

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Sistema para suporte de narrativas, acesso, partilha e visualização de informação multimédia

Ricardo Jorge Freire Dias

Relatório apresentado para a obtenção do Grau de Licenciado em Engenharia Informática pela Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia.

Lisboa, 25 de Setembro de 2007

Resumo

Este trabalho, desenvolvido no âmbito da disciplina de Projecto, teve como objectivo implementar um sistema para suporte de narrativas, acesso, partilha e visualização de informação. Para tal foi necessário desenvolver uma série de componentes que formam a parte *server-side* de um sistema que envolve também dispositivos móveis e informação geo-referenciada. Os componentes desenvolvidos foram: duas bases de dados de suporte ao sistema em questão, uma para o acesso e partilha de informação e a outra para as narrativas; uma interface de gestão *web* para inserir dados na base de dados de conteúdos informativos; um servidor que gere o acesso e partilha de informação assim como os eventos de jogo que constituem uma narrativa. Este servidor interage com os dispositivos móveis através de tecnologia de *web services*. Por fim, foi desenvolvida uma interface 3D para permitir a visualização e controlo de informação do sistema.

O trabalho foi maioritariamente implementado na linguagem C++, à excepção da interface *web* que foi implementada em PHP e as bases de dados que foram implementadas em SQL, num sistema de base de dados designado por PostgreSQL.

Em relação às narrativas interactivas, foi definida uma linguagem de descrição de narrativas assim como um motor de interpretação e execução da linguagem criada.

Conteúdo

1	Introdução	1
1.1	Contexto Académico	2
1.2	Descrição e Objectivos do Trabalho	2
1.3	Trabalho Relacionado	3
1.4	Mapa do Relatório	6
2	Arquitectura do Sistema	7
2.1	Arquitectura Global	8
2.2	Comunicação entre Componentes	9
2.3	Base de Dados	9
2.4	Interface Web	10
2.5	Servidor de Jogo	10
2.6	Interface 3D	10
3	Bases de Dados do Sistema	12
3.1	Base de Dados dos Conteúdos Informativos	13
3.1.1	Descrição das Entidades	16
3.2	Base de Dados dos Conteúdos do Jogo	20
3.2.1	Descrição das Entidades	22
3.3	Interface Web Service da Base de Dados	26
3.4	Biblioteca de Comunicação com PostgreSQL – PSQLib++	27
4	Interface Web	29
4.1	Apresentação da Interface Web	30
4.2	Implementação	33

5 Servidor de Jogo	36
5.1 Descrição das Funcionalidades do Servidor	37
5.1.1 Gestão de Sessões de Utilizadores	37
5.1.2 Gestão de Conteúdos Informativos	37
5.1.3 Gestão de Eventos de Jogo	37
5.1.4 Servidor de Web Services	38
5.2 Implementação dos Módulos	38
5.2.1 Implementação da Gestão de Sessões e Utilizador	38
5.2.2 Implementação da Gestão de Conteúdos Informativos	40
5.2.3 Implementação do Servidor de Web Services	40
5.3 Descrição e Definição do Motor de Jogo	41
5.3.1 Definição do Jogo	41
5.3.2 Identificação de Conceitos	42
5.3.3 Relações entre Conceitos	44
5.4 Implementação do Motor de Jogo	45
5.4.1 Definição da Linguagem EventLang	45
5.4.2 Interpretação da Linguagem EventLang	47
5.4.3 Execução dos Eventos de Jogo	57
5.4.4 Classes Dinâmicas	59
5.4.5 Implementação do Estado do Mundo e Informação dos Actores	59
5.5 Exemplo de Funcionamento de um Jogo	60
6 Interface de Visualização e Controlo 3D	67
6.1 Apresentação da Interface 3D e Funcionalidades	68
6.2 Implementação	69
6.2.1 Gestor do Espaço	71
6.2.2 Interface 2D	71
6.2.3 Entidade	71
6.2.4 Gestor de Sincronização	72
6.2.5 Parâmetros da Interface	72
7 Conclusões	73
7.1 Conclusões	74
7.2 Trabalho Futuro	74

Listas de Figuras

2.1	Diagrama de camadas da arquitectura do sistema	8
2.2	Diagrama de comunicação entre componentes.	9
3.1	Diagrama de entidades e relações da base de dados de conteúdos informativos.	14
3.2	Diagrama de entidade e relações dos <i>Layouts</i> de informação.	15
3.3	Diagrama de entidade e relações do histórico de inserções de conteúdos informativos.	15
3.4	Modelo de entidades e relações da base de dados do jogo.	21
3.5	Diagrama de classes da biblioteca PSQLib++.	28
4.1	Layout principal da interface web.	30
4.2	Layout da interface web durante a criação de um local.	31
4.3	Layout da interface web durante a criação de uma imagem.	32
4.4	Layout da interface web durante a visualização de uma imagem.	32
4.5	Layout da interface web durante a visualização de todas as imagens.	33
5.1	Diagrama de classes da gestão de sessões.	39
5.2	Árvore lógica dos <i>triggers</i> e pré-condições de dois eventos	43
5.3	Diagrama de classes da árvore lógica das condições	51
5.4	Manipulação da estrutura de dados dos eventos	58
5.5	Classes abstractas que definem as condições e os tipos de evento	60
5.6	Diagrama de classes do Estado do Mundo	61
5.7	Diagrama de classes da informação dos actores	62
6.1	Interface de Visualização 3D.	69
6.2	Diagrama de classes das classes mais importantes da Interface 3D.	70

Lista de Tabelas

4.1	Tabela de mapeamento entre tipos de atributos de tabelas SQL e componentes HTML de inserção de dados.	35
5.1	Tabela de exemplos de cada literal presente na gramática da linguagem EventLang.	50

Lista de Algoritmos

1	EventTree::CREATETREE	53
2	AndNode::EVALUATE	53
3	OrNode::EVALUATE	54
4	NotNode::EVALUATE	54
5	ConditionNode::EVALUATE	54
6	AndNode::HASSONS and OrNode::HASSONS	54
7	NotNode::HASSONS	54
8	ConditionNode::HASSONS	55
9	AndNode::REDUCTION	55
10	OrNode::REDUCTION	55
11	NotNode::REDUCTION	55
12	ConditionNode::REDUCTION	55
13	AndNode::ASSOCDEPEVENTS	55
14	OrNode::ASSOCDEPEVENTS	56
15	NotNode::ASSOCDEPEVENTS	56
16	ConditionNode::ASSOCDEPEVENTS	56
17	AndNode::CLEANSTATE	56
18	OrNode::CLEANSTATE	56
19	NotNode::CLEANSTATE	57
20	EventTree::RESOLVETRANSITIVITY	57
21	EventNode::RESOLVETRANSITIVITY	57
22	EventNode::PATHTO	58

1

Introdução

Conteúdo

1.1	Contexto Académico	2
1.2	Descrição e Objectivos do Trabalho	2
1.3	Trabalho Relacionado	3
1.4	Mapa do Relatório	6

Neste capítulo apresentam-se o resumo e os objectivos deste trabalho, que incide num sistema de suporte a narrativas interactiva e também de acesso e partilha de informação mulmédia que será implementado num local designado por Quinta da Regaleira. Por último, apresenta-se um plano das secções do relatório.

1.1 Contexto Académico

Este relatório foi escrito no contexto da disciplina de Projecto final de curso da licenciatura em Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa. Trata-se de uma disciplina semestral de 24 ECTS, tendo prevista uma carga horária semanal de cerca de 25 horas. Neste estágio a carga horária semanal média foi de 40 horas.

O estágio decorreu durante o período de 2 de Abril de 2007 até 28 de Setembro de 2007.

A orientação pedagógica deste projecto esteve a cargo do Professor Nuno Manuel Robalo Correia do Departamento de Informática e coordenador do *Interactive Multimedia Group* (IMG).

O projecto decorreu no *Interactive Multimedia Group* que faz parte do CITI (Centro de Informática e Tecnologias de Informação), que se encontra no Departamento de Informática. Este grupo tem como objectivo desenvolver investigação na área de processamento, interacção e apresentação de informação multimédia.

1.2 Descrição e Objectivos do Trabalho

Este trabalho insere-se no âmbito do projecto Regaleira-InStory, co-financiado pela Fundação Cultural Sintra e pelo programa POS-C. O projecto InStory tem o objectivo de implementar uma plataforma para suporte de narrativas interactivas, acesso e partilha de informação. A plataforma tem de ser o mais flexível possível pois tem de integrar vários tipos de dispositivos computacionais incluindo computadores *desktop*, PDA's e telemóveis.

O projecto InStory é desenvolvido num espaço cultural designado por Quinta da Regaleira, em Sintra. Este espaço é destinado maioritariamente a turistas. É um local com algum misticismo associado, ideal para a criação de histórias. Este local é muito rico em termos arquitectónicos pois possui vários monumentos, como o palácio da Regaleira, o Pátio dos Deuses, entre outros. Possui ainda vários jardins e grutas.

O objectivo deste trabalho é de ajudar os visitantes da Quinta da Regaleira durante a sua visita. Esta ajuda é feita através de um PDA ou de um telemóvel que esteja na posse do visitante. Durante a visita, o visitante irá receber informação do local onde se encontra. Por exemplo, se estiver a chegar ao palácio da Regaleira, automaticamente é enviada ao visitante, informação sobre o palácio. Esta informação contextual é constituída por texto e auxiliada por imagens, vídeos e sons.

Outra opção que o visitante tem ao seu dispor é de participar num jogo enquanto passeia pelo espaço. O objectivo é que o visitante, de alguma forma, seja levado ao encontro dos pontos mais importantes da Quinta da Regaleira enquanto se diverte a seguir o rumo do jogo. O jogo em causa é uma implementação de uma narrativa interactiva que difere de outro tipo de jogos pois, neste caso o visitante é o protagonista da história e pode, ao longo do jogo, fazer determinadas escolhas que influenciarão o rumo da história daí para a frente podendo inclusivé até mudar o respectivo final.

Um visitante que participe no jogo, terá que responder a perguntas, seguir pistas escondidas nos vários locais da Quinta da Regaleira, conversar com actores virtuais ou combater com eles, usando objectos que vai adquirindo ao longo do percurso de jogo.

Para concretizar os objectivos descritos, é necessário implementar uma série de componentes. Este trabalho irá implementar os seguintes componentes:

- **Base de Dados:** Serão implementadas duas bases de dados, uma para os conteúdos informativos que

são mostrados aos visitantes e outra para os conteúdos do jogo.

- **Interface Web:** Esta interface permite inserir dados dos conteúdos informativos.
- **Servidor de Jogo:** Este servidor gera todo o sistema, incluindo os conteúdos informativos e o progresso do jogo de cada visitante.
- **Interface 3D:** Esta interface serve para visualizar o estado do sistema, por exemplo a posição dos visitantes, que estão a participar num jogo, num mapa virtual.

As bases de dados que vão ser implementadas servem para guardar a informação que vai ser enviada para o visitante, quando este está perto de um local de interesse, e também o conteúdo dos eventos de jogo. A interface *web* tem como função facilitar a inserção dos dados na base de dados dos conteúdos informativos, principalmente na questão dos conteúdos multimédia, como imagens, vídeos e sons. O servidor de jogo é o *cérebro* do sistema. Este gera os utilizadores e as respectivas interacções com o conteúdo informativo e com os eventos do jogo. A interface 3D tem a função de permitir a visualização e controlar os conteúdos do sistema que se encontram nas bases de dados. Todos os componentes têm de interagir entre si, sendo necessária a criação de interfaces de comunicação bem definidas.

No projecto InStory está incluída a parte colaborativa dos visitantes, em que estes aumentam o repositório multimédia, enviando fotografias e vídeos. Também está incluída a criação de grupos sociais entre os visitantes o que permite explorar melhor a vertente social deste tipo de sistemas. Estes objectivos não estão incluidos no trabalho realizado, no âmbito deste projecto.

Em relação às narrativas, irá ser criada uma linguagem para as descrever. Esta linguagem será baseada numa descrição de composição de eventos que constituirão a narrativa dinâmica pretendida. Estes eventos serão constituídos por condições de despoletamento e por acções. O motor de jogo terá a função de saber interpretar a linguagem em questão e de executar os eventos pela ordem correcta.

1.3 Trabalho Relacionado

Narrativa Interactiva é um termo inventado por Chris Crawford. Este define narrativa interactiva como “Uma forma de entretenimento interactivo em que o jogador toma o papel de protagonista num ambiente de grande riqueza dramática”.

O conceito de utilizar narrativas aliadas a um sistema de jogo, de forma a que o utilizador faça também parte da história, surgiu nos anos 60 com o nome de Dungeons & Dragons e deu origem aos chamados *role playing games* (RPG's). De forma genérica, um RPG junta um grupo de jogadores para participarem activamente numa história contada pelo narrador, referido como *game-master* ou *dungeon-master* (DM). Cada jogador participa na aventura através de uma personagem ou alter-ego e as suas acções são adjudicadas pelo DM com base num sistema de regras e lançamentos de dados. Os jogadores recebem pontos pelas suas acções e esses pontos marcam a evolução da personagem e permitem-lhe desenvolver novas capacidades que possam ser úteis na solução de puzzles.

Inicialmente os jogos de aventura baseavam-se em papel, lápis e dados e necessitavam de um DM para controlar a evolução da aventura e as escolhas disponíveis aos jogadores. Posteriormente, o mesmo conceito foi adaptado para diferentes tipos de meio. Uma das primeiras adaptações consistiu numa série de livros onde o leitor decidia o passo seguinte de entre um número muito limitado de escolhas. Este sistema incluía combates, testes de perícia, força e até mesmo de sorte, tornando-se rapidamente popular como meio de

incentivar o gosto pela leitura. Alguns jogos de tabuleiro também resultaram de adaptações do conceito ao meio, como é o caso do jogo “Cluedo”.

Com a chegada da era do computador pessoal surgiram muitas adaptações e variantes do conceito original, sempre limitadas à capacidade computacional da altura. Desde as aventuras puramente baseadas em texto e MUD’s (*multi-user dungeons*) até à era de ouro das aventuras gráficas como a série Monkey Island (www.lucasarts.com), diversas aproximações ao conceito foram sendo apresentadas ao longo dos anos.

Na última década os RPG’s ressurgiram em força e, aliados à generalização dos acessos à Internet, deram origem ao fenómeno Massive Online Role-playing Game. O conceito original foi adaptado aos dias de hoje, substituindo o papel, caneta, dados e DM por um sistema computacional e aproximando jogadores de vários países e culturas. Exemplos actuais incluem os jogos Neverwinter Nights (www.bioware.com), World of Warcraft (www.blizzard.com) e Final Fantasy XI (www.square-enix.com).

As novas tecnologias digitais móveis têm contribuído grandemente para o desenvolvimento das narrativas digitais. Actualmente, a disponibilidade de informação digital geo-referenciada permite a criação de aplicações que, reconhecendo automaticamente a localização dos utilizadores, respondem adequadamente [9]. Em [12] é dada uma visão global das técnicas básicas de localização e apresentada uma taxonomia das propriedades dos sistemas de localização, incluindo uma compilação dos sistemas de localização existentes quer a nível comercial quer no campo da investigação. O artigo [8] analisa os diferentes meios de representação de informação espacial, apresentando e comparando diferentes tipos de modelos de localização.

O trabalho desenvolvido por Cavazza, Charles e Mead [2–4] tem como objectivo permitir que histórias com uma estrutura narrativa bem definida tenham o seu final alterado em consequência da interação do utilizador. Os autores descrevem um sistema em que o utilizador assiste a uma história e tem a oportunidade de influenciar o desenrolar da narrativa. Com o objectivo de manter a narrativa gerada coerente com o enredo imaginado pelo autor da história, as possibilidades de interacção são intencionalmente reduzidas. O papel do utilizador limita-se a mover objectos pelo cenário e sugerir comportamentos aos personagens. O autor da história descreve o comportamento de cada um dos personagens principais através de um grafo, que define hierarquicamente os possíveis planos que o personagem pode seguir para atingir seu objectivo na narrativa. As acções efectivamente executadas pelos personagens durante a história são calculadas por um algoritmo de planeamento que opera sobre este grafo.

Chris Crawford [5] criou um projecto designado por Storytron com o objectivo de criar ferramentas de *authoring* acessíveis a pessoas sem conhecimentos técnicos. No sistema de *authoring* SWAT, o autor da história define um conjunto de verbos (acções que podem ser executadas pelos personagens). Cada verbo possui associado uma lista de papéis que podem ser assumidos por diferentes personagens durante a sua execução. Por exemplo, o verbo “ofender” poderia ter associado dois papéis: o personagem ofendido e uma testemunha da ofensa. Para cada um destes papéis, há uma lista de verbos representando as possíveis reações. Assim, em resposta à ofensa, a personagem ofendida poderia “responder à ofensa” ou “ignorar a ofensa”. Para cada uma destas possíveis reações, o autor da história deve definir uma equação de inclinação, que determina a probabilidade de qual das reacções é executada. Este sistema também inclui outras funcionalidades incluindo um modelo de personalidade para os personagens e um algoritmo de bisbilhotice (*gossip*).

Leandro Barros [1] desenvolveu um sistema designado por Fabulator. Neste sistema uma história é constituída por uma sequência de acções, executadas por personagens, que são capazes de transformar o mundo. As sequências de acções são geradas por um algoritmo de planeamento. O algoritmo de planea-

mento trata o problema como se fosse um problema de pesquisa em espaço de estados e utiliza o algoritmo A* para o resolver. O utilizador controla um personagem que corresponde ao protagonista da história e todos os outros personagens são controlados pelo sistema. Sempre que o utilizador execute acções que invalidem o plano inicial é executado novamente o algoritmo de planeamento para gerar um novo plano.

O sistema Teatrix [14] foi desenvolvido com o objectivo de criação colaborativa de histórias por crianças. No Teatrix, alguns personagens da história são controlados pelas crianças que usam o sistema. Os outros personagens são agentes autónomos. Também existe um agente “director” omnisciente que pode inserir novos objectos e novas personagens, no mundo da história, controlando as suas acções no interesse de manter a coerência da história. O agente “director” não pode controlar as personagens controladas pelas crianças.

Na área dos sistemas baseados em localização, diversas experiências têm sido executadas. TOI - Traveller's On-line Information System foi um dos primeiros sistemas móveis de informação multimédia geo-referenciada e contextualizada a ser concebido [18]. O Georgia Tech desenvolveu o CyberGuide [10]. O MIT Laboratory for Computer desenvolveu um sistema de localização para aplicações móveis baseadas em localização e destinadas a espaços fechados [17]. A National Central University criou o protótipo de um guia turístico digital com conhecimento de localização [11]. A Universidade de Lancaster e a Universidade do Arizona desenvolveram um guia electrónico de Lancaster para dispositivos “handheld” [7]. A capacidade de adaptar um serviço às necessidades dos utilizadores torna-se cada vez vez mais importante, nomeadamente no que se refere às aplicações multimédia móveis.

No artigo [13] é descrito um protótipo de um sistema de narrativas digitais baseado nas tecnologias de jogos 3D e interfaces tangíveis. Este sistema proporciona novos meios de organizar, reutilizar e partilhar sequências de video, de forma interactiva e não linear, permitindo a re-experiência interactiva.

O projecto Exocog usa a Internet como um meio para implementar narrativas digitais interactivas [15]. Neste caso de estudo é criado um ambiente fictício e contada uma história através da manipulação do conteúdo verosímil de Web sites reais, permitindo analisar como as características especiais da Internet podem afectar e modificar a forma como se criam, contam e vivem as histórias (www.exocog.com). Os utilizadores podem acompanhar e influenciar o desenrolar de uma história, explorando e relacionando a informação disponibilizada, resolvendo enigmas e analisando o comportamento dos personagens através dos sites do jogo. O Exocog funde as características de um jogo e de uma narrativa digital, exigindo um comportamento activo por parte dos utilizadores.

No MIT foi desenvolvido um sistema para criação e participação em narrativas móveis contextualizadas [6, 16]. Este sistema permite aos utilizadores experimentarem o Mobile Cinema, que os leva numa viagem pelo espaço físico que os envolve, enquanto pedaços de uma história (na forma de sequências de video) vão surgindo no seu PDA, de acordo com os movimentos que executam e a respectiva localização. O sistema inclui ferramentas para definir a sequência da história e para a associar aos espaços físicos. Este sistema serviu de inspiração ao trabalho apresentado no presente artigo. Com base nos conceitos desenvolvidos no âmbito do projecto M-Views, pretendemos construir um novo sistema e expandi-lo através da integração de mecanismos adicionais para colaboração, registo, acesso a informação e outros tipos de actividades, incluindo jogos, que possuam diferentes requisitos computacionais.

1.4 Mapa do Relatório

O relatório está organizado da seguinte forma: no próximo capítulo será apresentada a arquitectura do sistema, descrevendo os vários componentes e as respectiva interacções. O capítulo seguinte descreve a implementação das bases de dados dos conteúdos informativos e de jogo. O capítulo 4 apresenta as funcionalidades da interface *web* e a respectiva implementação. O servidor de jogo que suporta as narrativas interactivas e outras actividades, é descrito em mais detalhe de seguida. No capítulo seguinte é apresentado a interface de visualização e controlo 3D, utilizada para visualizar o estado do sistema e por ultimo são apresentadas as conclusões e as direcções para trabalho futuro, com as quais se conclui o relatório.

