# DIGEO: Uma Ontologia de Jogos Digitais

por

Filipe Ghesla Silvestrim

UNIVERSIDADE DO VALE DO RIO DOS SINOS

FILIPE GHESLA SILVESTRIM

# DIGEO: Uma Ontologia de Jogos Digitais

Monografia apresentada como requisito parcial para a obtenção do grau de Tecnólogo em Jogos Digitais

Prof. Msc. João Ricardo Bittencourt
Orientador

São Leopoldo, novembro de 2009

*Dedico este trabalho aos meus pais, ao meu irmão e aos meus amigos verdadeiros.*
*Estes representam o alicerce da minha vida e das minhas aspirações.*

# AGRADECIMENTOS

Expresso os mais sinceros agradecimento a todas as pessoas que desempenham o papel de guias em meu aprendizado. Aprendizado esse, referente à vida e a maneira como a vivemos. Visto isso, é de grande importância salientar que estes mentores são os indivíduos responsáveis por terem me ajudado a galgar, por meio de muita batalha e persistência, os degraus que premeditam todas as conquistas em minha vida.

*If there will be one overriding evolution in game design (as distinct from game content), we would say it will be a move towards starting out with a generic design. This is consistent with (and indeed a logical consequence of) the software factory model. Such designs will allow designers to start arguing from the general to the specific, which will make the design concepts more portable and robust.*

— Andrew Rollings & Dave Morris

# SUMÁRIO


**LISTA DE ABREVIATURAS E SIGLAS . . . . . 8**

**LISTA DE FIGURAS . . . . . 10**

**LISTA DE TABELAS . . . . . 12**

**RESUMO . . . . . 13**

**ABSTRACT . . . . . 14**

**1 INTRODUÇÃO . . . . . 15**

**2 ENGENHARIA DO CONHECIMENTO . . . . . 18**
2.1 Ontologia . . . . . 18
2.1.1 Conceitos . . . . . 20
2.1.2 Construindo uma Ontologia . . . . . 22
2.1.3 Ontologias na prática . . . . . 23
2.2 Arquétipos Vs. Modelos . . . . . 25
2.3 Ontologias Presentes no Meio Digital . . . . . 26

**3 O DESENVOLVIMENTO DE JOGOS DIGITAIS . . . . . 28**
3.1 Game Design . . . . . 29
3.1.1 Classificações . . . . . 30
3.2 Desenvolvimento . . . . . 30

**4 TRABALHOS E FERRAMENTAS RELACIONADOS . . . . . 32**
4.1 Trabalhos Relacionados . . . . . 32
4.1.1 SDK Gameplay - Ferramenta Voltada para edição de Gameplay . . . . . 32
4.1.2 Scald - Um Motor Genérico de Role-Playing Games Computadorizados Configurável por Scripts . . . . . 33
4.1.3 Arquitetura Reusável Baseada em Mecânica de Jogo para Jogos Casuais . 34
4.1.4 FACHwerk: Um Framework para Desenvolvimento de Jogos Competitivos 35
4.1.5 Open Game Engine . . . . . 36
4.1.6 Game Ontology Project . . . . . 36
4.1.7 Digital Game Ontology . . . . . 37
4.1.8 Ontologia de Videojuegos . . . . . 38
4.2 Ferramentas Relacionadas . . . . . 38
4.2.1 RPG Maker XP . . . . . 38
4.2.2 Game Maker . . . . . 39

4.2.3 PlayCrafter . . . 40
4.2.4 Power Game Factory . . . 41
4.2.5 Unity 3D . . . 42
4.2.6 Torque Game Builder . . . 43
4.3 Tabelas Comparativas . . . 43

**5 ANÁLISE DE JOGOS DIGITAIS . . . 46**
5.1 Categorização . . . 46
5.1.1 Elementos . . . 47
5.1.2 Categoria Primária . . . 49
5.1.3 Categoria Secundária . . . 49
5.1.4 Singularidade . . . 51
5.2 Análises de Títulos . . . 54

**6 DIGEO: DIGITAL GAME ELEMENTS ONTOLOGY . . . 55**
6.1 Contextualização . . . 55
6.2 Escopo, Domínio e Método . . . 56
6.2.1 Questões de Competência . . . 56
6.3 Referencial Teórico . . . 57
6.4 Etimologia e Termos . . . 58
6.5 A ontologia . . . 58
6.6 Classes de níveis primários . . . 59
6.7 Classe "Behavior" e arvore hierárquica . . . 60
6.8 Classe "Description" e arvore hierárquica . . . 61
6.9 Classe "Interaction" e arvore hierárquica . . . 64
6.10 Conclusão . . . 66

**7 IMPLEMENTAÇÃO . . . 67**
7.1 A linguagem . . . 68
7.1.1 Elementos Extras . . . 68
7.2 Multi-Genre Game Engine . . . 71
7.2.1 Núcleo . . . 72
7.2.2 Elementos . . . 72
7.2.3 Plugins . . . 72
7.2.4 Interpretador . . . 73
7.3 Aplicação . . . 73
7.3.1 Peculiaridades do Desenvolvimento . . . 74
7.4 Conclusão . . . 75

**8 PONG: UMA APLICAÇÃO DA DIGEO E MUGE . . . 77**
8.1 O título: Pong . . . 77
8.2 Descrição Ontológica . . . 78
8.3 Implementação . . . 81
8.4 Criando a especifica XML via *DIGEO Language* . . . 82
8.5 Resultados obtidos . . . 85

**9 CONSIDERAÇÕES FINAIS . . . 87**

**A ANEXOS . . . . . 90**
A.1 Repostas das Questões de Competência da DIGEO . . . . . 90
A.2 Estrutura XML do *Pong DIGEO* . . . . . 91

**REFERÊNCIAS . . . . . 93**

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ACM | Association for Computing Machinery |
| ARMJC | Arquitetura Reusável Baseada em Mecânica de Jogo para Jogos Casuais |
| CRPG | Computer Role-Playing Game |
| DIGEO | Digital Game Elements Ontology |
| DLL | Dynamic Linked Library |
| FBX | FiLMBOX |
| GML | Gameplay Language |
| GOP | Game Ontology Project |
| GUI | Graphic User Interface |
| HTML | Hyper Text Markup Language |
| IDE | Integrated Development Environment |
| IA ou AI | Inteligência Artificial |
| IEEE | Institute of Electrical and Electronics Engineers |
| Indie | Independent video game development |
| IT ou TI | Tecnologia da Informação ou Informática |
| KBS | Knowledge-based systems |
| Mac | Macintosh |
| MDA | Mechanics, Dynamics, and Aesthetics |
| MIT | Massachusetts Institute of Technology |
| MuGE | Multi-genre Game Engine |
| ODE | Open Dynamics Engine |
| OGE | Open Game Engine |
| OGEd | Open Game Engine Editor |
| OGRE | Open Source 3D Graphics Engine |
| OOP | Object-oriented programming |

| | |
|---|---|
| OWL | Web Ontology Language |
| PC | Personal Computer |
| RAD | Rapid Application Development |
| RDF | Resource Description Framework |
| RGSS | Ruby Game Scripting System |
| RPG | Role-Playing Game |
| SDK | Software Development Kit |
| TGB | Torque Game Builder |
| XML | Extensible Markup Language |
| W3C | World Wide Web Consortium |
| WYSIWYG | What You See Is What You Get |

# LISTA DE FIGURAS


2.1 Captura de tela do software Protégé com as especificações de classe da ontologia proposta pelo trabalho. . . . . 24

3.1 Diferentes pontos de vista sob jogos digitais por meio da representação MDA. . . . . 29
3.2 Etapas presentes num modelo de desenvolvimento jogos. . . . . 31

4.1 Diagrama metodológico do processo de desenvolvimento do proposto trabalho. . . . . 32
4.2 Editor de desenvolvimento de gameplay. . . . . 33
4.3 Partes da arquitetura ARMJC, destacando suas possibilidades de relacionamento. . . . . 34
4.4 Interface do FACHwerk Launcher . . . . . 35
4.5 Editor WYSIWYG de recursos e mundo da OGE. . . . . 36
4.6 Interface visual do RPG Maker XP. . . . . 39
4.7 Programas parte do Game Maker. O da esquerda é o editor de tiles e o da direita é o editor de eventos e ações. . . . . 40
4.8 Interface visual da plataforma PlayCrafter. . . . . 40
4.9 IDE da Power Game Factory acessível principalmente a cliques (interações com o mouse). . . . . 41
4.10 Ambiente de desenvolvimento da Unity 3D. Mistura características de modeladores 3D e de sistemas de controle de comportamentos. . 42
4.11 Editor drag-and-drop da Torque Game Builder, possibilitando uma manipulação de forma intuitiva. . . . . 43

5.1 Títulos com características visuais parecidas. . . . . 47
5.2 Exemplo de elementos visuais presentes no título Super Mario World. 47
5.3 Tabela de alguns dos elementos, dentre eles concretos e abstratos, presentes no título Super Mário World. . . . . 48
5.4 Divisão, entre modelos, dos valores padrões presentes na categoria primária. . . . . 50
5.5 Divisão, entre modelos, dos valores padrões presentes na categoria secundária. . . . . 50
5.6 Agrupamentos dos valores possíveis aos elementos de cada categoria. 52
5.7 Detalhamento dos títulos de jogos digitais quanto à categorias na qual se inserem e também quanto a suas características de singularidade (customização dos valore padrões dos elementos de determinado modelo). . . . . 53

6.1 Classes de primeiro grau da hierarquia. "Thing" representa o ponto de partida de toda e qualquer ontologia. . . . . 59
6.2 Ramo hierárquico filho do elemento **Behavior**. . . . . 61
6.3 Ramo hierárquico filho do elemento **Description**. . . . . 62
6.4 Ramo hierárquico filho do elemento **Interaction**. . . . . 65

7.1 Modelo de desenvolvimento proposto para a criação de um título por meio da DIGEO. . . . . 67
7.2 Fluxo de desenvolvimento de um título mediante a aplicação do modelo proposto. . . . . 74
7.3 Meio a esse esboço evidencia-se a interdependência entre os códigos sendo que o MuGE não requer a instalação prévia de nenhuma tecnologia — esse papel é desempenhado pelos plugins. . . . . 75

8.1 Captura de tela referente ao título Pong clássico. . . . . 78
8.2 Diagrama *mind-map* que representa a estrutura de elementos ontológicos do título a ser produzido. . . . . 80
8.3 Uso prático do MuGE e de seu plugin padrão num projeto do XCode. 81
8.4 Título do jogo digital obtido mediante a interpretação da definição em XML do *Pong DIGEO*. . . . . 86

# LISTA DE TABELAS


2.1 Comparativo entre as principais definições de tipos de ontologias por Almeida e outros [1] . . . . . 21

2.2 Tabela com os nomes das linguagens de representação ontológica mais conhecidas e um breve resumo a respeito das mesmas. . . . . 24

4.1 Comparativo perante os objetivos dos trabalhos que visam uma padronização de definições ligadas ao desenvolvimento de jogos. . . . . 44

4.2 Comparativo entre as principais ferramentas que propiciam o desenvolvimento de jogos mediante um processo de fácil compreensão. . . 45

5.1 Títulos analisados e suas classificações respectivas. . . . . 54

8.1 Representação descritiva dos elementos presentes a descrição do título proposto. . . . . 79

# RESUMO

Com o advento da era digital, muitos conhecimentos ligados à área da computação tiveram um crescimento em termos de suas especificidades e em termos de complexidade quanto à produção das tecnologias. Com o decorrer do tempo, a TI, junto a uma grande diversidade de abordagens ligadas a estudos e pesquisas (gerando novas tecnologias e modelos práticos), assumiu uma área de estudos muito abrangente e bastante complexa aos olhos de indivíduos que não fazem parte deste mundo. Presente como uma sub-área da computação, os jogos digitais visam o entretenimento mediante o uso da intuição das pessoas. Esta é a visão de jogos digitais sob um ponto de vista de um usuário final. Em contraste, o desenvolvimento dos jogos digitais é uma abordagem muito mais complexa, desde que envolve conhecimentos, técnicos e formais, especificos. A par desse contexto, intenta-se a construção de um conhecimento padronizado e de fácil compreensão sobre a área do desenvolvimento de jogos digitais. A fim de validar esta idéia de uma maneira formal, propõe-se a criação de uma ontologia: sendo a especificação de um conceito. Com isso, cria-se a *Digital Game Elements Ontology* (DIGEO) com o propósito de preencher a lacuna existente entre os usuários de jogos digitais e o conhecimento técnico necessário para o desenvolvimento dos mesmos. Para validar a aplicabilidade da DIGEO, instaurou-se o motor de jogos MuGE que atua como interpretador à *DIGEO Language* (linguagem de marcação resultado da implementação prática da ontologia), no auxilio ao desenvolvimento de títulos de jogos digitais. Assim sendo, valida-se a utilidade da DIGEO em meio a sua aplicação prática de forma a ter sido obtido a implementação de um título de jogo digital — o *Pong DIGEO*, um *remake* do clássico Pong.

**Palavras-chave:** Ontologia. Jogos Digitais. Modelo Formal. Desenvolvimento de Jogos.

**DIGEO: A Ontology of Digital Games**

# ABSTRACT

With the arrival of the digital era, much of the knowledge related to IT had an increase in terms of its specificity and in terms of complexity in technology development. Having a great diversity of approaches when it comes to studies and research's (which results in new technologies and practical models), IT became a very extensive and complex area in the eyes of individuals that do not belong to this world. As a sub-area of IT, digital games aim at entertainment by using people intuition. This is a perception of digital games in a player's point of view. In contrast, the development of digital games are much more complex since it involves formal and technological knowledge. Having said that, it would be very useful, the creation of a standardized structure of easy comprehension when it comes to the digital game development knowledge. In order to validate this idea in a formal way, it is suggested the creation of an ontology: which is the specification of a concept. With this, *Digital Game Elements Ontology* (DIGEO) is created for purpose of closing the gap between the digital games users and the technical knowledge necessary for game development. In order to the practical use of DIGEO, a game engine called MuGE was created, acting as an interpreter of the *DIGEO Language* (a markup language as result of a practial implementation of the ontology), helping to build titles of digital games. Therefore, is validated the usefulness of the DIGEO via its practical application in the way of obtain a implementation of a title of digital game — the *Pong DIGEO*, a remake of the Pong classic.

**Keywords:** Ontology, Digital Games, Formal Model, Game Development.

# 1 INTRODUÇÃO

Atualmente, processos de desenvolvimento de jogos digitais, em sua maioria, utilizam modelos de desenvolvimento formais [2; 3], visando arquiteturas e interdependências de módulos padrões como jogabilidade e tratamento de entradas. Tal abordagem é extremamente válida para profissionais ou programadores que já estão familiarizados com o processo de desenvolvimento de jogos, porém, usuários que não compartilham de um conhecimento formal, ligado à área de desenvolvimento de jogos, podem apresentar certas dificuldades em compreender definições de um jogo digital e não saber como é composto e/ou desenvolvido.

Como certas abordagens [4; 5] feitas por profissionais da área de desenvolvimento de jogos, destaca-se o interesse e a necessidade de padronizar o conhecimento ligado aos estudos de tal segmento. Conforme veremos ao longo da pesquisa, a definição criada a partir da formatação, especificação de uma conceitualização, dos estudos relacionados ao processo do desenvolvimento de *games*[1] proporciona a compreensão e utilização de um conceito fundamentado em uma única pergunta: "Sob o ponto de vista de um desenvolvedor, qual a composição de um jogo digital?". A par deste tema, propõe-se a criação de uma ontologia que defina a composição, ou seja, quais os elementos[2] de um jogo digital numa abordagem formal do estudo de *game design* (seção 3.1) sob uma perspectiva de um desenvolvedor.

Frente a isso, a criação de uma especificação formal de um conhecimento compartilhado (no caso o desenvolvimento de jogos digitais) auxiliaria no processo de criação de novos jogos digitais de diversas maneiras: facilitando a análise e a criação de jogos digitais utilizando taxonomias e definições de fácil compreensão a indivíduos não familiarizados com estudos formais (o acesso aos dados é obtido na utilização de funções auxiliares que atuam com a finalidade em fornecer os valores através de um acesso semanticamente semelhante); possibilitando a criação e a implementação do conhecimento a um modelo de negócios; criando uma base de conhecimento acessível de forma livre e cooperativa.

O principal objetivo do trabalho é a criação da ontologia *Digital Games Elements Ontology* (*DIGEO*) — sendo conhecida em língua portuguesa como "Ontologia de Elementos de Jogos Digitais", a qual atua como responsável na identificação e definição dos elementos abstratos que compõem um jogo digital. O termo "elemento" é utilizado com a intenção em descrever entidades visuais, comportamentais e interativas, ou seja, membros que representam um jogo digital como instância. Em outras

[1] Estrangeirismo, em língua inglesa, da palavra em português "jogos digitais".

