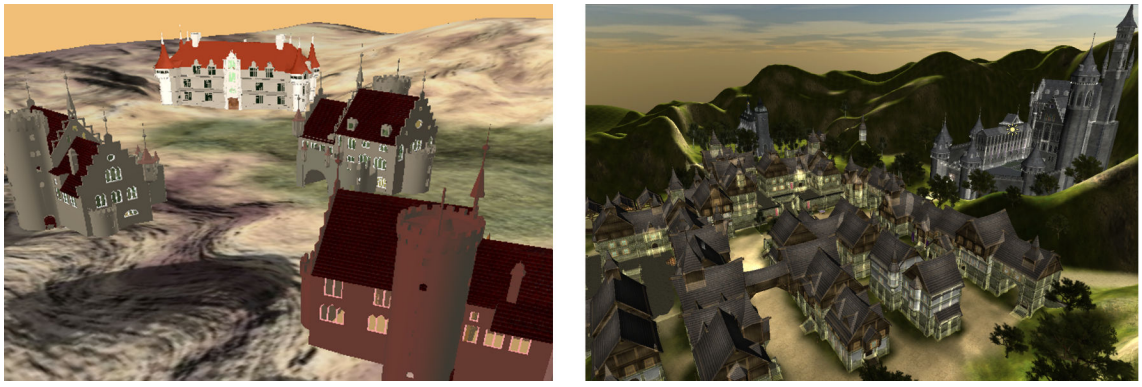
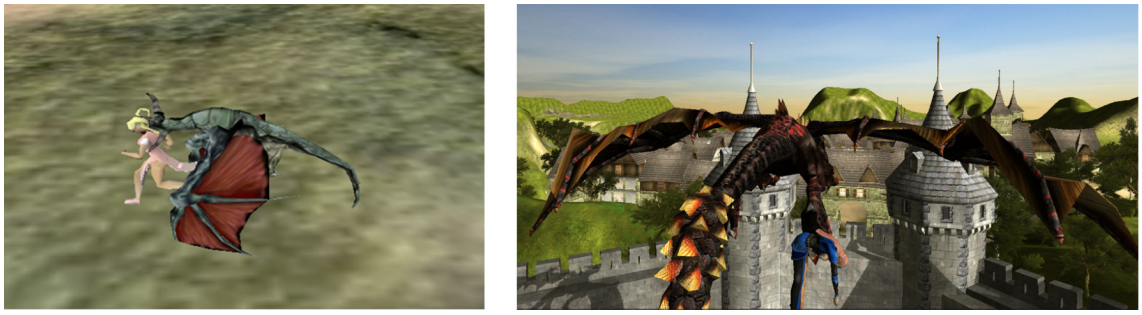


Figura 6.3: Comparação entre o cenário em diferentes versões do sistema de dramatização.

O aspecto de alguns eventos no sistema antigo e no novo estão apresentados na figura 6.4. Para que os eventos condissessem com o realismo dos modelos o comportamento dos atores teve que ser muito superior ao da versão anterior. O evento *Fight* é caracterizado em 6.4a; e o evento *Kidnap* em 6.4b.

(a)



(b)

Figura 6.4: Comparação entre alguns eventos da história nas diferentes versões do sistema de dramatização.

6.3 Desempenho com *Culling*

Com a utilização da técnica de *Culling* de cenário no novo sistema de dramatização pode-se notar um aumento significativo no desempenho da aplicação. A tabela 6.3 a

seguir apresenta uma comparação do sistema atual com diferentes níveis de *Culling* executados em um processador *QuadCore 2,4 GHz* com *4GB* de memória e uma placa gráfica *GeForce 8800GT*.

Tabela 6.3: Comparação da média de quadros por segundo da aplicação ao realizar um evento *Go(Brian,Church)* variando-se os níveis de *Culling*.

Nível de Culling	Média de quadros por segundo
Sem Culling	15,37
Com Culling de Objetos	18,08
Com Culling de Objetos e Terreno	36,29

6.4 Suporte a trabalhos futuros

Dentre os resultados obtidos com este trabalho, a obtenção de uma estrutura reutilizável foi o mais importante deles. *Storytelling* interativo é uma área em contínuo desenvolvimento e que possui diversas possibilidades de pesquisa. Por este motivo a criação de técnicas, ferramentas e conceitos é tão importante.

De forma semelhante a este trabalho pode-se extrair características da indústria cinematográfica e inseri-las em sistemas de *storytelling* para que este cresça em qualidade e capacidade de entretenimento. Algumas dessas características podem se aproveitar da estruturação definida aqui para fomentar o seu desenvolvimento. As próximas subseções irão expandir algumas dessas idéias.

6.4.1 Personagens Secundários

A maior parte dos enredos gerados automaticamente ou escritos manualmente possui poucos personagens principais, nos quais a história se foca e, geralmente, é através da visão de mundo deles que a narrativa é construída. Entretanto, esses personagens interagem de diversas maneiras com outros personagens. Aproveitando os exemplos citados na seção 4.1 pode-se perceber que em O Senhor dos Anéis os personagens principais são Frodo e Sauron, contudo, personagens como Sam, Aragorn e Gandalf são bastante apresentados durante a história e são muito importantes para a história do filme. Da mesma forma, pode-se considerar que o Yoshi é um personagem secundário na saga de Mario para resgatar a princesa.

Em um modelo de *storytelling* semelhante ao desenvolvido neste trabalho, personagens secundários podem ser inseridos entre eventos da história para que esta expanda a

sua capacidade de entretenimento. Desta forma personagens secundários com características reativas podem ser colocados em diversos locais e serem ativados ao perceberem a aproximação de algum personagem principal e realizem alguma ação com este personagem, como por exemplo: iniciar uma conversa ou uma luta e seguir o personagem em suas ações por certo período de tempo.

Desta forma pode-se aproveitar um grande cenário, onde os personagens principais devem realizar jornadas, para dispor personagens secundários pelo mundo, enriquecendo assim a história apresentada ao telespectador. Pode-se aproveitar o fato do cenário estar dividido em regiões e estas classificadas de acordo com alguns atributos. Os atributos podem ser utilizados para que um sistema automático de posicionamento de personagens secundários escolha qual deles irá ser colocado em cada região de acordo com os atributos da região e da personalidade do personagem. Além disso, o sistema de navegação pode ser utilizado para que os personagens secundários façam algumas jornadas pelo mundo, e acabem se encontrando com algum personagem principal e interagindo com este.

6.4.2 *Fillers*

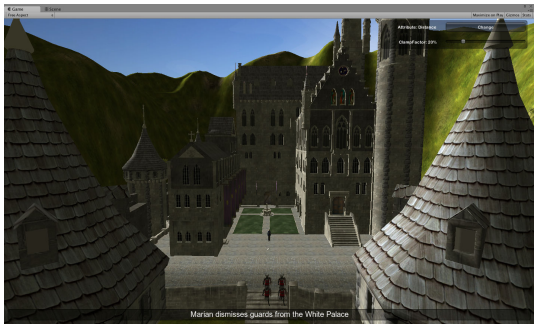
Diversas formas de entretenimento apresentam seções de sua história total que poderiam ser removidas totalmente sem que a linha geral da história fosse comprometida. Estas seções recebem o nome de *fillers* e são muito comuns em novelas e seriados. *Fillers* têm por objetivo inserir conteúdo de entretenimento por um determinado período de tempo para expandir a duração total da história. Entretanto, um *filler* não deve interferir no rumo que a história estava tomando.

Um sistema de *storytelling* pode utilizar *fillers* para aumentar o tempo e qualidade da dramatização da história sem a necessidade da geração de um enredo que possua um encadeamento lógico formalmente definido. Eles podem ser ativados entre algum evento da história e apresentar uma pequena narrativa sem consequências para o contexto corrente. Pode-se utilizar os momentos de transição entre cenas para ativar eventos em paralelo, por exemplo.

Novamente, pode-se aproveitar de um grande cenário e da divisão em regiões para dispor eventos que ativam os *fillers* pelo mundo. Estes não têm um princípio de localidade tão forte, mas podem se aproveitar dos atributos das regiões para manter uma coerência maior com o mundo e o *filler* a ser utilizado.

6.5 Apresentação de uma história completa

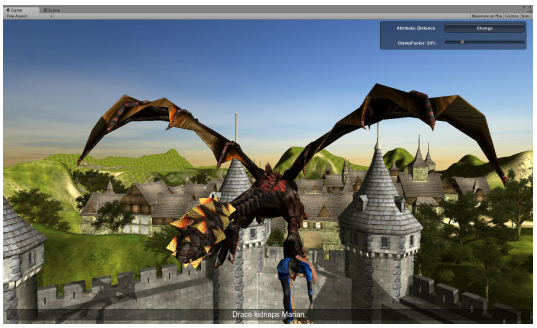
Para ilustrar os resultados obtidos com a ferramenta desenvolvida, nas figuras a seguir é apresentada uma série de *screenshots* provenientes da execução da aplicação. Os *waypoits* foram mantido somente para realçar o caminho seguido pelos personagens.