2

Arquitectura do Sistema

Conteúdo

2.1	Arquitectura Global	8
2.2	Comunicação entre Componentes	9
2.3	Base de Dados	9
2.4	Interface Web	10
2.5	Servidor de Jogo	10
2.6	Interface 3D	10

Neste capítulo iremos descrever a arquitectura global do sistema, assim como a arquitectura de cada componente do sistema.

A arquitectura deste sistema é cliente-servidor. Existe um servidor central que gera todo o tipo de conteúdos e disponibiliza uma interface, baseada em Web Services, para as comunicações com os clientes. Os clientes do sistema são bastante heterogéneos em termos de plataformas, incluindo: telemóveis, PDA's e computadores desktop.

2.1 Arquitectura Global

A arquitectura deste sistema é constituída por vários componentes, alguns implementados no contexto deste trabalho. O principal é um servidor central que controla a gestão de conteúdos multimédia, gestão de utilizadores e também o motor de jogo.

Este servidor guarda as informações numa base de dados que não tem de estar necessariamente no mesmo espaço físico que o próprio servidor. A razão da desacoplação entre a base de dados e o servidor é justificada pelo facto de a base de dados guardar informação que é utilizada pelos vários componentes do sistema e desta forma o servidor não é sobrecarregado pelos acessos à base de dados.

Também na arquitectura deste sistema, se incluem os clientes do jogo, que tanto podem ser PDA's como telemóveis, uma interface gráfica 3D e um site *web* com uma interface de gestão para os conteúdos multimédia da base de dados.

Na figura 2.1 podemos ver um diagrama de camadas da arquitectura do sistema com todos os componentes.

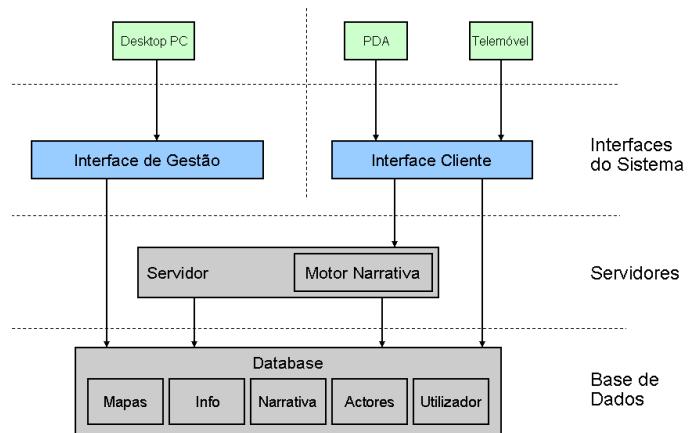


Figura 2.1: Diagrama de camadas da arquitectura do sistema

Os clientes implementados, actualmente são dois: um cliente para PDA já implementado e um cliente para telemóvel que ainda está a ser implementado. Estes clientes comunicam directamente com o servidor e são estes que funcionam como interface entre o utilizador e o jogo.

A interface gráfica 3D é utilizada para visualizar e controlar o estado do jogo. Esta utiliza a base de dados directamente para apresentar o estado do jogo e comunica com o servidor para interagir de forma a mudar algumas propriedades dos estado do jogo.

O site *web* tem como finalidade introduzir e visualizar os conteúdos multimédia respeitantes à Quinta da Regaleira, que são utilizados para informação contextual no nosso sistema.

De todos os componentes apresentados, apenas os clientes não foram implementados no contexto deste trabalho.

2.2 Comunicação entre Componentes

A comunicação entre os diferentes componentes do sistema foi fortemente influenciada pela heterogeneidade dos mesmos. Esta heterogeneidade dos componentes obrigou-nos a escolher um protocolo de comunicação de implementação fácil e ao mesmo tempo que satisfizesse os requisitos de eficiência do sistema. O protocolo escolhido teria que funcionar entre componentes de diferentes arquitecturas e diferentes sistemas de operação, e também teria de funcionar tanto em redes locais como redes de larga escala (*Internet*).

Dadas estas condições, (ou restrições), escolhemos como protocolo de comunicação a tecnologia de *web services*. Os *web services* são baseados em *remote procedure calls* em que os respectivos *stubs* serializam os dados dos parâmetros em XML e são transportados na rede utilizando o protocolo HTTP. Com estas características dos *web services* resolve-se o problema da heterogeneidade e da comunicação em redes de larga escala.

Para utilizar os *web services* como protocolo de comunicação usámos a ferramenta gSoap para gerar os *stubs* e *skeletons*. Esta ferramenta gera código em C++ a partir da descrição de um *web service* numa linguagem de descrição de *web services* denominada por WSDL. A razão principal para a utilização desta ferramenta é que esta ferramenta está optimizada para dispositivos de fraco processamento e pouca memória, como os PDA's, e também está optimizada para usar menos largura de banda nas comunicações [18].

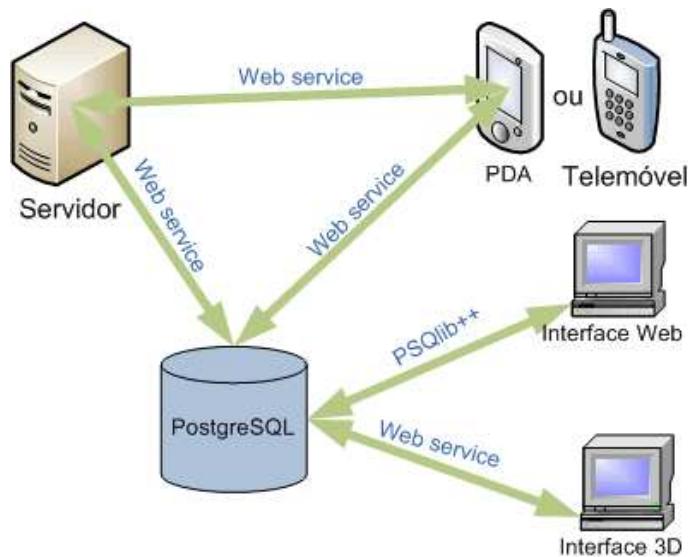


Figura 2.2: Diagrama de comunicação entre componentes.

2.3 Base de Dados

A base de dados do sistema foi implementada sobre um sistema de gestão de base de dados designado por PostgreSQL. A base de dados implementada, na verdade, corresponde a duas bases de dados, uma orientada para o conteúdo informativo da Quinta da Regaleira e a outra orientada para o conteúdo necessário à gestão do jogo interactivo.

A base de dados dos conteúdos informativos foi desenvolvida com duas finalidades: servir como

base de dados para interfaces *web* desenvolvidas pela Quinta da Regaleira e servir como repositório dos conteúdos informativos mostrados pela aplicação InStory. A base de dados dos conteúdos do jogo foi desenvolvida com a finalidade de dar suporte ao motor de jogo guardando todas as informações necessárias à manutenção do seu estado.

Sobre a base de dados encontra-se uma camada de software em C++ que implementa uma interface de comunicação com o sistema de base dados baseada em *web services*. A comunicação entre esta camada e o próprio sistema de gestão de base de dados é feita usando a biblioteca desenvolvida no âmbito deste projecto, implementada em C++, que funciona como *wrapper* para a interface do PostgreSQL na linguagem C.

No capítulo 3, iremos primeiro abordar a base de dados dos conteúdos informativos da Quinta da Regaleira e, de seguida, a base de dados dos conteúdos do jogo.

2.4 Interface Web

A interface *web* foi desenvolvida com o âmbito de ajudar na inserção de dados na base de dados respeitante aos conteúdos informativos da Quinta da Regaleira.

A interface foi desenvolvida em PHP e comunica directamente com o sistema de gestão de base de dados PostgreSQL. Esta permite a inserção, remoção e visualização de dados na respectiva base de dados.

No capítulo 4, iremos apresentar os pormenores de implementação da interface.

2.5 Servidor de Jogo

O servidor que controla o jogo interativo foi implementado em C++. Este servidor comunica com os clientes móveis (PDA's e telemóveis), e com a base de dados através de um interface baseada em *web services*. Este servidor implementa a gestão de utilizadores, gestão de conteúdos informativos e a gestão dos jogos. Para a gestão dos jogos o servidor inclui um interpretador de uma linguagem desenvolvida no âmbito deste projecto que descreve cada jogo. Esta descrição inclui o conjunto de eventos que formam o jogo, as condições de despoletamento de cada evento e também as consequências de cada evento.

No capítulo 5, iremos apresentar os pormenores de implementação dos diferentes tipos de funcionalidades que o servidor possui e também os detalhes de implementação da linguagem e do respectivo interpretador.

2.6 Interface 3D

A interface 3D foi desenvolvida com a finalidade de criar uma interface de visualização e controlo sobre o sistema. Esta interface foi implementada em C++ utilizando como *framework* o motor de jogo *open source* *Ogre 3D*, e permite visualizar os utilizadores que estão no decorrer do seu jogo, e os pontos de conteúdo informativo num mapa 3D. Esta visualização é feita à custa das coordenadas de GPS de cada utilizador e ponto de informação. Em relação aos pontos de informação, é inclusive possível mudar a sua posição a partir desta interface. Esta interface comunica apenas com a base de dados utilizando uma interface baseada em *web services*.

No capítulo 6, iremos apresentar os pormenores de implementação desta interface.

3

Bases de Dados do Sistema

Conteúdo

3.1	Base de Dados dos Conteúdos Informativos	13
3.2	Base de Dados dos Conteúdos do Jogo	20
3.3	Interface Web Service da Base de Dados	26
3.4	Biblioteca de Comunicação com PostgreSQL – PSQLib++	27

Neste capítulo iremos descrever a implmentação das duas base de dados deste sistema. Esta descrição será feita recorrendo a diagramas de entidades e relações e, sempre que necessário, dicionário de dados para descrição de atributos importantes para o contexto do trabalho.

3.1 Base de Dados dos Conteúdos Informativos

A base de dados dos conteúdos informativos foi desenvolvida com duas finalidades: servir como base de dados para interfaces *web* desenvolvidas pela Quinta da Regaleira e servir como repositório dos conteúdos informativos mostrados pela aplicação InStory.

A informação armazenada na base de dados está centrada numa entidade, denominada por Local, ou seja, toda a informação é dependente do contexto da localização. Um Local é caracterizado por uma localização definida em coordenadas de GPS, e a cada Local estão associados vários tipos de informações: textos, imagens, vídeos, sons, biografias e eventos.

As imagens, vídeos e sons têm atributos comuns e estes atributos comuns foram agrupados numa entidade designada por Media.

Os eventos e as biografias possuem cada, uma tabela de datas relacionadas com o respectivo evento ou com a respectiva biografia. Tanto o evento como a biografia têm um texto associado com a respectiva descrição.

A um conjunto de locais com uma determinada ordem é dado o nome de percurso. Esta entidade tem associada um tema que define o tema do percurso.

Existe uma entidade Glossário que descreve um conjunto de termos e que tem associado a cada descrição uma imagem de ilustração.

Todos os tipos de informações existentes estão associados a palavras chave como forma de indexação e a datas como forma de indexação temporal.

O diagrama de entidades e relações que define as associações entre as diferentes entidades descritas pode ser observado na figura 3.1.

Uma parte desta base de dados, que não está apresentada no diagrama da figura 3.1, está relacionada com a visualização dos conteúdos informativos nos diferentes sistemas de suporte, (PDA, desktop e telemóveis). No diagrama da figura 3.2, é apresentada a entidade *Layout* que para um determinado local e suporte, define a apresentação de conteúdos do tipo: texto, imagem, vídeo e som.

A relação entre o texto e o *layout*, e a *media* e o *layout* define a posição do texto e da media no *layout*.

Pode haver mais que um layout para um mesmo local e um mesmo suporte e, quando assim é, existe uma ordem entre eles.

Nesta base de dados também é mantido um histórico das inserções de conteúdos informativos (locais, textos, imagens, vídeos, sons, eventos e biografias). Neste histórico a informação que é mantida é o índice de quais os utilizadores que inserem informação e quando é que a inserem.

Este histórico é mantido automaticamente à custa de um trigger designado por *record_history*, que intercepta as inserções nas tabelas das entidades acima referidas e regista a informação necessária no histórico. O diagrama de entidades e relações do histórico pode ser observado na figura 3.3.

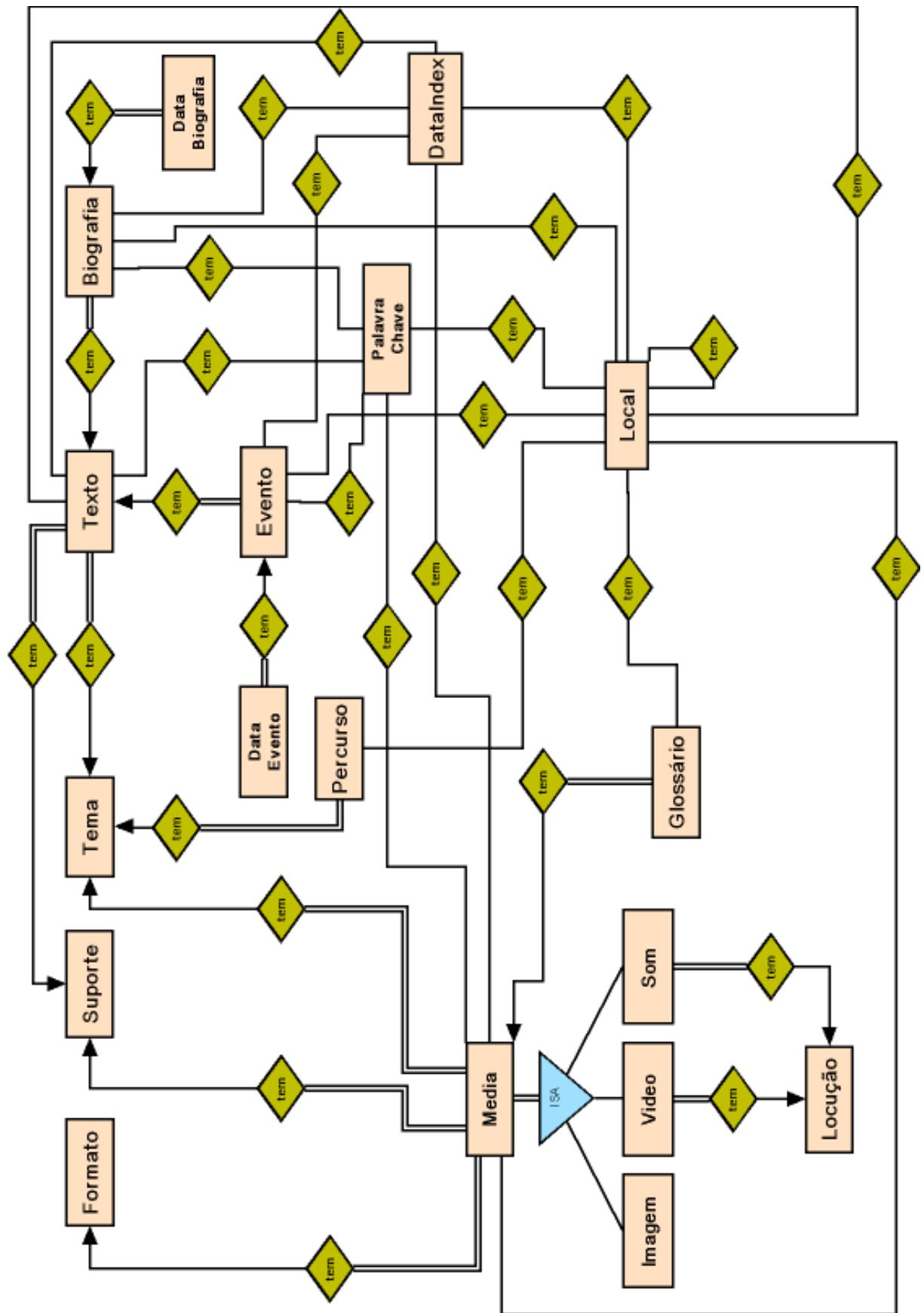


Figura 3.1: Diagrama de entidades e relações da base de dados de conteúdos informativos.

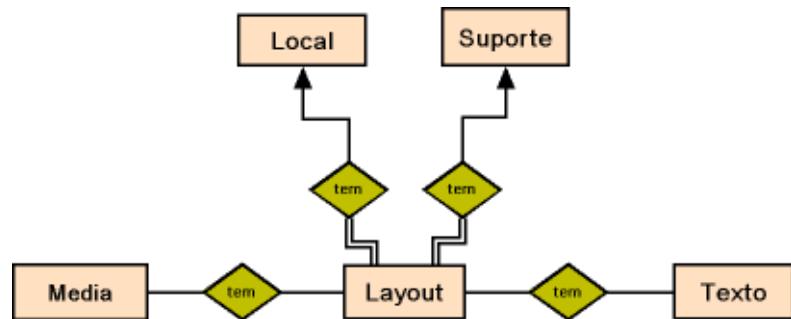
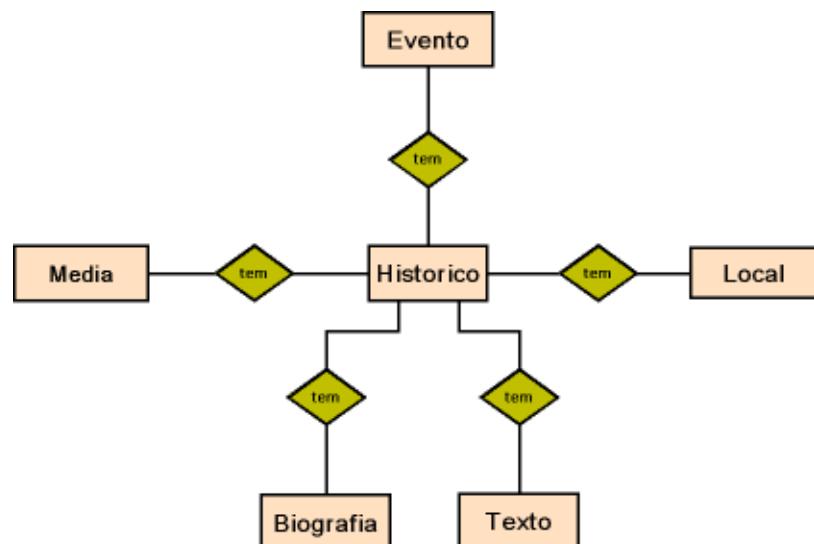
Figura 3.2: Diagrama de entidade e relações dos *Layouts* de informação.

Figura 3.3: Diagrama de entidade e relações do histórico de inserções de conteúdos informativos.

3.1.1 Descrição das Entidades

Local

Tabela que contém as localizações espaciais onde existe conteúdo informativo. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_local	Identificador único de um Local.
local	Nome de um Local.
latitude	Coordenada de GPS Latitude medida em graus.
longitude	Coordenada de GPS Longitude medida em graus.
altitude	Coordenada de GPS Altitude medida em metros.

Tema

Tabela que contém os vários tipos de temas que estão associados aos diferentes tipos de informação. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_tema	Identificador único de um Tema
tema	Nome de um Tema
description	Descrição de um tema.

Suporte

Tabela que contém os vários tipos de suporte que serão utilizados para visualizar os conteúdos informativos. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_suporte	Identificador único de um Suporte
suporte	Nome de um Suporte. Pode ser vários tipos: Desktop, PDA, MobilePhone, ...

Formato

Tabela que contém os vários tipos de formatos dos conteúdos audiovisuais. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_formato	Identificador único de um Formato
formato	Nome de um Formato. Pode ser vários tipos: mp3, jpeg, mpeg, ...

Locução

Tabela que contém os vários tipos de locuções dos conteúdos de áudio e de vídeo. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_locucao	Identificador único de um tipo de Locução
locucao	Nome de um tipo de Locução. Pode ser vários tipos: português, francês, inglês, ...

Media

Tabela que contém os conteúdos audiovisuais de qualquer tipo: imagem, vídeo e som. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_media	Identificador único de um tipo de Media.
filepath	Nome do ficheiro que contém os dados binários.
caption	Legenda descritiva.
copyright	Copyright do tipo de Media.
creation_date	Data de inserção na base de dados.
id_suporte	Identificador do suporte em que o tipo de media será visualizado.
id_formato	Formato do tipo de Media.
id_tema	Tema associado a este tipo de Media.

Imagen

Tabela que contém as imagens. Esta tabela estende a tabela Media. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_imagem	Identificador único de uma Imagem.
id_media	Identificador único de um tipo de Media.
resolution	Resolução de uma Imagem.

Vídeo

Tabela que contém os vídeos. Esta tabela estende a tabela Media. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_video	Identificador único de um Vídeo.
id_media	Identificador único de um tipo de Media.
id_locucao	Tipo de locução de um Som.
duration	Duração de um Vídeo.
resolution	Resolução de um Vídeo.

Som

Tabela que contém os sons. Esta tabela estende a tabela Media. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_som	Identificador único de um Som.
id_media	Identificador único de um tipo de Media.
id_locucao	Tipo de locução de um Som.
duration	Duração de um Som.

Percorso

Tabela que contém os percursos, sendo um percurso constituído por vários locais com uma determinada ordem entre eles. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_percorso	Identificador único de um Percurso.
percurso	Nome de um percurso.
duration	Duração aproximada de um percurso.
id_tema	Identificador de um Tema associado a um percurso.