[2] Neste trabalho o significado de elementos provém do estudo da matemática clássica, que evidencia um elemento como sendo um membro de um conjunto qualquer (como no caso aqui jogos digitais) composto por objetos distintos.

palavras, esta ontologia tende a expressar de forma mais clara e eficiente os padrões encontrados no mundo dos títulos de jogos digitais já produzidos. A possibilidade dessa abordagem, sendo uma definição prática e que utilize linguagens formais, é muito natural, pois apresenta um alto grau de sintonia com o mundo real, como neste caso, descrevendo exemplos reais de títulos de jogos digitais.

A fim de validar e enunciar de forma mais clara e específica à ontologia proposta, é necessária uma abordagem junto a diversas análises de estudos e ferramentas relacionadas. Considerando o fato que a presente monografia está inserida no curso *Tecnólogo em Jogos Digitais*, compreende-se a necessidade de implementar uma forma prática à teoria proposta, criando assim um enfoque transdisciplinar, cuja engenharia do conhecimento seja capaz de gerar uma base de conhecimentos formais e práticos, criando assim uma nova forma de ensino-aprendizagem num contexto de educação digital. O estudo dos Jogos Digitais têm de implementar e comprovar de forma prática e aplicável num cotidiano de um desenvolvedor, instâncias desse sistema. Após o desenvolvimento desse conhecimento, obter-se-á um conhecimento formal capaz de formalizar, frente ao desenvolvimento de uma linguagem própria, a *DIGEO Language* (seção 7.1) — uma linguagem de marcação que intenta agilizar o processo de desenvolvimento de jogos, baseando-se numa base de conhecimento padrão e formal.

Por isso, a fim de comprovar seu uso no mundo profissional ou educacional, será produzida a primeira versão de um motor de jogos que, através da interpretação da linguagem proposta, incentivará o desenvolvimento de aplicações usuais de jogos digitais. O *Multi-genre Game Engine* (MuGE) é o motor desenvolvido. O seu desenvolvimento proporciona tanto a programadores, prototipadores e projetistas de jogos, quanto a iniciantes na criação de jogos digitais um ambiente para criação de novos títulos de forma fácil e rápida, sem que hajam tarefas de programação que lancem mão do uso de teorias de jogos ou modelos estruturais — conhecimentos tópicos à área.

Visando à produção de um jogo digital, através de uma definição abstrata (conceitual, genérica e concisa), destaca-se a implementação de um modelo de desenvolvimento de jogos que utilize a DIGEO e o MuGE como forma a proporcionar um modelo de negócios que requer uma mão de obra de poucos desenvolvedores, isso pois tal metodologia (DIGEO + MuGE) possibilita curtos ciclos de estruturação e desenvolvimento de um título de acordo com a definição diagramática dos conceitos ontológicos ligados a um título (seção 8.4).

Certamente que um projeto desta magnitude é muito ambicioso e possibilita a criação de uma primeira versão da ontologia, bem como a implementação somente de recursos necessários à MuGE para que valide a aplicabilidade da teoria. Logo, faz-se necessário o desenvolvimento de um planejamento que cubra o espaço de tempo da monografia e uma futura aplicação no mercado de trabalho. Para isso, evidencia-se a necessidade de um desenvolvimento de uma base sólida e muito bem formalizada da ontologia, para que proporcione no futuro a continuação do desenvolvimento do MuGE com fins comerciais.

Com a finalidade de detalhar o processo de desenvolvimento das abordagens teóricas e práticas existentes nesta monografia, organiza-se o volume de uma maneira a envolver toda a teoria relacionada com o trabalho proposto. Dessa maneira, no primeiro capítulo, com o título **Engenharia do Conhecimento**, serão abordados assuntos que enquadram o pensamento e as teorias ligadas a ontologias, bem como

suas abordagens de forma teórica e prática. Posteriormente, em **O Desenvolvimento de Jogos Digitais**, o leitor será situado sob os estudos e as compressões sobre o desenvolvimento de jogos digitas de forma a serem explicadas as principais teorias e processos ligados, ao contexto teórico, à definição de desenvolvimento. Em seguida, em **Trabalhos e Ferramentas Relacionados**, serão feitas análises de ontologias de jogos já existes, bem como de trabalhos e ferramentas que possuem certas afinidades com as idéias propostas nesta monografia fazem-se de boa referência, bem como servem para abrir os horizontes quanto a existência de outras abordagens e de como elas são aplicadas. A fim de impulsionar a definição da ontologia, o capítulo **Análise de Jogos Digitais** expõe a análise de dezesseis jogos digitais sob um ponto de vista a categorizá-los em "categoria primária" e "categoria secundária" a fim de fazer uma análise que possibilitasse o futuro desenrolar da ontologia de uma forma mais contextual a taxonômica. Como ápice da monografia, tem-se o assunto da ontologia, em **DIGEO: Digital Game Elements Ontology**, abordando todas as informações relacionadas ao escopo da ontologia, bem como sua definição formal e minuciosa. Validando a aplicação da DIGEO, o capítulo **Implementação** é a etapa na qual serão descritas as abordagens de forma prática, sendo a implementação da linguagem de definição em XML da ontologia e o motor de jogos que interpreta a linguagem definida. A fim de exemplificar todas as propostas inferidas ao longo da monografia, a divisão da monografia possuidora do título **Pong: Uma Aplicação da DIGEO e MuGE** descreve a implementação da ontologia e do motor de jogos de forma a descrever e criar um *remake* do jogo Pong. Por último, em **Considerações Finais**, são externalizadas todas as conclusões obtidas atendendo ao desenvolvimento do trabalho, bem como os trabalhos futuros.

# 2 ENGENHARIA DO CONHECIMENTO

O processo de criação de novos títulos de jogos digitais baseia-se na aplicação da teoria dos jogos [6]. Os jogos, sejam digitais, sejam de tabuleiros, sejam desportivos requerem especificações comportamentais, interativas e descritivas — regras, objetivos, enredos — para fazer com que os mesmos existam como instâncias (títulos) presentes a um contexto global. Procurando entender como funciona a criação de novas instâncias de representações de conhecimentos formais, no caso específico de jogos digitais, salienta-se a importância do estudo do principal modelo de representação de conhecimento, isto é, a ontologia junto à forma de raciocínio primordial ao contexto de jogos — o *game design*.

Na busca por uma compreensão comum e compartilhada de conhecimentos específicos, a ser transmitido aos mais diversos públicos, destacam-se, no campo de pesquisa de sistemas digitais, propostas [7; 8] que objetivam a criação de um contexto que envolva o conhecimento de integração do meio ao desenvolvimento de formalismos, visando à resolução de problemas complexos os quais, usualmente, requerem um elevado nível de conhecimento humano. Com o objetivo de propor uma representação de informação que possibilite o compartilhamento e a reutilização de conhecimentos é destacado o estudo de ontologias. Tal estudo será abordado nas seções subseqüentes na presente pesquisa.

Visto que os estudos sobre conhecimentos compartilhados possuem um conceito central associado — domínio, faz-se necessária a análise do conhecimento junto a um conjunto de estudos que formalizam a teoria dos jogos digitais.

Visto que os estudos sobre conhecimentos compartilhados possuem um conceito central (domínio) associado, faz-se necessária, a análise do conhecimento junto a uma associação de estudos que formalizam a teoria dos jogos digitais. Por conseguinte, como área ou processo de essência teórica, no estudo descritivo e comportamental de um jogo, destaca-se o estudo do *game design*.

## 2.1 Ontologia

A consistência de um conhecimento formal é baseada em conceito de elementos — objetos, conceitos, entidades, relações [9], pertencentes ao contexto de domínio. Com isso uma conceitualização é identificada como sendo a visão abstrata e simplificada de uma informação a ser expressa. Consequentemente, como resultado de estudos e pesquisas ligados à área de conceitualização de conhecimento compartilhado, informações metodológicas [10] e ferramentas são apresentadas formalmente à comunidade com finalidades de operacionalizar o processo de desenvolvimento de *Sistemas Baseados em Conhecimento* (KBS) — sendo programas de computador

que utilizam o conhecimento (representado explicitamente) para resolver problemas complexos [11; 7; 10].

Assim, o conceito de ontologia faz-se presente desde a Grécia Antiga, período em que filósofos como Platão e Aristóteles faziam estudos relacionados à "natureza do ser, da existência, ou da realidade em geral, bem como das categorias de base do ser e suas relações" [12]. Logo, com o decorrer do tempo, o estudo de ontologias ganhou inúmeras áreas de abrangência, sendo alvo das mais variadas definições; não obstante, a existência de definições mesmo específicas (como a área de filosofia, presente no caso apresentado), existem definições também genéricas, como a proposta por Tom Gruber, em que enunciava que a ontologia "pode ser vista como o estudo da organização e da natureza do mundo, independentemente da forma de nosso conhecimento sobre ele" [13].

Dentre os tantos conceitos sobre ontologia, a definição voltada a "uma especificação explícita e formal de uma conceitualização compartilhada" [14] é escolhida como centro de estudo a esta pesquisa. Portanto, ao analisar tal definição, evidencia-se que uma ontologia serve para definir um vocabulário e um entendimento sobre a área de domínio de forma comum entre pessoas que compartilhem o interesse pelas informações contidas em tal situação.

Dessa forma, Mizoguchi [15] evidencia grupos de estudiosos que chegaram a um acordo a respeito de um conceito sobre ontologia, ou seja, eles desistem de uma definição comum sobre o significado da palavra em questão, uma vez que existem inúmeras divergências entre estudiosos a saber:

> *Ontologia é um conjunto de termos de conhecimento, incluindo o vocabulário, as interconexões semânticas, e algumas regras simples de inferência e lógica por algum tema específico.*
>
> — Hendler [16]

> *Uma ontologia é uma representação explícita de um entendimento comum de conceitos importantes em certo domínio de interesse.*
>
> — Kalfoglou [17]

> *Ontologias são acordos sobre a conceitualização compartilhada. Conceitualizações compartilhadas incluem frameworks para a modelagem de domínios de conhecimento; conteúdo especificado por protocolos para a comunicação entre os agentes de inter-operacionais; e acordos sobre a representação de teorias de domínio particular. No contexto de compartilhamento de conhecimento, ontologias são especificadas na forma de definições de um vocabulário de representação.*
>
> — Gruber [10]

Logo, faz-se uso de ontologias visando às facilidades de acesso ao conhecimento, a interoperabilidade entre formalismos, a possibilidade de comunicação e a troca de conhecimento de forma inteligível, se tratando de conhecimento técnico. A fim de evidenciar um contexto base (ponto de partida) para a especificação de uma conceitualização [18], destaca-se o estudo no campo da engenharia ontológica como sendo sucessor aos conceitos formais de KBS. No intuito de um estudo relacionado a um

conhecimento especifico, no caso da computação, utiliza-se a palavra ontologia como sendo a "descrição de conceitos e relações que podem existir para um agente ou uma comunidade de agentes" [19] ou também, conforme evidenciam, Studer, Benjamins e Fensel, "pesquisadores de IA, adotaram o termo ontologia para descrever o que pode ser computacionalmente representado no mundo em um programa" [7].

### 2.1.1 Conceitos

Para uma melhor compreensão das definições, principalmente no escopo computacional, entende-se a necessidade de uma abordagem dos elementos taxonômicos e sintáticos dos conceitos que abordam a mesma. A par dessa necessidade, conceitos formais em ontologia deveriam ser entendidos como objetos físicos ou lógicos e relacionamentos de acordo com um domínio de interesse [20]. Tais conceitos são prováveis substantivos (objetos) ou verbos (relacionamentos) em diversas sentenças que descrevem o domínio. Sendo assim, estudiosos, principalmente ligados à área de ciências da computação, mais especificamente em IA, salientam um glossário de definições e conceitos utilizados no meio de estudos ontológicos [21; 13] a saber:

**Domínio -** Área de abrangência ou conjunto de características que descrevem um contexto de estudo.

**Conceitualização -** Coleção de objetos, conceitos e outras entidades que se assumem existir em um domínio ou, também, uma visão abstrata e simplificada do mundo que desejamos representar para algum propósito.

**Classes ou conceitos -** Conjuntos, coleções, idéias ou tipos de dados ligadas a domínio de discursos.

**Slots, regras ou propriedades -** Particularidade ou característica que descreve recursos e atributos de um conceito.

**Frame -** Objetos ou conceitos.

**Papéis -** Relações binárias entre objetos.

**Taxonomia -** Categorização hierárquica ou classificação de entidades num mesmo domínio.

**Relações -** Representam o tipo de interação entre os conceitos de um domínio.

**Instâncias ou indivíduos -** Representam os elementos específicos das classes, ou seja, os próprios dados.

**Relacionamentos -** São as formas com que as instâncias se inter-relacionam; uma relação é uma propriedade que possui como valor outra instância da ontologia.

É interessante compreender as ligações conceituais que as ontologias possuem em relação a paradigmas orientados a objetos (OOP) e KBS's; tal relação fica explícita em citações como a que segue: "Uma ontologia, em conjunto de uma coleção de instâncias de classes, constituem uma base de conhecimento" [20].

Apesar da definição das características e dos elementos básicos, os mais diversos casos de uso de ontologias não apresentam uma estrutura comum, ou seja, as mesmas podem divergir a respeito das definições de acordo com a área/contexto de estudo. Visto isso, a fim de categorizar tais escopos e papéis, alguns estudiosos propõem definições de tipos, de acordo com seus relacionamentos, para que se definam as ontologias de forma mais sintética e explicativa — por categorias de uso — como podemos ver na tabela a seguir (Tabela 2.1), em que se compara os tipos mais comuns de ontologias.

Tabela 2.1: Comparativo entre as principais definições de tipos de ontologias por Almeida e outros [1]

| Abordagem | Classificação | Descrição |
|---|---|---|
| Quanto a função - Mizoguchi, Vanwel Kenhuysen & Ikeda (1995) | Ontologia de domínio | Reutilizáveis no domínio, fornecem vocabulário sobre conceitos, seus relacionamentos, sobre ativiades e regras que os governam. |
| | Ontolodia de tarefa | Fornecem um vocabulário sistematizado de termos, especificando tarefas que podem ou não estar no mesmo domínio. |
| | Ontologia gerias | Incluem um vocabulário relacionado a coisas, eventos, tempo, espaço, casualidade, comportamento, função, etc. |
| Quanto a grau de formalismo - Uschold & Gruninger (1996) | Ontologia altamente informais | Expressa livremente em linguagem natural. |
| | Ontologia semi-informais | Expressa em linguagem natural de forma restria e estruturada. |
| | Ontologia seiformais | Expressa em uma linguagem artificial definida formalmente. |
| | Ontologia rigirosamente formal | Os termos são definidos com semãntica formal, teoremas e provas. |
| Quanto a aplicação - Jasper & uschold (1999) | Ontologia de autoria neutra | Um aplicativo é escrito em uma única língua e depois convertido para uso em diversos sistemas, reutiliando-se as informações. |
| | Ontologia como especificação | Cria-se uma ontologia para um domínio, a qual é usada para documentação no desenvolvimanto de softwares. |
| | Ontologia de acesso comum á informação | Quando o vocabulário é inacessível, a ontologia torna a informação intelegível, proporcionando conhecimento compartilhado dos termos. |
| Quanto a estrutura - Haav & Lubi (2001) | Ontolodia de alto nível | Descrevem conceitos gerais a todos os elementos da ontologia (espaço, tempo, matéria,objeto, evento, ação, etc.) os quais são independentes do problema ou domínio. |
| | Ontologia de domínio | Descrevem o vocabulário a um domínio,como, por exemplo, medicina ou automóveis. |
| | Ontologia de terefa | Descrevem uma terefa ou atividade , como por exemplo, diagnóstico ou compras, mediante inserção de termos especializados na ontologia. |
| Quanto ao conteúdo - Van-Heijist | Ontologias terminológicas | Especificam termos que serão usados para representar o conhecimento em um domínio (por exemplo, os léxicos). |
| | Ontologia de informação | Especificam a estrutura de registros de bancos de dados (por exemplo, os esquemas de banco de dados). |
| | Ontologia de modelagem do conhecimento | Especificam conceitualizações do conhecimento, têm uma estrutura interna semanticamente rica, e são refinadas para uso no domínio do conhecimento que descrevem. |
| | Ontologia de aplicação | Contêm as definições necessárias para modelar o conhecimento em uma aplicação. |
| | Ontologia de domínio | Expressam conceitualizações que são específicas para um determinado domínio do conhecimento. |
| | Ontologias genéricas | Similares às ontologias de domínio, mas os conceitos que as definem são consideradas genéricas e comuns a vários campos. |
| | Ontologia de representação | Explicam as conceitualizações que estão por trás dos formalismos de represntação do conhecimento. |

### 2.1.2 Construindo uma Ontologia

A construção de uma nova ontologia, ou representação formal, genérica e compartilhada de certa área de conhecimento, como neste caso — jogos digitais, pode-se compreender de diversas maneiras como segue: compartilhar o entendimento comum da estrutura da informação entre pessoas ou agentes de software, permitir a reutilização do domínio de conhecimento, fazer suposições explícitas do domínio, separar conhecimento de domínio do conhecimento operacional, analisar o conhecimento de domínio, compartilhar conhecimento e estruturas de informação entre pessoas e agentes de software, explicitar hipóteses ou ainda a reutilização do conhecimento de um domínio.