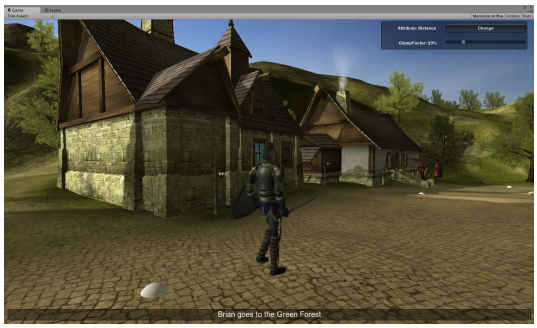
(a) *ReduceProtection(Marian,PrincessCastle);*

(b) *Go(Draco,PrincessCastle);*

(c) *Attack(Draco,PrincessCastle);*

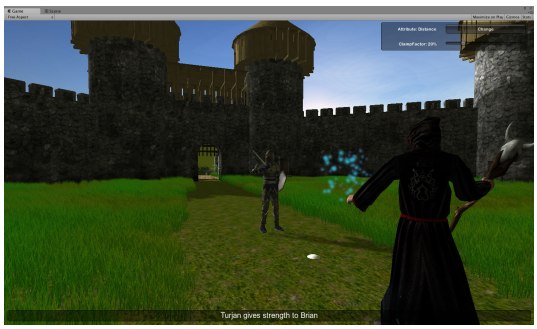


(d) *Kidnap(Draco,Marian);*

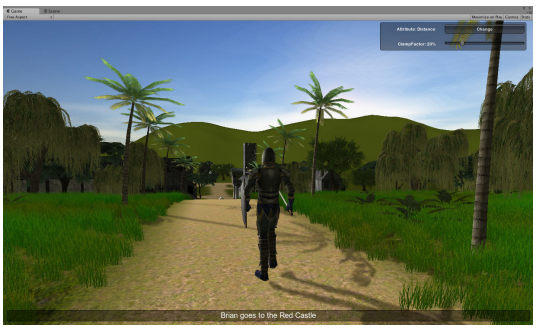


(e) *Go(Brian,TurjanCastle);*

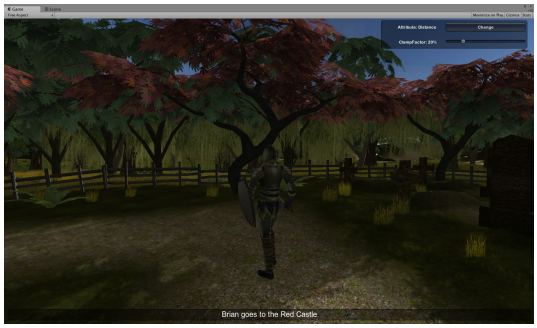
(f) *Go(Brian,TurjanCastle);*



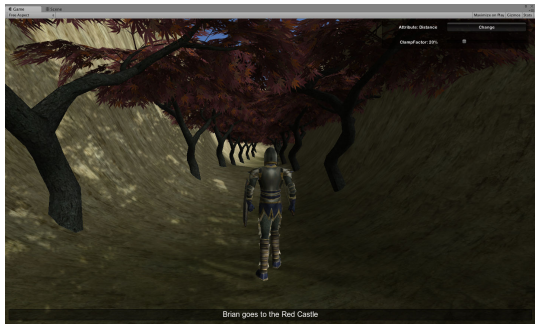
(g) *Donate(Turjan,Brian);*



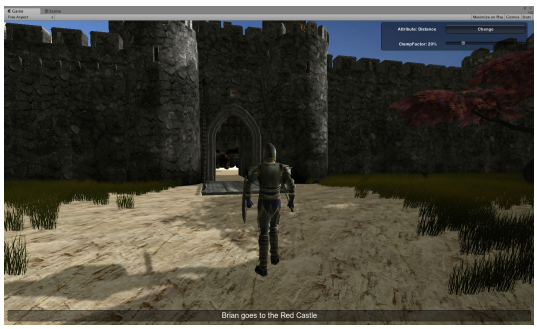
(h) *Go(Brian,DracoCastle);*



(i) *Go(Brian,DracoCastle);*

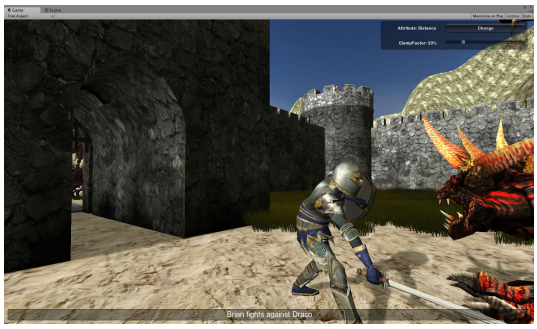


(j) *Go(Brian,DracoCastle);*



(k) *Go(Brian,DracoCastle);*

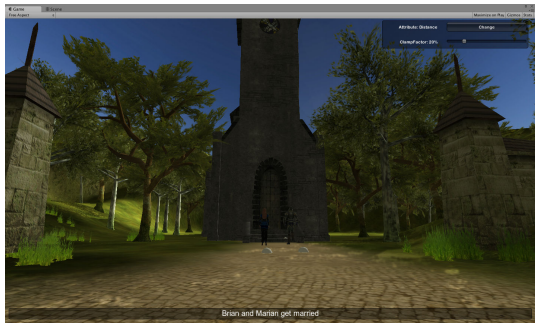
(l) *Attack(Brian,DracoCastle);*



(m) *Fight(Brian,Draco);*

(n) *Kill(Brian,Draco);*

(o) *Free(Brian,Marian);*



(p) *Marry(Brian,Marian);*

7 CONCLUSÃO

O objetivo deste trabalho foi a elaboração de um sistema de *storytelling* que seja capaz de entreter um usuário por um período razoável de tempo. Para atingir este objetivo foram desenvolvidas técnicas para aumentar o tempo e a qualidade da dramatização do sistema de *storytelling* interativo Logtell. Expandiu-se o tempo de dramatização ao ampliar o evento *Go*, tornando-o mais longo, possibilitando que um determinado personagem percorra um mundo grande e com diversas características.

Mesmo que a dramatização de um sistema de *storytelling* possa ser expandida mais que a técnica apresentada aqui, a proposta almeja ser um passo inicial nesta direção. A expansão do cenário parece ser uma proposta interessante para aumentar o tempo de dramatização e conteúdo em um modelo de storytelling baseado em roteiro.

A estrutura de dados Treeph se mostrou uma solução adequada para o problema de busca de caminhos em grandes mundos virtuais, graças as suas características organizacionais hierárquicas das árvores e o desempenho em busca de caminhos de grafos. Acredita-se que regiões e *waypoints* forma uma combinação para se utilizar em um algoritmo de busca de caminhos multi-nível.

A escolha da Unity3D como *GameEngine* possibilitou um grande salto na qualidade do material e na velocidade de desenvolvimento deste. Juntamente com o algoritmo de *Culling* a Unity3D foi eficiente para permitir a utilização do aplicativo de storytelling em tempo real.

Os resultados obtidos se mostraram satisfatórios, possibilitando que o sistema desenvolvido possua tempo total e qualidade de dramatização bastante superior ao sistema anterior (POZZER, 2005).

7.1 Trabalhos futuros

Este trabalho tem como meta servir de base para que sejam desenvolvidas novas técnicas de extensão do tempo e qualidade de *storytelling*. A subdivisão de um cenário em regiões é útil para o acréscimo de conteúdo reativo, ou seja, dependendo da posição de algum personagem ou evento pode-se alavancar algum outro evento. A seguir são apresentados tópicos de pesquisas que podem fazer uso do trabalho aqui desenvolvido.

- Personagens Secundários: A utilização de personagens secundários e figurantes pode tornar uma história mais dinâmica, tirando o centro de ação de apenas alguns poucos personagens. Estes personagens poderiam ser ativados quando percebessem a aproximação de algum personagem principal e tomassem alguma atitude para tornar a história mais atrativa, entretanto, sem alterar o enredo geral da história.

- *Fillers*: textitFillers são laços de história que não adicionam nem removem fatos importantes ao enredo geral. Servem apenas para preencher o tempo de dramatização e apresentar material de boa qualidade gráfica. Muitos filmes, seriados e novelas apresentam diversos *Fillers* durante o seu desenrolar. Estes também podem se aproveitar da estrutura de regiões para serem ativados.

7.2 Publicações

A seguir são apresentadas os artigos relacionados a este trabalho de graduação que foram publicados previamente e durante o seu desenvolvimento.

- LIMA, E. S.; POZZER, C. T.; DALLA FAVERA, E. C.; DORNELLAS, M. C.; CIARLINI, A. E. M.; FEIJO, B.; FURTADO,A. L. *Support Vector Machines for Cinematography Real-Time Camera Control in Storytelling Environments*. Em: VIII Simpósio Brasileiro de Jogos e Entretenimento Digital - SBGames 2009, 2009, Rio de Janeiro.

- DALLA FAVERA, E. C.; POZZER, C. T. *Expandindo Storytelling: Aumentando o Tempo de Dramatização ao Utilizar Cenários Grandes*. Em: Congresso Regional de Iniciação Científica e Tecnológica em Engenharia - CRICTE, 2009, Joinville-SC.

- AZEVEDO, V. C.; DALLA FAVERA, E. C.; POZZER, C. T.; CIARLINI, A. E. M. FEIJO, B.; PASSOS, E. B. *Using Navigation Meshes to Improve Storytelling Dramatization*. Em: Simpósio Brasileiro de Jogos e Entretenimento Digital, 2008, Belo Horizonte.

REFERÊNCIAS

ALDRICH, J.; CHAMBERS, C.; NOTKIN, D. Component-Oriented Programming in ArchJava. In: OOPSLA 2001 WORKSHOP ON LANGUAGE MECHANISMS FOR PROGRAMMING SOFTWARE COMPONENTS, 2001. **Proceedings...** [S.l.: s.n.], 2001.

ARIJON, D. **Grammar of the Film Language.** [S.l.]: Communication Arts Books, Hasting House, Publishers, New York, 1976.

AZEVEDO, V. C.; DALLA FAVERA, E. C.; POZZER, C. T.; CIARLINI, A. E. M.; FEIJO, B.; PASSOS, E. B. Using Navigation Meshes To Improve Storytelling Dramatization. In: VII BRAZILIAN SYMPOSIUM ON COMPUTER GAMES AND DIGITAL ENTERTAINMENT, BELO HORIZONTE - MG, 2008. **Anais...** [S.l.: s.n.], 2008.