Glossário

Tabela que contém os termos de um Glossário. Os atributos da tabela estão descritos na tabela seguinte.

Atributo	Descrição
id_termo	Identificador único de um termo.
termo	Nome de um termo.
description	Descrição de um termo.
id_imagem	Identificador da imagem associada a um termo.

Texto

Tabela que contém os textos informativos. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_texto	Identificador único de um Texto.
title	Título de um Texto.
autor	Autor de um Texto.
texto1	Texto de nível 1 que suporta até 200 caracteres.
texto2	Texto de nível 2 que suporta até 500 caracteres.
hline texto3	Texto de nível 3 que não tem limite de caracteres.
id_tema	Identificador de um Tema associado a um Texto.
id_suporte	Identificador de um Suporte no qual será visualizado o Texto.

Biografia

Tabela que contém as biografias. Cada biografia tem associado um texto com a respectiva biografia e também um conjunto de datas relacionadas. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_biografia	Identificador único de uma Biografia.
name	Nome do indivíduo do qual é escrita a biografia.
birth_date	Data de nascimento.
death_date	Data de falecimento.
id_texto	Identificador do Texto que está associado a uma Biografia.

Evento

Tabela que contém os eventos. Cada evento tem associado um texto com a respectiva descrição do evento e também um conjunto de datas relacionadas com o evento. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_evento	Identificador único de um Evento.
eventtitle	Título de um Evento.
id_texto	Identificador do Texto que está associado a um Evento.

Keyword

Tabela que contém as palavras chave que são associadas aos vários tipos de informação. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_keyword	Identificador único de uma Palavra Chave.
keyword	A respectiva Palavra Chave.

DateIndex

Tabela que contém as datas que são associadas aos vários tipos de informação. Os atributos da tabela estão descritos na tabela seguinte.

Atributo	Descrição
id_date	Identificador único de uma Data.
year	Ano.
month	Mês.
day	Dia.

Layout

Tabela que contém as apresentações dos conteúdos informativos dos diferentes locais. Os atributos da tabela estão descritos na tabela seguinte.

Atributo	Descrição
id_layout	Identificador único de um Layout.
id_suporte	Identificador do suporte de um Layout.
id_local	Identificador do local de um Layout.
ordem	Número que indica a ordem de um Layout num conjunto de Layouts do mesmo local e mesmo suporte.

3.2 Base de Dados dos Conteúdos do Jogo

A base de dados dos conteúdos do jogo foi desenvolvida com a finalidade de dar suporte ao motor de jogo, guardando todas as informações necessárias à manutenção do seu estado.

Para se poder desenvolver uma narrativa interactiva é preciso existir uma história (enredo) e actores. Dentro do conjunto dos actores de uma história, baseada na localização, têm de se considerar os actores reais, que são os protagonistas, e os actores virtuais.

No caso concreto deste sistema, a base de dados vai armazenar as várias histórias e os respectivos actores de cada história.

Cada actor possui um conjunto de propriedades e objectos. Os actores reais estão associados a um grupo. Os actores virtuais possuem uma representação que pode ser de vários tipos: imagem, vídeo e som. Cada objecto tem propriedades e também uma representação, do mesmo tipo que um actor virtual.

Os actores reais estão associados à história em que estão a participar através de uma relação que define o percurso espacial do utilizador no espaço físico. Os actores virtuais estão associados à história a que pertencem através de uma relação que define a posição de cada personagem virtual, no espaço físico, em cada instante.

Cada história está associada a uma temática e a um mapa, que corresponde à representação do espaço físico onde os actores reais se movimentam. Associado a cada história estão também os eventos da história, estes eventos são as partes diferentes de uma história que permitem a um actor real adquirir informação sobre o enredo e também tomar decisões que poderão modificar o rumo da história.

Cada evento tem associado um ou mais *layouts* que definem a apresentação do evento ao actor real. Estes *layouts* têm a mesma estrutura dos *layouts* da base de dados dos conteúdos informativos.

Nesta base de dados fica armazenada o histórico dos eventos em que cada actor real participa durante a sua participação na história.

O diagrama de entidades e relações que define as relações entre as diferentes entidades descritas pode ser observado na figura 3.4.

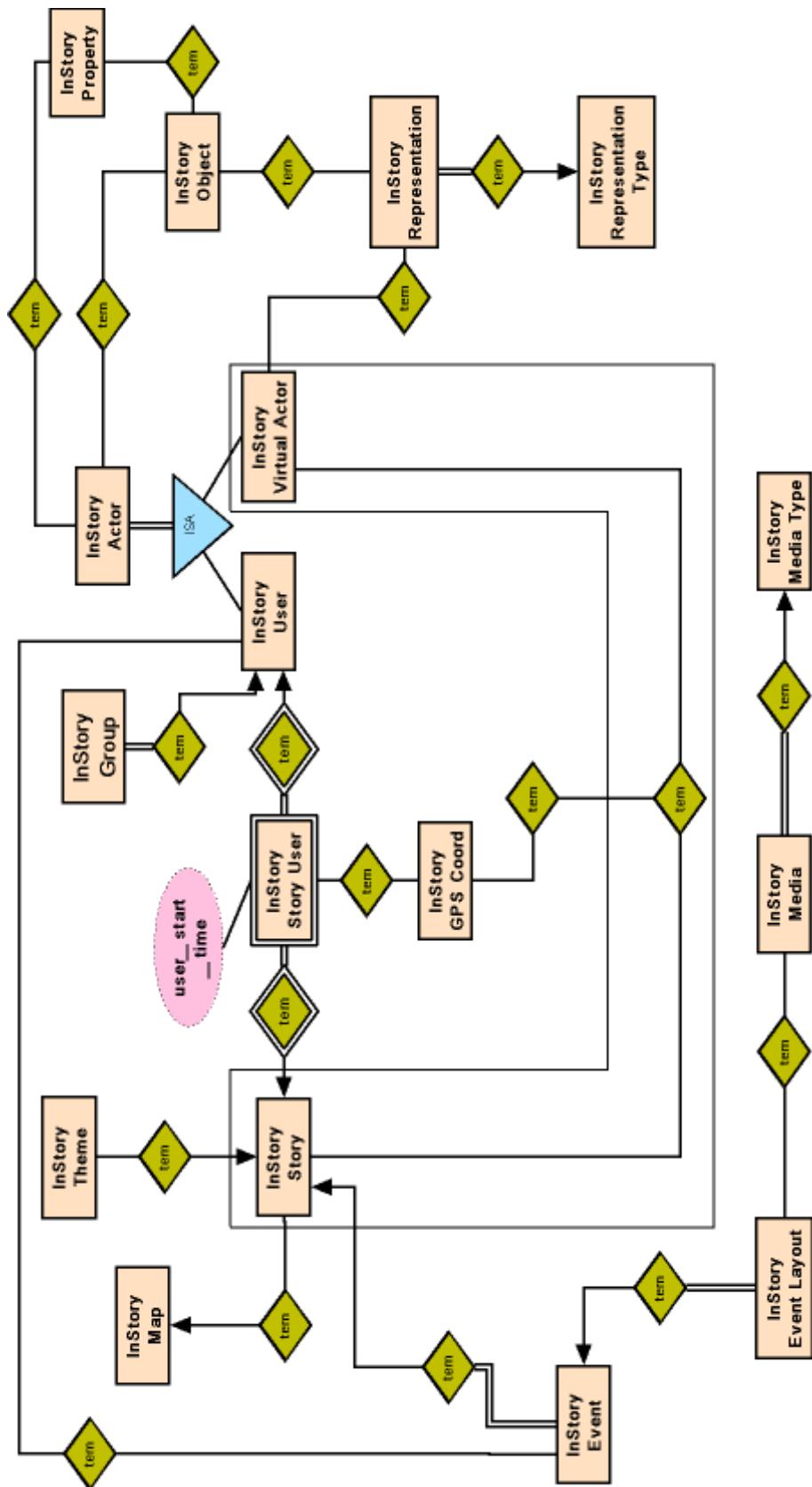


Figura 3.4: Modelo de entidades e relações da base de dados do jogo.

3.2.1 Descrição das Entidades

Instory_Theme

Tabela que contém os diferentes temas de cada história. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_theme	Identificador único de um tema.
theme_name	Nome de um tema.
theme_description	Descrição de um tema.

Instory_Map

Tabela que contém os diferentes mapas de cada história. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_map	Identificador único de um mapa.
description	Descrição de um mapa.
filepath	Nome do ficheiro onde se encontra a imagem que representa um mapa.
map_width	Largura de um mapa em <i>pixels</i> .
map_height	Altura de uma mapa em <i>pixels</i> .
src_latitude	Coordenada de GPS Latitude do canto superior esquerdo, de um mapa, medida em graus.
src_longitude	Coordenada de GPS Longitude do canto superior esquerdo, de um mapa, medida em graus.
scale_latitude	Factor de escala entre a altura de um mapa em <i>pixels</i> e a distância real em quilómetros.
scale_longitude	Factor de escala entre a largura de um mapa em <i>pixels</i> e a distância real em quilómetros.
north_angle	Ângulo que o eixo que define a altura, de um mapa, faz com o Norte.

Instory_Story

Tabela que contém as histórias. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_story	Identificador único de uma história.
story_title	Titulo de uma história.
story_description	Descrição de uma história.
id_map	Identificador do mapa de uma história.
id_theme	Identificador do tema de uma história.
story_file	Ficheiro onde se encontra o <i>script</i> que define o controlo dos eventos de uma história.

Instory_Group

Tabela que contém os grupos de utilizadores. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_group	Identificador único de uma grupo.
group_name	Nome de um grupo.
group_description	Descrição de um grupo.

Instory_Propriety

Tabela que contém os diferentes nomes de propriedades. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_prop	Identificador único de uma propriedade.
prop_name	Nome de uma propriedade.

Instory_Representation_Type

Tabela que contém os diferentes tipos de representações dos objectos e dos actores virtuais. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_rep_type	Identificador único de um tipo de representação.
type_name	Nome de um tipo de representação.

Instory_Representation

Tabela que contém as representações dos objectos e dos actores virtuais. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_represent	Identificador único de uma representação.
id_rep_type	Identificador do tipo de representação.
data_file	Ficheiro onde se encontra a representação em formato binário.

Instory_Object

Tabela que contém os objectos. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_object	Identificador único de um objecto.
obj_name	Nome de um objecto.

Instory_GPS_Coord

Tabela que contém todas as diferentes coordenadas de GPS utilizadas no sistema. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_gps	Identificador único de uma coordenada GPS.
latitude	Coordenada de GPS Latitude medida em graus.
longitude	Coordenada de GPS Longitude medida em graus.
altitude	Coordenada de GPS Altitude medida em metros.

Instory_Actor

Tabela que contém os actores, tanto reais como virtuais. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_actor	Identificador único de um actor.
actor_name	Nome de um actor na história.

Instory_User

Tabela que contém os actores reais, esta tabela estende da tabela Instory_Actor. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_actor	Identificador único de um actor.
id_user	Identificador único de um actor real.
user_name	Nome de um actor real (Utilizador).
user_pass	Password de um actor real.
user_ip	Endereço IPv4 da ultima ligação ao sistema.
id_group	Identificador do grupo a que um actor real pertence.

Instory_Virtual_Actor

Tabela que contém os actores virtuais, esta tabela estende da tabela Instory_Actor. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_actor	Identificador único de um actor.
id_vactor	Identificador único de um actor virtual.

Instory_Story_User

Tabela que contém as sessões dos utilizadores que participaram, ou participam, numa história. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_story	Identificador de uma história.
id_user	Identificador de um actor real.
user_level	Nível de progressão na história de um utilizador.
user_start_time	Estampilha temporal do inicio de participação numa história de um utilizador.
user_end_time	Estampilha temporal do fim de participação numa história de um utilizador.

Instory_Event

Tabela que contém os eventos de uma história. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_event	Identificador único de um evento.
event_name	Nome de uma evento.
event_description	Descrição de um evento.
id_story	Identificador da história a que um evento pertence.

Instory_Media_Type

Tabela que contém os diferentes tipos de media. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_media_type	Identificador único de um tipo de media.
media_type	Nome de um tipo de media. Pode ser imagem, video, som ou texto.

Instory_Media

Tabela que contém os conteúdos media. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_media	Identificador único de um media.
id_media_type	Identificador do tipo de media.
data_file	Ficheiro onde se encontra o conteúdo media no formato binário.

Instory_Support

Tabela que contém os diferentes tipos de suporte: Desktop, PDA, MobilePhone. Os atributos estão descritos na tabela seguinte.

Atributo	Descrição
id_support	Identificador único de um suporte
support	Nome de um suporte. Pode ser vários tipos: Desktop, PDA, MobilePhone e outros.

Instory_Event_Layout

Tabela que contém os *layouts* dos eventos de uma história. Os atributos da tabela estão descritos na tabela seguinte.

Atributo	Descrição
id_layout	Identificador único de um <i>layout</i> .
id_support	Identificador do suporte para o qual um <i>layout</i> está destinado.
vibrate	Valor binário que indica se, caso o dispositivo de suporte suportar, o dispositivo vibra na ocorrência do <i>layout</i> .
id_event	Identificador de um evento que está associado a um <i>layout</i> .

3.3 Interface Web Service da Base de Dados

A interface de comunicação em *web services* utilizada na comunicação com os clientes InStory, o servidor de jogo e a interface 3D, está definida num ficheiro WSDL designado por `psqlib.wsdl`. Este ficheiro representa todas as funções que podem ser invocadas remotamente pelos outros componentes do sistema.

O ficheiro `psqlib.wsdl` é compilado utilizando uma aplicação do pacote gSOAP designada por `wsdl2h` que gera uma ficheiro `.h` que contem código na linguagem C++ com algumas extensões do gSOAP onde define os protótipos das funções e os respectivos tipos abstractos de dados, necessários para a invocação das respectivas funções. Este ficheiro `.h` é posteriormente compilado com uma aplicação, do pacote gSOAP, designada por `soapcpp2` que gera os respectivos *stubs* e *skeletons* que permitem a invocação remota destas funções.

A comunicação entre os código gerado dos *stubs* e as bases de dados é feita usando uma biblioteca designada por `PSQLib++` que iremos descrever na secção seguinte.

De seguida apresentaremos os nomes das funções da interface e uma breve descrição de cada uma.

- **GetInStoryLocals:** Devolve todos os locais existentes na base de dados de conteúdos informativos.
- **GetGroups:** Devolve a lista dos grupos de utilizadores existentes na base de dados do jogo.
- **GetGames:** Devolve a lista dos jogos, disponíveis ao utilizador, existente na base de dados do jogo.
- **ValidateUser:** Dado um nome de utilizador e uma palavra chave valida se o utilizador está presente na base de dados do jogo.
- **SetUserInfo:** Actualiza a informação respeitante a um utilizador como o seu nome, o endereço IP e o grupo a que pertence.
- **SetUserGame:** Associa um dado utilizador a um jogo gravando na base de dados a data e hora do momento.
- **EndUserGame:** Finaliza a associação entre um dado utilizador e um jogo gravando na base de dados do jogo a data e hora do momento.
- **SetUserGPS:** Actualiza a posição em coordenadas GPS de um dado utilizador.
- **GetLocalLayouts:** Devolve, para um dado local, o *layout* desse local que é constituído por imagens, textos, vídeos e sons.

- `GetSupports`: Devolve uma lista de suportes existentes na base de dados dos conteúdos informativos.
- `GetMaps`: Devolve a lista de mapas existentes na base de dados do jogo.
- `GetMap`: Devolve a informação respeitante a um mapa.
- `GetUsersCurrentInfo`: Devolve a posição, em coordenadas GPS, de todos os utilizadores que estão a jogar.
- `GetUserPath`: Devolve a lista de posições, em coordenadas GPS, que um dado utilizador já percorreu.
- `SetEventDone`: Grava na base de dados do jogo que um dado utilizador efectuou um dado evento de um dado jogo.
- `GetActorInfo`: Devolve a informação respeitante a um actor de um jogo, nomeadamente as suas propriedades e os objectos que possui.
- `GetGameInfo`: Devolve o ficheiro de *script* que descreve os eventos de um dado jogo.
- `GetVirtualActorGPS`: Devolve a posição corrente em coordenadas GPS de um dado actor virtual.
- `UpdateActorInfo`: Actualiza na base de dados a informação respeitante a um actor nomeadamente as suas propriedades e os objectos que possui.
- `GetEventLayout`: Devolve, para um dado evento de jogo, o respectivo *layout* que é constituído por imagens, textos, vídeos e sons.
- `SetLocalGPS`: Actualiza a posição em coordenadas GPS de um local existente na base de dados de conteúdos informativos.

Estas funções são utilizadas pelos vários componentes do sistema mas algumas são apenas específicas para cada componente.

3.4 Biblioteca de Comunicação com PostgreSQL – PSQLib++

A comunicação entre a interface de *web services* em C++ e a base de dados PostgreSQL é feita à custa de uma biblioteca desenvolvida na linguagem C++ que tem como base a biblioteca oficial disponibilizada pelo PostgreSQL desenvolvida na linguagem C.

A biblioteca foi desenvolvida com o objectivo de facilitar na programação de aplicações, na linguagem C++, que necessitam de aceder à base de dados PostgreSQL, desenvolvendo para esse efeito um conjunto de classes com funcionalidades de visualização dos dados retornados pelas *queries* de SQL.

A classe `PSQLibDatabase` controla a ligação à base de dados. O construtor desta classe tem como parâmetros as informações necessárias para efectuar a ligação à base de dados. Esta classe permite executar comandos SQL directamente sobre a base de dados. Os comandos SQL, executados nesta classe, que retornam resultados, como o comando `select`, retornam sob a forma de uma classe chamada `PSQLibResultIterator` que funciona com um iterador de linhas dos resultados retornados pelas *queries*.

A classe `PSQLibResultIterator` pode funcionar como um iterador de linhas ou pode retornar um vector com os resultados por coluna, tendo que se indicar o nome da coluna para seleccionar a respectiva.

A outra classe desta biblioteca, `PSQLibTable`, funciona como se uma instância desta classe fosse uma tabela existente na base de dados, podendo inserir ou remover colunas e também obter as linhas da respectiva tabela indicando a linha e a coluna pretendida.

O diagrama de classes desta biblioteca pode ser observada na figura 3.5 de modo a compreender melhor as relação entre as classes descritas.

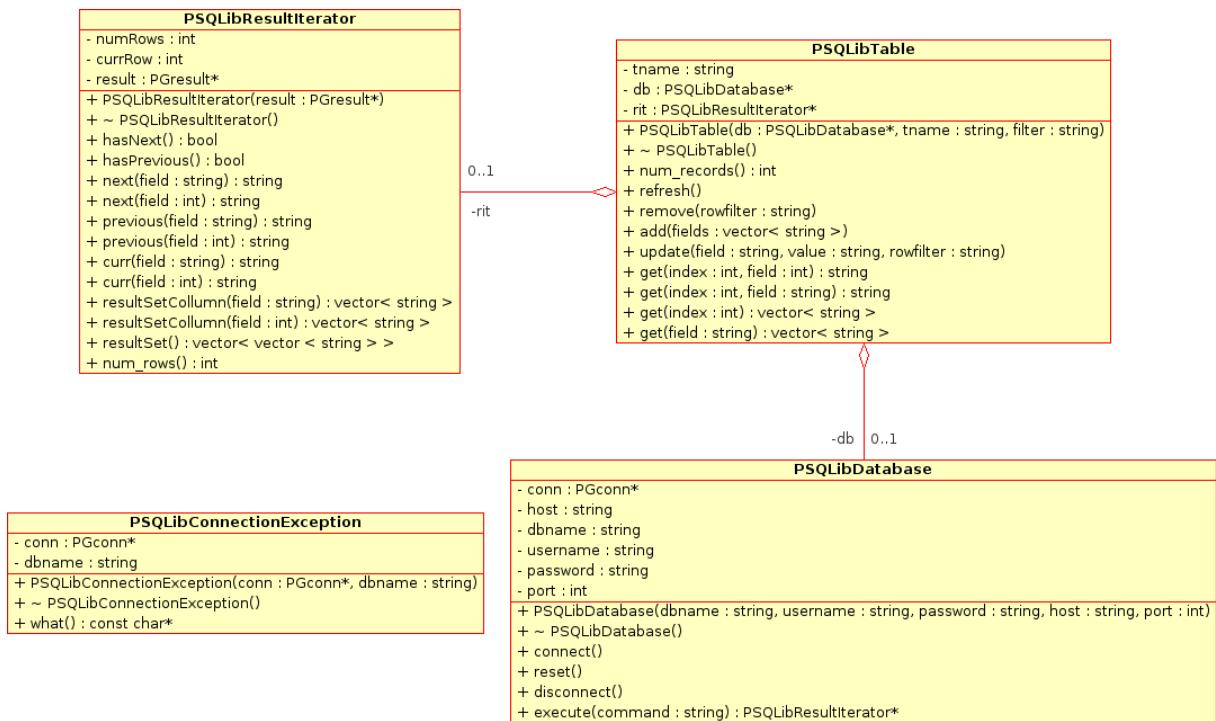


Figura 3.5: Diagrama de classes da biblioteca PSQLib++.