Não há uma maneira correta ou padronizada de se desenvolver uma ontologia, porém, na aplicação de um processo desburocratizado e acessível de desenvolvimento, aconselha-se o uso a partir da prática de sistemas passo-a-passo.

A seguir, enumeram-se alguns passos, de acordo com definições formais existentes [20; 8], necessários à compreensão de pesquisas para a criação de novas ontologias, como segue:

1. **Determinar o domínio e o alcance -** Esta etapa ajuda a obter uma visão clara da cobertura da ontologia e a definição prevista. Nesse caso, há diversas perguntas sobre a informação existente na definição que fornecem respostas e orientações quanto à manutenção da mesma. Em pesquisas que envolvam contextos de engenharia ontológica, utilizam-se questões de competência, mais conhecidas como *meta-questions*, ou seja, são questionamentos que guiam o desenvolvimento de uma ontologia. A respeito disso, temos como exemplo as seguintes perguntas: "Qual é a característica?", "Como a característica é herdada?", "Qual é a taxonomia adequada?", "Quais são definições de classe ou instância?", "O que é um papel?", "O que leva esta especificação a não ser apenas uma coleção de coisas?", "Quais os relacionamentos existentes?", "Qual a diferença entre um objeto e um processo?", dentre outras indagações.

2. **Considerar a reutilização de ontologias existentes -** Reutilizar ontologias existentes é uma boa prática, pois proporciona uma evolução de conhecimentos e/ou cogita a aplicabilidade dos conhecimentos lançando mão de outro ponto de vista

3. **Enumerar os termos importantes -** A terminologia, na formalização de nomes e processos, muitas vezes resulta na criação de uma taxonomia apropriada. Inicialmente é definida para dar continuidade (com o uso de uma sintaxe compartilhada) a construção das especificações de determinado conceito.

4. **Definir classes e hierarquias -** Definir e identificar, nesta etapa, os conceitos gerais e as classes — camadas e hierarquias dos conceitos relacionados.

5. **Definir propriedades (*slots*), atributos e relações -** Descrever, de forma explícita (a estrutura interna dos conceitos intrínsecos e extrínsecos) as partes e os relacionamentos com outras classes ou instâncias.

6. **Definir aspectos e restrições dos *slots* -** São definições ligadas ao tipo de restrição, de valores dos *slots* e suas cardinalidades.

7. **Criando instâncias -** Incluir o preenchimento dos valores dos *slots* para cada instância criada nos passos anteriores. Observa-se que, ao se construir ou se investigar uma ontologia, deve-se tomar a precaução de não definir casos específicos de instâncias, pois desta forma estar-se-ia limitando o campo de ação da mesma.

8. **Separar conhecimento de um domínio do conhecimento operacional -** Focar o estudo, restringir o público alvo e criar objetivos claros são passos importantes para o desenvolvimento de um conhecimento sólido e preciso. A ontologia não deve conter todas as possibilidades de informação sobre o domínio.

9. **Análise -** Aplicar o uso da ontologia em modelos práticos e analisar se o conhecimento obtido é satisfatório ao ser relacionado com o resultado esperado.

Os passos descritos anteriormente podem ser considerados em três principais etapas. Primeiramente a *especificação*, responsável por identificar terminologias, objetivos primários, propósitos, nível de granulação e escopo. Seguido da *conceitualização*, na qual deve-se organizar e estruturar, numa maneira semi-formal, o conhecimento adquirido durante a fase de anterior, utilizando um conjunto de representações que compreenda tanto especialistas no conhecimento de domínio quanto leigos. Por fim, têm-se a fase de *implementação*, sendo o momento de aplicar (via o uso de um ambiente de desenvolvimento que pode envolver ferramentas gráficas e linguagens de descrição) a ontologia à uma representação formal, bem como a implementação dos produtos das fases anteriores: conceitos, hierarquias, relacionamentos, modelos.

### 2.1.3 Ontologias na prática

Com a finalidade de se obter um ciclo de desenvolvimento completo e facilitar o desenvolvimento de ontologias, pesquisadores utilizam linguagens de representação e ambientes gráficos para o desenvolvimento [8].

De acordo com as diversas (Tabela 2.2) linguagens de representação computacional de ontologias, destaca-se a linguagem *Resource Description Framework* (RDF) [22] como sendo o componente base para o desenvolvimento de outras linguagens. Com isso, sabemos que todo o conhecimento desenvolvido através dos processos de criação de uma ontologia, serão armazenados e descritos por meio de linguagens e formatos compreensíveis a futuros pesquisadores e curiosos.

Para ajudar a desenvolver, organizar e obter uma representação visual dos conceitos, como estrutura, especializações, propriedades, restrições, relacionamentos, ligados às ontologias, evidenciam-se diversos editores gráficos, dentre os quais salienta-se o software *Protégé* [23] (Figura 2.1) como a principal ferramenta visual para fins de engenharia ontológica.

Protégé é uma plataforma livre e de código aberto, com suporte a diversas linguagens de representação que fornece uma comunidade de usuários em crescimento, com um conjunto de ferramentas para a construção de modelos de domínio e conhecimento de aplicações baseadas em ontologias. Esta tem por objetivo ajudar e facilitar a definição de classes, propriedades, taxonomias, restrições e também os dados (instâncias) referentes à base de conhecimento. Com o uso de tal plataforma, existem duas principais maneiras de se modelar uma ontologia: uma delas é via o editor Protégé-Frames [24] e a outra é Protégé-OWL [25].

Tabela 2.2: Tabela com os nomes das linguagens de representação ontológica mais conhecidas e um breve resumo a respeito das mesmas.

| Nome | Descrição |
|---|---|
| KIF | Baseada numa lógica de primeira ordem |
| Ontolingua | Baseada na KIF, possui uma representação baseada em frames |
| Loom | Baseada em lógicas de descrição |
| SHOE | Construída como uma extensão do HTML |
| XOL | Pequeno subconjunto de primitivas do OKBC protocolo chamado OKBC-Lite descritos via XML |
| RDF | Linguagem baseada numa rede semântica para descrever recursos Web |
| RDF Schema | Extensão da RDF incluindo uma representação baseada em frames |
| OIL | Baseado em lógicas descritivas incluindo representação de primitivas baseada em frames |
| DAML+OIL | Última versão da linguagem DAML (DARPA Agent Markup Language) |
| OWL | Linguagem de consulta padrão que possibilita a centralização de coleções de dados RDF descentralizados |

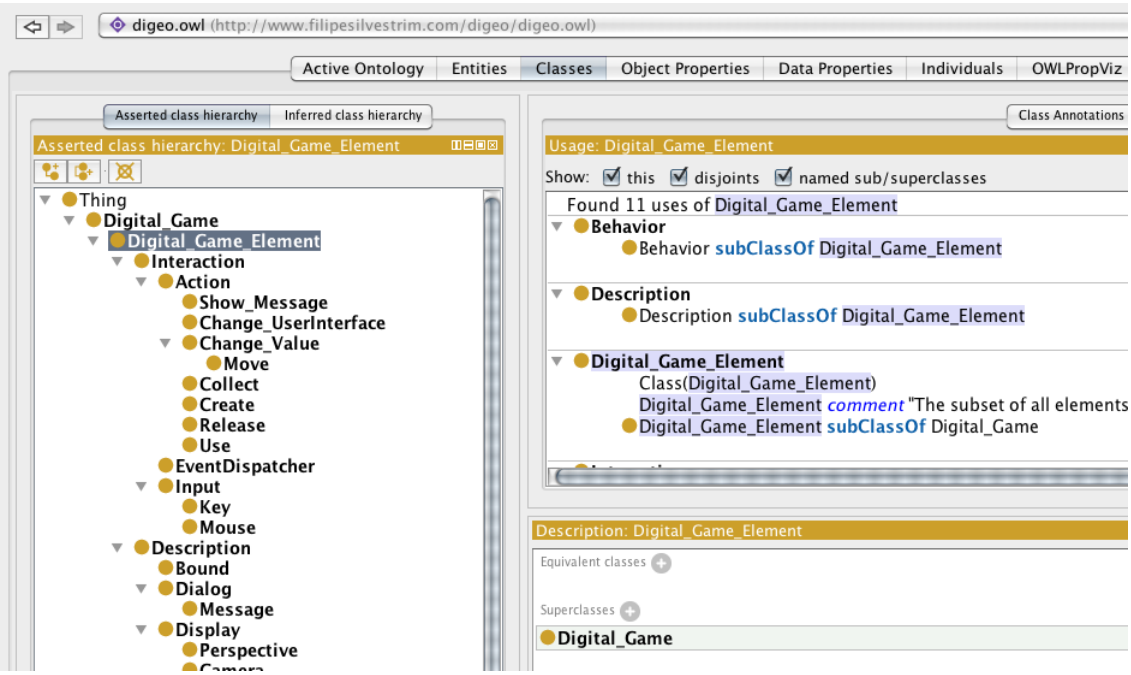


Figura 2.1: Captura de tela do software Protégé com as especificações de classe da ontologia proposta pelo trabalho.

## 2.2 Arquétipos Vs. Modelos

Arquétipos, na filosofia segundo Aurélio [26], define-se por ser um "modelo ideal, inteligível, do qual se copiou toda coisa sensível: para Platão, a idéia do bem é o arquétipo de todas as coisas boas da natureza". A partir desta definição, entende-se o significado de arquétipo como um modelo criado para simbolizar um protótipo ideal a algo. Analisando o estado da arte de um arquétipo, Beale [27] transcreve arquétipos para o escopo de engenharia do conhecimento, como sendo pedaços de conhecimento que indicam como representar conceitos ou informação.

Segundo o modelo proposto por Beale, o uso do termo arquétipo é utilizado para denominar um modelo de níveis de conhecimento, o qual define estruturas de informações válidas para os mais diversos propósitos, ou seja, para permitir que usuários possam expressar formalmente seus conceitos diante de um domínio específico; permitir que sistemas da informação guiem e validem o *input*[1] do usuário durante a criação e modificação da informação em tempo real, garantindo que todas as instâncias da informação obedeçam aos requisitos do domínio; garantir a interoperabilidade, em nível de conhecimento, não somente no que se refere à estrutura de dados e, finalmente, para provir uma base bem definida de consulta eficiente de dados complexos.

Por sua vez, modelos também conhecidos por *templates*, são definidos como uma especificação direta e localizada de arquétipos — dentro de um contexto de ampla abrangência. Os modelos são utilizados para fins de construção e validação de dados sob assuntos, enquanto os arquétipos são geralmente utilizados como exemplos genéricos, provenientes da composição em diversas possibilidade de combinação de dados. Assim, modelos são utilizados para restringir as escolhas dos arquétipos para fins locais ou específicos, enquanto arquétipos suportam a definição de reutilizar dados contidos em certos domínios. Por fim, entende-se que modelos não afetam a estrutura de arquétipos, uma vez que as informações contidas em um arquétipo servem de referência para um modelo — que tem sua informação dependente da linguagem descritiva do arquétipo.

Impondo-se um olhar analítico quanto ao uso de modelos e arquétipos, sugere-se uma novo metodologia (mais direto e menos burocrático) a respeito do uso de *templates* como sendo dados independentes de uma linguagem de arquétipo. Dessa forma, os mesmos serviriam ao propósito de explicitar de forma restritiva os dados de um conjunto de informação, como no nosso caso uma ontologia. Transcrevendo todo esse ideal de arquétipos e modelos a um mundo digital, contemplando-se valores como metodologia de desenvolvimento, sugere-se o uso de modelos num escopo menos abstrato, ou seja, sugere-se utilizar o conceito de *template*, proposto por Beale, cortando-se a conexão entre os mesmos e arquétipos, proporcionando assim, através do uso dessa teoria modificada, uma categorização entre elementos e instâncias de uma ontologia (verificar seção 5.1).

Sendo assim, a modificação proposta se comportaria semelhante a um nicho ecológico, pois um *template* teria o papel de representar (restritivamente) o conjunto de ações que as instâncias desempenhariam, bem como as relações entre as classes. Numa comparação clássica, o habitat representa o "endereço" da espécie e o nicho ecológico eqüivale à "profissão". Espelhando-se tal comparação para o ambiente de

[1]Entrada de dados via interação do usuário. Digitar no teclado ou mexer no mouse é um tipo de *input*.

jogos digitais, teríamos o nicho representado pelos categorias de um jogos digital (ver seções 5.1.2 e 5.1.3).

## 2.3 Ontologias Presentes no Meio Digital

Convenções de nomenclaturas e definições ligadas à cultura digital podem ser interpretadas [28] como formas convencionais de ontologias. Para uma descrição e/ou formalização de conhecimentos, necessita-se uma compreensão profunda entre entidades e relações num determinado escopo. Há pesquisadores, como Gelatti [29] e Zagal [30], que idealizaram suas próprias ontologias com o intuito de especificar jogos digitais.

O primeiro pesquisador destaca a importância de uma padronização entre *game engines*[2] via a criação de uma nomenclatura e definições, iniciando a proposta de uma futura ontologia que permita a reutilização de conhecimentos de desenvolvimento de jogos sobre o domínio de motores de jogos a fim de evitar a "reinvenção da roda" e possibilitando a introdução de padrões que permitam a interoperabilidade entre aplicações. Com uma postura oposta a Gelatti, o outro pesquisador propõem uma linguagem ontológica para a análise de jogos. Como resultado, tem-se a criação do *Game Ontology Project* (GOP), sendo criada uma *framework*[3] para descrição, análise e estudo de jogos. O estudo é obtido através da definição de uma hierarquia de conceitos abstratos via análise de diversos títulos de jogos digitais.

Devido ao fato de ontologias para jogos digitais terem certo grau elevado de abstração a ponto de ser difícil a definição formal de elementos, constatou-se [28; 31] um interesse elevado a proposições de ontologias informais, de maneira a serem intituladas de ontologias "estranhas" ou "vagas". Para Yun e outros [31], cujos estudos estão ligados ao escopo de jogos educacionais, os conceitos e as relações entre os mesmos, numa ontologia tradicional, são nítidas.

Há, no entanto, muitas incertezas no mundo real e, para sanar esta falta de informações, pesquisadores evidenciam um modelo de descrição ontológico chamado *fuzzy ontology*, ou ontologia vaga, que tem por objetivo atuar como uma ponte entre as definições modeladas pela ontologia e o mundo real. Tal pensamento é de grande utilidade quando se necessita uma abordagem mais prática e solidificada. Na vida real, necessita-se encurtar a distância entre as definições formais de uma ontologia de jogos digitais e sua aplicabilidade prática. Para isso, o estudo dos conceitos de ontologias vagas faz-se eficiente.

Por fim, a importância da criação de um modelo de descrição de elementos de jogos digitais, já fora evidenciada por Jupiter e Wilson [32] no desenvolvimento da proposta "Digital Game Ontology", baseada no projeto Music Ontology [33], que sugere uma ontologia para multimídia, e utilizando conceitos do projeto GOP. O modelo sugerido aborda um modelo de conceitos e as suas relações, porque descreve um jogo digital como uma atividade — a ação de jogar, ao invés de descrevê-lo similar a um monobloco ou a uma peça simplesmente etiquetada por ser uma mídia digital. Então, além de elementos externos a um jogo digital como desenvolvedores, publishers, gêneros, tal ontologia possui como essência o comportamento de como

[2]Motores de jogos. Uma ou mais bibliotecas, de códigos em linguagens de programação, que visam suas aplicabilidades proporcionando ambientes mais propícios ao desenvolvimento de jogos.

[3]Uma *framework* é uma estrutura básica e conceitual utilizada para resolver ou endereçar a solução de problemas complexos.

um usuário interage com um jogo digital. Ainda sob um olhar informal, José Luis González também idealiza [34] uma ontologia com esse foco, chamando-a de "Ontologia de Videojuegos" em que se categoriza e se descreve os elementos presentes em um jogo digital sob o ponto de vista de usuário. Assim, a ontologia de José Luis González é representada como a descrição de um jogo digital sem embasamento teórico ou analítico.

Dentre as diversas propostas analisadas, destaca-se a continuidade de forma online e colaborativa do projeto GOP. Muitos estudos de ontologias vagas mostram-se cada vez mais presentes em publicações importantes como ACM [35] ou IEEE [36]. Concluindo a revisão e estado atual destes estudos, nota-se a descontinuidade [37] e o baixo avanço [38] no projeto Digital Game Ontology.

# 3 O DESENVOLVIMENTO DE JOGOS DIGITAIS

Os Jogos Digitais assinalam um impacto muito marcante na vida das pessoas, de modo a serem utilizados de diversas formas — influenciando o cotidiano de um *gamer*[1] como ferramenta de entretenimento ou como forma de aprendizagem [39], socializando pessoas e interagindo diariamente com milhões de pessoas ao redor do mundo [40]. Tais características fazem com que o interesse e a procura por cursos e livros de desenvolvimento de jogos cresçam cada vez mais [41]. Não obstante, existem muitos fatos que, muitas vezes, acabam não aparecendo na mídia, mas que abordam as grandes estatísticas de desistências [42] neste campo de estudo. Estes dados são reflexos de uma realidade de pessoas sem um prévio conhecimento das abordagens que envolvem o estudo da criação e desenvolvimento de jogos digitais.