CAVAZZA, M.; CHARLES, F.; MEAD, S. J. Character-Based Interactive Storytelling. **IEEE Intelligent Systems**, Piscataway, NJ, USA, v.17, n.4, p.17–24, July 2002.

CHARLES, F.; LUGRIN, J.; CAVAZZA, M.; MEAD, S. Real-time camera control for interactive storytelling. In: GAME-ON, LONDON, UK, 2002. **Anais...** [S.l.: s.n.], 2002.

CIARLINI, A. E. M.; BARBOSA, S. D. J.; CASANOVA, M. A.; FURTADO, A. L. Event Relations in plan-based plot composition. In: VII BRAZILIAN SYMPOSIUM ON COMPUTER GAMES AND DIGITAL ENTERTAINMENT - SBGAMES 2008, BELO HORIZONTE - MG, 2008. **Anais...** [S.l.: s.n.], 2008.

CIARLINI, A.; POZZER, C. T.; FURTADO, A. L.; FEIJÓ, B. A logic-based tool for interactive generation and dramatization of stories. In: ACM SIGCHI INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTER ENTERTAINMENT TECHNOLOGY - ACE 2005, VALENCIA, SPAIN, 133-140, 2005. **Anais...** [S.l.: s.n.], 2005.

CIARLINI, A.; VELOSO, P.; FURTADO, A. A Formal Framework for Modelling at the Behavioural Level. In: THE TENTH EUROPEAN-JAPANESE CONFERENCE ON INFORMATION MODELLING AND KNOWLEDGE BASES, SAARISELKä, FINLAND, 2000. **Proceedings...** [S.l.: s.n.], 2000.

COURTY, N.; LAMARCHE, F.; DONIKIAN, S.; MARCHAND, E. A cinematography system for virtual storytelling. In: INTERNATIONAL CONFERENCE ON VIRTUAL STORYTELLING, TOULOUSE, FRANCE, 2003. **Proceedings...** [S.l.: s.n.], 2003.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische Mathematik**, [S.l.], v.1, n.1, p.269–271, December 1959.

DORIA, T. R.; CIARLINI, A. E. M.; ANDREATTA, A. A Nondeterministic Model for Controlling the Dramatization of Interactive Stories. In: ACM MM2008 - 2ND ACM WORKSHOP ON STORY REPRESENTATION, MECHANISM AND CONTEXT - SRMC08, 2008, VANCOUVER, CANADA, 2008. **Proceedings...** [S.l.: s.n.], 2008.

FROHLICH, P. H. **Component-Oriented Programming Languages**: why, what, and how. 2003. Tese (Doutorado) — University Of California, Irvine.

GRASBON, D.; BRAUN, N. A morphological approach to interactive storytelling. In: FLEISCHMANN, M.; STRAUSS, W., EDITORS, PROCEEDINGS: CAST01, LIVING IN MIXED REALITIES, SANKT AUGUSTIN, GERMANY, P. 337-340, 2001. **Anais...** [S.l.: s.n.], 2001.

GUNN, S. R. **Support vector machines for classification and regression**. [S.l.]: Faculty of Engineering, Science and Mathematics School of Electronics and Computer Science, 1998.

HALPER, N.; HELBING, R.; STROTHOTTE, T. **Managing the Trade-Off Between Constraint Satisfaction and Frame Coherence**. 2001.

HAWKINS, B. **Real-Time Cinematography for Games (Game Development Series).** [S.l.]: Charles River Media, Inc., Rockland, MA, USA, 2004.

HE, L.; COHEN, M.; SALESIN, D. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In: ACM SIGGRAPH '96, 217-224, 1996. **Proceedings...** [S.l.: s.n.], 1996.

JOHNSON, D. B. Efficient Algorithms for Shortest Paths in Sparse Networks. **J. ACM**, New York, NY, USA, v.24, n.1, p.1–13, January 1977.

LIMA, E. E. S.; POZZER, C. T.; DALLA FAVERA, E. C.; d'ORNELLAS, M. C.; CIARLINI, A.; FEIJÓ B.AND FURTADO, A. L. Support Vector Machines for Cinematography Real-Time Camera Control in Storytelling Environments. In: VIII BRAZILIAN SYMPOSIUM ON COMPUTER GAMES AND DIGITAL ENTERTAINMENT - SBGAMES 2009, 2009. **Anais...** [S.l.: s.n.], 2009.

LOWY, J. **Programming .net Components**. [S.l.]: O'reilly Media, 2003.

MASCELLI, J. **The Five C's of Cinematography**: motion picture filming techniques. [S.l.]: Silman-James Press, Los Angeles, 1998.

MATEAS, M.; STERN, A. Towards integrating plot and character for interactive drama. In: DAUTENHAHN, K., EDITOR, SOCIALLY INTELLIGENT AGENTS: THE HUMAN IN THE LOOP, AAAI FALL SYMPOSIUM, TECHNICAL REPORT, P. 113-118, 2000. **Anais...** [S.l.: s.n.], 2000.

NEWMAN, R. **Cinematic game secrets for creative directors and producers.** [S.l.]: Focal Press, UK, 2008.

POZZER, C. T. **Um Sistema para Geração, Interação e Visualização 3D de Histórias para TV Interativa**. 2005. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro.

SPIERLING, U.; BRAUN, N.; IURGEL, I.; GRASBON, D. Setting the scene: playing digital director in interactive storytelling and creation. In: COMPUTER AND GRAPHICS 26, 31-44., 2002. **Anais...** [S.l.: s.n.], 2002.

TARJAN, R. E. **Data Structures and Network Algorithms**. [S.l.]: Society for Industrial & Applied, 1983.

TYAGI, S. A Comparative Study of SVM Classifiers and Artificial Neural Networks Application for Rolling Element Bearing Fault Diagnosis using Wavelet Transform Preprocessing. In: WORLD ACADEMY OF SCIENCE, ENGINEERING AND TECHNOLOGY VOLUME 33, P. 319-327, 2008. **Proceedings...** [S.l.: s.n.], 2008.

UNITY3D. **Disponível em**: http://unity3d.com/ [acessado em novembro 2009]. 2009.

YOUNG, R. Creating interactive narrative structures: the potential for ai approaches. In: AAAI SPRING SYMPOSIUM IN ARTIFICIAL INTELLIGENCE AND INTERACTIVE ENTERTAINMENT, PALO ALTO, CALIFORNIA. AAAI PRESS, 2000. **Anais...** [S.l.: s.n.], 2000.

APÊNDICE A MANUAL DE CONSTRUÇÃO E CONFIGURAÇÃO DE CENÁRIO NA UNITY3D

Visando o desenvolvimento de trabalhos futuros que façam uso da estrutura de cenário construída aqui o processo de criação e configuração de um cenário na Unity3D será descrito a seguir. Os requisitos necessários para a construção são o pacote de scripts da Treeph e a *GameEngine* Unity3D.

- Criando um projeto:

  O primeiro passo é a criação de um novo projeto dentro da Unity3D e carregar o pacote de scripts da Treeph. Pacotes são carregados através do comando *Import Package* no menu *Assets*. A figura A.1 representa o resultado inicial do projeto criado contendo o pacote de *Scripts* e os *Assets* necessários para a construção de um terreno.

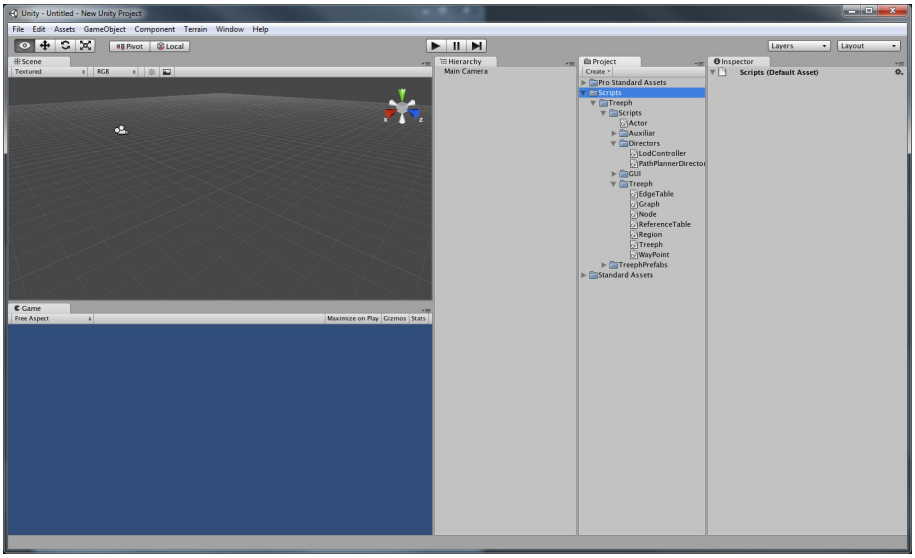


Figura A.1: Novo projeto na Unity contendo o pacote de arquivos da Treeph.

- Criando terrenos:

O side da Unity3D fornece o tutorial para a criação de terrenos na Unity3D. Nessa etapa são criados os terrenos que irão representar as regiões de mais alto nível. A figura A.2 apresenta alguns terrenos criados.