4

Interface Web

Conteúdo

4.1 Apresentação da Interface Web	30
4.2 Implementação	33

Neste capítulo iremos descrever os pormenores de implementação da interface de gestão web da base de dados dos conteúdos informativos.

Esta interface de gestão foi implementada em PHP com o intuito de gerir os conteúdos multimédia existentes na base de dados descrita na secção 3.1.

4.1 Apresentação da Interface Web

A interface de gestão, de suporte *web*, foi desenvolvida com o intuito de facilitar a inserção e manutenção dos conteúdos multimédia pertencentes à Quinta da Regaleira. Para esse efeito a interface foi implementada utilizando a linguagem de programação, orientada para a *web*, designada por PHP.

Esta interface, devido à importância da informação que mantém, possui requisitos de segurança dos dados. Assim foi necessário implementar um controlo de acessos à interface, por parte dos utilizadores, por forma a salvaguardar os dados que se inserem com esta interface.

O controlo de acessos tem três níveis de segurança diferentes:

- Controlo total: é permitido inserir, remover e visualizar.
- Controlo de escrita: é permitido inserir e visualizar.
- Controlo de leitura: é permitido visualizar.

Estes tipos de controlos de acesso permitem que existam utilizadores que têm apenas como função inserir dados na base de dados sem poder remover o que já existia.



Figura 4.1: Layout principal da interface web.

Na figura 4.1 podemos observar que a interface tem um menu lateral onde se encontram todos os tipos de dados que o utilizador pode inserir ou visualizar.

Cada entrada do menu lateral indica o tipo de dados que se pretendem inserir, remover ou visualizar. Quando o utilizador carrega numa das entradas do menu lateral, surgem as opções para essa entrada. As opções de cada entrada podem ser diferentes apenas tendo todas em comum a opção de criação. Na figura 4.2 podemos observar o *layout* que surge quando o utilizador pretende efectuar operações sobre locais. Neste caso ainda temos mais opções que estão relacionadas com o facto de podermos efectuar associações, dos vários tipos de informação da base de dados, com um respectivo local.

A inserção de um novo tipo de informação é sempre feita recorrendo a um formulário com caixas de texto e listas de escolha múltipla. Todas as caixas de texto estão relacionados directamente com os campos

The screenshot shows the 'Base de Dados InStory' interface. On the left, a sidebar lists various database categories: Inicio, Locais, Criar Local, Ver Locais, Associar Imagens, Associar Videos, Associar Sons, Associar Textos, Associar Eventos, Associar Biografia, Associar Glossário, Imagens, Videos, Sons, Textos, Eventos, Biografias, Glossário, Percursos, Tabelas Auxiliares, Palavras Chave, and Datas Índice. The 'Criar Local' link is currently selected. The main content area is titled 'Criação de um novo Local'. It contains four text input fields labeled 'Local', 'Latitude', 'Longitude', and 'Altitude'. Below these is a section titled 'Associar Palavras Chave' with a 'CheckButton' and a 'Palavra Chave' input field. At the bottom is a 'Criar' button.

Figura 4.2: Layout da interface web durante a criação de um local.

da tabela correspondente ao tipo de informação que se está a inserir. Por vezes nem todos os campos da tabela aparecem no formulário, nestes casos os respectivos campos ocultos são preenchidos automaticamente pelo sistema. As listas de escolha múltipla correspondem a chaves externas que a tabela possui para outra tabela e na lista de escolha múltipla aparecem os registo da respectiva tabela referenciada. Existe também uma lista de palavras chave que se podem escolher se estiverem associadas ao tipo de informação que se está a inserir no momento. Na figura 4.3 pode ser observado um formulário de criação de uma imagem com todos estes componentes que foram descritos.

A visualização dos registo de cada tipo de informação pode ser feita em dois modos:

- Ver Um a Um.
- Ver Todos.

No modo **Ver Um a Um** o utilizador visualiza a informação de um registo de um determinado tipo de informação. Esta visualização é feita à custa de um formulário preenchido com a respectiva informação. No final do formulário existem duas setas de navegação para se poder navegar pelos vários registo do mesmo tipo de informação. Este modo de visualização pode ser visto na figura 4.4.

No modo **Ver Todos** o utilizador visualiza a informação de todos os registo de um tipo de informação, na forma tabular. Neste modo não são mostradas todas as informações de um determinado registo. Este modo de visualização pode ser visto na figura 4.5.

Base de Dados InStory

Criação de uma nova imagem

<ul style="list-style-type: none"> Início Locais Imagens <ul style="list-style-type: none"> Criar Imagem Ver Imagens Vídeos Sons Textos Eventos Biografias Glossário Percursos Tabelas Auxiliares Palavras Chave Datas Índice 	Ficheiro: <input type="text"/> Browse... Legenda: <input type="text"/> Copyright: <input type="text"/> suporte: <input type="text" value="PDA"/> formato: <input type="text" value="jpeg"/> tema: <input type="text" value="História"/> Resolução: <input type="text"/> Associar Palavras Chave: <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: #90EE90;">CheckButton</th> <th style="background-color: #90EE90;">Palavra Chave</th> </tr> </thead> <tbody> <tr> <td><input checked="" type="checkbox"/></td> <td>Palavra 1</td> </tr> </tbody> </table> <p>Criar</p>	CheckButton	Palavra Chave	<input checked="" type="checkbox"/>	Palavra 1
CheckButton	Palavra Chave				
<input checked="" type="checkbox"/>	Palavra 1				

[Início](#) > [Imagens](#)
 IMG - Interactive Multimédia Group 2007 Contacto

Figura 4.3: Layout da interface web durante a criação de uma imagem.

Base de Dados InStory

Ver Imagens

<ul style="list-style-type: none"> Início Locais Imagens <ul style="list-style-type: none"> Ver Imagens Ver Um a Um Ver Todos Vídeos Sons Textos Eventos Biografias Glossário Percursos Tabelas Auxiliares Palavras Chave Datas Índice 	
	Legenda: <input type="text" value="jardim"/> Copyright: <input type="text" value="yes"/> Formato: <input type="text" value="jpeg"/> Suporte: <input type="text" value="PDA"/> Tema: <input type="text" value="História"/> Resolução: <input type="text" value="800x600"/>

[Início](#) > [Imagens](#)
 IMG - Interactive Multimédia Group 2007 Contacto

Figura 4.4: Layout da interface web durante a visualização de uma imagem.

Ficheiro	Legenda	Data Criação	Suporte	Tema	Visualizar	Remover
Garden.jpg	jardim	2007-07-25 20:46:01.529082	PDA	História		X
Waterfall.jpg	casca	2007-07-25 20:46:01.533121	PDA	Arte		X
Forest Flowers.jpg	jardim	2007-07-25 20:46:01.534142	PDA	Arquitectura		X

Início > Imagens
IMG - Interactive Multimédia Group 2007 Contacto

Figura 4.5: Layout da interface web durante a visualização de todas as imagens.

4.2 Implementação

Antes de descrevermos a implementação dos componentes da interface *web*, como o controlo de acessos, a inserção e a visualização de informação, é necessário descrever como a interface obtém os dados da base de dados descrita na secção 3.1.

A interface desenvolvida comunica directamente com a base de dados PostgreSQL através de uma classe denominada por DB. Esta classe foi desenvolvida utilizando como base a API de comunicação entre o PHP e o PostgreSQL, com a finalidade de facilitar o desenvolvimento da interface.

Cada opção do menu lateral da interface é implementada num ficheiro PHP e estes ficheiros implementam todas as funcionalidades de cada opção (inserção, remoção e visualização do dados). Existem mais alguns ficheiros que controlam outras funcionalidades da própria interface como as definições da interface (caminhos do sistema de ficheiros e credenciais das base de dados), o controlo de acessos e o desenho da própria interface.

As definições da interface têm como função permitir a portabilidade da interface. Estas definições incluem os caminhos no sistema de ficheiros onde se encontra a própria raiz da interface e os directórios onde são guardados os conteúdos multimédia, e as credenciais necessárias à ligação à base de dados dos conteúdos informativos. Estas definições estão representadas sob a forma de variáveis globais que estão disponíveis nos restantes ficheiros da interface. O ficheiro PHP que inclui estas definições é `settings.php`.

O controlo de acessos da interface *web* é implementado à custa do próprio controlo de acessos do PostgreSQL. Os utilizadores que podem aceder à interface têm de ser utilizadores existentes no gestor de utilizadores do PostgreSQL.

O controlo de acessos desta interface tem dois componentes:

- Autenticação.
- Permissões.

A autenticação nesta interface é feita após o utilizador submeter as suas credenciais. Nesse momento o sistema compara o *hash* da palavra chave inserida pelo utilizador com o *hash* existente na tabela de sistema do PostgreSQL, pg_shadow, que está associada ao nome de utilizador inserido pelo utilizador. Este *hash* é feito concatenando a palavra “md5” ao resultado da técnica de hash md5 aplicada à concatenação da palavra chave com o nome de utilizador.

```
password_hash = ''md5'' + md5(password + username)
```

A comparação dos *hash*'s é feita utilizando uma *query* SQL à base de dados. Caso essa *query* retorne um resultado significa que o utilizador inseriu credenciais válidas e fica guardada, em variáveis de sessão do PHP, a informação do utilizador e a indicação de que a autenticação foi válida. A partir desse momento o utilizador pode começar a utilizar as funcionalidades disponíveis da interface. O código PHP que implementa o processo de autenticação de utilizadores está no ficheiro login.php.

A gestão de permissões da interface tem como função gerir as operações de inserção, remoção e visualização sobre os dados informativos que são manipulados pela interface. Estas permissões estão presentes na associação entre utilizadores e as tabelas na própria base de dados PostgreSQL. Cada utilizador, quando criado pelo *script* de criação da base de dados dos conteúdos informativos, tem definido para cada tabela as permissões de leitura, inserção e remoção. Estas permissões reflectem-se directamente na interface web através das opções, do menu lateral, disponíveis a cada utilizador que variam conforme o utilizador pode ou não inserir, remover ou visualizar.

O menu lateral é construído pelo código PHP existente no ficheiro menu.php. Neste ficheiro processa-se uma série de testes às permissões do utilizador corrente de modo a apresentar ou ocultar a respectiva opção. Cada teste é feito utilizando um conjunto de funções da classe DB:

- can_insert(\$user, \$table): verifica se o utilizador \$user pode inserir registos na tabela \$table.
- can_delete(\$user, \$table): verifica se o utilizador \$user pode remover registos na tabela \$table.

Não existe uma função para a permissão de visualização, porque nesses casos são de utilizadores que não pertencem ao sistema e como tal não têm credenciais de autenticação.

O ficheiro header.php, que controla os acessos directos às páginas da interface, utiliza estas funções e também a indicação se o utilizador está autenticado para permitir, ou não, o acesso à respectiva página.

Os componentes HTML dos formulários de inserção e visualização são gerados de forma automática. No caso dos formulários de inserção, a geração automática tem em conta os tipos dos atributos da respectiva tabela e cria os vários componentes HTML de inserção de dados (caixas de texto, listbox's, etc...) necessários à inserção de dados em todos os atributos, à excepção dos que são chaves primárias. Os atributos que são chaves externas também originam um componente HTML, em que se pode escolher um registo da tabela referenciada por essa chave externa.

Estes componentes de inserção de dados dependem do tipo dos atributos de cada tabela mas também podem ser definidos manualmente. O mapeamento entre o tipo do atributo de uma dada tabela e o componente HTML a que dá origem pode ser visto na tabela 4.1.

Tipo do Atributo	Componente HTML
varchar	Caixa de texto 1 linha.
float8	Caixa de texto 1 linha.
int4	Caixa de texto 1 linha.
time	Caixa de texto 1 linha.
date	Caixa de texto 1 linha.
text	Caixa de texto de várias linhas.
file	Caixa de texto 1 linha com botão para escolha de ficheiro no sistema de ficheiros.
Chave Externa	Caixa de escolha múltipla.

Tabela 4.1: Tabela de mapeamento entre tipos de atributos de tabelas SQL e componentes HTML de inserção de dados.

Utilizando o mapeamento de tipos descrito na tabela 4.1 e dada uma tabela da base de dados foi possível, de forma automática gerar formulários de inserção de dados para cada tipo de dados informativos que se pretendia inserir. Cada ficheiro PHP correspondente a uma entrada do menu lateral, na sua opção de inserção de dados, invoca o gerador automático de formulários de inserção sobre a tabela correspondente. Esta invocação é feita através da classe DB invocando a função `toHTMLForm($table, $renames, $retypes)` em que `$table` é o nome da tabela, `$rename` é um vector indexado que permite renomear o nome dos atributos da tabela e `$retypes` é um vector indexado que permite trocar o tipo dos atributos da tabela. Esta função gera o código HTML com o formulário pretendido.

Os formulários de visualização são de dois tipos: formulário preenchido ou tabela. Um formulário preenchido apenas diz respeito a um registo de uma dada tabela. Para apresentar este tipo de visualização é usado o mesmo processo dos formulários de inserção, com a diferença que o formulário é preenchido automaticamente com um registo da respectiva tabela. Além de o formulário ser preenchido com a informação de um registo ainda inclui duas setas no final do formulário para navegar entre os registos. No caso em que é apresentado o primeiro registo da tabela apenas é apresentada uma seta para a direita.

A visualização dos dados em forma de tabela tem como finalidade poder-se observar o conjunto total dos registos de uma tabela. Esta visualização em forma de tabela é feita automaticamente dado um iterador de *queries*. Um iterador de *queries* é representado pela classe QIterator. Uma instância desta classe é resultado da invocação da função `query($query)`, da classe DB, em que `$query` corresponde à query SQL que queremos submeter à base de dados. A classe QIterator permite-nos iterar sobre os registos de um dada tabela ou também apresentar todos os registos sob a forma de uma tabela HTML. Este tipo de visualização permite, a utilizadores com as permissões necessárias, apagar registos. A apresentação da tabela HTML com os registos de uma dada tabela da base de dados é feita invocando a função `toHTMLTable($headers, $additional, $hrefs, $user)`, da classe QIterator, em que `$headers` é um vector com os nomes que ficam no cabeçalho da tabela (estes nomes correspondem aos nomes dos atributos da tabela com a mesma ordem e sem contar com as chaves primárias) `$additional` é um vector indexado para adicionar colunas especiais à tabela como imagens, vídeos, sons, ou até a opção de remoção, `$hrefs` é um prefixo de endereço para reencaminhar cada linha da tabela para a respectiva visualização em formulário preenchido ou para outra finalidade e `user` que nesta versão não está a ser utilizado.

Internamente a função `toHTMLTable`, quando é invocada com a opção de remoção em cada linha, verifica se o utilizador tem permissão para remover registo da respectiva tabela e consoante o resultado apresenta ou não a opção de remoção.

5

Servidor de Jogo

Conteúdo

5.1	Descrição das Funcionalidades do Servidor	37
5.2	Implementação dos Módulos	38
5.3	Descrição e Definição do Motor de Jogo	41
5.4	Implementação do Motor de Jogo	45
5.5	Exemplo de Funcionamento de um Jogo	60

Neste capítulo iremos descrever a implementação de um servidor com a função de gerir toda a parte lógica do sistema, gerindo um sistema de sessões de utilizadores e gerindo os conteúdos informativos. Também iremos descrever a definição de um jogo baseado em narrativas interactivas e identificar os conceitos e as suas relações que o ajudam a construir. Para isso iremos descrever os detalhes de implementação da linguagem criada para descrever um jogo.

5.1 Descrição das Funcionalidades do Servidor

O servidor de jogo é o componente mais importante do sistema. Pode ser visto como o componente central que dita o comportamento dos restantes componentes, para isso utiliza uma interface em *web services* para a comunicação. O servidor tem quatro módulos que de seguida iremos enumerar e descrever. Os módulos são:

- Gestão de Sessões de utilizadores.
- Gestão de Conteúdos Informativos.
- Gestão de Eventos do Jogo ou Motor de Jogo.
- Servidor de Web Services.

5.1.1 Gestão de Sessões de Utilizadores

O servidor foi implementado a pensar na existência de vários utilizadores, a aceder ao sistema em concorrência. Cada utilizador, durante a sua interacção com o sistema tem um estado associado que varia de utilizador para utilizador, suportado pelo conceito de sessão. Uma sessão está sempre associada a um único utilizador. Em cada sessão são guardadas todas as informações necessárias para manter o estado da progressão do respectivo utilizador.

5.1.2 Gestão de Conteúdos Informativos

O utilizador, ao longo da sua progressão no espaço físico (neste caso a Quinta da Regaleira), vai recebendo informação contextual do local onde se encontra. O envio da informação contextual é feito pelo servidor, em resposta à recepção das coordenadas GPS da posição do utilizador.

O servidor mantém a informação correspondente aos locais (secção 3.1.1) que se encontram na base de dados dos conteúdos informativos. O servidor só envia a informação contextual caso as coordenadas GPS da posição do utilizador se encontrem num raio de N metros centrado num local, sendo N um valor definido na configuração do servidor.

A informação contextual enviada pelo servidor, é apenas um identificador do local que o cliente terá de utilizar para obter o *layout* da base de dados dos conteúdos informativos.

5.1.3 Gestão de Eventos de Jogo

A gestão de eventos de jogo é o módulo mais complexo deste trabalho. O utilizador durante a sua visita será interrompido com eventos de jogo que podem depender da sua localização ou de outros factores como intervalos de tempo entre eventos. O servidor suporta vários jogos diferentes em simultâneo, ou seja, um utilizador pode estar a participar num jogo diferente de outro utilizador. Os jogos são definidos numa linguagem de *scripting*, definida e implementada no contexto deste trabalho. O servidor é capaz de interpretar a linguagem de definição dos jogos e gera uma estrutura de dados dinâmica com os eventos, de forma a os poder enviar pela ordem correcta ao utilizador.

5.1.4 Servidor de Web Services

O servidor é uma aplicação *stand-alone* e como tal foi necessário implementar um servidor para atender os pedidos HTTP provenientes das invocações das funções definidas pela interface de *web services*. Este módulo foi implementado com suporte para múltiplos pedidos (*multi-threaded*). Dado que é por este módulo que se recebem os pedidos dos restante componentes do sistema, este necessita de comunicar com os restantes módulos para executar os respectivos pedidos.

5.2 Implementação dos Módulos

Nesta secção iremos descrever a implementação dos módulos do servidor. O servidor inicia com a criação do *thread* do servidor de web services. Este *thread* é que comanda o servidor, pois todas as acções são despoletadas por ele. Os restantes módulos são representados por variáveis estáticas incluídas na classe `InStoryEngine`. Esta classe possui uma função designada por `startEngine` que instancia as respectivas variáveis. Esta instanciação só é realizada após a leitura de um ficheiro de configuração do servidor onde se definem parâmetros importantes para o funcionamento do mesmo. A classe `InStoryEngineSettings` processa o ficheiro de configuração e fica com os respectivos parâmetros guardados em memória para as restantes classes do programa poderem aceder.

De seguida, iremos descrever a implementação de cada módulo em mais pormenor, à excepção do módulo de gestão de eventos de jogo que será descrito em secções mais à frente.

5.2.1 Implementação da Gestão de Sessões e Utilizador

A classe que gera as sessões de utilizadores é designada por `InStorySessionManager`. Esta classe implementa três funções:

- `void createSession(InStoryUser *user, string support)`
- `void destroySession(string key)`
- `InStorySession *getSession(string key)`

A primeira função `createSession`, como o nome indica, cria uma sessão. Esta função recebe como parâmetros um utilizador e o tipo de suporte do cliente (PDA ou telemóvel). Cada sessão criada fica guardada numa tabela de dispersão indexada pelo nome de utilizador. Este nome de utilizador é usado como parâmetro de entrada para as outras duas funções: `destroySession` e `getSession` que como os nomes indicam, uma destrói uma sessão existente e a outra devolve uma sessão existente.

Uma sessão é definida pela classe `InStorySession`. Quando a sessão é criada, é iniciado um *thread*, que é utilizado para, periodicamente, a informação da progressão do utilizador no jogo ser sincronizada na base de dados e também para obter informação do próprio jogo, como a posição corrente dos actores virtuais.

Uma sessão guarda informações importantes para manter o estado de progressão do utilizador. Essas informações são:

- O utilizador.