Neste campo de ação, mesmo com a existência de diversas propostas formais e informais, de modelos para agilizar, padronizar e evoluir a maneira com que os jogos digitais são concebidos, desenvolvidos e entregues, nota-se a falta de um conhecimento teórico de forma acessível a um público não familiarizado com nesse campo de ação. Devido à carência de formas mais acessíveis e inteligiveis de conhecimentos relacionados ao estudo do desenvolvimento de jogos, sugere-se uma análise teórica sobre os principais conteúdos no escopo do desenvolvimento de jogos com um objetivo de criar uma proposta que amplie e abstraia a assimilação desse conhecimento específico a um público mais genérico.

Nesse sentido, sugere-se um modelo que aborde o entendimento formal de games de uma maneira mais genérica, não de forma a fomentar a abordagem de desenvolvimento, mas como uma porta de entrada à construção de uma especialização nesse campo de atuação. Como um exemplo de abordagem que aspira diminuir a distância entre indivíduos presentes no nicho dos jogos digitais (game designers, game developers, críticos, pesquisadores, estudantes) tem-se o modelo MDA[2] [43] (Figura 3.1), cuja proposta é criar uma abordagem formal para o entendimento de games[3] acreditando no esclarecimento e fortalecimento do processo iterativo entre tais indivíduos. Dessa forma, a abordagem tende a facilitar a decomposição de conhecimentos ligados às teorias de desenvolvimento de jogos. De forma resumida, Hunicke e outros [43] propõem a *framework* MDA como sendo uma ferramenta que efetue a tarefa de diminuir a discrepância de informações e pontos de vista entre o público alvo já mencionado. Contextualiza-se sua composição, feita por intermédio de três camadas conceituais: *mecânica*, como sendo todas ações, *comportamentos* e

[1] Usuário ou jogador de jogos digitais.

[2] Mecânica, Dinâmica e Estética.

[3] Como já dizia Raph Koster, autor do livro A Theory of Fun, "uma aprendizagem pode ser problemática dependendo de como fora nos direcionada a aprendizagem".

mecanismos de controle de um jogador a um jogo; Estética, como sendo um modo de descrição de gameplay; *dinâmica*, sendo a essência do desenvolvimento da estética.

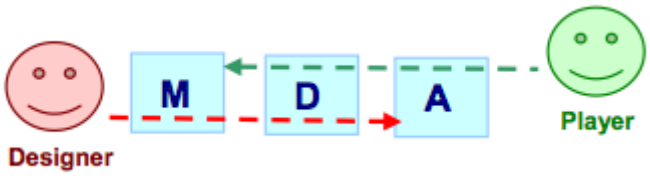


Figura 3.1: Diferentes pontos de vista sob jogos digitais por meio da representação MDA.

## 3.1 Game Design

> Não é necessário um treinamento formal para se tornar um game designer.
>
> — Raph Koester [44]

Apesar da afirmação, *game design* não é um assunto de fácil abordagem de acordo com alguns autores [45; 46; 47], pois os mesmos dizem que antes de entender o que é o *game design* é necessária a compreensão (superficial ou não) de áreas de processo criativo, de psicologia, da arte, de tecnologia e de negócios, em suma, todas as áreas relacionadas diretamente ou indiretamente com *games*. Isso é fato de acordo com o estado da arte abordado na teoria do MDA, na qual o mesmo relata, e se faz presente num contexto global também, que todos os conceitos se interligam e afetam diretamente o resultado final.

A compreensão de todos esses conceitos num contexto de jogos digitais é uma tarefa árdua obtida mediante tentativas e erros, pois não existe uma teoria ou modelo formal que represente todos esses conceitos, tanto que o que existe são propostas de diferentes escolas teóricas. Isso é uma realidade, pois entender *game design* é uma tarefa que requer um estudo sobre a história dos jogos e sobre os conceitos, ligados à área, de forma linear. Então, de forma resumida, *game design* é o estudo por trás do que o game será [48], ou seja, são as decisões e idéias de uma concepção, criação e desenvolvimento de um jogo. Numa visão analítica, o campo de game design envolve as decisões de regras, *look and feel* (estética), temporais, riscos, tarefas, recompensas, penalidades e tudo mais que envolva a experiência do jogador é responsabilidade do *game designer*.

Alguns autores como Andrew Rollings e Ernest Adams [49] ainda definem *game design* como sendo o processo de criação de um *game* a partir do zero, ou seja, imaginar como o jogo será, definir como ele funcionará, descrever os elementos (conceituais, funcionais, artísticos, entre outros) que farão parte do jogo e transmitir essa informação para o time que construirá o jogo. Porém, nota-se, após certas interações com o meio do desenvolvimento de jogos digitais, que *game design*[4] não é só um processo, mas sim uma área de estudo que explica de forma conceitual como um jogo é/será e todas as teorias e conceitos que os envolvem.

[4]Profissional da área do *game design*.

Ao analisar todos os aspectos que envolvem a criação de um jogo, bem como o papel de um *game designer* fica a pergunta: Mas o que caracteriza um jogo digital? Na tentativa de esclarecer, a possível resposta à pergunta, a característica principal de um jogo digital é ser um sistema formal e fechado que engaja (através de conflitos e resultados desiguais) jogadores [50]. Visto isso, tem-se que um sistema formal de jogo digital é composto por elementos formais, integrados através de um sistema dinâmico de regras.

> Um jogo sem jogadores, sem um objetivo, sem regras ou procedimentos não é um jogo de forma alguma. Jogadores, objetivos, procedimentos, regras, recursos, conflitos, limites e resultados, isto é, a essência do que os jogos são e uma forte compreensão de suas inter-relações em potencial é a base do game design.
>
> — Tracy Fullerton [50]

### 3.1.1 Classificações

Quando se fala de classificações de jogos, a primeira palavra do meio a ser comentada é *gênero*, pois este conceito tem por objetivo classificar video games quanto ao seu *gameplay* (jogabilidade), tais como, recursos, características, objetivos, visualização, perspectiva do jogador, estratégias de jogabilidade. Deve-se, entretanto, tomar muito cuidado com a aplicação de tal classificação, pois de acordo com Mark Wolf, "a idéia de gêneros não existe sem dificuldades, bem como a definição de que exatamente é constituído um gênero, sobrepõe-se entre gêneros e o fato de que gêneros estão sempre no mesmo fluxo que o de novas palavras sendo pronunciadas" [51]. Nesse sentido, percebe-se que a definição de gêneros é muito abstrata e confusa.

Enfatizando a análise de Thomas Apperley [52], ele argumenta que gênero é uma categoria de que necessita ser repensado sob uma perspectiva mais crítica, pois se nota a falta de informações relacionadas a essa categoria bem como a abordagem sobre categorizações. Tal situação não se refere somente a video-games, mas também a filmes, dentre outras categorias visto que não há uma definição formal e aplicada sobre o que é gênero ou o que tal segmento se relaciona.

Outra abordagem muito comum ao meio é a classificação de jogos em estilos [53]; classificação que, muitas vezes, se confunde com a definição de gêneros. Por outro lado, diferente de gêneros são os estilos que se classificam quanto às características de um jogo, quanto a elementos visuais, quanto às diferenças de narrativas e conteúdos presentes no jogo. Porém, dependendo da classificação proposta, a taxonomia relacionada pode ter significados antagônicos.

## 3.2 Desenvolvimento

Dentre os diversos modelos [54] de desenvolvimento de softwares, quando se espelha o uso em jogos, dá-se ênfase na fase de design (Figura 3.2), visto ser a fase responsável pelo detalhamento teórico e comportamental do produto final: o *game*. É nessa etapa, em jogos nomeados de *game design*, que os elementos e características de um jogo são descritas [55]. Para uma assimilação com o mundo real, o autor sugere uma abordagem intitulada "Vaso de Barro", em que propõe a obtenção do produto final dado um ciclo de produção criativo de uma jazida de matéria prima

(poço de idéias) em que é retirado um pedaço amorfo, passando por diversas etapas[5] até tomar sua forma final.

Figura 3.2: Etapas presentes num modelo de desenvolvimento jogos.

Nota-se a dificuldade na compreensão sobre a diferença do *game developer* (desenvolvedor de jogos) ao *game designer* num contexto de desenvolvimento. No entanto, com base em estudos [56] formais, pode-se afirmar que *game developer* é qualquer pessoa que tem envolvimento com a construção do jogo como um todo, podendo ser os engenheiros de software, animadores ou modeladores gráficos, músicos, desenvolvedores, produtores, escritores; já *game designer* é apenas uma especialização de game developer. Conceitualmente, a etapa de desenvolvimento de um jogo possui como *front-end*[6] o *game design*, pois, desde o momento cuja idéia é concebida até a fase de pré-produção, o mesmo atua como a principal peça do desenvolvimento, tanto que nesse processo são construídos os documentos[7] de Concept Doc, Proposal Doc e Design Doc como formalização da idéia do jogo de forma geral.

Enfim, desenvolver um jogo é um longo percurso, pois, mesmo as etapas cruciais do desenvolvimento como especificação, codificação e testes, tende ou a desprender visto que leva muito tempo, ou a envolver muitas pessoas além de comprometer muito dinheiro a fim de colocar um produto competitivo no mercado. Conforme profissionais do meio, a parte ilusória do desenvolvimento de jogos digitais é a burocracia técnica e teórica na criação de jogos desde protótipos e web-games até AAA[8], atualmente, visto que o ideal seria começar o desenvolvimento de um jogo a partir de rabiscos e não por intermédio de etapas presentes no meio profissional.

[5]Processos de modelagem (do pensamento e das idéias), refinamentos, personificação e refinamento.

[6]Etapa inicial de um dado processo.

[7]Documentos que servem para apresentar formalmente o jogo para produtores, *publishers* (editor/empresário que compra a idéia e banca o jogo), equipe de desenvolvimento, respectivamente.

[8]Jogos de alta qualidade, com alto orçamento.

# 4 TRABALHOS E FERRAMENTAS RELACIONADOS

Previamente a uma abordagem formal, faz-se necessária a compreensão da ontologia, neste momento, devido a uma visão mais comparativa diante de propostas semelhantes que este trabalho propõe, ou seja, a *DIGEO*, bem como sua aplicação a um modelo prático, nomeado de *MuGE*. Com interesse de tornar o conhecimento ligado à área de desenvolvimento de jogos acessíveis e padronizados atendendo o reconhecimento sob a composição de títulos de jogos digitais como instância, criou-se a ontologia *Digital Games Elements Ontology* (DIGEO). Conforme especificação de um sistema baseado em conhecimento, propõe-se uma implementação prática ao desenvolvimento de um motor de jogos intitulado *Multi-genre Game Engine* (MuGE).

Percebe-se, claramente, a necessidade de pesquisas sobre possíveis trabalhos e ferramentas relacionados, com a finalidade de pesquisar e absorver o conhecimento já retido nesse campo de atuação, bem como analisar de forma crítica todos os prós e contras sob o ponto de vista relativo à implementação da ontologia proposta nesta pesquisa.

Com a intenção de se fundamentar a criação da ontologia, bem como o modelo de desenvolvimento, evidencia-se a necessidade do estudo de abordagens teóricas e práticas que encerrem certo nível de similaridade com o resultado esperado na DIGEO ou no MuGE. Para tanto, será utilizado um processo iterativo de pesquisa, reconhecimento de escopo e análise conforme representado no gráfico abaixo (Figura 4.1).

Figura 4.1: Diagrama metodológico do processo de desenvolvimento do proposto trabalho.

## 4.1 Trabalhos Relacionados

### 4.1.1 SDK Gameplay - Ferramenta Voltada para edição de Gameplay

Contreras [57] criou um conjunto de ferramentas que facilitam a implementação de qualquer tipo de *gameplay* em jogos digitais. Dentre estas ferramentas, destaca-se

a biblioteca GameplayLib[1] e um editor (Figura 4.2) — simulador visual de *gameplay*.

No contexto, destaca-se a implementação da linguagem *Gameplay Language* (GML), uma linguagem de marcação baseada em XML que se insere nas idéias resgatadas do artigo "Um Modelo de Interações para Definição de Regras de Jogos" [58]. A linguagem oferece uma sintaxe baseada em termos formais de *gameplay*: Elements, Objects, Classes, State Game, Generators, Triggers, Behaviors, Actions e Perceptions.

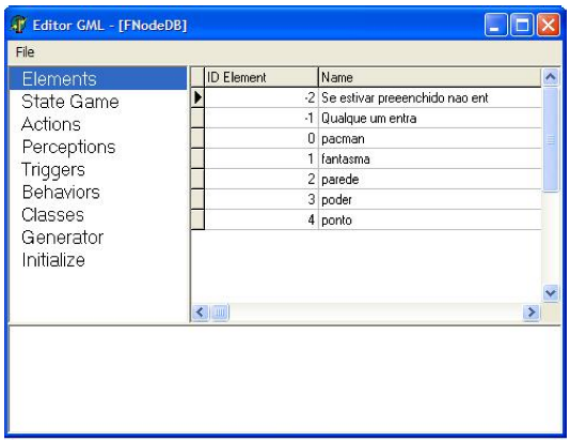


Figura 4.2: Editor de desenvolvimento de gameplay.

A DIGEO possui semelhança, de forma teórica, com o *SDK Gameplay* porque o mesmo foi projetado a fim de possibilitar a pessoas com menos conhecimento tecnológico para que possam também usufruir das ferramentas oferecidas pelo SDK. O desenvolvimento rápido de protótipos de jogos permite que decisões de continuação, alteração ou descontinuação de um projeto possam ser tomadas antes do mesmo ser finalizado.

A essência de tal projeto, além de ser específica, diverge da proposta da DIGEO, pois, mesmo que o projeto possibilite a pessoas que possuam um conhecimento de jogos e de suas interatividades que não dominam o conhecimento tecnológico a criação de regras, as mesmas teriam que ter um conhecimento básico de regras e conceitos formais ao *gameplay*.

### 4.1.2 Scald - Um Motor Genérico de Role-Playing Games Computadorizados Configurável por Scripts

O projeto *Scald*, segundo Costi [59], focou-se no desenvolvimento de um motor de jogos para a criação de outros jogos no estilo CRPGs para desenvolvedores, possibilitando, assim, a diversificação das regras utilizadas pelo sistema, produzindo também a criação que CRPGs de diversos estilos. Esta diversificação é possível por meio do uso de arquivos externos codificados via *linguagens de script*[2], as quais tornam a codificação mais simples e eficiente — possibilitando tanto que o motor quanto que o título sejam independentes de plataforma.

Costi afirma que "a complexidade na criação de um jogo está ligada diretamente às regras nas quais ele é baseado, não importando o seu gênero". A partir de tal

[1]Um conjunto de código em C++ que tem como intuito auxiliar na implementação de regras no formato XML.

[2]Linguagens de programação interpretadas por outras tecnologias. Também chamadas de linguagem de extensão.

inferência, o mesmo fomenta a utilização de ferramentas que auxiliem na criação de jogos, mais especificamente RPGs, salientando a idéia de que a criação de jogos se torne mais fácil, independente de plataformas e possibilitando uma quantidade maior de jogadores bem como a redução de tempo no desenvolvimento.

Com uma abordagem mais voltada aos desenvolvedores, o projeto é dividido em duas partes: a primeira é responsável pelas necessidades dos jogos, como por exemplo, desenhar na tela e a segunda se compromete em customizar as regras, de forma flexível, do RPG. Ainda assim, devido à utilização dos arquivos contendo *scripts* para a descrição de regras, o autor afirma que é necessária uma tecnologia que permita um fácil entendimento dessas descrições de características, pois tal entendimento é dado através de uma ontologia que não é abordada e seu uso é vago e obscuro. Ainda assim, numa visão mais análoga ao desenvolvimento desta pesquisa, idéias de uma linguagem de especificação através de linguagens de marcação (como no caso o XML) e uma divisão da arquitetura por meio dos módulos do motor que servirão como bases para possíveis consultas.

### 4.1.3 Arquitetura Reusável Baseada em Mecânica de Jogo para Jogos Casuais

Diferentemente dos trabalhos vistos anteriormente, Amaral [60] propõe uma *Arquitetura Reusável Baseada em Mecânica de Jogo para Jogos Casuais*[3] capaz de auxiliar o desenvolvedor a ingressar no mercado como desenvolvedor de jogos. Conforme evidencia o mesmo, inúmeras oportunidades na área são fornecidas a desenvolvedores iniciantes, devido ao aumento de investimentos e da lucratividade potencial, diferentemente das oportunidades existentes em grandes empresas: a aspiração de todo iniciante.