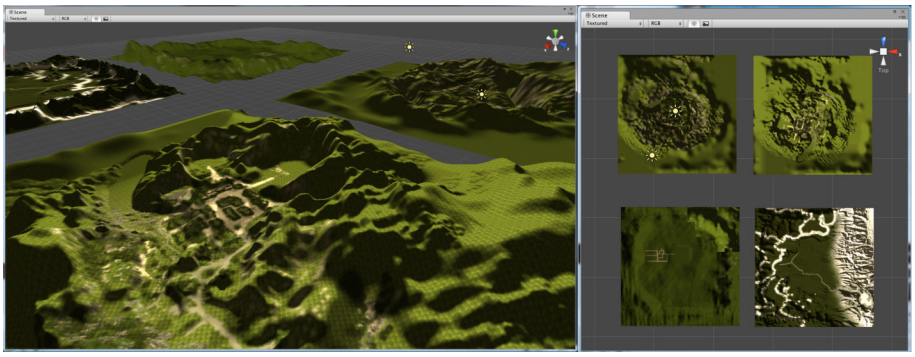


Figura A.2: Terrenos criados na Unity.

- Posicionando objetos:

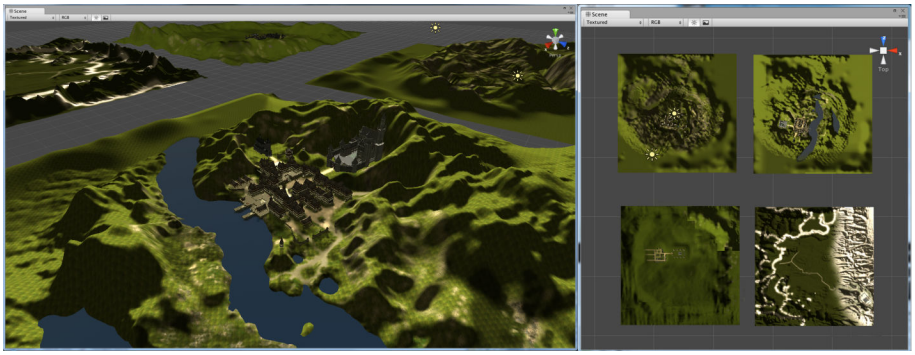
  O processo de modelagem do cenário se encerra com o posicionamento dos objetos que farão parte do mundo virtual. Nesta etapa casas, castelos, torres e quaisquer outros objetos que sejam ligados ao cenário são colocados. Para que o algoritmo de *Culling* funcione corretamente a hierarquia do cenário deve ser organizada da seguinte maneira:

  – Todos os objetos de um determinado terreno devem ser filhos de um mesmo *GameObject* de nome *ScenarioObjects*.

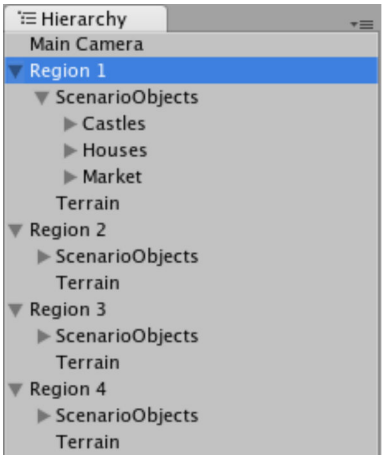
  – Todos os terrenos devem ter o nome de Terrain.

  – Ambos itens acima devem ser filhos de um mesmo *GameObject*.

  A figura A.3a demonstra o posicionamento de diversos objetos nos terrenos do item anterior. Em A.3b é apresentada a hierarquia do cenário para os terrenos do item anterior.



(a)

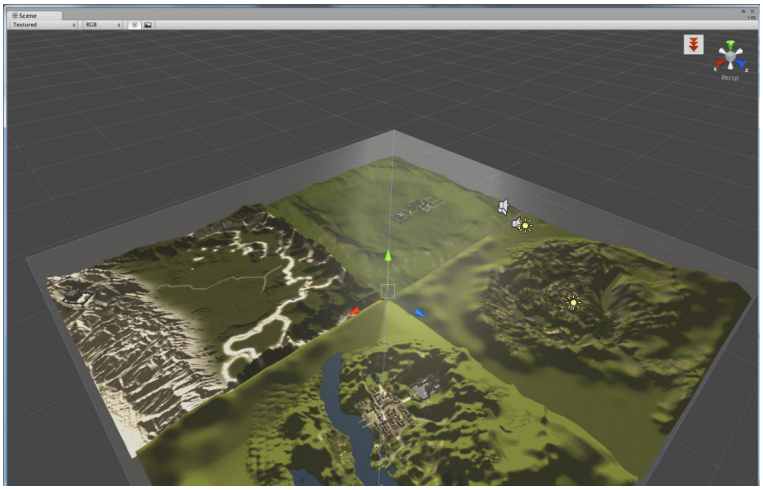


(b)

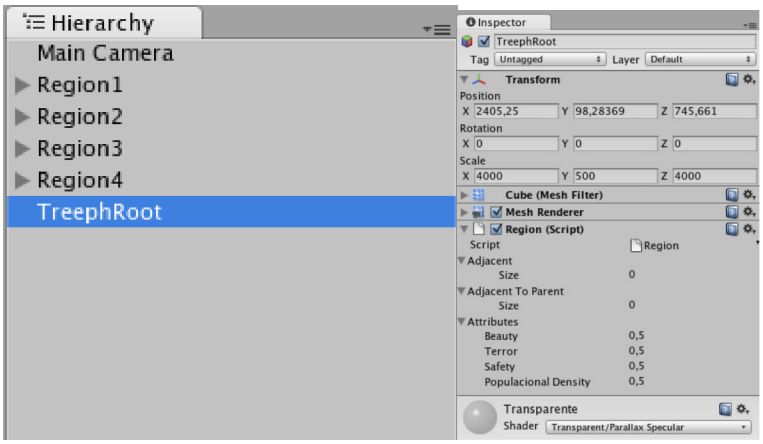
Figura A.3: Cenário contendo objetos. a) Visão em perspectiva. b) Vista Superior

- Criando regiões:

  A criação de regiões consiste na divisão do cenário em áreas com características semelhantes e dividir estas em outras menores. Inicia-se criando um *GameObject* vazio e com o nome de *TreephRoot*. Para facilitar o processo de identificação de regiões indica-se que ao invés de se criar um *GameObject* vazio crie-se um *Cube* e adicione-se um material semi-transparente. Desta forma pode-se identificar facilmente a região dentro do Editor. Escala-se esse *Cube* até que ele englobe a área que se quer delimitar na região, no caso da *TreephRoot* todos os terrenos gerados. Após o posicionamento do *GameObject* de forma correta adiciona-se o script *Region.cs* nele. A figura A.4 apresenta essa etapa.



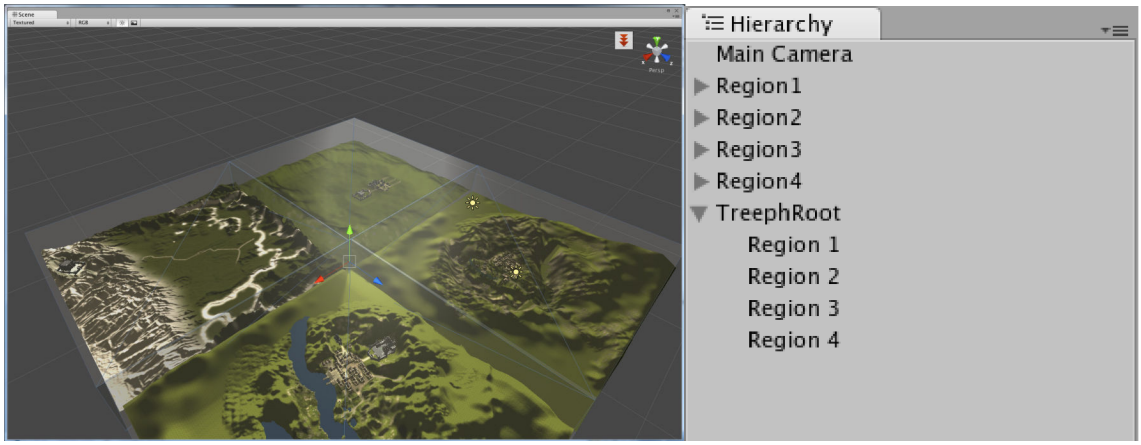
(a)



(b) (c)

Figura A.4: Criação de regiões. a) Região *TreephRoot* que engloba todas as outras; b) Hierarquia de *GameObjects*; c) *Script Region* adicionado na região *TreephRoot*.

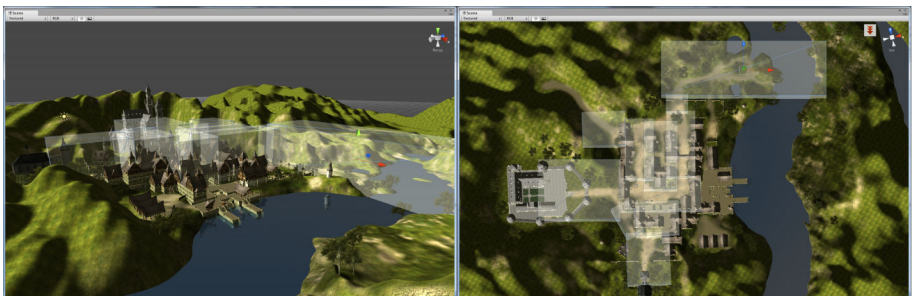
O processo de criação de regiões é repetido para cada terreno, delimitando uma região para cada um deles. Cada nova região deve ser filha da região que ela especializou. Pode-se desabilitar o *MeshRenderer* de cada um dos *cubes* para facilitar a visualização. A figura A.5 apresenta as regiões delimitadas por cubos semi-transparentes.