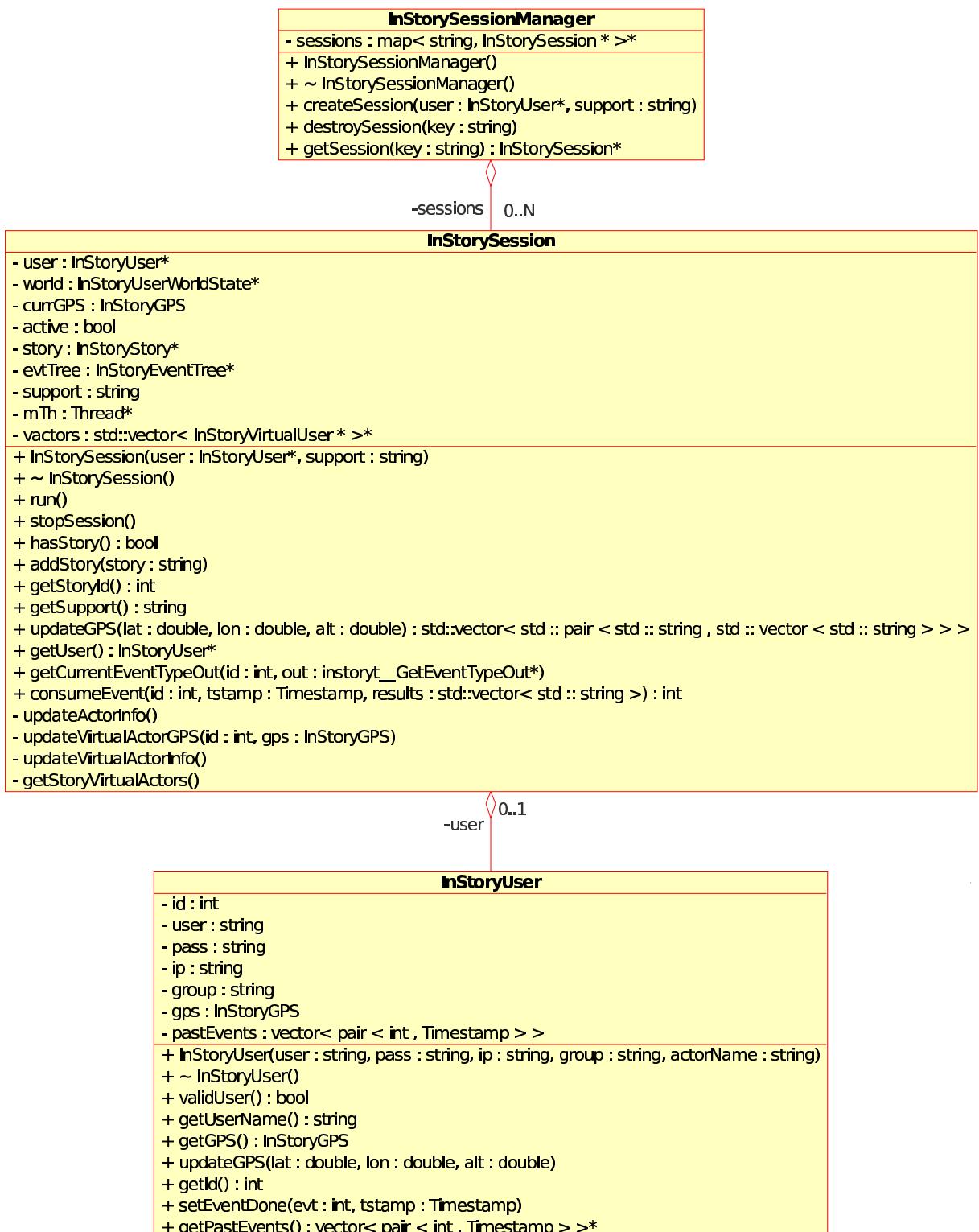


Figura 5.1: Diagrama de classes da gestão de sessões.

- O estado do mundo (este tópico é abordado na secção 5.3.3).
- A posição corrente do utilizador em coordenadas GPS.
- Uma instância do jogo em que está a participar.
- Uma lista dos actores virtuais com as suas posições.

A classe `InStorySession` é inclusive utilizada para avançar na progressão do jogo através da função `consumeEvent`. Esta função será explicada aquando da descrição da implementação do motor de jogo.

A classe `InStoryUser` define um utilizador do sistema. Esta classe guarda as credencias do utilizador, a sua posição em coordenadas GPS, o grupo a que pertence e o histórico dos eventos que já superou.

Na figura 5.1 pode ser observado o diagrama de classes com estas três classes acima descritas.

5.2.2 Implementação da Gestão de Conteúdos Informativos

A gestão de conteúdos informativos é implementada na classe `InStoryContext`. Esta classe quando é instanciada acede à base de dados dos conteúdos informativos e obtém a lista de todos os locais existentes com as suas respectivas localizações.

Sempre que é recebida a informação da posição do utilizador, é invocada a função `getNearLocals` desta classe que devolve um vector com os locais que estão no raio da posição do utilizador.

Este módulo é bastante mais simples que outros, pois apenas tem de devolver os identificadores dos locais ao cliente.

5.2.3 Implementação do Servidor de Web Services

O servidor de Web Services é um servidor que atende pedidos HTTP, protocolo usado pelas invocações de *web services*. Para poder processar pacotes HTTP utilizamos a biblioteca disponibilizada pelo gSOAP. O gSOAP possui primitivas para processar este tipos de pedidos mas não tem suporte para processar vários pedidos em simultâneo. A classe `InStorySOAPServer` foi implementada de forma a utilizar a biblioteca gSOAP, para processar pedidos HTTP, com suporte *multi-threaded*.

O funcionamento do servidor é muito simples. Quando é criada uma instância da classe `InStorySOAPServer` é iniciado um *thread* com o ciclo da recepção dos pedidos HTTP. Por cada pedido HTTP é criado uma instância da classe `InStorySOAPRequest` que processa o respectivo pedido. Esta classe inicia um *thread* para processar o respectivo pedido, de forma a não bloquear o ciclo da classe `InStorySOAPServer`. Desta forma é possível processar vários pedido em paralelo.

A interface de *web services* definida por este servidor encontra-se no ficheiro `instory.wsdl`. De seguida, iremos descrever cada função da interface.

- `LoginUser`: Dado um nome de utilizador, uma palavra chave, um grupo e um nome (alcunha) de actor é efectuada a operação de *login* no servidor.
- `LogoutUser`: Dado um nome de utilizador é efectuada a operação de *logout* do servidor.
- `Groups`: Devolve todos os grupos registados na base de dados do jogo.

- **Games:** Devolve todos os jogos disponíveis na base de dados dos jogo.
- **SelectGame:** Dado um nome de utilizador e um nome de um jogo inscreve o utilizador no respectivo jogo.
- **UpdateUserGPS:** Dado um nome de utilizador e coordenadas GPS actualiza a posição do respectivo utilizador e devolve uma lista de conteúdos contextuais e uma lista de eventos.
- **GetEvent:** Dado o identificador de um evento, devolve a informação relativa ao evento.
- **ConsumeEvent:** Dado o identificador de um evento e o resultado desse mesmo evento, devolve o identificador do evento que se segue.

5.3 Descrição e Definição do Motor de Jogo

O motor de jogo é o módulo que constrói os jogos a partir de ficheiros de *script* e controla todas as interacções dos utilizadores durante cada evento do jogo. Para percebermos melhor como funciona o motor de jogo é necessário perceber o que é um jogo e como este funciona. Nas secções seguintes iremos definir os conceitos que caracterizam um jogo.

5.3.1 Definição do Jogo

Uma história interactiva baseada na localização, difere de uma história tradicional porque a personagem principal é o próprio utilizador do jogo, tendo desta forma de se criar vários caminhos possíveis para o desenvolvimento da acção criando assim uma história não linear.

No contexto deste sistema, cada utilizador será a personagem principal na história em que estará a participar, e como tal poderá interagir com outras personagens da mesma história, sendo que estas tanto podem ser reais como virtuais.

O desenvolvimento de uma história, com base neste tipo de jogo, será feita recorrendo a acontecimentos que são despoletados pela personagem principal ou pela ocorrência de outros acontecimentos. Estes acontecimentos são definidos pela alteração que produzem no estado do mundo, relativo à personagem principal.

De forma a ganhar algum dinamismo e permitir acções complexas, cada personagem possui um estado próprio. Esse estado é caracterizado por um conjunto de propriedades que poderão ser diferentes para cada tipo de história. Estas propriedades são pares com um nome e um valor que pode variar ao longo do jogo. As personagens também possuem uma lista de objectos que poderão ser utilizados pela personagem contra ela própria ou contra outras personagens.

Os acontecimentos que ocorrem durante o jogo podem ser de vários tipos: simples comentários de um narrador omnipresente, em que não há interacção explícita com a personagem, diálogos com outros personagens, questões de escolha múltipla, batalhas entre personagens e acções mais específicas que o personagem terá de executar, como por exemplo tirar uma fotografia a um objecto ou local.

5.3.2 Identificação de Conceitos

Para podermos especificar um jogo, baseado em narrativas, é necessário identificar os conceitos principais que se utilizam na construção de uma história não linear. Os conceitos fundamentais para o tipo de jogo ou história descrito na secção 5.3.1, são:

- Personagens.
- Objectos.
- Propriedades.
- Acontecimentos.

De seguida definimos cada conceito e de que forma estão ligados ao sistema que gera o jogo.

Personagens

Este conceito, que passaremos a chamar *Actor*, abrange dois tipos: actores reais, a que chamaremos apenas actores, e actores virtuais. Os actores são o conceito central do jogo pois são eles que vão controlar a progressão do jogo através da sua posição no mundo real e interacção com o mundo virtual.

A posição de um actor, ou até de um actor virtual, no mundo real é dada através de coordenadas GPS. A cada instante o sistema conhece a posição de todos os actores presentes no sistema. A interacção entre um actor e o mundo virtual é feita através de um dispositivo móvel, como um PDA ou um telemóvel.

Um actor virtual também tem associada uma posição no espaço real descrito em coordenadas GPS. Cada actor virtual não interage com outros actores, apenas reage a interacções. A sua representação corresponde a uma imagem mas pode ser extensível a outros tipos mais complexos de representação. A posição de um actor virtual não é estática. Ao longo do tempo, a posição é alterada de acordo com um calendário especificado *a priori*. Este calendário descreve a posição de um actor virtual num determinado espaço de tempo.

Objectos

Este conceito representa os objectos que poderão existir no mundo virtual e que auxiliarão os actores no desenrolar do jogo. Cada objecto possui um nome que identifica o objecto, a quantidade de objectos deste tipo (este atributo tem de ser interpretado no contexto em que os actores possuem mais do que um objecto do mesmo tipo), e uma representação que pode ser feita de várias formas.

Propriedades

Este conceito representa as propriedades que são utilizadas tanto nos actores como objectos. É um conceito muito simples, pois cada propriedade apenas possui um nome e um valor. Desta forma podemos criar diferentes tipos de propriedades nas diferentes histórias, ficando a cargo da lógica do jogo a responsabilidade de dar semântica a cada propriedade.

Acontecimentos

Este conceito, que passaremos a chamar de *Evento*, é constituído por duas partes: o *quando*, que corresponde à definição do despoletamento (*trigger*) do evento, e o *como*, que corresponde à definição das ações que serão executadas pelo evento que afectará, como consequência, o estado do mundo de um actor.

O *trigger* de um evento pode ser definido pela acção, ou acções, que desencadeiam o evento e por pré-condições que têm de ser satisfeitas no momento do desencadeamento do evento.

As ações que estão definidas para desencadear um evento são as seguintes:

- O actor chegar a uma localização.
- Ter ocorrido um evento há pelo menos N segundos.
- O actor chegar ao alcance de um actor virtual.

As pré-condições que os eventos terão eventualmente de satisfazer são as seguintes:

- Ter ocorrido um evento.
- O actor possuir um objecto.

Tanto as ações de desencadeamento, como as pré-condições, podem ser combinadas entre si com conjunções ou disjunções. Por exemplo, podemos dizer que um evento pode ocorrer quando o actor chega a uma localização ou quando tiverem passado sessenta segundos após outro evento. Outro exemplo com pré-condições seria que um evento pode ocorrer quando o actor chega a uma localização e o actor tem um objecto e já ocorreu outro evento. Também se pode negar uma pré-condição, por exemplo não ter ocorrido um evento ou o actor não ter um objecto.

As ações e pré-condições ligam-se sempre com uma conjunção. Desta forma, mesmo que sejam executadas as ações necessárias ao desencadeamento do evento, se este não satisfizer as pré-condições, não poderá ser executado.

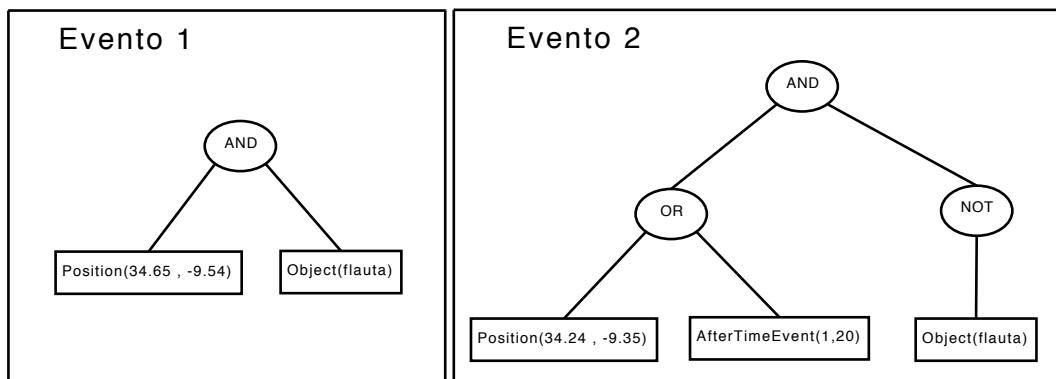


Figura 5.2: Árvore lógica dos *triggers* e pré-condições de dois eventos

A figura 5.2 mostra as árvores lógicas de dois eventos que definem os *triggers* e as pré-condições dos eventos. Cada nó interno das árvores corresponde às operações lógicas **and**, **or** ou **not**, e cada folha das árvores corresponde aos *triggers* ou pré-condições dos eventos.

As ações executadas pelo evento podem ser definidas como um *workflow* de três fases:

- *Output*.
- *Input*.
- Processamento.

A fase de *Output* é caracterizada pelo envio, do servidor para o actor, de informação relativa ao evento. Esta informação pode-se manifestar de várias maneiras como texto, imagem, som e vídeo.

A fase de *Input* é caracterizada pelo envio, do actor para o servidor, de informação relativa ao evento. Esta informação enviada está fortemente dependente da fase de *Output*, pois tipicamente a informação enviada corresponde a uma resposta à fase de *Output*.

Por vezes pode não existir fase de *Input*, como nos casos em que a fase de *Output* apenas disponibiliza informação ao actor, sem se querer nenhuma interacção posterior por parte deste.

A fase de Processamento é caracterizada pelo processamento da informação obtida na fase de *Input*. Mais uma vez, esta fase está fortemente dependente da fase de *Output* pela mesma razão da fase de *Input*. Caso não exista fase de *Input* não haverá fase de Processamento pois não existe informação para processar. Como consequência deste processamento, o estado do mundo, na perspectiva do actor, poderá ser modificado.

5.3.3 Relações entre Conceitos

Depois de identificados os conceitos na secção anterior (5.3.2), existem algumas relações entre eles que iremos de seguida descrever. Cada actor (real ou virtual) possui uma lista de objectos. Estes objectos vão sendo adicionados ao actor ao longo do jogo, ou podem estar desde logo, na posse do actor. Cada actor também possui uma lista de propriedades. Estas propriedades, no início do jogo, têm valores padrão, sendo que ao longo do jogo os valores das propriedades poderão sofrer alterações em consequência dos eventos que vão ocorrendo.

Cada objecto possui uma lista de propriedades, cujos valores são estáticos ao longo do jogo, isto é, nunca serão modificados. As propriedades dos objectos poderão eventualmente influenciar os valores das propriedades do actor que possui o respectivo objecto.

Um actor real necessita de manter um registo dos eventos que já ultrapassou e para isso possui uma lista de identificadores desses, e um *timestamp* associado a cada identificador que indica quando foi ultrapassado o respectivo evento.

A definição do estado do mundo na perspectiva de um actor é dada pelo conjunto de vários conceitos, de forma a retirar o máximo de informação possível para se poderem validar as acções de desencadeamento e précondições dos *triggers* dos eventos.

Estado do Mundo

O estado do mundo é único para cada actor do jogo. Este é construído com base na informação retirada do próprio actor. O estado do mundo possui:

- Informação do Actor:
 - Lista de Objectos.

- Lista de Propriedades.
- Posição (coordenadas GPS).
- Lista de eventos já superados.
- Lista de actores virtuais que estão ao alcance da posição do actor.
- Memória (estrutura de dados para manter variáveis locais criadas pelo *script* do jogo).

O estado do mundo é utilizado pelos eventos para validar as suas condições de desencadeamento. Além disso, a fase de Processamento de cada evento poderá modificar o estado do mundo.

5.4 Implementação do Motor de Jogo

Para descrever a implementação do motor de jogo vamos seguir a ordem do fluxo de execução da aplicação, originado quando um utilizador entra no sistema (*login*) e escolhe um jogo.

O utilizador ao escolher o jogo, desencadeia uma série de acções no servidor. Primeiro o servidor tem que obter da base de dados do jogo o *script* que o define. Depois tem de interpretar a linguagem e gerar uma árvore de eventos e após esta operação, o jogador fica pronto a jogar. De seguida, descreveremos o processo de interpretação da linguagem e a geração da árvore de eventos.

5.4.1 Definição da Linguagem EventLang

A linguagem EventLang foi criada para descrever os eventos que constituem um jogo. Um evento é constituído por duas partes: as condições de despoletamento e as acções de consequência.

A sintaxe da linguagem é baseada na declaração de classes em C++. No exemplo seguinte podemos observar a declaração de um evento.

```
event 1 {
    trigger:
        position(35.5, 35.5)

    precondition:
        hasobject(flauta)

    type dialog {
        input {
            [ "Esta mensagem vai aparecer no cliente!"]
        }
        consequence {
        }
    }
}
```

A declaração de um evento é feito com a keyword **event**, com um argumento numérico que identifica o evento. Dentro do bloco de um evento temos três secções identificadas pelas keywords: **trigger**, **precondition** e **type**. A secção **trigger** define as acções que despoletam o evento. No exemplo anterior a condição

era as coordenadas GPS da posição do utilizador serem 35.5 graus de latitude e 35.5 graus de longitude. A secção **precondition** são as condições que tem de ser satisfeitas para o evento poder ser despoletado, mesmo que as condições de despoletamento já estejam satisfeitas. No exemplo anterior a condição era o utilizador ter a flauta na sua lista de objectos. A secção **type** define o tipo do evento e as respectivas acções. À frente da keyword **type** é necessário um argumento que indica qual o tipo de evento que se está a definir. Dentro desta secção está o **input** que define o conteúdo dinâmico que é enviado para o cliente, e o **consequence** que processa a resposta, ao evento, enviada pelo cliente. Apenas no **consequence** é possível modificar o estado do mundo. No exemplo anterior, o evento é do tipo diálogo e portanto na secção **input** é enviada uma mensagem para o cliente apresentar ao utilizador. A secção **consequence** está vazia pois não existe resposta para processar.

As condições que são inseridas no **trigger** e no **precondition** não são obrigatoriamente restritas ao **trigger** ou ao **precondition**, ou seja, as condições que estão no **trigger** podem estar no **precondition** e vice-versa. Isto porque quando é avaliado a árvore lógica destas condições, para o despoletamento do evento, é criada uma conjunção das condições do **trigger** e do **precondition**. A razão da separação é só simplicidade de leitura.

As condições que se podem inserir não fazem parte da linguagem. Estas podem ser encaradas como funções que são definidas no início do ficheiro de *script*. Na verdade a implementação da condição encontra-se numa classe C++, que tem de estender obrigatoriamente de uma classe abstracta que define as funções que têm que ser implementadas por todas as classes que representam cada condição. No inicio do ficheiro de *script* apenas se define o mapeamento entre o nome da condição que se utiliza no ficheiro e a respectiva classe.

```
define position as PositionCondition
```

Os tipos de eventos também não fazem parte da linguagem e são definidos da mesma forma que as condições. Logo existe uma classe que representa cada tipo de evento. Dependendo do tipo de evento, o **input** também será diferente. No **input** apenas se podem enviar informações para o cliente. A informação que é enviada pode ser de dois tipos, cadeia de caracteres ou variáveis que podem representar listas de objectos, o valor de uma propriedade ou um objecto.

Para listar a informação que é para ser enviada para o cliente utilizam-se os parênteses rectos e insere-se, separado por vírgulas, os diferentes tipos de informação que se pretendem enviar.

Tanto no **input** como no **consequence** estão disponíveis variáveis que representam os objectos possuídos pelo utilizador e também os valores das propriedades do utilizador. No **consequence** é possível criar variáveis através de uma simples afectação: teste = 40. Estas variáveis sobrevivem à passagem de eventos, ou seja, se definirmos uma variável no **consequence** de um evento essa variável estará disponível tanto no **input** como no **consequence** de outro evento.

No **consequence** é possível processar a resposta do utilizador a um evento. A resposta está disponível sob a forma de uma variável designada por `event<número do evento>.result`. Desta forma é possível aceder às respostas de cada evento. Como já foi dito é possível criar variáveis e realizar operações sobre elas como somar ou subtrair, dependendo do tipo da variável. Foi também implementada a operação condicional `if` para ajudar na expressividade da linguagem.

As variáveis que representam os objectos do utilizador e a suas propriedades são da forma: `user.objects` para os objectos e `user.properties.<nome da propriedade>` para as propriedades.

Como o objectivo da linguagem é descrever os eventos de um jogo, existe mais uma construção na

linguagem, importante para ajudar a descrever as relações entre eventos. A keyword **chain** permite criar eventos compostos, ou seja, permite definir um evento que na verdade é a composição de vários eventos. A partir do momento em que o primeiro evento da composição ocorre, todos os eventos dessa composição irão ocorrer sem nunca ser interrompidos por eventos que não pertençam à composição.