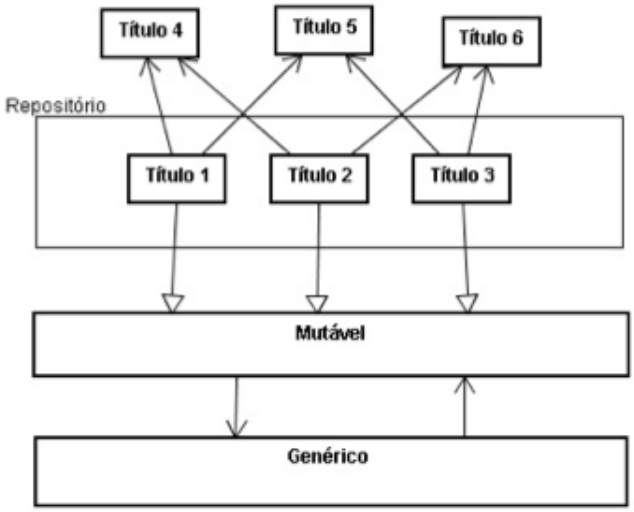


Figura 4.3: Partes da arquitetura ARMJC, destacando suas possibilidades de relacionamento.

A arquitetura proposta tem como propósito acelerar o processo de produção de jogos casuais através do encapsulamento e reaproveitamento do código fonte. O modelo é apresentado através de uma implementação por meio da linguagem de programação *Actionscript 2.0*, visando à produção de web games com modelo de

[3]Num futuro será referida com o acrônimo ARMJC.

negócio baseado em propaganda. Porém, apesar da implementação ser feita para um caso específico, o modelo teórico possibilita uma abordagem genérica suficiente para a aplicação a qualquer gênero de jogos casuais, independente da plataforma em que este fosse desenvolvido.

Através de uma abordagem aos módulos: módulo genérico, ao gênero para o qual foi produzido, e módulo mutável que contém as especializações e os relacionamentos das mesmas com a parte genérica. Nota-se (visto que a conexão entre a parte genérica e as mutáveis fora implementada) apenas a necessidade da programação das partes mutáveis, possibilitando assim o reaproveitamento completo da parte genérica ao gênero (Figura 4.3). Há interesse, nesse trabalho, em estudar e reaproveitar conceitos relacionados ao projeto de maneira a proporcionar um modelo de negócio útil junto a uma aceleração do tempo de desenvolvimento, diminuindo assim os custos de produção e resultando num modelo mais rentável.

### 4.1.4 FACHwerk: Um Framework para Desenvolvimento de Jogos Competitivos

No campo de ação de *frameworks*, Manfroi [61] criou uma baseada em conceitos de arquiteturas de motores de jogos, com um mecanismo de conexão entre jogadores para a criação de partidas multi-jogadores.

Ainda numa abordagem genérica a jogos *multi-players* (multi-jogadores), o projeto possibilita a criação de qualquer jogo contextualmente similar. Para isso, os aspectos que definem a *framework* são uma organização em áreas principais de acesso, divisão das classes em módulos lógicos, representação física dos objetos de jogo em estruturas de dados e as relações entre estes e as classes que os gerenciam.

Uma abordagem interessante, e já vista em outros projetos (como SDK Gameplay e Scald), é a representação do jogo através de XML, que possibilita a definição (*profile*) do mundo do jogo e, consequentemente, todos seus objetos: *entities*, *objects*, *resources*, .... Conforme observa o autor, "um *profile* pode definir tanto objetos numa escala pequena como um simples recurso de som com seus atributos de volume e repetição, ou também objetos de grande escala, como por exemplo a própria partida referenciando a descrição de objetos e instanciando-os no mundo como entidades".

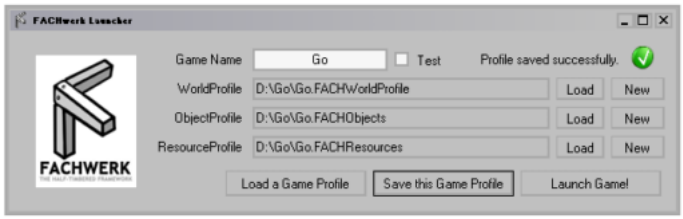


Figura 4.4: Interface do FACHwerk Launcher

O *framework* atende requisitos de ser específico para jogos *online* de competição entre jogadores, possuir cenários fechados como tabuleiros ou mapas, ser simples, possuir pequeno e robusto porte, e ser acessível para desenvolvedores na licença — documentação e extensibilidade. Dentre estes requisitos, todos ajudam na definição e criação de um jogo, porém, o recurso mais marcante para o autor da monografia, porém ironicamente de menor especificação teórica, foram os *profiles* — um ponto que servirá como base na construção da implementação do modelo prático chamado de MuGE, da DIGEO.

### 4.1.5 Open Game Engine

O *Open Game Engine* (OGE) [62] é um projeto iniciado em janeiro de 2006, com objetivo de desenvolver e distribuir uma *game engine* que inclua diversos módulos que facilitam o desenvolvimento de jogos, como física, *input*, *network*, *script*, entre outros. Propondo diminuir a dificuldade de portes para outras plataformas, uma vez que o motor é codificado utilizando a linguagem C++, o projeto permite a codificação de fases de jogo a partir de arquivos de *linguagens de script* da linguagem de marcação XML. Como o nome já menciona, uma *open engine* é baseada em sistemas e subsistemas sob licenças *open sources*[4].

A proposta sugerida nesta pesquisa monográfica, é utilizar como fonte de idéias o principal foco da OGE: construção de um motor capaz de se adaptar a múltiplos gêneros de jogos digitais (*fps*, *space shooter*, *indoor*, *outdoor*, ...). Ainda, quanto a idéias a serem aproveitadas, a OGE possui um editor (Figura 4.5) chamado *OGEd* que fornece a usuários que utilizam sistemas baseados em WYSIWYG, como a ODE [63] e a Ogre [64], um editor visual com funcionalidades de edição (de recursos e de mundo) com *scripts*, objetos, *GUI*, terreno, .... Sob uma análise mais técnica, encontram-se muitas dependências de código, pois se utiliza muitas bibliotecas externas tais como a Ogre, ODE, Bullet, RakNet, dificultando a portabilidade de seu núcleo.

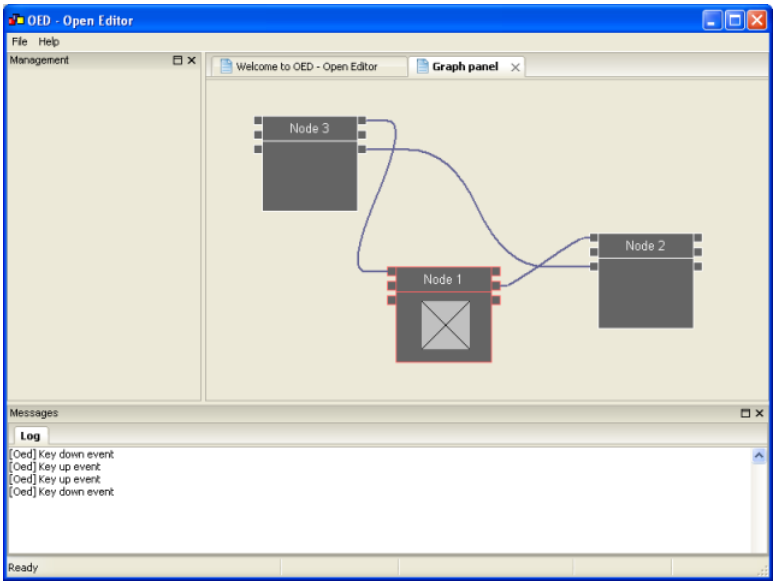


Figura 4.5: Editor WYSIWYG de recursos e mundo da OGE.

### 4.1.6 Game Ontology Project

Dentre os trabalhos analisados, é chegado o momento de analisar trabalhos com foco no campo de ação de ontologias de jogos digitais. Tais pesquisas são necessárias pois, possivelmente, ajudarão na construção da DIGEO. No contexto de projetos de estudos de jogos, destaca-se o projeto *Game Ontology Project* [65] (GOP) que propõe uma *framework* para descrição, análise e estudo de jogos. Isso acontece por meio da definição de conceitos abstratos numa hierarquia[5] oriunda de uma análise de vários

[4]De código aberto a comunidade.
[5]Lembrando que tal conceito é a essência da definição de uma Ontologia.

jogos em específico. Como alicerce da ontologia, existem definições de classes num formato mais genérico: *interface*, *rules*, *goals*, *entities*, e *entity manipulation*.

Com o intuito de proporcionar uma linguagem de projeto a *game designers*, a abordagem do GOP vem ao encontro de propiciar uma ontologia que identifique elementos estruturais importantes de jogos e suas relações, organizados de forma hierárquica. Numa visão global, este projeto se assemelha a proposta do DIGEO, não fosse pelo fato de propor apenas uma abordagem descritiva a partir de uma terminologia usual a jogadores (como *level*, chefe, inimigos). Visto isso, nota-se que neste caso, mesmo com uma boa documentação, a ontologia proposta descreve suas classes somente através de análises de títulos de jogos digitais via perspectiva de um jogador, perdendo, numa visão comparativa com a proposta do DIGEO, a fidelidade de um objetivo específico e conciso em relação a um *game* como produto final sob perspectiva de desenvolvedores de jogos. Observa-se, ainda, a carência de definições de propriedades e relacionamentos entre as classes descritas.

Apesar da análise crítica, vale uma comparação construtiva quanto à abordagem proposta pelo projeto que não se detém à descrição de regras para criação de jogos, mas encerra como objetivo primordial a identificação de características abstratas de elementos projetuais através de exemplos — análises de jogos já publicados. Destaca-se ainda a busca pela resposta "do que é o que é um jogo?" não de forma a classificá-lo conforme características e/ou mecânicas, mas sim a fim de descrever o espaço de design[6] dos jogos.

Dentre as meta-questões relacionadas à criação da ontologia são incluídas as questões: "Como podemos entender a interatividade em jogos?", "Como o *gameplay* é abordado no contexto de jogos digitais?", "Que regras existem no espaço relativo aos jogos?". Assim sendo, o propósito do GOP é categorizar, de forma ontológica, um jogo de acordo com o que vemos, ou seja, mais especificamente coisas que atuem no *gameplay* do jogo causando reações e efeitos que possam categorizar títulos no espaço da totalidade dos games.

### 4.1.7 Digital Game Ontology

O interesse nos estudos de jogos como forma de arte, marketing ou relacionamentos entre jogos digitais e a ludologia[7] levou os pesquisadores Chan e Yuen da Universidade politécnica de Hong Kong a publicarem o trabalho "Digital Game Ontology: SemanticWeb Approach on Enhancing Game Studies" [32] com esperança de representar, num formato ontológico, a maioria dos conceitos pertencentes ao evento de jogar ao invés do jogo como uma peça interativa.

Para solidificar o conceito do evento de jogar, a ontologia possui um referencial teórico e descritivo proveniente da *Music Ontology* e do GOP, com o objetivo de criar uma base para a descrição do *Digital Game Ontology* sob a forma de representação dos conceitos de game como uma peça de mídia e as atividades e experiências de usuário envolvidas. Este projeto proporciona um entendimento dos dados do jogo similar a uma peça, ou seja, inclui informações dos desenvolvedores, publisher, gênero, conceitos de desenvolvimento, processos, *gameplay* entre outros neste mesmo sentido.

A conclusão que se chega, a partir do estudo e análise do projeto, cujo propósito

[6] A palavra design aqui é utilizada de forma semelhante ao significado em *game design*

[7] do Latim, *ludus*, com o signoficado de jogos digitais. É a disciplina que estuda os jogos e o ato de jogar.

é projetar a possibilidade de uma tabela de dados relacionada à implantação de um banco de dados para pesquisa de jogos digitais é que sua aplicabilidade é bastante útil no campo de pesquisa e de mercado, possibilitando, assim, uma visão mais apurada frente a títulos já existentes. Demonstra-se grande interesse quanto ao projeto, porém evidencia-se [37] a descontinuidade e falta de suporte relacionado ao mesmo.

### 4.1.8 Ontologia de Videojuegos

Aspirando à construção de uma ontologia de jogos digitais que caracterize e gere conhecimentos relacionados aos elementos que formam os games, José Luis González propõe uma *Ontologia de Videojuegos* [34] que se baseia na interação do usuário Vs. a máquina num espaço relacionado aos jogos digitais.

A partir de observações e pesquisas, encontra-se apenas uma documentação da ontologia num formato OWL/HTML [66] sem nenhum exemplo de uso relacionado ao foco central nem tampouco existe alguma publicação formal, fato que se fazia presente nas duas ontologias apresentadas anteriormente. A par disso, e analisando a estrutura da ontologia, entende-se que o desenvolvimento da mesma fora resultado de conclusões pessoais do autor sem nenhum embasamento teórico-formal explícito. Entende-se a ontologia proposta por González quando é analisada a árvore ontológica da mesma — a partir de seus três elementos principais: "Game-Core", "Game-Engine" e "Game-Interface", numa abordagem taxonômica voltada a um objetivo de desenvolvimento, momento em que o objetivo do projeto é a análise da interação jogador Vs. *game*.

## 4.2 Ferramentas Relacionadas

Para entender a abordagem no desenvolvimento de games de uma forma mais visual, citam-se ferramentas que auxiliam no desenvolvimento de uma forma mais compreensiva. É sabido que não são as melhores ferramentas de criação de um jogo AAA, porém, cada uma delas, possui uma especialidade e também assume capital importância à proposta, quem dentre as principais, algumas serão abordadas a seguir. A importância de tais análises sob um ponto de vista de pessoas que não orbitam o conhecimento relacionado ao *game design* se dá na maneira em que os dados presentes a tal escopo são apresentados e manipulados de forma ao uso de uma nomenclatura mais amigável.

### 4.2.1 RPG Maker XP

O *RPG Maker XP* [67], bem como a maioria das ferramentas RAD[8], é um editor gráfico (Figura 4.6) que possibilita que um usuário leigo, sem prévio conhecimento das técnicas de desenvolvimento de jogos, possa criar jogos no estilo RPG voltado à possibilidade de se adicionar elementos visuais a mapas divididos em forma de grid (*tilesets*) e também customizando os acontecimentos do jogo mediante o uso de comandos de eventos pré-definidos. O editor suporta programação por *script* via a *Ruby Game Scripting System* (RGSS).

Apesar da restrição visual de criar apenas jogos em 2D, o poder da ferramenta é muito interessante, visto que, diante do propósito, permite a criação de jogos

[8]Desenvolvimento de aplicações de forma rápida.

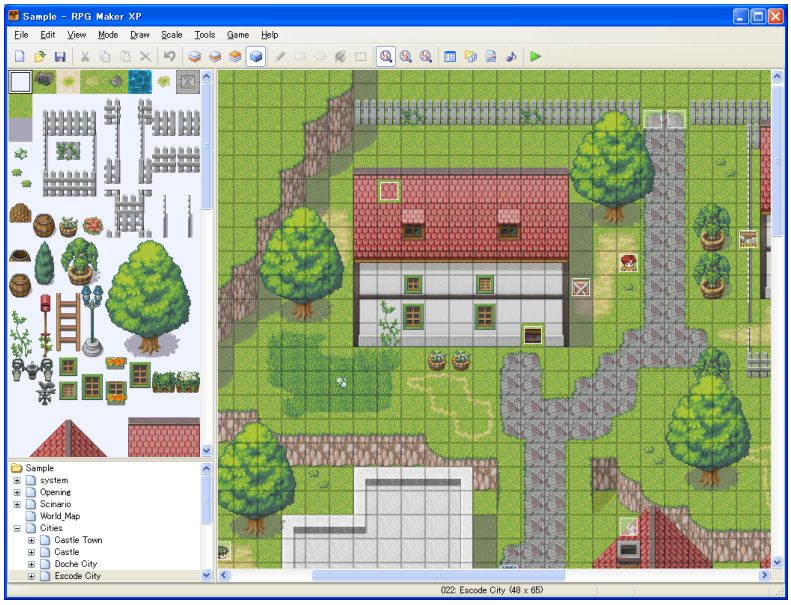


Figura 4.6: Interface visual do RPG Maker XP.

muito similares aos populares títulos dos consoles de 16-bit, tais como Zelda, Chrono Trigger, Breath of Fire e Final Fantasy, do console Super Nintendo, por exemplo.

Em uma análise comportamental, a ferramenta atende à necessidade de criar jogos a partir de uma interface visual e ainda permite a edição de comportamento, tudo isso sem perder o foco nos elementos visuais. A ferramenta usada mostrou-se muito fácil e intuitiva, porque assim é possível o desenvolvimento de protótipos jogáveis em apenas alguns minutos.

### 4.2.2 Game Maker

A ferramenta de criação de jogos *Game Maker* [68] é um SDK proprietário, embora possa ser utilizada de forma gratuita, porém com certas limitações com suporte a linguagem de *script Game Maker Language* ou via utilização de DLL's, renderização 2D por tiles isométricos e retangulares e suporte limitado a 3D. Todos os recursos utilizados para criação dos jogos são organizados em diretórios, incluindo pequenos programas (Figuras 4.7) tais como editores de imagens, sons, scripts e fases, para auxiliar o desenvolvimento. O software ainda permite a externalização dos recursos criados com o objetivo de que sejam utilizados em outros jogos ou fora do programa.