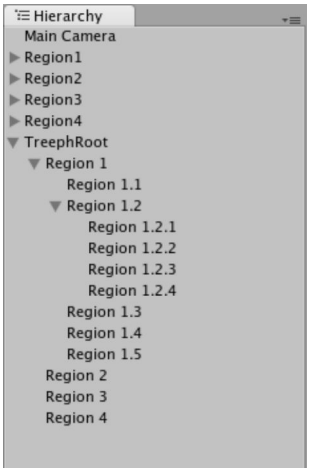
(a) (b)

Figura A.5: Criação das regiões de mais alto nível. a) Região *TreephRoot* dividida em 4 sub-regiões que englobam cada um dos terrenos; b) Hierarquia de *GameObjects*.

O processo é continuado até que todas as áreas de interesse tenham sido divididas em regiões. A figura A.6 apresenta a sub-divisão de uma região em várias menores.



(a)



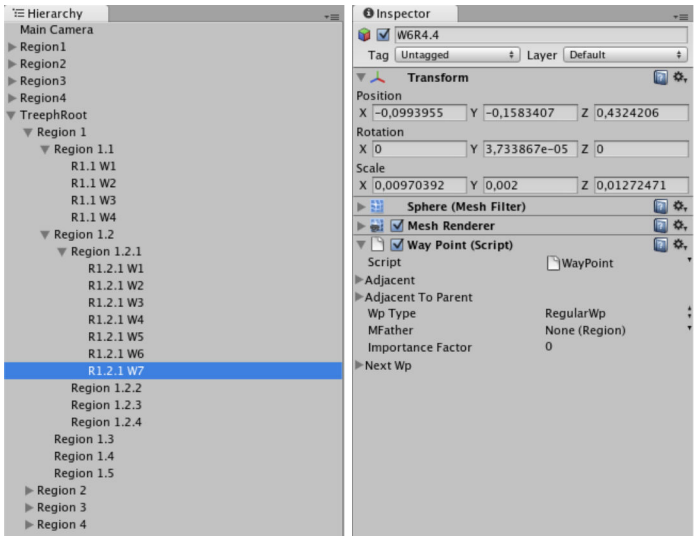
(b)

Figura A.6: Subdivisão das regiões. a) Visão em perspectiva e superior da divisão da Região 1; b) Hierarquia de *GameObjects* apresentando esta divisão.

- Dispondo *wayPoints*:

Após o processo de criação de regiões ter sido estabelecido é necessário posicionar os *waypoints* no terreno. O pacote da *Treeph* dispõem de um *prefab* de *waypoint*, este pode ser utilizado para dispor todos os *waypoints* do terreno. Inicia-se instanciando um *prefab* no terreno e utilizando as ferramentas de movimentação para que o seu posicionamento fique correto. Distribui-se os *waypoints* pelos locais que devem ser percorridos pelos atores. Todos os *waypoints* devem ser filhos da região correspondente. A figura A.7 apresenta alguns *waypoints* posicionados no terreno juntamente com a sua estrutura hierárquica.

(a)



(b) (c)

Figura A.7: Disposição dos *Waypoints*. a) *Waypoints* no editor de cenário; b) Hierarquia de *GameObjects* demonstrando a disposição de *waypoints*; c) *Script WayPoint* adicionado aos objetos.

- Conectando regiões:

  Esta etapa visa estabelecer as relações de adjacência entre regiões e a adjacência

paternal entre sub-regiões. Ao observar-se o *Inspector* de uma região nota-se que o *scrpit* da região possui os campos *Adjacent* e *Adjacent to Parent*, ao expandi-los a variável *size* é exposta. Esta variável determina o número de nós adjacenetes esta região terá, seu valor inicial é zero, mas quando um valor natural diferente de zero é assimilado á esta variável surge uma lista de *n* elementos, cada um desses elementos deve ser preenchido com alguma região adjacente aquela região. A figura A.8 apresenta a configuração de uma região.

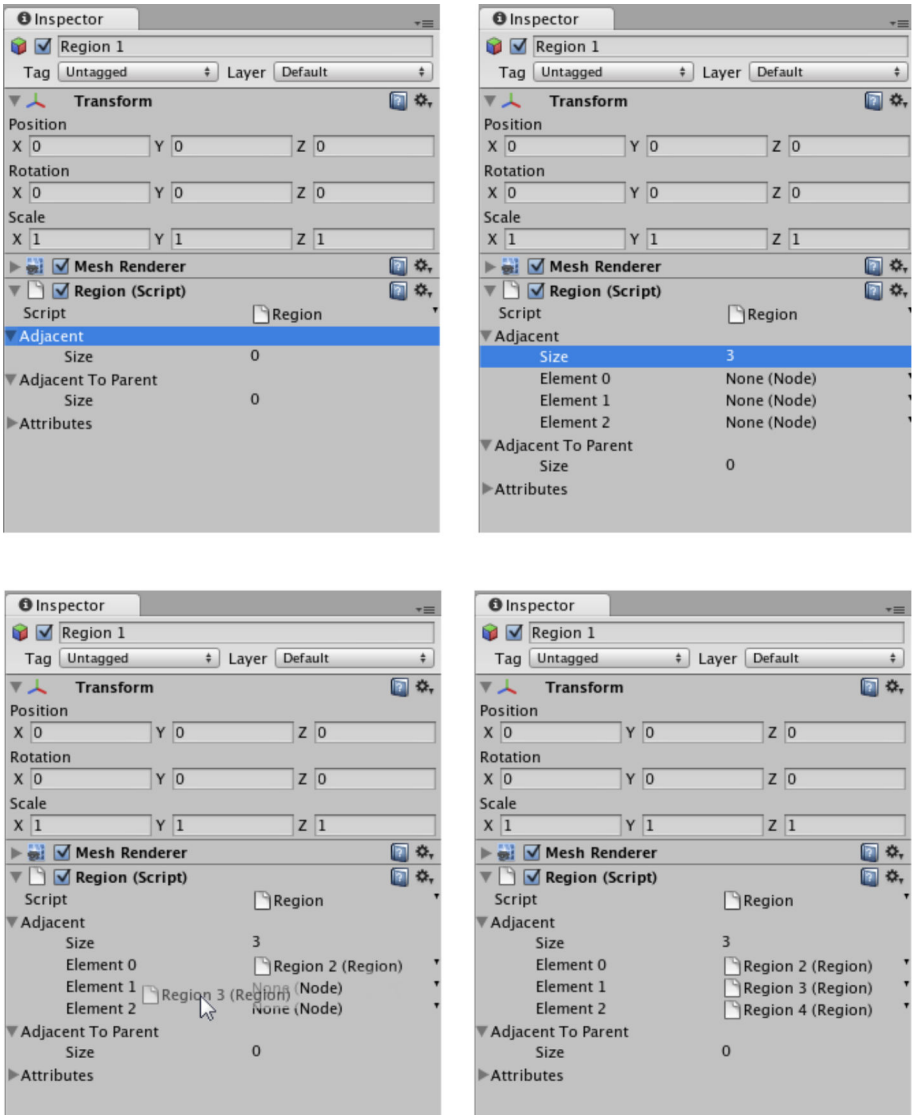
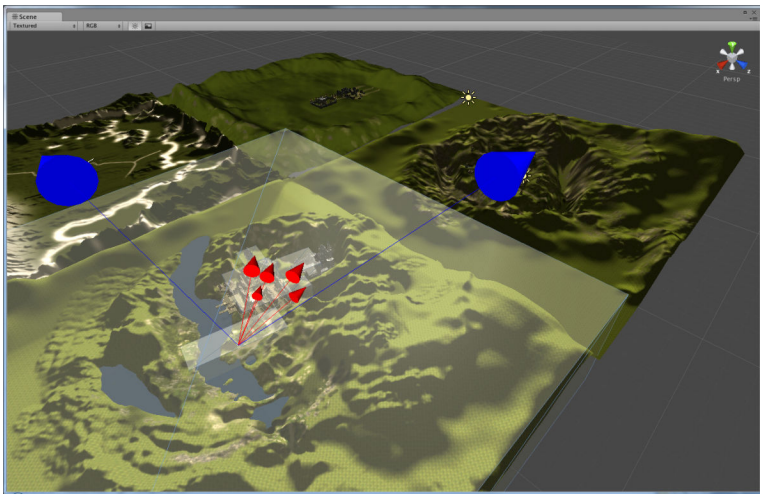


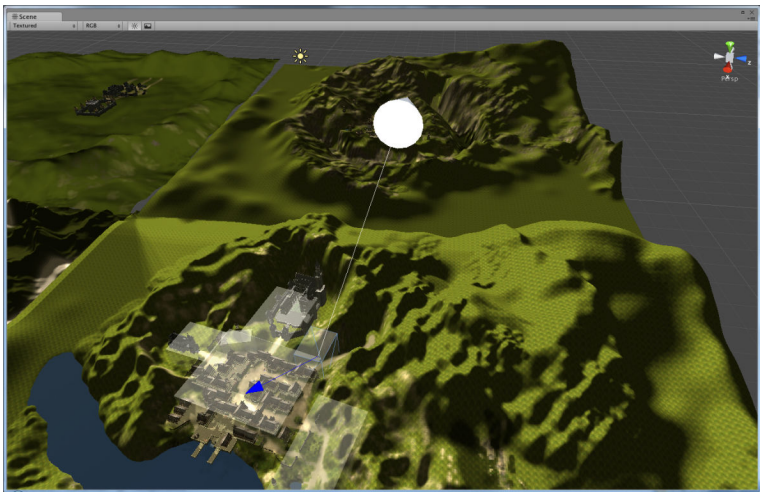
Figura A.8: Processo de conexão de regiões.