Imaginemos que temos sete eventos definidos e não indicamos condições de **precondition**. Podemos utilizar a seguinte construção para definir um evento composto, indicando a ordem de execução dos eventos.

```
chain 1->7->2->[3,4->5]->6
```

O que esta construção indica é que, quando o evento 1 for consumido logo a seguir vem o evento 7 e depois vem o evento 2. Depois do evento 2 pode vir ou o evento 3 ou o evento 4. Se for o evento 3 a ser consumido então logo a seguir virá o evento 6, mas se for o evento 4 primeiro a ser consumido então a seguir virá o evento 5 e depois do evento 5 ser consumido virá o evento 6. De notar que os eventos 3 e 4 são concorrentes, pois poderão estar disponíveis para serem consumidos em simultâneo e caso um deles seja consumido ficará sempre o outro evento por consumir, significando que caso esta situação não seja desejável é necessário inserir algumas condições no **precondition** de cada um.

Por vezes precisamos que um evento, que após ser consumido, volte a estar disponível, por exemplo, eventos de diálogos de informação ou para a criação de ciclos. Para se concretizar esta situação, na declaração de um evento, basta inserir atrás de **event** a keyword **survive**. Desta forma o evento que tiver esta construção poderá ser continuamente consumido enquanto as suas condições de despoletamento forem satisfeitas. Agora um exemplo com todas as construções que foram descritas:

```
survive event 5 {
    precondition:
        lastevent(4)
    type useobject {
        input {
            ["Use o objecto certo para vencer Ares",
             user.objects, user.properties.Life,
             ares.properties.life]
        }
        consequence {
            user.objects -= event5.result;
            if (event5.result == "flauta") {
                ares.properties.life = 0;
            }
            else {
                user.properties.Life -= lostpoints;
            }
        }
    }
}
```

5.4.2 Interpretação da Linguagem EventLang

Nesta secção iremos descrever o processo de interpretação da linguagem que origina a árvore de eventos. Para começar iremos apresentar a gramática independente de contexto no formato BNF (Backus Naur

form) da linguagem.

```

<input> ::=   <EOL>
            | <input> <define>
            | <input> <event>
            | <input> <eventchain>

<define> ::= "define" <STRING> "as" <string>

<event> ::=   <props> "event" <INT> "{" <decl> "type" <type> "}"
            | <props> "event" <INT> "{" <decl> "}"

<props> ::=   <EOL>
            | "survive"

<eventchain> ::= "chain" <eventlist>

<eventlist> ::=   <eventent>
                  | <eventent> "->" <eventlist>

<eventent> ::=   <INT>
                  | "[" <eventconcurr> "]"

<eventconcurr> ::=   <eventlist>
                  | <eventlist> "," <eventconcurr>

<decl> ::=   <EOL>
            | <trigger>
            | <trigger> <precondition>
            | <precondition>

<trigger> ::= "trigger" ":" <trigdecl>

<precondition> ::= "precondition" ":" <predecl>

<trigdecl> ::=   <trigtype>
                  | <trigdecl> "and" <trigdecl>
                  | <trigdecl> "or" <trigdecl>
                  | "(" <trigdecl> ")"

<trigtype> ::= <STRING> "(" <arrayval> ")"

<predecl> ::=   <pretype>
                  | <predecl> "and" <predecl>
                  | <predecl> "or" <predecl>
                  | "not" <predecl>
                  | "(" <predecl> ")"

<pretype> ::= <STRING> "(" <arrayval> ")"

```

```

<arrayval> ::=  <val>
              | <val> "," <arrayval>

<val> ::=   <FLOAT>
          | <INT>
          | <STRING>

<type> ::=   <STRING> "{" <input> "}"
           | <STRING> "{" <consequence> "}"
           | <STRING> "{" <input> <consequence> "}"

<input> ::= "input" "{" <inputdecl> "}"

<inputdecl> ::=  "[" <list> "]"
                 | <STRING>
                 | "if" "(" <boolexp> ")" <inputdecl>
                 | "if" "(" <boolexp> ")" <inputdecl> "else" <inputdecl>

<list> ::=  <exp>
            | <list> "," <exp>

<consequence> ::= "consequence" "{" <conseqdeclseq> "}"

<conseqdeclseq> ::=  <conseqdecl>
                      | <conseqdecl> ";" <conseqdeclseq>

<conseqdecl> ::=  <STRING> "=" <exp>
                  | <STRING> "+=" <exp>
                  | <STRING> "-=" <exp>
                  | "if" "(" <boolexp> ")" <conseqdecl>
                  | "if" "(" <boolexp> ")" "{" <conseqdeclseq> "}" "else" "{" <conseqdeclseq> "}"
                  | <EOL>

<boolexp> ::=  <exp>
                | <exp> "=" <exp>
                | <exp> "!=" <exp>
                | <exp> "<" <exp>
                | <exp> ">" <exp>
                | <exp> "<=" <exp>
                | <exp> ">=" <exp>

<exp> ::=   <FLOAT>
          | <INT>
          | <STRING>
          | <BOOL>
          | <QUOTED_STRING>
          | <exp> "+" <exp>
          | <exp> "-" <exp>

```

Na tabela 5.1 encontram-se exemplos de utilização dos literais presentes na gramática da linguagem. O interpretador da linguagem foi implementado usando um gerador de *parsers* designado por yacc. Este gerador pega na gramática, definida numa sintaxe um pouco diferente da descrita acima, e juntamente com extensões de código C++, para ajudar a criar a árvore de sintaxe abstracta, gera um interpretador da linguagem.

Literal	exemplo
STRING	user.objects
INT	234.
FLOAT	33.4
BOOL	true ou false
QUOTED_STRING	"isto é uma string \n mudança de linha"

Tabela 5.1: Tabela de exemplos de cada literal presente na gramática da linguagem EventLang.

Durante a fase de interpretação são construídas várias árvores de sintaxe abstracta pois a avaliação das árvores é feita em diferentes fases e também de maneiras diferentes.

Processo de criação de eventos

Os eventos são representados pela classe abstracta `IEventNode`. Nesta classe ficam guardadas as árvores de sintaxe abstracta das condições de **trigger** e **precondition** e das acções de **input** e **consequence**. Esta classe também representa um nó da árvore de eventos, que vamos criar para obter a ordem de execução de eventos.

Na fase de interpretação da linguagem, são criados os vários eventos (instâncias da classe `EventNode`) e adicionando um a um numa lista de eventos. Cada evento é construído com um apontador para uma árvore lógica constituída pelas condições presentes no **trigger** e **precondition**, como está representado na figura 5.2. Esta árvore é constituída pelas classes que estão representadas no diagrama de classes da figura 5.3.

A raiz da árvore lógica é um objecto da classe `LogicNode` assim como os restantes nós da árvore. As folhas da árvore são da classe `ConditionNode`, pois esta classe é que possui a condição para ser avaliada. Esta condição é representada por uma classe abstracta designada por `EventCondition`. Todas as condições estendem desta classe. Portanto a árvore lógica é constituída por operadores lógicos, que correspondem aos nós internos e pelas condições que obrigatoriamente têm de devolver um valor lógico quando são avaliadas. Existe apenas uma restrição nesta versão do sistema, em relação aos operadores lógicos: o operador **not** tem de preceder sempre uma folha da árvore.

Além do apontador para a árvore lógica, um evento quando é criado também leva um apontador para um objecto da classe abstracta `ASTTypeNode` que representa o tipo do evento. Esta guarda internamente as árvores de sintaxe abstracta correspondentes ao **input** e **consequence**.

Criação da árvore de eventos

Quando terminada a interpretação do código que define um jogo, ficamos com uma lista de objectos, da classe `IEventNode`, todos com as respectivas árvores lógicas e as árvores de sintaxe abstracta do **input** e **consequence**.

O passo seguinte é criar a árvore de eventos que indica quais os eventos que poderão ocorrer ao utili-

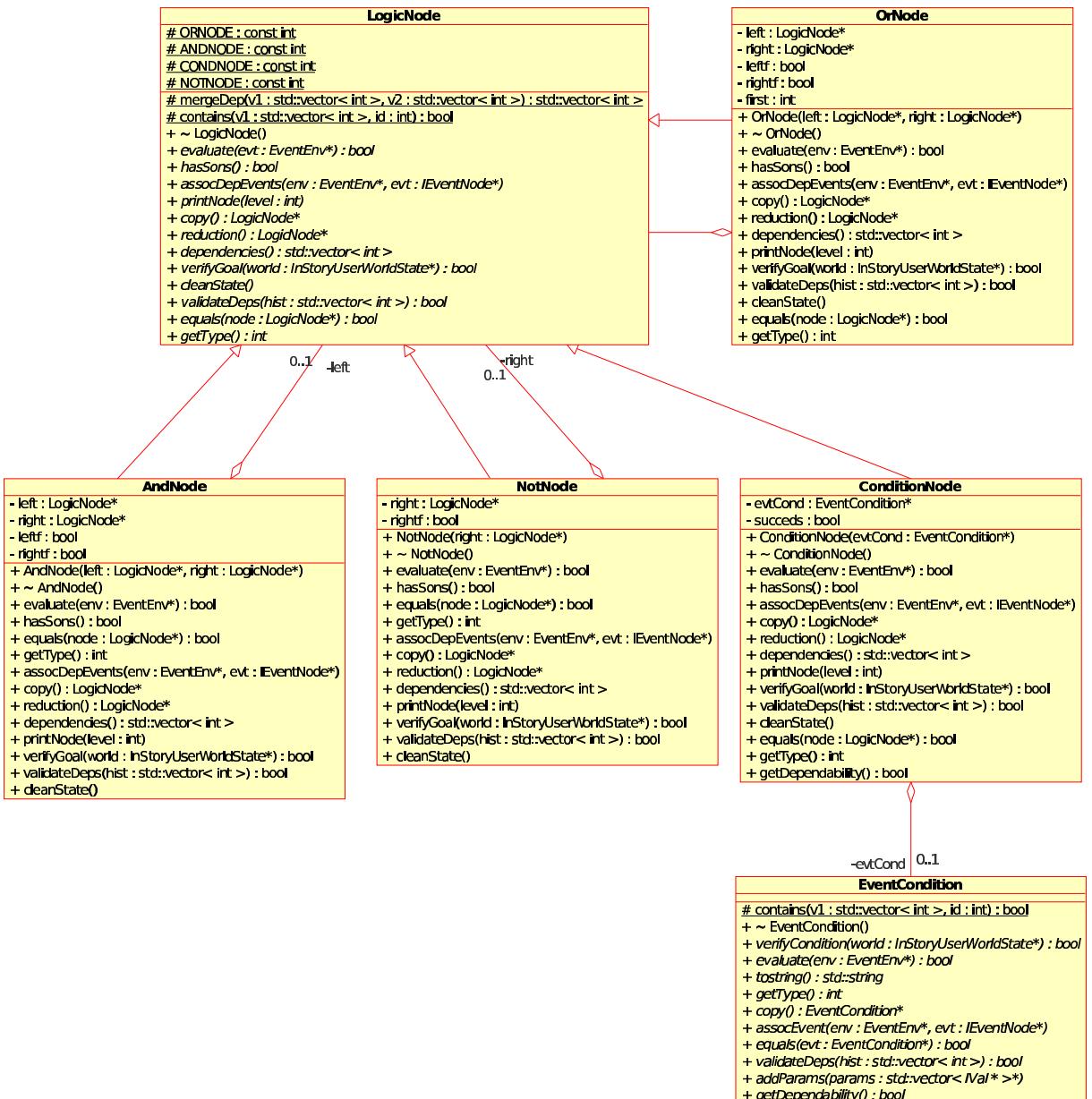


Figura 5.3: Diagrama de classes da árvore lógica das condições

zador em cada momento. A lista dos eventos encontra-se dentro de um objecto da classe EventTree. Esta classe é inicializada pelo interpretador e usando a função addEventNode insere os eventos um a um numa lista que possui. Para gerar a árvore dos eventos, invoca-se a função createTree, e esta gera numa estrutura de dados, interna à classe, a respectiva árvore.

O objectivo do algoritmo implementado na função createTree é usar a informação das árvores lógicas de cada evento e construir uma ordem pela qual os eventos podem ocorrer, por exemplo se tivermos um evento identificado pelo valor 1 e outro evento identificado pelo valor 2, e a condição de **trigger** do evento 1 for uma condição de posição.

```
event 1 {
    trigger:
        position(34.5,34.5)
}
```

E a condição de **trigger** do evento 2 for uma condição que indica que o evento só pode acontecer depois do evento 1.

```
event 2 {
    trigger:
        afterevent(1)
}
```

Então a ordem pela qual os eventos ocorrerão será primeiro o evento 1 e só depois poderá ocorrer o evento 2. Assim sendo, a árvore correspondente a esta situação seria: a raiz como evento 1 e o evento 2 como filho do evento 1. O exemplo descrito é muito simples. Na verdade, as árvores lógicas, utilizando os três operadores lógicos, definem ordens muito complexas entre os eventos.

De seguida apresentamos o algoritmo implementado na função createTree (Algoritmo 1), descrito em pseudo-código. Na implementação da função createTree são invocadas funções que pertencem a outras classes, mas também são aqui descritas em pseudo-código.

O resultado do algoritmo fica guardado no ambiente. O ambiente é representado pela classe EventEnv. Esta classe representa uma estrutura de dados Pilha em que cada posição da Pilha corresponde a uma instância da classe EventEnv. Cada objecto desta classe guarda uma lista de eventos. Durante a execução do algoritmo vão sendo inseridos eventos no objecto que está no topo da pilha e o topo da pilha vai sendo renovado por novos objectos. No final da execução do algoritmo, no fim da pilha encontra-se as raízes das árvores de eventos. Na verdade, a árvore de eventos, é uma lista de árvores em que cada uma corresponde ao rumo que cada utilizador pode seguir durante o jogo.

Algoritmo 1 EventTree::CREATETREE

Input: *LEvent*

```

1: env  $\Leftarrow \emptyset$ 
2: LNewEvent  $\Leftarrow \emptyset$ 
3: LErase  $\Leftarrow \emptyset$ 
4: while size(LEvent)  $> 0 do
5:   for i = 0 to size(LEvent) - 1 do
6:     evt  $\Leftarrow LEvent[i]$ 
7:     root  $\Leftarrow evt.root$ 
8:     if EVALUATE(root, env) then
9:       if HASSONS(root) then
10:        newevt  $\Leftarrow copyEvent(evt)$ 
11:        newevt.back  $\Leftarrow REDUCTION(root)$ 
12:        ASSOCDEPEVENTS(newevt.root, newevt, env)
13:        insert(LNewEvent, newevt)
14:        CLEANSTATE(newevt.root)
15:      else
16:        evt.back  $\Leftarrow REDUCTION(root)$ 
17:        ASSOCDEPEVENTS(root, evt, env)
18:        insert(LNewEvent, evt)
19:        insert(LErase, evt.id)
20:      end if
21:    else
22:      CLEANSTATE(root)
23:    end if
24:  end for
25:  for i = 0 to size(LErase) - 1 do
26:    erase(LEvent, LErase[i])
27:  end for
28:  clear(LErase)
29:  for i = 0 to size(LNewEvent) do
30:    insert(env, LNewEvent[i])
31:  end for
32:  clear(LNewEvent)
33:  env  $\Leftarrow beginScope(env)$ 
34: end while
35: RESOLVETRANSITIVITY(env)
36: return env$ 
```

Algoritmo 2 AndNode::EVALUATE

Input: *root* and *env*

```

if root.leftf and root.rightf then {Na primeira execução de EVALUATE neste nó, esta condição é sempre satisfeita}
  l  $\Leftarrow EVALUATE(root.left, env)
  r  $\Leftarrow EVALUATE(root.right, env)
  if l and r then
    root.leftf  $\Leftarrow FALSE$ 
    root.rightf  $\Leftarrow FALSE$ 
    return true
  end if
end if
return false$$ 
```

Algoritmo 3 OrNode::EVALUATE

Input: *root* and *env*

```

l  $\Leftarrow$  FALSE
r  $\Leftarrow$  FALSE
if root.leftf then {Na primeira execução de EVALUATE neste nó, esta condição é sempre satisfeita}
    l  $\Leftarrow$  EVALUATE(root.left, env)
end if
if root.rightf then {Na primeira execução de EVALUATE neste nó, esta condição é sempre satisfeita}
    r  $\Leftarrow$  EVALUATE(root.right, env)
end if
if l and r then
    root.first  $\Leftarrow$  2 {Na primeira execução de EVALUATE neste nó, o valor de root.first é 1}
end if
if l or r then
    if l and (not HASSONS(root.left)) then
        if root.first = -1 then
            first  $\Leftarrow$  0
        end if
        root.leftf  $\Leftarrow$  FALSE
    end if
    if r and (not HASSONS(root.right)) then
        if root.first = -1 then
            first  $\Leftarrow$  1
        end if
        root.rightf  $\Leftarrow$  FALSE
    end if
    return true
end if
return false

```

Algoritmo 4 NotNode::EVALUATE

Input: *root* and *env*

```

if GETDEPENDABILITY(root.right) then {Sabendo que root.right é um ConditonNode}
    root.rightf  $\Leftarrow$  not EVALUATE(root.right, env)
    return root.rightf
else
    root.rightf  $\Leftarrow$  EVALUATE(root.right, env)
    return root.rightf
end if

```

Algoritmo 5 ConditionNode::EVALUATE

Input: *root* and *env*

```

root.succeeds  $\Leftarrow$  EVALUATE(root.evtCond, env) {O EVALUATE de uma condição está dependente da implementação da condição mas a semântica é: se a condição depender de outro evento então verifica se o evento existe no Ambiente env e se existir retorna true, caso contrário retorna false}
return root.succeeds

```

Algoritmo 6 AndNode::HASSONS and OrNode::HASSONS

Input: *root*

```

return HASSONS(root.left) or HASSONS(root.right)

```

Algoritmo 7 NotNode::HASSONS

Input: *root*

```

return not root.rightf

```

Algoritmo 8 ConditionNode::HASSONS

Input: *root*
return *not root.succeds*

Algoritmo 9 AndNode::REDUCTION

Input: *root*
if *root.leftf* **then**
return *AndNode(REDUCTION(root.left), REDUCTION(root.right))*
else
return NIL
end if

Algoritmo 10 OrNode::REDUCTION

Input: *root*
if (*not root.leftf*) and (*root.rightf*) **then**
return *REDUCTION(root.left)*
else
if *root.leftf* and (*not root.rightf*) **then**
return *REDUCTION(root.right)*
else
if (*not root.leftf*) and (*not root.rightf*) **then**
if *root.first = 0* **then**
return *REDUCTION(root.right)*
else
if *root.first = 1* **then**
return *REDUCTION(root.left)*
else
if *root.first = 2* **then**
return *OrNode(REDUCTION(root.left), REDUCTION(root.right))*
end if
end if
end if
end if
end if
end if
return NIL

Algoritmo 11 NotNode::REDUCTION

Input: *root*
return *NotNode(REDUCTION(root.right))*

Algoritmo 12 ConditionNode::REDUCTION

Input: *root*
return *ConditionNode(copyCond(root.evtCond))*

Algoritmo 13 AndNode::ASSOCDEPEVENTS

Input: *root, evt* and *env*
if (*not root.leftf*) and (*not root.rightf*) **then**
ASSOCDEPEVENTS(root.left, evt, env)
ASSOCDEPEVENTS(root.right, evt, env)
end if

Algoritmo 14 OrNode::ASSOCDEPEVENTS

Input: *root*, *evt* and *env*

```

if (not root.leftf) then
    ASSOCDEPEVENTS(root.left, evt, env)
end if
if (not root.rightf) then
    ASSOCDEPEVENTS(root.right, evt, env)
end if
```

Algoritmo 15 NotNode::ASSOCDEPEVENTS

Input: *root*, *evt* and *env* {Esta função não faz nada pois se uma condição que depende de outro evento está negada então deixa de estar dependente}

Algoritmo 16 ConditionNode::ASSOCDEPEVENTS

Input: *root*, *evt* and *env*

```

ASSOCEVENT(root.evtCond, evt, env) {A função ASSOCEVENT é implementada pelas diferentes condições, mas a semântica é: se a condição for dependente de outro evento então obtém o evento de que depende, do Ambiente, e associa-se a este como seu filho e associa-o, a si próprio como seu pai.}
```

Algoritmo 17 AndNode::CLEANSTATE

Input: *root*

```

CLEANSTATE(root.left)
CLEANSTATE(root.right)
root.leftf  $\Leftarrow$  TRUE
root.rightf  $\Leftarrow$  TRUE
```

Algoritmo 18 OrNode::CLEANSTATE

Input: *root*

```

CLEANSTATE(root.left)
CLEANSTATE(root.right)
root.leftf  $\Leftarrow$  TRUE
root.rightf  $\Leftarrow$  TRUE
root.first  $\Leftarrow$  -1
```

Algoritmo 19 NotNode::CLEANSTATE

Input: *root*
CLEANSTATE(root.right)
root.rightf \Leftarrow FALSE

Algoritmo 20 EventTree::RESOLVETRANSITIVITY

Input: *env*
temp \Leftarrow *env*
while *temp* \neq NULL **do**
 for *i* = 0 to *size(temp)* – 1 **do**
 evt \Leftarrow *temp[i]*
 RESOLVETRANSITIVITY(*evt*)
 end for
 temp \Leftarrow *endScope(temp)*
end while

Algoritmo 21 EventNode::RESOLVETRANSITIVITY

Input: *evt*
temp \Leftarrow []
for *i* = 0 to *size(evt.parents)* – 1 **do**
 for *j* = 0 to *size(evt.parents)* – 1 **do**
 if *evt.parents[i].id* \neq *evt.parents[j].id* **then**
 size \Leftarrow PATHO(*evt.parents[j], evt.parents[i].id*)
 if *size* > 0 **then**
 insert(temp, parents[i])
 end if
 end if
 end for
end for
for *i* = 0 to *size(temp)* – 1 **do**
 removeSon(temp[i], evt.id)
end for

5.4.3 Execução dos Eventos de Jogo

A partir do momento em que já se possui a lista de árvores de eventos, podem-se executar os eventos do jogo. A classe que fica responsável por manipular a estrutura de dados dos eventos, ao longo do jogo, é designada por `InStoryEventTree`. Esta classe possui duas funções muito importantes:

- `getAvailableEvents`: Esta função devolve os eventos que podem ocorrer no momento.
- `setEventDone`: Esta função manipula a estrutura de dados dos eventos de forma a consumir os eventos.