Para validar o uso e as funcionalidades da ferramenta, houve testes de uso que evidenciaram certa facilidade no desenvolvimento de protótipos básicos executados por pessoas que já possuíam um conhecimento técnico; porém, por outro lado, houve muita dificuldade aos leigos nessa questão. Como fatores negativos, evidenciaram-se limitações no desenvolvimento de forma técnica, sendo necessário o uso da linguagem de scripts para construir jogos amadores, e também teóricos, sem documentações que possibilitem a iniciantes começarem de forma fácil os estudos relacionados à ferramenta.

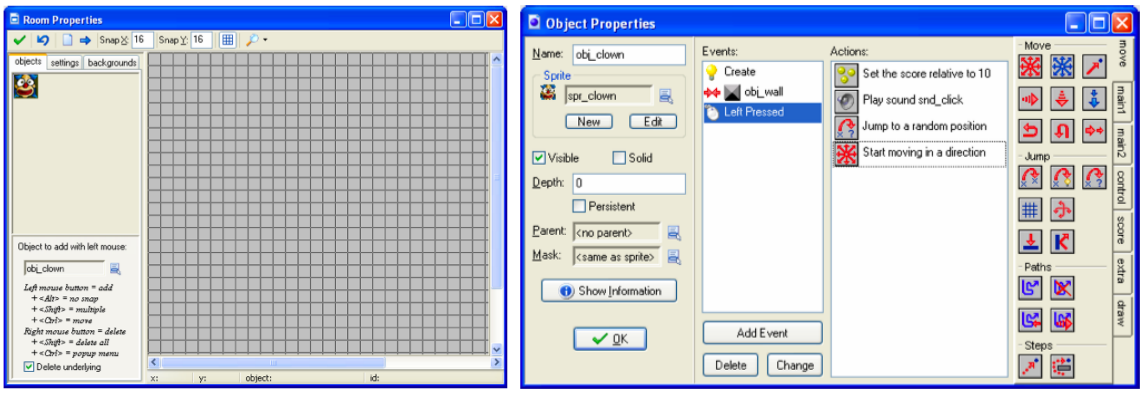


Figura 4.7: Programas parte do Game Maker. O da esquerda é o editor de tiles e o da direita é o editor de eventos e ações.

### 4.2.3 PlayCrafter

Diferentemente das ferramentas anteriormente mencionadas, porém muito similar a outras plataformas de desenvolvimento web como Popfly [69] e Sploder [70], o *PlayCrafter* [71] é um portal com uma ferramenta de desenvolvimento de jogos e publicações dos títulos criados — os quais são desenvolvidos com a tecnologia Adobe Flash [72]. Desenvolvendo jogos com visualização 2D e física, esta plataforma proporciona a criação de jogos através de uma dinâmica *drag and drop* (arrastar e soltar) de peças visuais customizáveis (Figura 4.8).

Figura 4.8: Interface visual da plataforma PlayCrafter.

Com mais de 5.000 jogos produzidos, o grande diferencial desta ferramenta está na possibilidade que ela proporciona em editar e personalizar jogos já criados e publicados por outros usuários. Até o momento, existem dezesseis diferentes tipos de jogos sustentados através de um modelo de templates, dentre os quais tem-se, *golf*, *breakout*, *shooters*, *platformers* e *puzzles*. Como forma de atrair novos usuários, existe um modelo de negócio baseado em propaganda na hora de carregar o jogo.

Como ponto central de forte interesse ao desenvolvimento do modelo teórico, envolvendo o MuGE, temos a proposta principal do PlayCrafter que é "modificar a maneira com que os jogos são criados por reduzir drasticamente os custos e a

*expertise* (conhecimento especializado) necessária para o desenvolvimento de um jogo".

Mesmo com a ajuda de mini tutoriais, em forma de passo-a-passo, a ferramenta requer certo grau de conhecimento sobre games para um uso intuitivo da mesma no desenvolvimento dos jogos. Após um reconhecimento básico sobre os objetos existentes na ferramenta, o usuário pode lançar mão do uso da técnica de desenvolvimento via *play-test*[9].

### 4.2.4 Power Game Factory

Com o objetivo de tornar mais compreensível a criação de jogos, sob um ponto de vista de *game design* o *Power Game Factory* [73] é um software (Figura 4.9) desktop para criação de *side scrolling action games* (jogos do gênero de ação com uma visão rolagem lateral) para Macintoshs (computadores Apple).

O presente projeto tem como diferencial a possibilidade de se criar jogos muito similares a títulos parecidos com os principais lançamentos dos consoles de 8-bit e 16-bit, num formato 2D, porém com gráficos e sons muito superiores, entretanto, sem a necessidade de se programar uma linha de código.

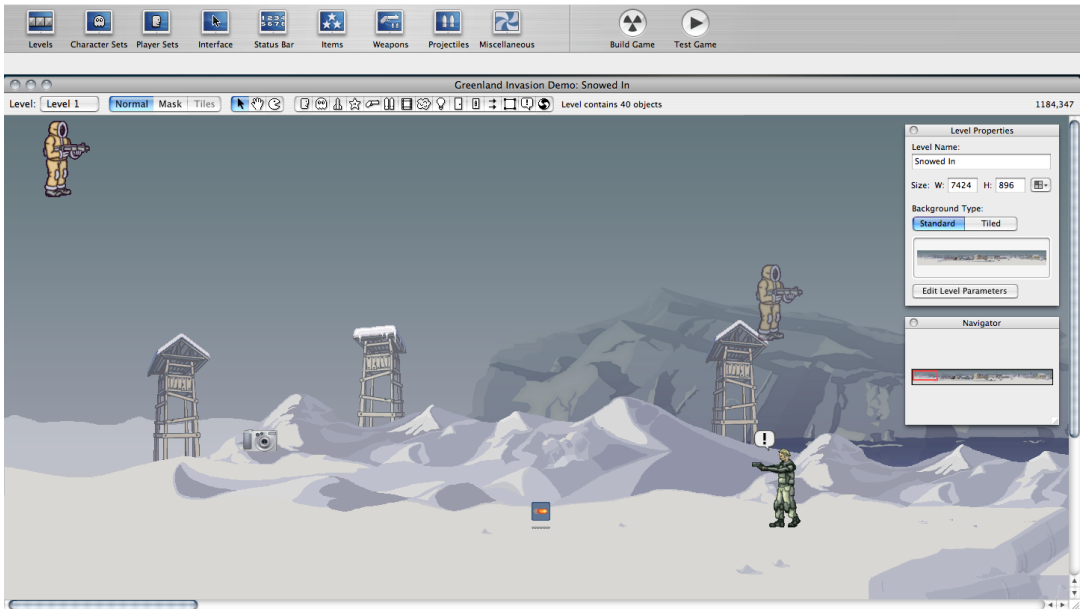


Figura 4.9: IDE da Power Game Factory acessível principalmente a cliques (interações com o mouse).

Como fonte de inspiração quanto ao objetivo de tornar o desenvolvimento de jogos acessível a maioria dos indivíduos sem conexão à área, o presente software apresenta de forma vitoriosa a aplicabilidade deste objetivo via um dos principais propósitos: a eliminação das exigências comuns ligadas à compreensão dos conhecimentos sobre a programação de jogos, visto que o desafio é superar as dificuldades técnicas de se projetar um jogo através de um foco ligado totalmente ao design e customização de conteúdos a partir de um título.

[9]Técnica que aborda o desenvolvimento através de partes que são executadas e testadas.

### 4.2.5 Unity 3D

Unity [74] é uma ferramenta para desenvolvimento de jogos e publicação multiplataforma, dependente da instalação de um *plugin*[10], desenvolvida com o propósito de ser uma ferramenta RAD, visto que possibilita a criação de jogos 3D num âmbito profissional. Atuando sob vários núcleos, a ferramenta possibilita a manipulação de animações, a implementação de física utilizando a Agéia PhysXTM, dentre outros recursos.

As vantagens dessa tecnologia são as facilidades na criação e manipulação de elementos utilizando a IDE de criação (Figura 4.10), bem como a independência de manipulação de baixo nível de tecnologias de visualização. A exemplo disso, contamos com as tecnologias de renderização OpenGL e DirectX, unidos à liberdade do uso de *scripts*, dentre eles têm-se a linguagem JavaScript, porém a baixa presença do *plugin* no mercado, somado ao fator de ser uma tecnologia relativamente nova e uma comunidade ainda pequena servem como pequeno contraponto para a análise da mesma.

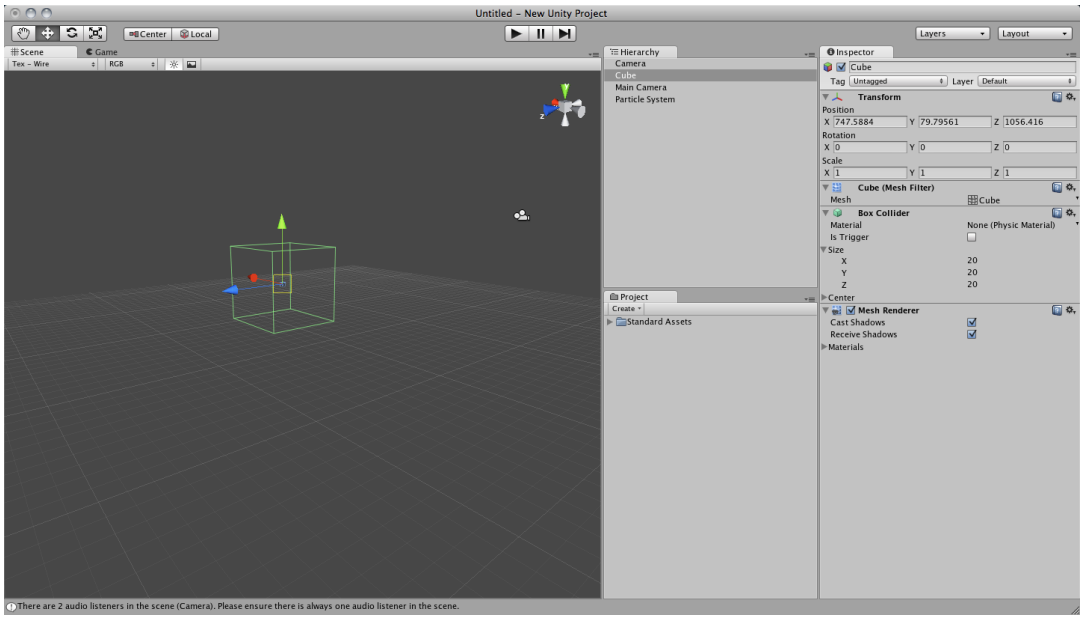


Figura 4.10: Ambiente de desenvolvimento da Unity 3D. Mistura características de modeladores 3D e de sistemas de controle de comportamentos.

Mesmo com a falta de suporte ao desenvolvimento de jogo *multiplayer* ou a falta de suporte a video, esse ambiente de desenvolvimento proporciona uma curva de aprendizagem exponencial aliado a uma IDE muito acessível e de fácil manipulação de modelos 3D a partir de ferramentas profissionais como Maya, Cinema 4D, Cheetah 3D ou qualquer programa que exporte para a extensão FBX, incluindo softwares como Lightwave 3D, Softimage e 3D Studio Max.

Com o intuito de fomentar essa idéia, o ideal é tentar produzir uma ferramenta com o propósito de modelar o estado da arte em um ambiente 3D interativo; a Unity 3D proporciona suporte avançado à parte de renderização, física, som e todos os recursos principais necessários à criação de um jogo AAA para plataforma Web, Mac, Windows, IPhone e futuramente Wii.

[10]Programa de computador que interage com uma aplicação servidora — neste caso, um browser.

### 4.2.6 Torque Game Builder

Para uma abordagem similar à Unity, a *Torque Game Builder* (TGB) [75] é um SDK para a criação de jogos, desenvolvida pela *Garage Games*. Baseada na *Torque Game Engine*, um dos primeiros motores de jogos (lançamento em 2000) a serem utilizados de forma popular dentre outros diversos estúdios de jogos, a TGB possui duas versões atuais de editores, 2D e 3D, liberadas publicamente para compra em julho de 2008 e agosto de 2009 respectivamente, afirmando-se como ferramentas poderosas no desenvolvimento de jogos através do uso de seus editores visuais, entre eles o de mundo e interface num formato 2D (um dos pontos contrastantes quanto a Unity3D que não suporta nativamente este tipo de objetos visuais), através do modelo WYSIWYG.

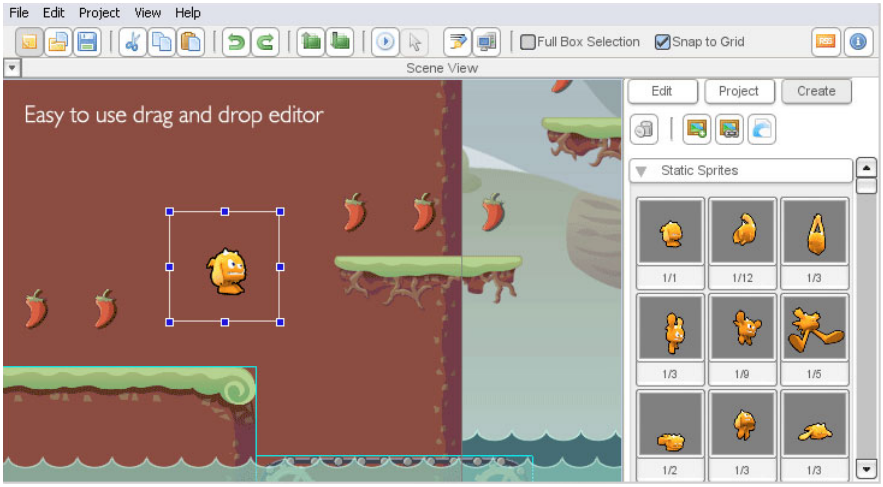


Figura 4.11: Editor drag-and-drop da Torque Game Builder, possibilitando uma manipulação de forma intuitiva.

Com uma interface visual amigável (Figura 4.11) e uma boa documentação, o kit de desenvolvimento proporciona uma excelente interface de desenvolvimento para desenvolvedores de jogos já atuantes no mercado. Com um aspecto de ferramenta profissional, o TGB necessita, para interações mais avançadas, o uso de uma linguagem própria chamada *TorqueScript* muito parecida com a sintaxe do C++.

Através das ferramentas propostas pela TGB é possível a publicação de jogos para diversas plataformas, tais como PC, Mac, Xbox 360, Wii, iPhone e na Web. Ainda, salienta-se uma proposta muito interessante a desenvolvedores avançados, visto que há a possibilidade de acesso ao código fonte dos motores por trás do SDK, dada liberdade de customizar e alterar códigos fonte.

## 4.3 Tabelas Comparativas

Numa tentativa de comparar os diversos trabalhos e ferramentas existentes, criaram-se duas tabelas (4.1 e 4.2) com colunas que evidenciam as diferenças das propostas. A análise abre espaço à discussão da importância do desenvolvimento de uma ontologia que aborde o tema *desenvolvimento dos jogos digitais*, mas com uma aplicação prática.