A relação de adjacência paterna é configurada da mesma maneira, entretanto ela indica regiões filhas que possuem conexões com regiões de nível maior. O *script* da região facilita a verificação das relações entre as regiões pois permite visualizar as relações entre regiões via editor, quando a aplicação inicia a sua execução

ao clicar-se em alguma região o editor exibe flechas que indicam a conectividade das regiões. Cada flecha possui uma cor: Vermelhas indicam regiões filhas, Azuis indicam regiões adjacentes, e Brancas indicam regiões com adjacência parental. A figura A.9 apresenta as flechas exibidas no editor.



(a)



(b)

Figura A.9: Exemplo de flechas apresentadas no editor. a) Vermelhas indicam paternalidade, Azuis indicam adjacência; b) Flecha branca indica adjacência paterna.

- Conectando *waypoints*:

  O processo de conexão de *waypoints* é bem semelhante ao das regiões. Apresentando, inclusive um algoritmo semelhante para visualizar a conectividade através de flechas. A figura A.10 apresenta a adjacência em *waypoints* e a figura A.11 apresenta as flechas de adjacência de *waypoints*.

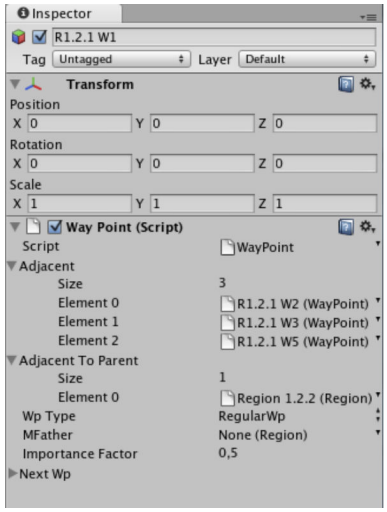


Figura A.10: Conexão de *waypoints*

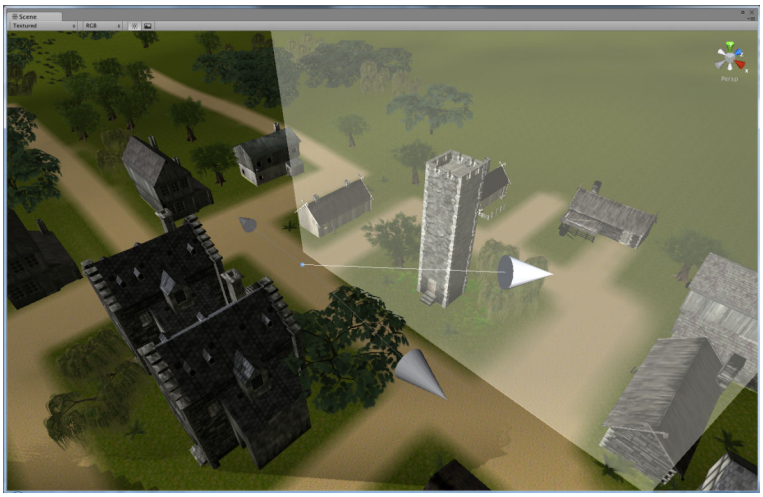


Figura A.11: Flechas representando adjacência (cinza) e adjacência paterna (branca).

- Criando e editando atributos:

  Atributos influênciam o caminho obtido pelo sistema de planejamento de caminhos. Para criar atributos novos ou modificar os atributos existentes deve-se abrir o arquivo *Treeph.cs*. Neste arquivo está presente a definição da classe *AttributesStruct*, para adicionar-se um atributo deve-se adicionar um item no enum *Attributes* com o nome do novo atributo. A seguir cria-se uma variável do tipo *public float* com o mesmo nome do atributo e inicializa-se o seu valor com 0,5f. Para finalizar modifica-se a função get e set adicionando-se dois *cases* nos respectivos *switches*. A figura A.12 apresenta o arquivo antes e depois da inserção de um novo atributo, realçando os locais onde o código foi modificado.

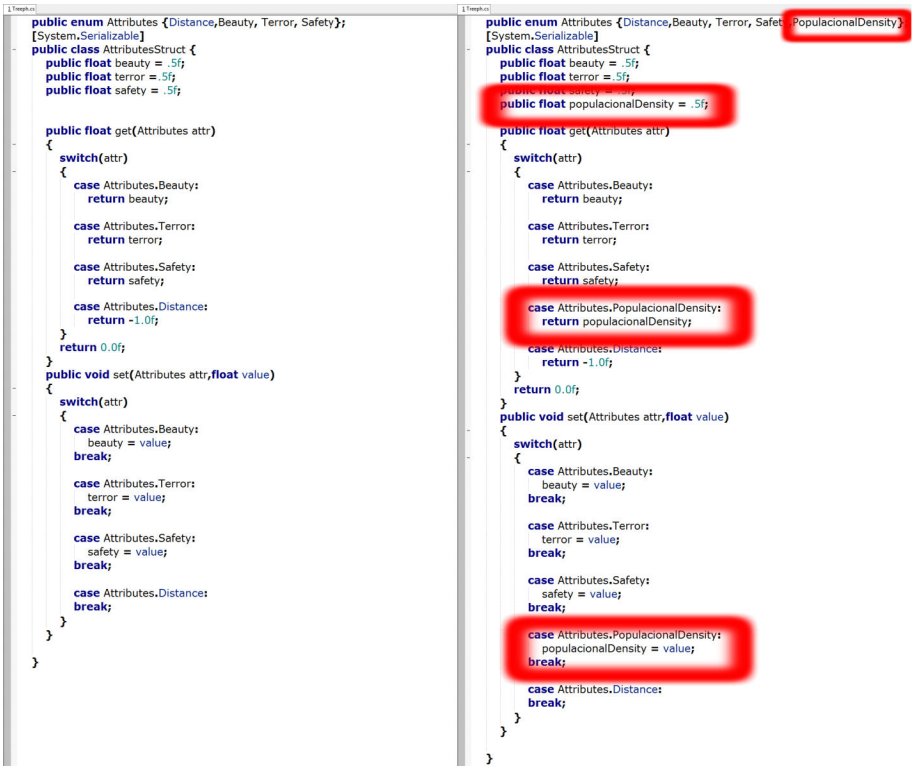


Figura A.12: Trechos de código a serem modificados ao adicionar-se novos atributos.

- Configurando atributos de cada região:

  Posteriormente a criação de atributos deve-se atribuir valores a cada atributo em regiões folha. O processo de atribuição é feito através do editor no *script* da região. Neste *script* existe um campo de nome *Attributes*, ao expandi-lo pode-se visualizar os atributos definidos para o sistema e o respectivo valor de cada um. Os valores devem ser estar entre 0 e 1 e quão menor o número maior é este atributo. Por exemplo, uma região com atributo beleza com valor 0,1 é muito mais bela que uma com valor 0,6. Somente as regiões folhas devem ser configuradas pois quando o aplicativo é inicializado, os valores das outras regiões são automaticamente calculados baseado no valor das suas regiões filhas. A figura A.13 apresenta a configuração de alguns atributos em uma região.

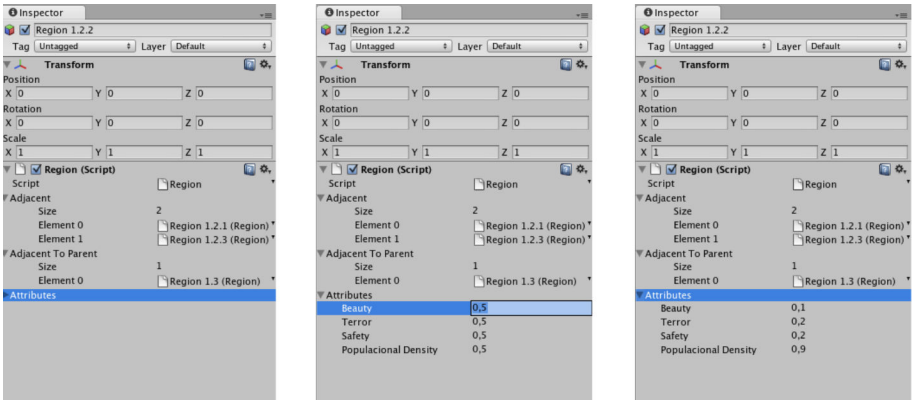


Figura A.13: Processo de configuração de atributos em regiões.

- Configurando fator de importância de *waypoints*:

  O sistema de corte de caminhos leva em consideração o fator de importância de cada *waypoint* para determinar a sua chance de aparecer no caminho final. Este fator é configurado via editor e deve possuir um valor entre 0 e 1, onde 0 indica pouca importância e 1 muita importância. A figura A.14 apresenta a configuração de um *waypoint*.

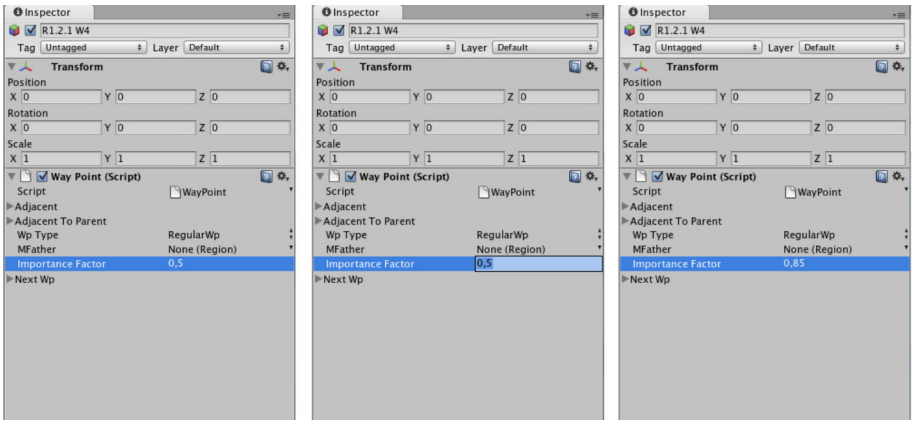


Figura A.14: Processo de configuração do fator de importância em *waypoints*.