A função `getAvailableEvents`, para devolver os eventos que podem ocorrer no momento tem de avaliar as condições. A avaliação é feita invocando a função `verifyGoal`, da raiz da árvore lógica das condições. Esta função recebe como argumento o estado do mundo (uma instância da classe `InStoryUserWorldState`), e se as condições forem satisfeitas então, o evento pode ocorrer no momento.

Algoritmo 22 EventNode::PATHTO

```

Input: evt and id
  if evt.id = id then
    return 1;
  end if
  acum ← 0
  for i = 0 to size(evt.parents) do
    acum ← acum + PATHTO(evt.parents[i], id)
  end for

```

A função `setEventDone` é um pouco complexa pois implementa um conjunto de regras importantes para se poder manipular a lista das árvores de eventos. Quando um evento é consumido, a lista vai herdar os filhos desse nó, mas para poder herdar é necessário que os filhos satisfaçam duas restrições: o filho que vai ser herdado não existir na própria lista e os eventos dos quais o filho depende já terem sido todos consumidos.

Em relação à primeira restrição, caso já exista um evento na lista com o mesmo identificador que o filho, que vai ser herdado, faz-se uma fusão dos dois eventos. Esta fusão de eventos apenas envolve a junção das duas árvores lógicas através de uma disjunção, pois o restante conteúdo dos eventos é comum. Em relação à segunda restrição, se o filho depender exclusivamente de dois eventos e apenas um deles já tiver sido consumido, o filho não pode ser herdado e é apagado da estrutura de dados. Podemos apagar o evento porque de certeza que ele ainda existe na estrutura de dados como filho do evento do qual depende.

Na figura 5.4, podemos observar um exemplo de como funciona a estrutura de dados quando se consumem eventos. Neste exemplo podemos observar que o evento 2 depende do evento 1 e do evento 3 mas não exclusivamente, ou seja, ele depende do evento 1 **ou** do evento 3, e dai poder ser logo herdado quando o evento 1 é consumido.

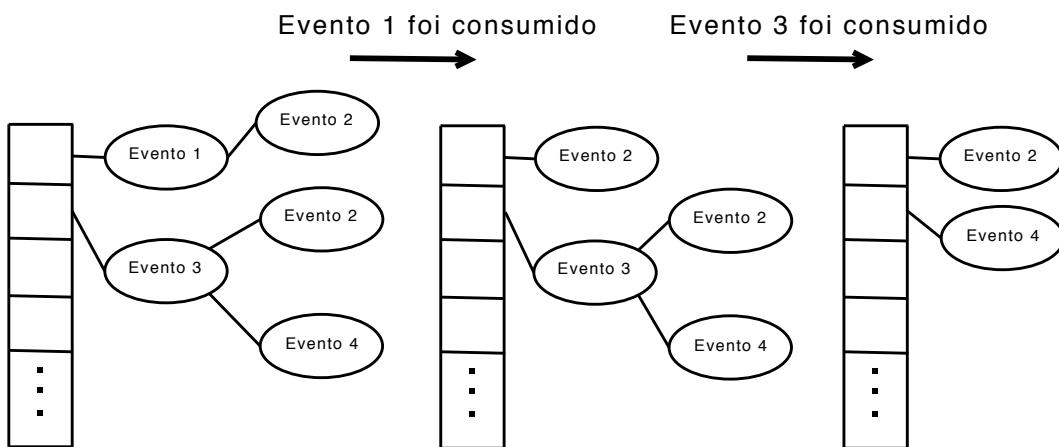


Figura 5.4: Manipulação da estrutura de dados dos eventos

A função `getAvailableEvents` é invocada sempre que o cliente envia para o servidor a posição do utilizador. Como resposta a esta informação é enviada para o cliente a lista de eventos disponíveis. Caso haja algum evento disponível o cliente escolhe o primeiro evento, pede informação sobre este ao servidor e pede o respectivo *layout* à base de dados do jogo. Quando o servidor recebe o pedido de informação de um evento, processa-se o envio do conteúdo da secção de `input` do evento. O conteúdo da secção de `input`, está sob a forma de uma árvore de sintaxe abstracta que é avaliada para retirar a informação que se

pretende enviar para o cliente. Depois de enviada essa informação para o cliente, este envia para o servidor a confirmação da execução do evento, juntamente com uma eventual resposta ao evento, caso este assim o exigisse. O servidor por sua vez avalia a árvore de sintaxe abstracta da secção de **consequence** por forma a actualizar o estado do mundo e por ultimo invoca a função `setEventDone` para a consumir da estrutura de dados.

5.4.4 Classes Dinâmicas

As condições que se inserem nas secções de **trigger** e **consequence**, e os tipos de eventos não estão limitados ao que se implementou neste trabalho. As condições e os tipos são representados por classes que se podem adicionar à aplicação dinamicamente. Durante a interpretação da linguagem, quando o *parser* lê o identificador de uma condição, ou de um tipo de evento, utiliza um mecanismo de carregamento dinâmico de classes implementado pelas classes `EventConditionFactory` e `ASTTypeNodeFactory`. Estas duas classes, através do nome da classe, carregam para memória uma instância da classe, cujo apontador devolvem de volta ao *parser* para continuar o seu processo.

As classes que representam as condições ou os tipos de eventos têm de respeitar algumas regras para o sistema de carregamento dinâmico poder funcionar. No caso das condições, as classes têm de estender a classe abstracta `EventCondition` e implementar todas as funções abstractas. No caso dos tipos de eventos, as classes têm de estender a classe abstracta `ASTTypeNode` e implementar todas as funções abstractas. Na figura 5.5 pode ser observado as respectivas funções das duas classes.

As condições necessitam de argumentos, como o caso da condição de posição que necessita da latitude e longitude. Como não é possível passar os argumentos ao construtor da instância da classe, que é carregada dinamicamente, os argumentos são passados num vector invocando a função `addParams` com o respectivo vector em argumento. Há um pormenor que é necessário conhecer: a ordem dos argumentos descritos no *script* da linguagem é inversa à ordem dos argumento do vector. No caso da condição de posição vem primeiro a latitude e depois a longitude e no vector a ordem é inversa.

As classes dinâmicas além de terem que implementar todas as funções abstractas declaradas na suas super classes, ainda tem que definir uma função que é utilizada pelas classes responsáveis pelo carregamento dinâmico. A função que necessitam de implementar é a seguinte:

```
extern "C" {
    EventCondition *maker() {
        return new PositionCondition();
    }
}
```

Esta função implementa a invocação do construtor da classe que é carregada. Neste caso trata-se da classe `PositionCondition` mas a função é igual para todas as classes, tendo apenas de mudar o respectivo construtor. Para o sistema de carregamento funcionar, todas as classes têm de estar num directório definido num parâmetro do servidor, e todas elas têm de estar compiladas como bibliotecas.

5.4.5 Implementação do Estado do Mundo e Informação dos Actores

O estado do mundo é implementado pela classe `InStoryUserWorldState`. Os membros desta classe podem ser observados no diagrama de classes da figura 5.6. Por cada utilizador existe uma instância desta

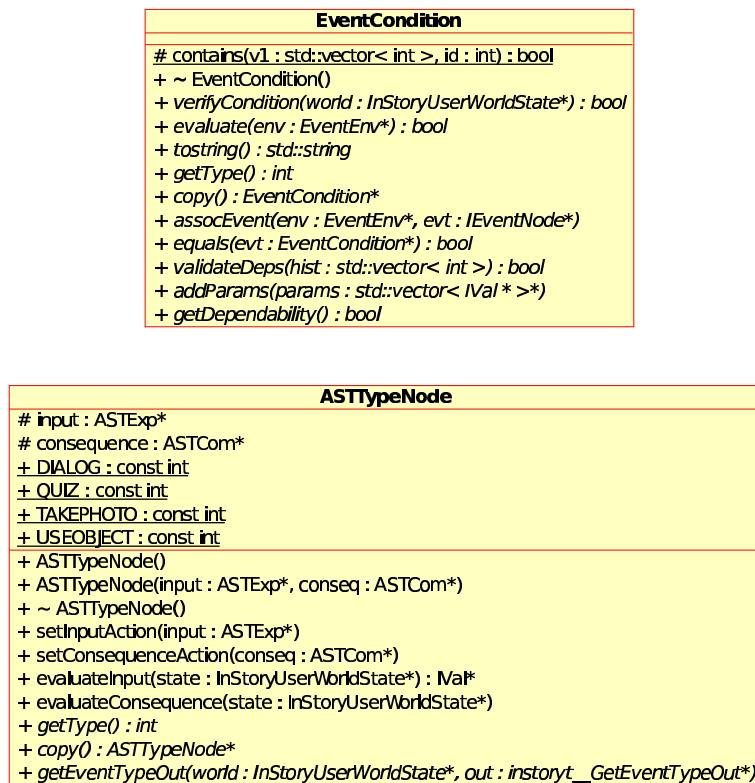


Figura 5.5: Classes abstractas que definem as condições e os tipos de evento

classe que é utilizada pelos eventos para verificar as condições e avaliar as acções.

A informação dos actores, nomeadamente as suas propriedades e os objectos que possuem são representados pelas classes que estão apresentadas no diagrama de classes da figura 5.7. Os actores virtuais e os objectos possuem uma lista de objectos da classe `InStoryRepresentation`. Essa lista contém a representação, tanto de um objecto como de um actor virtual. A representação pode ser de várias formas. Por exemplo, uma imagem ou um vídeo ou um som ou a combinação dos anteriores. O que define a representação é a implementação da classe base que estende da classe `InStoryRepresentation` e desta forma é possível acrescentar novos tipos de representação.

5.5 Exemplo de Funcionamento de um Jogo

Nesta secção apresentamos a demonstração da interacção entre um cliente e o servidor durante um jogo. Para demonstrar mostramos o *script* do jogo e, de seguida, a interacção do cliente com o servidor a consumir eventos. Este jogo chama-se a *batalha* e é definido da seguinte maneira:

```

define position as PositionCondition
define afterevent as AfterEventCondition
define dialog as ASTTypeDialog
define useobject as ASTTypeUseObject
define lasteventresult as LastEventResultCondition
define hasobject as ObjectCondition
define lastevent as LastEventCondition

```

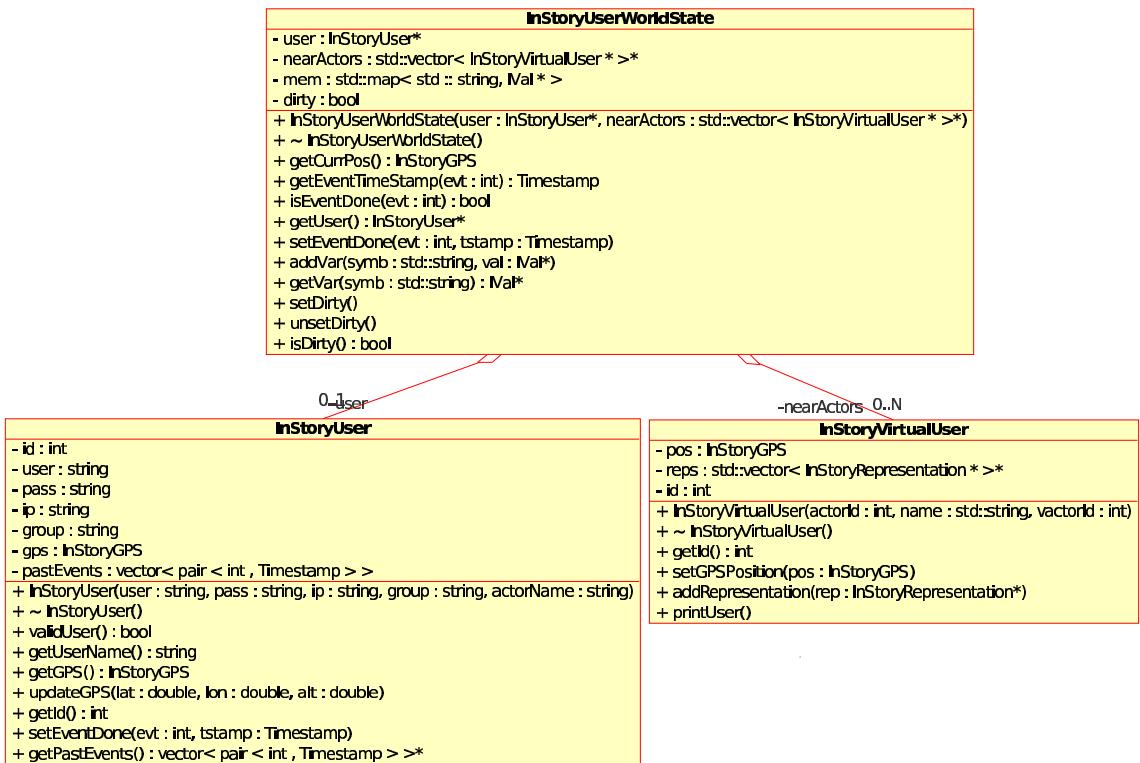


Figura 5.6: Diagrama de classes do Estado do Mundo

```

define quiz as ASTTypeQuiz

event 1 {
    trigger:
        position(34.5, 34.5)
    precondition:
        hasobject(flauta)
    type dialog {
        input {
            ["Saudações, eu sou o DEUS Aries \nE vou-te mandar pelos ares :p"]
        }
        consequence {
            ares.properties.life = 200;
        }
    }
}
event 7 {
    type quiz {
        input {
            ["Escolha a opção:", "Lutar", "Fugir"]
        }
    }
}
event 2 {
    precondition:

```

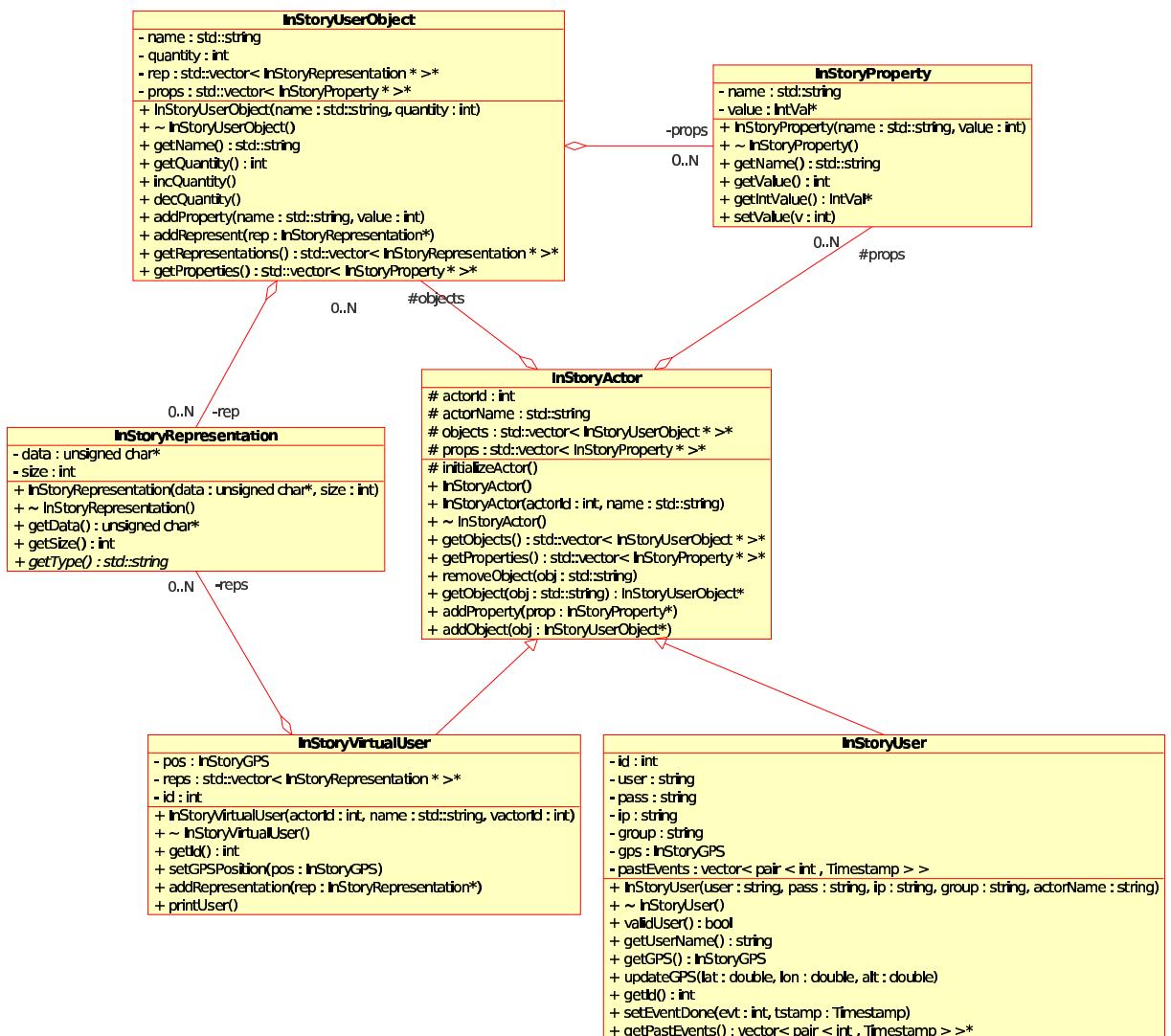


Figura 5.7: Diagrama de classes da informação dos actores

```

lasteventresult(Lutar)
type useobject {
    input {
        [ "Use o objecto certo para vencer Aries",
          user.objects, user.properties.Life,
          ares.properties.life]
    }
    consequence {
        user.objects -= event2.result;
        lostpoints = 50;
        if (event2.result == "flauta") {
            ares.properties.life = 0;
        }
        else {
            user.properties.Life -= lostpoints;
        }
    }
}
event 3 {
    precondition:
        lasteventresult(flauta)
    type dialog {
        input {
            [ "Parabéns, consegui vecer Aries"]
        }
    }
}
survive event 4 {
    precondition:
        (not lastevent(4)) and
        hasobject(flauta) and
        (not afterevent(3))
    type dialog {
        input {
            [ "Esse objecto não é eficaz contra Aries \nAcabou de perder ",
              lostpoints, "pontos de vida"]
        }
    }
}
survive event 5 {
    precondition:
        lastevent(4)
    type useobject {
        input {
            [ "Use o objecto certo para vencer Aries",
              user.objects, user.properties.Life,
              ares.properties.life]
        }
    }
}

```

```

consequence {
    user.objects -= event5.result;
    if (event5.result == "flauta") {
        ares.properties.life = 0;
    }
    else {
        user.properties.Life -= lostpoints;
    }
}
}

event 6 {
    precondition:
        afterevent(3)
    type dialog {
        input {
            [ "Como derrotou Aries tem direito à sua espada" ]
        }
        consequence {
            user.objects += "espada";
        }
    }
}
}

chain 1->7->2->[3,4->5]->6

```

Este jogo é bastante simples tendo o utilizador que combater com o Deus Aries que apenas pode ser derrotado utilizando a flauta. O jogo começa quando o utilizador chega à posição: latitude igual a 34.5 e longitude igual a 34.5. Nesta altura o cliente pede a informação sobre o evento 1:

```
<client> infoevent 1
```

E obtém a resposta:

```

<server>
Type: Dialog
Text:
    Saudações, eu sou o DEUS Aries
    E vou-te mandar pelos ares :p
Objects:

```

O campo **Type** é o tipo de evento que neste caso é um diálogo. Nesta altura o cliente diz ao servidor que quer consumir o evento:

```
<client> consumeevent 1
```

E o servidor envia a seguinte resposta a indicar se existe logo de seguida outro evento:

```
<server> NextEvent: 7
```

Daqui para a frente apresentamos a interacção, de forma contínua, sem interrupções.

```
<client> infoevent 7

<server>
Type: Quiz
Text:
    Escolha a opção:
    Lutar
    Fugir
Objects:

<client> consumeevent 7 Lutar

<server> NextEvent: 2

<client> infoevent 2

<server>
Type : UseObject
Text:
    Use o objecto certo para vencer Aries
    User: 200
    Aries: 200
Objects:
    potion = 5
    Life = 50
    sword = 1
    Atack = 5
    flauta = 1
    Atack = 100

<client> consumeevent 2 sword

<server> NextEvent: 4

<client> infoevent 4

<server>
Type : Dialog
Text:
    Esse objecto não é eficaz contra Aries
    Acabou de perder 50 pontos de vida
Objects:

<client> consumeevent 4

<server> NextEvent: 5
```

```
<client> eventinfo 5

<server>
Type : UseObject
Text:
    Use o objecto certo para vencer Aries
    User: 150
    Aries: 200
Objects:
    potion = 5
    Life = 50
    flauta = 1
    Atack = 100

<client> consumeevent 5 flauta

<server> NextEvent: 3

<client> infoevent 3

<server>
Type : Dialog
Text:
    Parabéns, conseguiu vecer Aries
Objects:

<client> consumeevent 3

<server> NextEvent: 6

<client> infoevent 6

<server>
Type : Dialog
Text:
    Como derrotou Aries tem direito à sua espada
Objects:

<client> consumeevent 6

<server> NextEvent:
```

A interacção termina quando o servidor indica que já não existe mais nenhum evento que pode ocorrer naquele momento. Podemos ver nesta interacção que as respostas a eventos influenciam a ocorrência dos próximos eventos. No caso do evento 2 e do evento 5 em que se escolhe o objecto para atacar Aries, dependendo do tipo de objecto, ocorrem diferentes eventos em resposta.