Tabela 4.1: Comparativo perante os objetivos dos trabalhos que visam uma padronização de definições ligadas ao desenvolvimento de jogos.

| Nome | Foco | Base Teórica | Linguagem Relacionada | Flexibilidade | Portabilidade | Documentação |
|---|---|---|---|---|---|---|
| SDK Gameplay | Descrever *gameplay* | "Um Modelo de Interações para Definição de Regras de Jogos" [58] | GML | Limita-se à padronização de regras de jogos | Dependente da implementação de um interpretador ao GML | Boa documentação mediante um website (Acessível por `http://www.drecon.com.br/sdkgameplay/`) |
| Scald | Descrever regras e interações ligadas ao gênero RPG | Não possui | XML | Permite a criação de jogos com características similares ao gênero | Dificultada quando não existe especificação formal do XML | Somente o que especificado na tese que o descreve |
| ARBMJJC | Generalizar a reutilização de código em relação a títulos do gênero Puzzle | Não possui | ActionScript 2.0 | Sua abordagem de estrutural permite uma expansão do modelo a outros gêneros | Não possui, uma vez que a tecnologia é proprietária | Somente o que especificado na tese que o descreve |
| FACHwerk | Descrever jogos multijogador | Não possui | XML | Possibilita a criação de qualquer jogo contextualmente similar | Dificultada quando não existe especificação formal do XML | Somente o que especificado na tese que o descreve |
| OGE | Criar um motor de jogos genérico | Não possui | C++ | Pode ser compilado em diversas plataformas mediante o uso de um compilador | Restrita, uma vez que seus códigos são dependentes à outros | Boa base de documentação (Visto no na wiki de url `http://sourceforge.net/apps/mediawiki/oge/index.php?title=Main_Page`) quanto a estrutura e modelo teórico da aplicação. Falta de exemplos práticos. |
| GOP | Propor uma ontologia para jogos | Não possui | OWL | Como toda e qualquer ontologia, pode ser extensível. Não deixa explícito qual sua licença de distribuição | É aplicável de qualquer forma prática | Excelente documentação vista seu website (URL `http://www.gameontology.org/index.php/Main_Page`) |
| Digital Game Ontology | Aplicar a ontologia GOP sob um olhar a jogos digitais | Não possui | Não fora constatada | Como toda e qualquer ontologia, pode ser extensível. Não deixa explícito qual sua licença de distribuição | É aplicável de qualquer forma prática | Não foram encontradas documentações atuais, somente o paper que descreve. |
| Ontologia de Videojuegos | Descrever ontologicamente jogos digitais | Não possui | OWL | Licenciada sobre JLG | É aplicável de qualquer forma prática | Somente uma definição OWL/HTML com pouco detalhamento quanto aos dados relacionados |
| DIGEO | Criar uma ontologia de jogos digitais sob a perspectiva de um desenvolvedor | *Game Design* | *DIGEO Language* (baseada em OWL/XML) | Como é um modelo teórico através da especificação de uma ontologia sobre licença MIT, a mesma pode ser extendida ou modificada para outros propósitos | Possibilita a implementação da teoria em qualquer plataforma | Documentação teórica através desta monografia e do website da ontologia (Acessível pela url `http://www.filipesilvestrim.com/digeo/`) |

Tabela 4.2: Comparativo entre as principais ferramentas que propiciam o desenvolvimento de jogos mediante um processo de fácil compreensão.

| Nome | Licença | Dependências | Restrições | Linguagem | Documentação | Ambiente de desenvolvimento | Plataformas publicadas |
|---|---|---|---|---|---|---|---|
| RPG Maker XP | Proprietária | Nenhuma | Visualização 2D; objetos limitados; linguagem de script | Ruby Game Scripting System | Juntamente ao software | Windows | Windows |
| Game Maker | Proprietária | Nenhuma | Visualização 2D; objetos limitados; linguagem de script | Game Maker Language ou C++ (mediante criação de DLL's) | Juntamente ao software | Windows | Windows |
| PlayCrafter | Proprietária, porém gratuito para uso e publicação | Flash Player | Visualização 2D; objetos limitados | Não existe | Tutoriais in-game | Web, mediante o uso da plataforma PlayCrafter | Web |
| Power Game Factory | Proprietária | Nenhuma | Visualização 2D sidescrolling | Não existe | Juntamente ao software e por meio de projetos de exemplo | Mac OS | Mac OS |
| Unity 3D | Proprietária, sendo gratuito para desenvolvedores Indie e paga para os demanais | Unity Player | Não possui suporte nativo a GUI 2D | Javascript, C# ou Boo script | Juntamente ao software; comunidade grandiosa e muito soporte na web | Windows e Mac OS | Web, Mac, Windows, IPhone e futuramente Wii |
| Torque Game Builder | Proprietária | Torque Player | Linguagem de programação própria | TGB Scripting Language | Juntamente ao software; comunidade não muito grande | Windows e Mac | PC, Mac, Xbox 360, Wii, iPhone e na Web |
| MuGE | Proprietária, porém a especificação da *DIGEO Languague* é livre meio a licença MIT | Nenhuma | Necessita da criação de um interpretador do XML | C++ e XML | Documentação presente à monografia atendendo as especificações da DIGEO e da *DIGEO Language* | Não é limitado | Dependente da implementação do Interpretador |

# 5 ANÁLISE DE JOGOS DIGITAIS

Dentre os diversos indivíduos que atuam no nicho de jogos digitais (gamers, desenvolvedores de jogos, game designers, roteiristas, designers, curiosos, jogadores imparciais), questiona-se "qual seria a receita para se descrever jogos de uma forma padrão e acessível a vários nichos?". Pensando nisso, surgira o interesse na criação de uma primeira classificação, não com intuito de ser complexa ou completa, mas com o objetivo de sincronizar os estudos nesse projeto de forma a compreender como funciona um jogo e qual é a sua composição sob o aspecto conceitual, porém livre de complexidades teóricas. Consequentemente, com o resultado da análise, teremos dados consistentes para esboçar uma classificação de títulos de jogos digitais por intermédio de análises adjuntas à taxonomias e definições populares.

Com uma abordagem estratégica, as análises são obtidas às custas de um estudo abrangendo diversos jogos, numa visão de acordo com a relevância de títulos de jogos digitais, selecionamos dezesseis títulos (Tabela 5.1) com características e relevâncias diferenciadas, isto é, as mercadológicas, as tecnológicas, as visuais e as comportamentais.

Como acontece em outras áreas de estudos, idealiza-se o estudo de jogos digitais de uma forma mais unificada de acordo com algumas propostas existentes [4; 5]. Tais unificações seja de uma forma seja de outra, agregam informações importantes para estudos mais genéricos a respeito da área, tornando o conhecimento acessível a pessoas que não atuam nesse segmento. Ao se adquirir um conhecimento dos diversos tipos de jogos digitais, seja esse conhecimento proveniente de bancos de dados voltados a análises seja de artigos acadêmicos, prossegue-se o estudo das teorias necessárias à criação da DIGEO por meio de uma categorização/classificação de jogos digitais de acordo com diversas perspectivas e olhares analíticos dos usuários.

O interesse em esboçar tal classificação serve, para que, de forma teórica, consigamos conceber a composição de um jogo digital sob uma perspectiva intuitiva à compreensão de usuários iniciantes — sem vínculos a especificações formais e técnicas da área (como *game design*, tecnologias, teoria de jogos, ...).

## 5.1 Categorização

No decorrer das análises, evidenciaram-se padrões de dados comportamentais e visuais dos quais pode-se propor uma abordagem de categorizá-los utilizando uma nomenclatura mais acessível do que gêneros ou estilos. A definição dessas categorizações é alcançada através de observações de diversos estudiosos [52; 54; 76] possibilitando, assim, a declaração de uma composição inicial a respeito de elementos de base e decorativos, estruturando também a identidade de um título como

instância. Desse modo, apoiando-se no conceito de Andrew Rollings, infere-se que "uma das maneiras mais concretas de se obter uma perspectiva concreta sobre tecnologia é conhecer a diferença entre tecnologias de base e decorativas" [49], nesse caso, os jogo digitais, como sendo uma união dos conceitos que classificam um jogo em *categoria principal* (seções 5.1.2) e *categoria secundária* (seção 5.1.3).

Como forma de validar essa classificação proposta, avaliam-se três jogos (Figura 5.1) de forma detalhada quanto características primárias e secundárias. Em uma primeira análise, somente através da imagem estática pode-se concluir que os mesmos possuem características de base semelhantes, pois estes podem ser descritos esteticamente como similares — levando em consideração o desconhecimento das classificações propostas, eles são validados como possuidores de um estilo semelhante. Para esclarecimento apenas, assinala-se que a referida análise será retomada após um compreensão completa dos significados de categorias primárias e secundárias.

Figura 5.1: Títulos com características visuais parecidas.

### 5.1.1 Elementos

Consideram-se elementos de jogos digitais aqueles objetos ou *tokens* [54] (taxonomia proposta por Rollings e Morris) supervisionados, discretamente pelo computador [53]. De forma explícita, infere-se como elementos sendo as propriedades ou características, concretas ou abstratas, comuns à maioria dos jogos.

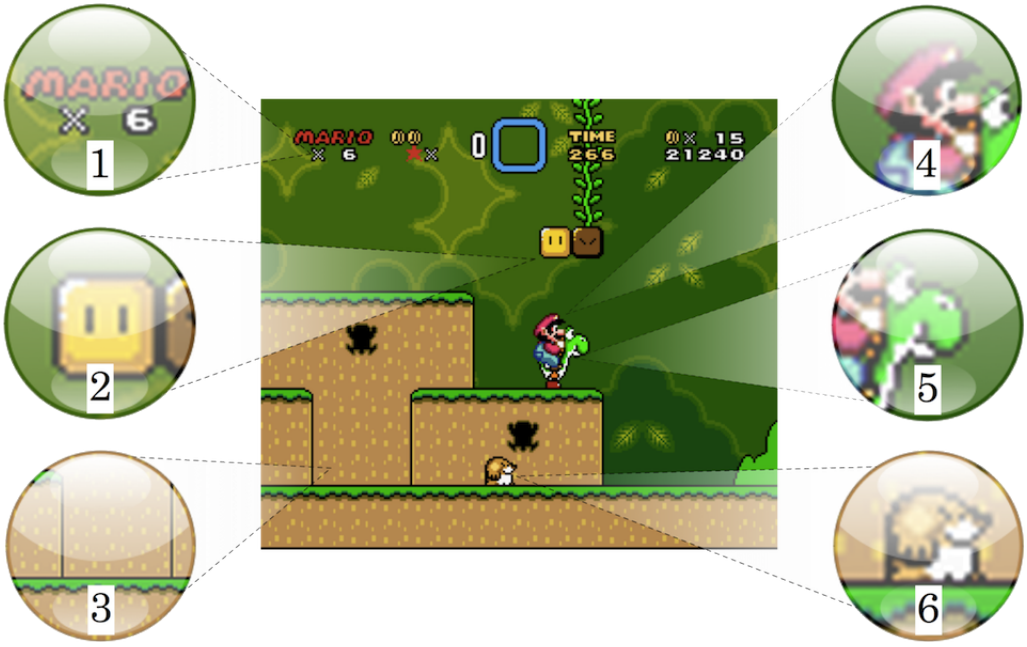


Figura 5.2: Exemplo de elementos visuais presentes no título Super Mario World.

A partir da definição formal de elementos de jogos digitais, abre-se um leque a inúmeras interpretações e implementações dos mesmos. No caso da classificação proposta neste capítulo, os elementos são componentes que influenciam na descrição e/ou interação do jogo, ou seja, são aqueles que criam/modificam a identidade de um título. Observa-se, por exemplo, na análise do jogo Super Mário World [77] (Figura 5.2), diversos elementos visuais (numerados) que modificam o *gameplay* e, com isso, a identidade do jogo. Se num exercício mental fosse instigada a modificação ou remoção de um desses elementos, teríamos uma mudança na percepção do jogo, comparando nas descrições anteriores. Ainda, a cerca da análise do título, vê-se a necessidade de uma adaptação das percepções para elementos que não sejam somente visuais. Ao ser aberto o leque de elementos a serem reconhecidos, a partir de uma percepção global e abstrata no âmago de um jogo digital, transcreve-se os elementos existentes conforme se observa na figura 5.2 para a tabela 5.3.

A partir da análise iniciada no parágrafo anterior, pode-se sustentar que o ponto de vista de uma análise (como o caso mencionado que limita-se à percepção limitada somente a elementos visuais) influencia na maneira cuja uma instância de um jogo se manifesta. Sendo assim, patenteia-se a criação de elementos limitados, de acordo com a categorização proposta, a fim de focar o estudo e a compreensão a cerca de serem obtidas percepções harmônicas a respeito de quais elementos serão analisados.

| ▼ Elements | |
|---|---|
| **Camera** | Tracked |
| **Viewpoint** | Side |
| **User Interface** | Info Bar |
| **Physics** | ☑ |
| **Input** | Keyboard |
| **Entity** | Mario, Yoshi, Enemy |
| **Static Object** | World |
| **Dynamic Object** | Block |

Figura 5.3: Tabela de alguns dos elementos, dentre eles concretos e abstratos, presentes no título Super Mário World.

Com uma compreensão clara a respeito do papel dos elementos de jogos digitais na classificação e categorização dos mesmos, discute-se o uso desses recursos parametrizáveis, a fim de criar divisões taxonômicas do uso dos mesmos com uma

classificação que lança mão de modelos de forma genérica e adaptável.

Como pode se observar nas tabelas 5.4 e 5.5 utiliza-se uma nomenclatura usual a estilos e gêneros a fim de criar classificações paramétricas e pré-definidas, ou seja, ao invés de consumir tempo demasiado na atribuição de valores (Figura 5.6) aos elementos, para assim os qualificar, emprega-se o uso de modelos com valores pré-definidos de acordo com as categorias já sugeridas. Tal abordagem serve como uma camada de abstração à classificação de jogos digitais alicerçada em elementos já definidos.

É importante deixar claro que, nesta etapa do trabalho, não haverá uma abordagem minuciosa de cada elemento pertencente a essas divisões, uma vez que os mesmos serão retomados na seção 6.6 de forma a explicar o conceito, valor e forma de uso.

### 5.1.2 Categoria Primária

Esta categoria serve para descrever as características primárias de um título, ou seja, aquelas características de base que estruturam e moldam um jogo digital. A especialização de um jogo é descrita pelo conjunto dessas características, bem como seus respectivos valores. A partir dos títulos analisado, observa-se a presença de elementos descritivos, do título como um todo, que expressam como objetivo principal o levantamento das características pormenorizadamente (sendo limitadas pelo número de elementos presentes à mesma) de um jogo sem levar em conta o seu *gameplay*. Visto isso, traçou-se uma definição que explora a quantidade mínima de elementos (Figura 5.4) necessários para definir algum jogo quanto a sua *categoria primária*: sendo expressa por intermédio do conceito de *setting* [53], sendo a definição ou plano de fundo de um jogo, representando o mundo que será explorado pelo usuário e personagens, ou seja, é a representação do mundo de jogo a qual os personagens irão existir e interagir. São aspectos que precedem precedem os estereótipos de um jogo — são as definições que fazem do título um local habitável.

### 5.1.3 Categoria Secundária

Como já mencionado na seção 5.1, tal categoria serve para descrever as características decorativas, ou também chamadas de secundárias, cujos elementos compostos nessa categoria são dependentes à categoria primária, visto que um título não pode existir sem uma categoria primária, mas sim pode abrir mão de sua categoria secundária. Como base de apoio à categoria primária, há a descrição do jogo a partir de um composto de elementos comportamentais e interativos.

Essa categoria contém elementos (Figura 5.5) cujos propósitos descrevem um título composto de objetos de *gameplay*, elementos que perturbam um sistema harmônico (jogo) pela inferência de forças de ação e reação. Como fundamento dessa categoria presencia-se a existência de elementos que estruturam a jogabilidade de um título mediante a análise de aspectos como interatividade, linearidade, controle dos personagens, colaboração e imersão.

| Main Category | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **Default** | **Adventure** | **Action** | **Sport** | **Puzzle** | **Strategy** | **RPG** | **Music** |
| ▼ Elements | | | | | | | | |
| **User** | Nothing | Entity | Entity | Entity | Cursor | Entity | Entity | Nothing |
| **Score** | Goal | Quest | Goal | Goal | Problem Solve | Challange | Quest | Hit |
| **Bound** | World | Region | World | Room | Board | Region | World | None |
| **Physics** | ☐ | ☐ | ☑ | ☑ | ☐ | ☐ | ☐ | ☐ |
| **Task** | ☐ | ☐ | ☑ | ☑ | ☑ | ☐ | ☑ | ☐ |
| **Level** | ☐ | ☐ | ☑ | ☐ | ☑ | ☐ | ☐ | ☐ |
| **Dialog** | ☐ | ☑ | ☐ | ☐ | ☐ | ☐ | ☑ | ☐ |

Figura 5.4: Divisão, entre modelos, dos valores padrões presentes na categoria primária.

| Secundary Category | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **Default** | **Platform** | **FPS** | **RTS** | **Action-Adventure** | **Racing** | **RPG** | **Artillery** |
| ▼ Elements | | | | | | | | |
| **Camera** | Fixed | Tracked | Tracked | Movable | Tracked | Tracked | Tracked | Tracked |
| **Viewpoint** | Top | Side | First Person | Isometric | Top | Top | Thrid Person | Thrid Person |
| **Time** | No | Free | No | Free | No | Limited | Turn | Turn |
| **Map** | Visible | Visible | Visible | Fog of War | Visible | Visible | Visible | Visible |
| **Main Entity** | Character | Character | Character | Character | Character | Car | Character | Character |
| ▼ Action Elements | | | | | | | | |
| **Collect** | ☐ | ☐ | ☐ | ☑ | ☑ | ☐ | ☑ | ☑ |
| **Change Value (Combat)** | ☐ | ☐ | ☐ | ☑ | ☑ | ☐ | ☑ | ☐ |
| **Show Message** | ☐ | ☑ | ☐ | ☑ | ☐ | ☑ | ☑ | ☑ |
| **Use (ie.: weapon)** | ☐ | ☑ | ☑ | ☐ | ☐ | ☐ | ☐ | ☑ |
| **Release** | ☐ | ☑ | ☑ | ☑ | ☐ | ☐ | ☑ | ☐ |

Figura 5.5: Divisão, entre modelos, dos valores padrões presentes na categoria secundária.

### 5.1.4 Singularidade

Mesmo diante de uma categorização, nota-se a existência de características divergentes à divisão sistemática de elementos e seus valores. Assim sendo, para garantir a unicidade de cada título, procura-se expandir a compreensão dos elementos. Sob um olhar mais atento às categorizações e aos elementos compostos, evidencia-se uma discrepância nos valores dos elementos; essa diversificação dos valores de elementos é o fator que irá distinguir um título a outro categorizado como tendo uma descrição similar quanto a seus modelos — valores similares no quesito categoria primária e secundária.