- Configurando sistema de *Culling*:

  O sistema de *Culling* permite um ganho de desempenho na aplicação. Para criar o sistema deve-se instanciar um *GameObject* vazio a partir do menu e adicionar o *script LodController*. Este *script* possui o campo *High Region List* ao expandi-lo pode-se colocar o número de terrenos distintos que representam as regiões de mais aglto nível. A seguir arrasta-se os respectivos *GameObjects* que contém os terrenos e os objetos das regiões. A figura A.15 apresenta a configuração do *LodController*.

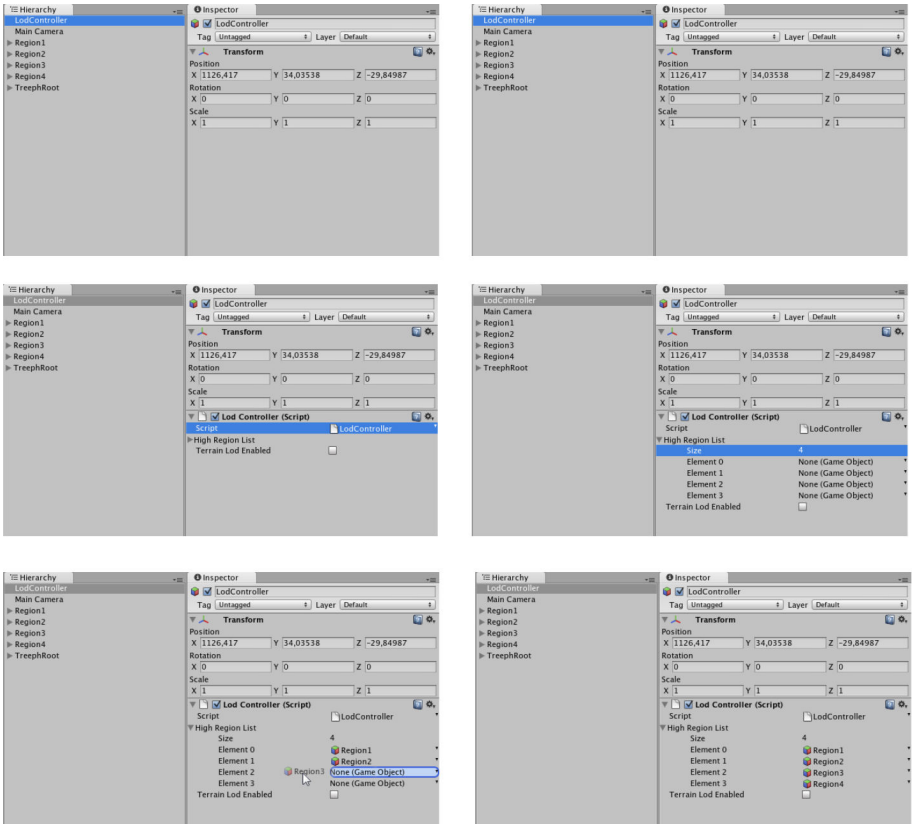


Figura A.15: Processo de configuração do *script LodController*.

A variável *Terrain Lod Enabled* pode ser marcada para que o sistema de *Culling* remova o terreno junto com o *Culling* dos objetos. Em geral deixa-se esta variável marcada, entretanto, caso o tempo de carregamento do cenário na troca de regiões se torne muito grande pode-se desabilitá-la para que esse tempo diminua, contudo, pode afetar a taxa de quadros da aplicação.

Para finalizar o processo o sistema de *Culling* deve relacionar cada cenário a sua respectiva região. Para tal deve-se adicionar um *script* em cada *GameObject* que comporta o terreno e seus objetos. O *script* é o *RegionConnector*. Após adicioná-lo ajustá-se o campo *Correspondent Region* ligando-o à respectiva região. A figura A.16 apresenta esta etapa.

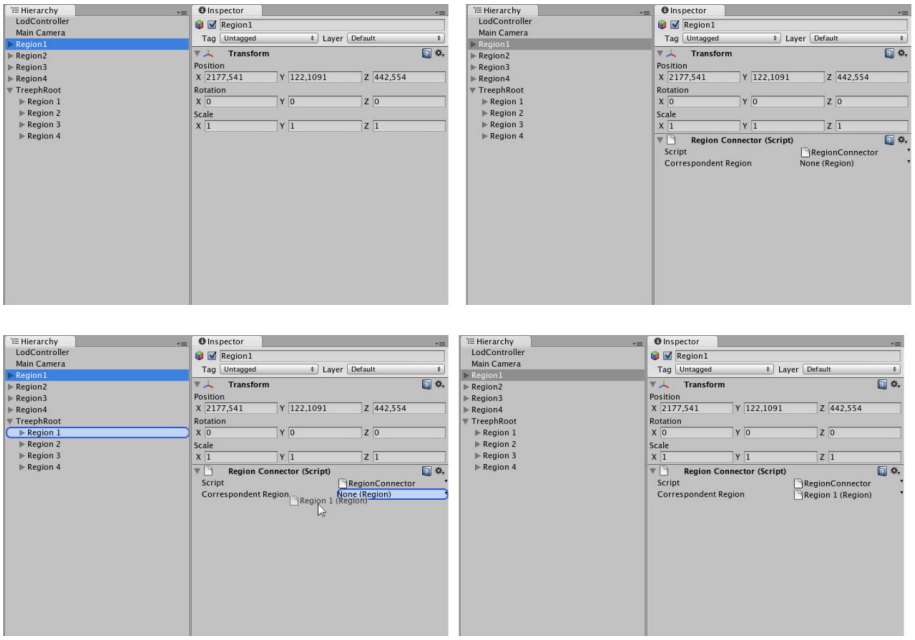
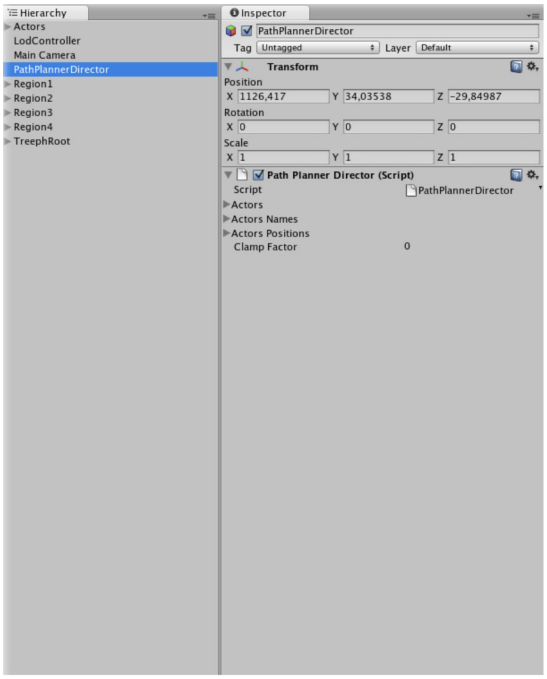


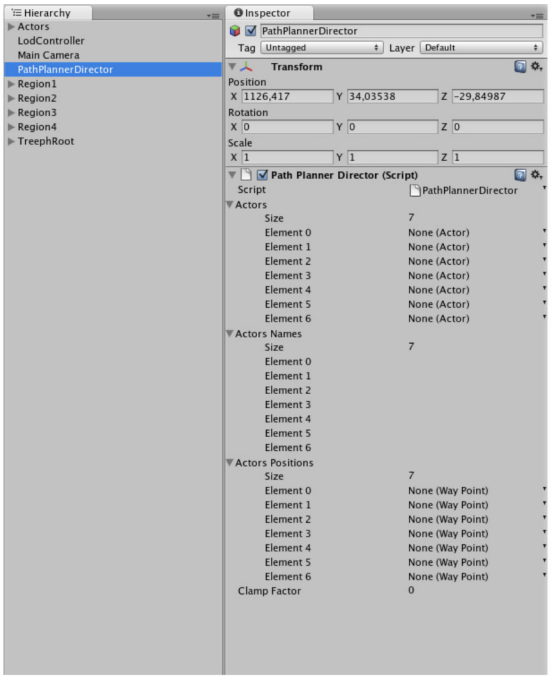
Figura A.16: Processo de configuração do *script RegionConnector*.