6

Interface de Visualização e Controlo 3D

Conteúdo

6.1 Apresentação da Interface 3D e Funcionalidades	68
6.2 Implementação	69

Neste capítulo iremos descrever os pormenores de implmentação da interface de visualização e controlo 3D. Esta interface foi implementada com o intuito de visualizar e controlar os conteúdos do sistema que se encontram nas bases de dados. Com esta interface é possível visualizar de forma espacial todos os conteúdos que possuam informação associada, incluindo os jogadores do jogo e os pontos de informação contextual.

6.1 Apresentação da Interface 3D e Funcionalidades

Esta interface 3D foi implementada com a finalidade de proporcionar novas formas de visualizar informação em narrativas interactivas. O sistema para suporte de narrativas interactivas, que este projecto implementa, possui uma propriedade importante, a relação com o espaço. Os utilizadores, ao interagirem com o sistema, movimentam-se no espaço real, logo torna-se necessário que se encontrem formas interessantes e inovadoras de se poder visualizar os dados gerados por sistemas como este.

Esta interface funciona apenas sobre a informação contida nas base de dados descritas no capítulo 3, ou seja, utiliza a informação contida nas bases de dados para representar o estado actual do sistema.

Nesta interface temos vários elementos que pretendemos visualizar. Os elementos são os seguintes:

- Mapa.
- Jogadores.
- Pontos de Informação.
- Janelas 2D.

Alguns destes elementos são estáticos no sentido em que não podemos interagir com eles e outros permitem que haja interacção directa a partir desta interface.

O mapa corresponde ao espaço físico onde os jogadores se encontram a jogar. Este mapa é carregado a partir da informação guardada na base de dados do jogo, podendo-se escolher vários mapas desde que estes se encontrem na base de dados.

Os jogadores, como o nome indica, correspondem aos utilizadores que interagem com o sistema no âmbito do jogo. Um jogador é um elemento estático pois apenas podemos observar a sua posição no mapa e ver o seu estado do decorrer do jogo, sem poder fazer qualquer interacção com este.

Os pontos de informação correspondem aos locais (secção 3.1.1) da base de dados dos conteúdos informativos. Os locais possuem vários tipos de informação associados como imagens, texto, sons e vídeos e como tal é possível nesta interface visualizar uma pequena parte desta informação nomeadamente uma imagem e um texto. Este elemento permite interacção directa, podendo mudar-se a posição do ponto de informação utilizando esta interface.

As janelas 2D são usadas para facilitar a interacção com o sistema que por vezes se pode tornar algo complexo por ser a três dimensões.

Na figura 6.1 podemos observar os elementos descritos. Nesta figura encontram-se quatro pontos de informação e um jogador, todos eles sobre o mapa da Quinta da Regaleira.

O tempo de actualização dos dados, que a interface obtém das base de dados, pode ser definido pelo utilizador fora da interface. Por omissão a posição dos jogadores é actualizada de cinco em cinco segundos e dos pontos de informação de quarenta em quarenta segundos.

Nesta interface 3D utiliza-se o rato e o teclado para navegação sobre esta. O rato controla um ponteiro, com que se pode interagir com o menu de escolha de mapas, e permitindo seleccionar um ponto de informação e arrastá-lo para outra localização. Com a ajuda de uma tecla e do rato podemos navegar pelo espaço 3D da interface e utilizar a roda de *scroll*, para aumentar ou diminuir a profundidade no espaço 3D.



Figura 6.1: Interface de Visualização 3D.

6.2 Implementação

A interface foi implementada na linguagem de programação C++ em ambiente Windows. Para implementarmos o espaço 3D desta interface de visualização e controlo foi necessário recorrer a um motor gráfico *open source*, utilizado na implementação de jogos de computador, denominado por OGRE 3D.

Este motor gráfico facilitou muito no desenvolvimento da interface pois permite-nos abstrair das primitivas gráficas 3D de baixo nível uma vez que ao utilizar o paradigma de programação *Object Oriented* foi mais fácil estruturar a interface por forma a torna-la extensível.

A arquitectura da interface é constituída por vários componentes:

- Gestor do Espaço: Este componente gera todos os elementos que estão contidos na interface incluindo os elementos 2D e os 3D.
- Interface 2D: Este componente é responsável por todas as janelas 2D que são apresentadas no ecrã.
- Entidade: Este componente representa todos os elementos que estão contidos no espaço 3D.
- Gestor de Sincronização: Este componente gera a actualização da informação que se encontra nas bases de dados.
- Parâmetros da Interface: Este componente guarda todos os parâmetros de configuração da interface.

Todos estes componentes são constituídos por uma ou mais classes C++ e estão associados entre si.

De seguida iremos descrever em detalhe a função e constituição de cada componente desta interface.

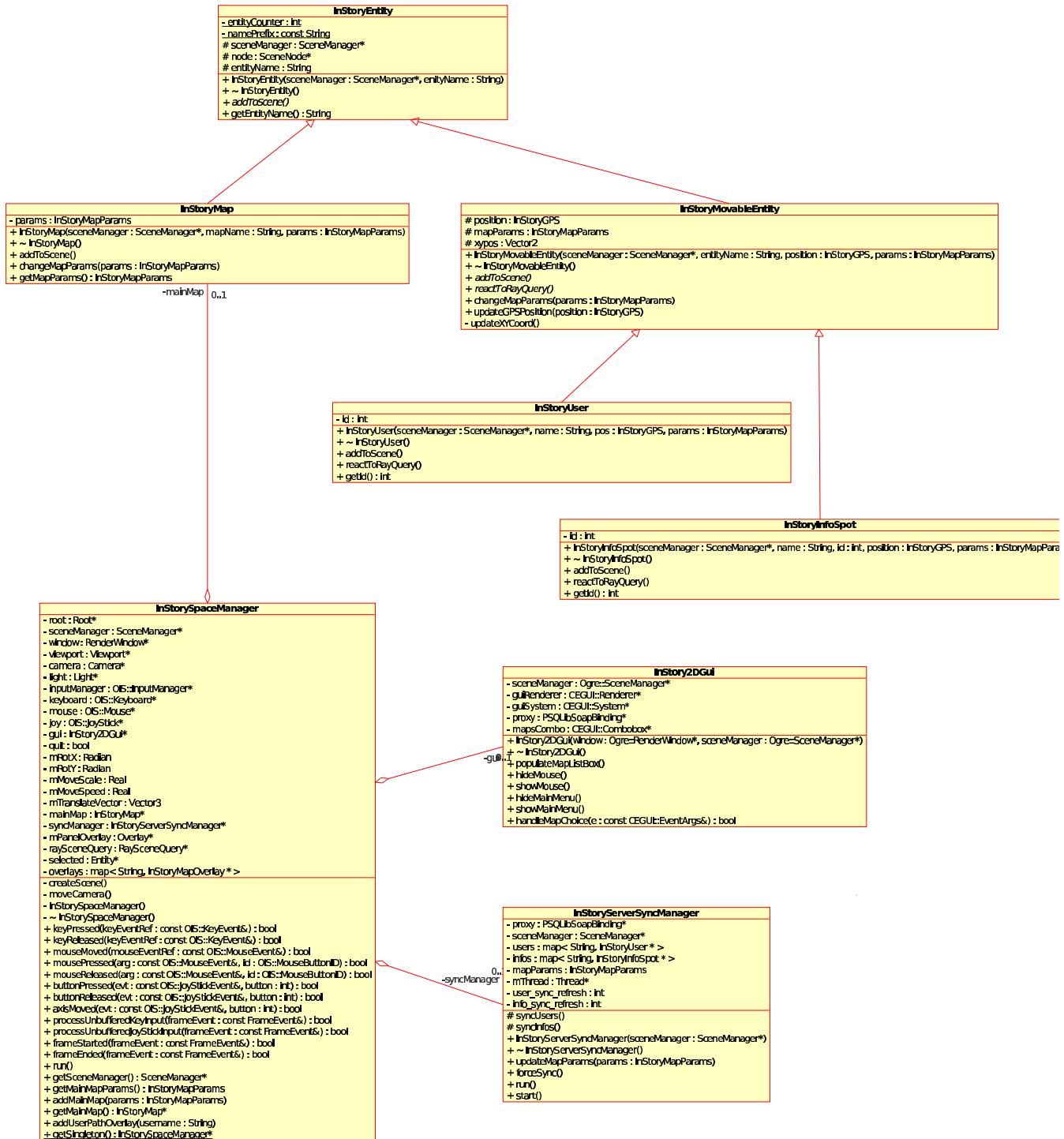


Figura 6.2: Diagrama de classes das classes mais importantes da Interface 3D.

6.2.1 Gestor do Espaço

O gestor do espaço é representado por uma única classe designada por `InStorySpaceManager`. Esta classe, só por si, controla o ciclo principal do programa, o desenho dos elementos gráficos e ainda todo o *input* dos periféricos.

O ciclo principal do programa é controlado por duas funções que são invocadas no inicio e no fim de cada *frame*. Estas duas funções chamam-se `frameStarted` e `frameEnded`.

Os eventos criados pelo teclado e pelo rato são tratados por um conjunto de funções que a classe implementa. Os eventos são gerados por uma biblioteca externa ao OGRE mas que funciona em conjunto com este, designado por OIS (Output Input System).

No construtor desta classe é inicializado o gestor de sincronização. Este insere, através do gestor de espaço, os elementos gráficos que são apresentados no ecrã.

6.2.2 Interface 2D

A interface 2D é implementada utilizando uma biblioteca externa ao OGRE designada por CEGUI. Esta biblioteca implementa vários componentes na construção de interfaces baseadas em janelas.

A utilização deste tipo de interfaces serve como mais um meio simples de comunicação entre o utilizador e o sistema. Neste caso concreto, implementamos uma janela com uma *listbox* que contém todos os mapas que o utilizador pode escolher para visualizar. e também inclui um botão de saída da aplicação.

A classe que gera as janelas é a classe `InStory2DGui`. Esta classe gera todos os controlos das janelas incluindo os eventos relacionados com estas.

6.2.3 Entidade

As entidades nesta interface são todos os elementos que são apresentados no ecrã, como o mapa, os pontos de informação e os utilizadores. Cada elemento é representado por uma classe e todas elas pertencem a uma hierarquia de classes em que a classe base é `InStoryEntity`. Esta classe implementa uma função abstracta `addToScene` que define a maneira como o elemento é adicionado ao espaço 3D.

O mapa é uma instância da classe `InStoryMap` que estende directamente da classe `InStoryEntity`, como se pode observar na figura 6.2.

Todos os elementos que dependem de uma localização espacial, como os pontos de informação e os utilizadores, pertencem à classe `InStoryMovableEntity`. Esta classe define variáveis e funções que tratam do mapeamento entre as coordenadas GPS e as respectivas coordenadas do espaço 3D. Esta classe também define uma função abstracta `reactToRayQuery`, que tem como finalidade ser invocada sempre que o utilizador observe um destes elementos no centro do ecrã. No caso dos pontos de informação, quando a função `reactToRayQuery` é invocada, é mostrado ao lado do ponto de informação, uma descrição do contexto do local com uma imagem e algum texto. Esta função pode ser encarada como um *handler* que é invocado quando o evento: *o elemento estar no centro da câmara*, é gerado.

Os pontos de informação são representados pela classe `InStoryInfoSpot` e esta estende da classe `InStoryMovableEntity` e como tal esta classe é que implementa as funções abstractas das suas super-classes, como a função `addToScene` e a função `reactToRayQuery`. Os utilizadores são representados pela

classe `InStoryUser`, que estende igualmente da classe `InStoryMovableEntity` e implementa também as respectivas funções abstractas.

Estas hierarquias de classes podem ser observadas no diagrama de classes da figura 6.2.

6.2.4 Gestor de Sincronização

O gestor de sincronização é representado pela classe `InStoryServerSyncManager`. Esta classe implementa as funcionalidades de sincronização da informação nas bases de dados.

A sincronização é implementada com um *thread* que sincroniza a informação necessária em intervalos de tempo. Os intervalos de tempo são variáveis para os diferentes tipos de informação. Para elementos como utilizadores, o intervalo de tempo é mais curto do que para os pontos de informação pois os utilizadores estão em constante movimento. Os elementos da interface que são sincronizados periodicamente são:

- Utilizadores.
- Pontos de Informação.

Os outros elementos são sincronizados com a base de dados quando a aplicação assim o necessita.

6.2.5 Parâmetros da Interface

Os parâmetros da interface são representados pela classe `InStoryServerGUIParams`. As variáveis desta classe são estáticas, estão inicializadas por omissão no próprio código e estão disponíveis para o resto das classes. Os parâmetros que esta classe define são:

- `user`: Nome de utilizador utilizado pelo gestor de sincronização para aceder às bases de dados.
- `pass`: Palavra chave utilizada pelo gestor de sincronização para aceder às bases de dados.
- `user_sync_refresh_time`: Intervalo de tempo da sincronização de utilizadores.
- `info_sync_refresh_time`: Intervalo de tempo da sincronização dos pontos de informação.
- `resources_path`: Caminho do sistema de ficheiros local onde se encontram os recursos necessários para o funcionamento da interface, como texturas e ficheiros de *scripting*.
- `cegui_layout_file`: Ficheiro que define a apresentação das janelas 2D.



Conclusões

Conteúdo

7.1 Conclusões	74
7.2 Trabalho Futuro	74

Neste capítulo serão apresentadas as conclusões sobre o trabalho desenvolvido e algumas propostas de trabalho futuro.

7.1 Conclusões

A realização deste projecto obrigou o estudo de várias áreas da informática pois este assim o exigia. Começamos por implementar a base de dados dos conteúdos informativos e a interface *web* para que a Quinta da Regaleira pudesse desde logo começar a inserir informação na base de dados. Tal não se verificou desde logo, e a interface *web* foi sofrendo alterações ao longo do trabalho.

A base de dados do jogo foi o que fizemos a seguir. Fizemos uma versão preliminar que foi sendo alterada conforme se definia as regras do jogo. O servidor foi implementado na mesma altura. Começamos por implementar a gestão de sessões de utilizadores e a gestão dos conteúdos informativos. Esta componente exigiu algum esforço pois a implementação partiu do zero. Quando ficou implementada a gestão de utilizadores do servidor, implementamos a interface 3D pois a interface 3D só observa a base de dados dos jogo para apresentar informação. A interface 3D ficou muito simples e por falta de tempo não se implementaram mais funcionalidades, além das descritas no capítulo 6.

Por último implementamos a gestão dos eventos do jogo. Esta parte do trabalho foi a que teve um período de implementação maior. Começamos por pesquisar algumas linguagens interpretadas em *runtime* para descrever os eventos mas acabamos por decidir implementar a nossa própria linguagem de eventos. Esta linguagem foi construída incrementalmente, adicionando novas construções à medida que iam sendo necessárias. Ao mesmo tempo fomos implementando a interacção do cliente e servidor no modo de jogo.

O algoritmo implementado para gerar a árvore de eventos (secção 5.4.2), é de bastante complexidade e sendo a linguagem de implementação C++, que não possui um *garbage collector*, foi impossível manter a execução do algoritmo sem fugas de memória (variáveis que não são libertadas). Para remediar a situação e dado que a execução do algoritmo demora algum tempo, o algoritmo só executa uma vez por cada jogo que for carregado, ou seja, só é gerada uma árvore de eventos por cada jogo e sempre que temos mais de um utilizador a jogar é feita uma cópia da árvore não sendo necessário executar o algoritmo.

7.2 Trabalho Futuro

Existem vários hipóteses de trabalho futuro neste projecto. O mais importante talvez seja a criação de uma ferramenta de *authoring* para criação do jogo. Esta ferramenta teria de ser capaz de criar o *script* de jogo com os eventos e as respectivas condições e acções, e ainda o conteúdo visual de cada evento. Esta criação devia ser feita de modo visual, com diagramas de blocos, pois o objectivo seria que as pessoas sem conhecimentos técnicos pudessem editar novos jogos. Esta ferramenta ainda é de mais importância pelo facto de neste momento ser necessário criar os *scripts* à mão e todos os conteúdos dos eventos terem de ser inseridos na base de dados à custa de *queries SQL*.

Outra componente a evoluir seria a interface 3D. Esta interface tem um bom potencial para criar novas formas de visualização de sistemas como este. Seria interessante poder interagir com os próprios utilizadores através desta interface, através de mensagens ou outra forma de comunicação, e também ter mapas completamente a três dimensões para se poder navegar pelos próprios edifícios do local.

Por ultimo, deveria ser melhor explorada a ideia das contribuições dos utilizadores com imagens e vídeos. Este trabalho não implementou esta componente mas não será muito difícil incorporar esta ideia no trabalho realizado. feito.

Bibliografia

- [1] Leandro Motta Barros and Soraia Raupp Musse, *Introducing narrative principles into planning-based interactive storytelling*, ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology (New York, NY, USA), ACM Press, 2005, pp. 35–42.
- [2] M. Cavazza, F. Charles, and S. J. Mead, *Characters in search of an author: Ai-based virtual storytelling*, ICVS 2001: International Conference on Virtual Storytelling.
- [3] M. Cavazza, F. Charles, and Steven J. Mead, *Character-based interactive storytelling*, IEEE Intelligent Systems, special issue on AI in Interactive Entertainment, vol. 17, Los Alamitos, United States, 2002, pp. 17–24.
- [4] Marc Cavazza, F. Charles, and S. J. Mead, *Emergent situations in interactive storytelling*, ACM Symposium on Applied Computing, ACM Press, Madrid, Spain, 2002, pp. 1080–1085.
- [5] Chris Crawford, *Chris Crawford on interactive storytelling*, New Riders, Berkeley, 2004.
- [6] M. Crow, P. Pan, L. Kam, and G. Davenport, *M-views: A system for location-based storytelling*, In Proceedings of Ubiquitous Computing (Ubicomp) (Seattle, Washington), October 2003, pp. 31–34.
- [7] N. Davies, K. Cheverst, K. Mitchell, and A. Efrat, *Using and determining location in a context-sensitive tour guide*, IEEE Computer, 2001, pp. 35–41.
- [8] S. Domnitchevai, *Location modeling: State of the art and challenges*, Workshop on Location Modeling for Ubiquitous Computing, Ubicomp 2001, Atlanta, Georgia, September 30- October 2 2001.
- [9] R. Want et. al, *Expanding the horizons of location-aware computing*, IEEE Computer, August 2001, pp. 31–34.
- [10] S. Long et al, *Rapid prototyping of mobile context-aware applications: The cyberguide case study*, In proceedings of Mobicom 96 (New York), ACM Press, 1996, pp. 97–107.
- [11] Yu-Chee et al, *Location awareness in ad hoc wireless mobile networks*, IEEE Computer, 2001, pp. 31–34.
- [12] J. Hightower and G. Borriello, *Location systems for ubiquitous computing*, Computer, vol. 34, IEEE Computer Society Press, August 2001, pp. 57–66.
- [13] N. Lin, K. Mase, Y. Sumi, and Y. Katagiri, *Interactive storytelling with captured video*, Ubicomp 2004, Adjunct Proceedings (Interactive poster), Ubicomp 2004, Nottingham, England, September 2004.

BIBLIOGRAFIA

- [14] Isabel Machado, Rui Prada, and Ana Paiva, *Bringing drama into a virtual stage*, CVE '00: Proceedings of the third international conference on Collaborative virtual environments (New York, NY, USA), ACM Press, 2000, pp. 111–117.
- [15] J. Miller, *Storytelling evolves on the web: Case study: Exocog and the future of storytelling*, Interactions, vol. 12, ACM Press, 2005, pp. 30–47.
- [16] Pengkai Pan, *Mobile cinema*, Ph.D. thesis, MIT Media Laboratory, 2004.
- [17] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, *The cricket location-support system*, 6th ACM International Conference on Mobile Computing and Networking (Boston), ACM Press, 2000, pp. 32–43.
- [18] Robert van Engelen, *Code generation techniques for developing light-weight xml web services for embedded devices*, SAC '04: Proceedings of the 2004 ACM symposium on Applied computing (New York, NY, USA), ACM Press, 2004, pp. 854–861.