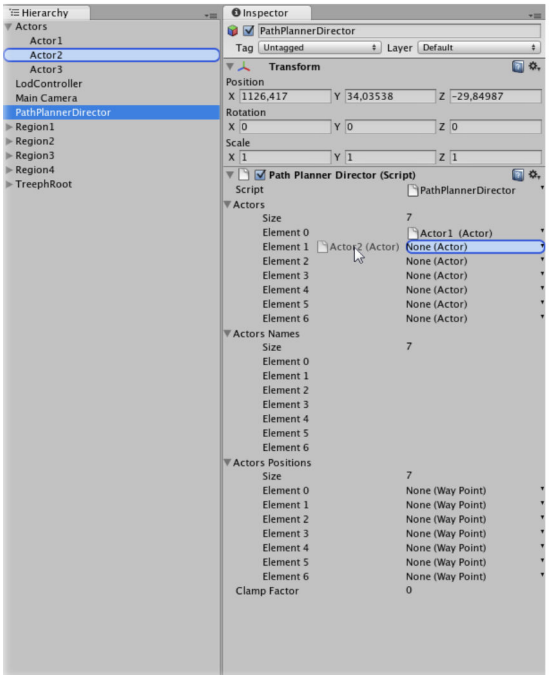
- Instanciando e configurando o *PathPlannerDirector*:

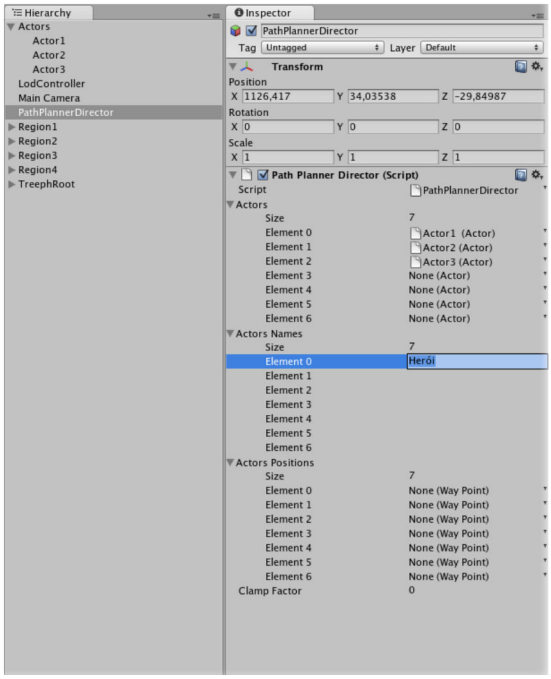
  O passo final para o estabelecimento do sistema de navegação é a configuração do *PathPlannerDirector*. Ele irá forncer a interface para que caminhos sejam obtidos e irá manter informações sobre os atores e locais do cenário. Cria-se um *GameObject* vazio através do menu e adiciona-se o *script PathPlannerDirector*. Este *script* possui quatro campos: *Actors*, coresponde a lista de personagens do sistema; *Actors Names*, corresponde a lista de nomes de atores e locais a serem produzidos pelo sistema de geração de história; *Actor Positions*, corresponde a lista de *waypoints* que cada nome disposto em *Actors Names* está posicionado; *Clamp Factor*, valor que determina o fator de corte de caminhos.

  Para preencher cada uma das listas deve-se simplemente arrastar os atores e *waypoints* e escrever os respectivos nomes. A figura A.17 apresenta a configuração do *PathPlannerDirector*.


Hierarchy
Actors
LodController
Main Camera
PathPlannerDirector
Region1
Region2
Region3
Region4
TreephRoot
Inspector
PathPlannerDirector
Tag Untagged
Layer Default
Transform
Position
X 1126,417 Y 34,03538 Z -29,84987
Rotation
X 0 Y 0 Z 0
Scale
X 1 Y 1 Z 1
Path Planner Director (Script)
Script PathPlannerDirector
Actors
Actors Names
Actors Positions
Clamp Factor 0


Hierarchy
Actors
LodController
Main Camera
PathPlannerDirector
Region1
Region2
Region3
Region4
TreephRoot
Inspector
PathPlannerDirector
Tag Untagged
Layer Default
Transform
Position
X 1126,417 Y 34,03538 Z -29,84987
Rotation
X 0 Y 0 Z 0
Scale
X 1 Y 1 Z 1
Path Planner Director (Script)
Script PathPlannerDirector
Actors
Size 7
Element 0 None (Actor)
Element 1 None (Actor)
Element 2 None (Actor)
Element 3 None (Actor)
Element 4 None (Actor)
Element 5 None (Actor)
Element 6 None (Actor)
Actors Names
Size 7
Element 0
Element 1
Element 2
Element 3
Element 4
Element 5
Element 6
Actors Positions
Size 7
Element 0 None (Way Point)
Element 1 None (Way Point)
Element 2 None (Way Point)
Element 3 None (Way Point)
Element 4 None (Way Point)
Element 5 None (Way Point)
Element 6 None (Way Point)
Clamp Factor 0


Hierarchy
Actors
Actor1
Actor2
Actor3
LodController
Main Camera
PathPlannerDirector
Region1
Region2
Region3
Region4
TreephRoot
Inspector
PathPlannerDirector
Tag Untagged
Layer Default
Transform
Position
X 1126,417 Y 34,03538 Z -29,84987
Rotation
X 0 Y 0 Z 0
Scale
X 1 Y 1 Z 1
Path Planner Director (Script)
Script PathPlannerDirector
Actors
Size 7
Element 0 Actor1 (Actor)
Element 1 Actor2 (Actor) None (Actor)
Element 2 None (Actor)
Element 3 None (Actor)
Element 4 None (Actor)
Element 5 None (Actor)
Element 6 None (Actor)
Actors Names
Size 7
Element 0
Element 1
Element 2
Element 3
Element 4
Element 5
Element 6
Actors Positions
Size 7
Element 0 None (Way Point)
Element 1 None (Way Point)
Element 2 None (Way Point)
Element 3 None (Way Point)
Element 4 None (Way Point)
Element 5 None (Way Point)
Element 6 None (Way Point)
Clamp Factor 0


Hierarchy
Actors
Actor1
Actor2
Actor3
LodController
Main Camera
PathPlannerDirector
Region1
Region2
Region3
Region4
TreephRoot
Inspector
PathPlannerDirector
Tag Untagged
Layer Default
Transform
Position
X 1126,417 Y 34,03538 Z -29,84987
Rotation
X 0 Y 0 Z 0
Scale
X 1 Y 1 Z 1
Path Planner Director (Script)
Script PathPlannerDirector
Actors
Size 7
Element 0 Actor1 (Actor)
Element 1 Actor2 (Actor)
Element 2 Actor3 (Actor)
Element 3 None (Actor)
Element 4 None (Actor)
Element 5 None (Actor)
Element 6 None (Actor)
Actors Names
Size 7
Element 0 Herói
Element 1
Element 2
Element 3
Element 4
Element 5
Element 6
Actors Positions
Size 7
Element 0 None (Way Point)
Element 1 None (Way Point)
Element 2 None (Way Point)
Element 3 None (Way Point)
Element 4 None (Way Point)
Element 5 None (Way Point)
Element 6 None (Way Point)
Clamp Factor 0

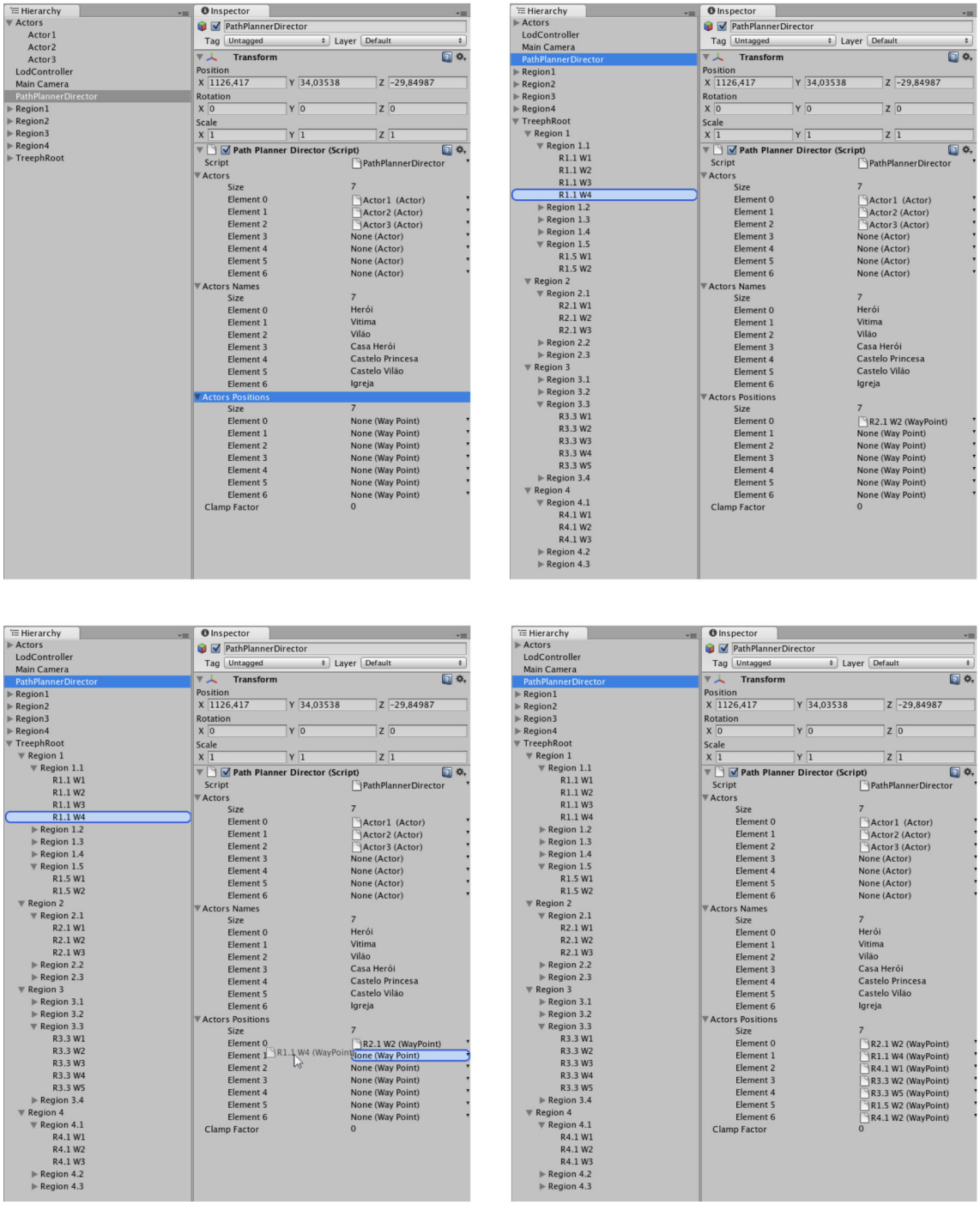


Figura A.17: Processo de configuração do *script PathPlannerDirector*.