Inicialmente, por intermédio dos jogos analisados, obtém-se a possibilidade de atribuir valores consistentes aos elementos presentes nas duas categorias, como pode ser verificado na figura 5.6. Salienta-se, em tal momento, a possibilidade de adicionar e modificar valores uma vez que se pretenda descrever títulos com características inexistentes até o momento, para que sejam criados novos modelos ou customizações.

Entende-se por customizações a alteração de um valor existente a um elemento de um modelo específico. Como exemplo, a modificação do valor do elemento *Physics* para *Yes*, de modelo de categoria primária *Default* (Figura 5.4), possibilita-se a criação de um título diferenciado dos demais — mesmo quando os títulos são classificados como presentes ao mesmo modelo.

Nesse momento já se percebe a capacidade de distinguir os três títulos (Figura 5.1) que, sob um olhar gráfico, encerram características similares de maneira a utilizar a categorização proposta pelo trabalho. Como resultado da aplicação dos estudos abordados nesse capítulo, categorizam-se os títulos de acordo com os valores descritos na figura 5.7.

Figura 5.6: Agrupamentos dos valores possíveis aos elementos de cada categoria.

**GTA II**

| | |
|---|---|
| ▼ Characteristcs | |
| **Main Category** | Action |
| **Secondary Category** | Action-Adventure |
| ▼ Main Categ. Customizations | |
| **Physics** | ☐ |
| **Dialog** | ☑ |
| ▼ Sec. Categ. Customizations | |
| **Show Message** | ☑ |
| **Use (ie.: weapon)** | ☑ |
| **Release** | ☑ |

**Zelda** : A link to the past

| | |
|---|---|
| ▼ Characteristcs | |
| **Main Category** | Action |
| **Secondary Category** | Action-Adventure |
| ▼ Main Categ. Customizations | |
| **Physics** | ☐ |
| **Level** | ☐ |
| **Dialog** | ☑ |
| ▼ Sec. Categ. Customizations | |
| **Camera** | Movable |
| **Show Message** | ☑ |
| **Use (ie.: weapon)** | ☑ |
| **Release** | ☑ |

**Pokemon Red**

| | |
|---|---|
| ▼ Characteristcs | |
| **Main Category** | RPG |
| **Secondary Category** | RPG |
| ▼ Main Categ. Customizations | |
| **Score** | Challange |
| ▼ Sec. Categ. Customizations | |
| **Camera** | Movable |
| **Viewpoint** | Top |
| **Time** | No |
| **Use (ie.: weapon)** | ☑ |

Figura 5.7: Detalhamento dos títulos de jogos digitais quanto à categorias na qual se inserem e também quanto a suas características de singularidade (customização dos valore padrões dos elementos de determinado modelo).

## 5.2 Análises de Títulos

Observar a existência de elementos como componentes que caracterizam de jogos digitais é um desafio marcante quanto a necessidade da criação de uma etimologia "popular", genérica e descritiva. Sendo assim, através da análise dos jogos escolhidos, os mesmos são enquadrados (Tabela 5.1) de acordo com as suas categorias, cobiçando-se assim, a criação de uma nomenclatura que possa servir como base às classes da ontologia proposta. Visto isso, pode-se inferir que um jogo digital é formulado mediante a computação de todos seus elementos, tendo seu resultado modificado devido a característica paramétrica de cada elemento.

Tabela 5.1: Títulos analisados e suas classificações respectivas.

| Título | Categoria Primária | Categoria Secundária |
|---|---|---|
| Pong | Sport | Default |
| Luxor III | Puzzle | Default |
| Halo : Combat Evolved | Action | FPS |
| Spore | Strategy | RTS |
| Warcraft III | Strategy | RTS |
| World of Goo | Puzzle | Default |
| Tetris Zone | Puzzle | Default |
| Full Throttle | Adventure | Default |
| Zelda : A Link to the Past | Action | Action-Adventure |
| Super Mario World | Action | Platform |
| Rock N Roll Racing | Sport | Racing |
| Pokemon Red | RPG | RPG |
| Guitar Hero : Legends Of Rock | Music | Default |
| GTA II | Action | Action-Adventure |
| Worms 3D | Strategy | Artillery |
| Need For Speed Pro Street | Sport | Racing |

# 6 DIGEO: DIGITAL GAME ELEMENTS ONTOLOGY

O desenvolvimento de uma ontologia que descreva a presença de elementos presentes a composição de jogos digitais sob a perspectiva de um desenvolvedor é proposto como uma forma especializada de pesquisa à ontologias de IA. A *Digital Game Elements Ontology* (DIGEO), ou em português, *ontologia de elementos de jogos digitais*, é uma ontologia que visa permitir uma compreensão e comunicação de forma compartilhada de domínio público sobre o desenvolvimento de jogos digitais. Com isso, cria-se uma linguagem para especificação de conceitos dos elementos que compõem os jogos digitais. A presente abordagem transcende a visão/interpretação de um *gamer* (jogador), uma vez que a ontologia proposta serve, a partir das mesmas definições e vocabulários, como base para o desenvolvimento acessível de jogos.

No campo de ação prático a respeito da ontologia o usuário tem a possibilidade de descrever um jogo digital, seja a fim de desenvolvimento seja a fim de análise, por meio de uma compreensão padronizada e de fácil uso/compreensão. Isso acontece, pois, apesar de todo e qualquer jogo digital ser descrito de diversos pontos de vista ou do desenvolvedor ou de um jogador casual, toda e qualquer descrição/definição a ser adquirida com o emprego da DIGEO, há uma padronização de pontos de vista e é composta de uma *etimologia* semelhante à de um dicionário de um conhecimento específico.

Observa-se que o desenvolvimento da ontologia faz-se presente ao longo do capítulo, apresentando, de forma indireta todas as etapas descritas na seção 2.1.2. Por fim, a implementação da ontologia utiliza o software *Progeté* e a linguagem *Web Ontology language* (OWL) [66; 8] como tecnologias a fim de sustentar sua construção de forma visual e amparar sua aplicabilidade computacional, respectivamente.

## 6.1 Contextualização

Todo e qualquer jogo digital pode ser descrito de diversas maneiras: sob o ponto de vista de usuário; da interatividade entre usuário e jogo; dos elementos visuais de um jogo; dos elementos que compõem um jogo e outras formas. De todas as maneiras para descrever um jogo, a menos explorada, num contexto global, é a ausência de uma descrição mais prática e de fácil entendimento aos principiantes nessa área.

A experiência humana entre jogadores e jogos digitais é compreendida cada vez mais como uma relação de mutualismo, cujo desenvolvimento dos jogos depende do *feedback* e da visão que um jogador possuir sobre o jogo e vice-versa. Observando-se tal relação, demonstra-se interesse em ampliar a abrangência e o alcance do conhecimento técnico nessa área para um público alvo de pessoas externas ao mundo acadêmico ou profissional por intermédio de uma linguagem coloquial e de fácil

compreensão.

Sob um olhar antropológico, como afirma Malaby [78], "levando em conta a popularidade que os jogos vêm conseguindo ao longo dos anos, que existem novas oportunidades para que consigamos aprimorar o entendimento e a relação da nossa sociedade com os jogos", evidencia-se um forte interesse em descartar formalismos. Com isso, o desejo de ampliar a extensão do conhecimento formal sobre desenvolvimento de jogos aspira-se à criação de uma linguagem que permeie no limiar entre o conhecimento popular e a academia de jogos digitais; a implementação desse modelo de expressão torna hábil a aplicação de duas grandes escolas — atuantes nesse contexto, sendo elas a narratologia e a ludologia, na forma prática de seus conhecimentos.

A partir de uma reflexão dos questionamentos e das propostas, nota-se que até mesmo num contexto mais formal e acadêmico, *game designers* demonstram predisposição e interesse [45] à criação de uma linguagem formal e padronizada à área. A principal dificuldade faz-se presente a partir da falta de um vocabulário unificado à descrição de jogos existentes e para a criação de novos. Apesar da existência de diversos formalismos [5; 79; 56; 2] intencionados à criação de terminologias, categorizações, padrões de design, nenhum destes projetos foca-se, exclusivamente, na formalização de uma linguagem ou modelo de estudo destinado a sanar problemas de padronização no desenvolvimento de jogos digitais. A par de tal carência, propõe-se, para sanar esse lacuna, a ontologia DIGEO.

## 6.2 Escopo, Domínio e Método

Antes de iniciar o desenvolvimento de uma ontologia, deve-se situar o assunto de forma a entender a proposta formal da ontologia diante da finalidade (escopo), abrangência (domínio) e conjunto de regras básicas (método). O principal objetivo desta ontologia é "tornar o conhecimento do desenvolvimento de jogos acessível, padronizado e formal — por meio de uma linguagem de fácil compreensão, a um público que abrange desde profissionais até curiosos relacionados a área".

Naturalmente, o conceito e o conjunto de regras que expressam a ontologia figuram sob a combinação de indivíduos das classes da ontologia como componentes que se relacionam via conexões e regras de restrições, abordadas no próximo capítulo, que apresenta uma forma de utilização da mesma. Essa ontologia não deve conter informações genéricas sobre jogos digitais ou mesmo informações que não estejam ligadas ao desenvolvimento de jogos de forma analítica ou criacional.

### 6.2.1 Questões de Competência

De acordo com a abordagem teórica, (Seção 2.1.2), a fim de direcionar o desenvolvimento da DIGEO utiliza-se aqui a abordagem de meta-questions visto que elas servem como guia para manter o foco central na proposta. Algumas das respostas às questões podem divergir ou tomar rumos diferenciados com o decorrer do processo criacional da ontologia, isso porque elas já atuaram no seu papel principal: limitar o escopo do modelo. Apesar de algumas das questões terem um cunho global, elas estão limitadas ao domínio da ontologia.

De acordo com o domínio desta ontologia, pôde-se enunciar as questões de competência relacionadas, a saber: "Qual é a composição de um jogo?", "Como um jogo é desenvolvido?", "Quais as relações entre os elementos dentro de um jogo?", "Quais

são definições de elementos e instâncias dos mesmos?", "O que leva esta especificação a não ser apenas uma taxonomia?", "O que leva esta especificação a não ser apenas um agrupamento de definições?", "Quais os relacionamentos existentes?", "Qual a diferença entre um elemento e um componente?", "Qual será a cobertura do domínio?", "Por que utilizar a ontologia?", "Qual a aplicabilidade da ontologia?", "Para que tipo de perguntas a ontologia deve providenciar as respostas?". Após terem sido respondidas todas as questões a respeito da ontologia (Anexo A.1), pode-se então enunciar quais são, de acordo com o domínio da mesma, os papéis (função) principais da ontologia:

1. Ter uma definição que precede, no escopo de desenvolvimento, definições de ontologias de jogos já existentes;
2. Vocabulário comum e padrão
3. Adicionar o entendimento do que são elementos;
4. Possibilitar a descrição da composição de jogos digitais sob um olhar de *game design* por meio do uso de elementos;
5. Tornar explícito e acessível as definições a respeito dos elementos necessários para o desenvolvimento de jogos digitais;
6. Formalizar a composição de um jogo digital;
7. Explicar o funcionamento semântico e as relações das classes da ontologia;
8. Auxiliar o desenvolvimento de jogos para ser menos burocrático, diminuindo o tempo e tornando mais compreensível;

## 6.3 Referencial Teórico

É válido uma consideração a respeito de trabalhos relacionados (Capítulo 4), pois os mesmos ajudam a aperfeiçoar e a ampliar as fontes existentes em um domínio específico e tarefa. Baseando-se nas idéias centrais da ontologia GOP, a ontologia aqui proposta descreve elementos de jogos digitais sob um olhar teórico, a partir de estudos de *game design*, utilizando-se um vocabulário que mistura taxonomias apresentadas nos estudos GOP, *Digital Game Onthology*, entre outros fundamentos.

A idealização de um modelo teórico, diferenciado e que padronize o vocabulário, bem como a aplicação da teoria do desenvolvimento de jogos numa implementação viável ao desenvolvimento de jogos frente a uma linguagem padronizada é a aspiração de muitos estudos [80; 81; 82]. Mesmo existindo diversas propostas de ontologias ou modelos de desenvolvimento, poucas proporcionam a criação de aplicativos ou linguagens que facilitem o desenvolvimento de jogos de uma forma a impulsionar a aprendizagem do conhecimento relacionado ao desenvolvimento de jogos digitais — desde protótipos até jogos semi-profissionais.

O formalismo em que cada ontologia é expressa, não tem forte impacto na prática, desde que elas sejam sistemas de representação de conhecimento, pois elas podem evoluir de acordo com o passar do tempo. Apesar dessa afirmação, o uso de ontologias com uma base formal, como é o caso da DIGEO, uma vez que usa formalizações de *game design*, serve como um fundamento sólido para tradução de

definições em objetos de uso real, como é o caso dos "elementos", os componentes reais, presentes em um título de jogo digital.

Mesmo com o auxílio de ontologias e estudos já existentes, aqui, a ontologia será desenvolvida a partir do zero, uma vez que o foco propõe o uso de forma a descrever e consolidar elementos sob um aspecto implemental — abordagem que diverge ao olhar de outras ontologias ligadas a áreas do meio digital. É importante observar que o uso das análises (Capítulo 5) servirá como um embasamento ao reconhecimento de padrões, recursos e nomenclaturas comuns aos jogos analisados, possibilitando a descrição dos mesmos sob a forma de elementos.

## 6.4 Etimologia e Termos

Etimologia é a parte da gramática que trata da história ou origem das palavras e da explicação do significado de palavras através da análise dos objetos que a integram. A partir de tal definição e da pergunta "qual a diferença entre taxonomia e conceitos?", sabe-se que uma ontologia contém uma taxonomia, bem como seus conceitos. Em geral, uma terminologia ou a criação de uma outra, raramente é obtida sem nenhuma fonte de consulta. Então, o vocabulário e os termos existentes na ontologia proposta são frutos do estudo de *game design*, dos trabalhos relacionados, mas principalmente da análise de elementos (classificados em categorias) conforme fora declarado na seção 5.1.

Diferentemente de outras ontologias de jogos, a DIGEO utiliza uma taxonomia ligada a área de estudo de jogos digitais, porém, o vocabulário é organizado de forma a descrever as interações e características dos elementos. Com o desenvolvimento da ontologia, há uma divisão de conhecimento explícito, sendo os elementos de um jogo composto em três grandes grupos: descritivos (seção 6.8), comportamentais (seção 6.7) e interativos (seção 6.9).

## 6.5 A ontologia

Em síntese, a DIGEO é uma ontologia que identifica elementos abstratos que compõem um jogo digital sob uma perspectiva de desenvolvimento. A função primária da mesma é a de tornar o conhecimento dos elementos ligados ao desenvolvimento de jogos digitais acessível e compreensível ao maior número de indivíduos. Em conseqüência de sua existência, um vocabulário comum e de uma aplicação prática, torna o conhecimento inteligível em nível de desenvolvimento a todo indivíduos que possuem interesse à área. Com isso, aspira-se à aplicação da mesma na criação, seja através de processos seja através de uma tecnologia, de títulos reais.

Para a formalização e criação desse "guia de desenvolvimento de jogos"[1], necessitam-se criar as classes e posicioná-las na hierarquia da ontologia. A definição de classes, propriedades e restrições serão abordados mutuamente, porque as classes, por si só, não fornecerão informações suficientes à descrição da estrutura interna dos conceitos.

É importante que utilizando um método descritivo *top-down*, ele distingue a classificação genérica primeiro e depois especializa de acordo com as características, aplicações e relações dos mesmos, ou seja, a partir de informações genéricas, no caso da DIGEO "definir um jogo a partir de elementos de jogos digitais", inicialmente

[1]Comentário feito por pessoas leigas à área do desenvolvimento de jogos quando apresentados à DIGEO.

se criam os elementos e a partir novos exemplos não se adicionam novas classes, no máximo propriedades. Já no caso da GOP, a mesma aborda um desenvolvimento *bottom-top* no momento em que novas classes são adicionadas atendendo à análise de novos títulos de jogos digitais. As classes são organizadas hierarquicamente, de forma a questionar se, por serem instâncias de uma classe, os objetos irão (necessariamente), por definição, ser instâncias de classes pai na hierarquia.

Na abordagem visual das classes, observa-se o símbolo ⓒ, na cor preta, à frente de toda e qualquer definição de classe, já o símbolo ⓟ, na cor cinza, descreve as propriedades da classe em que ele está inserido, através da formatação tabular[2] hierárquico. As propriedades contam, ainda, com algumas particularidade como o *range*, ou também conhecido em programação como restrição de tipo. O range será representado através da restrição, entre parênteses, numa formatação subscrita (ex.: $_{\text{(list)}}$ ), antes da definição do propriedade. Os valores padrões lançam mão da nomenclatura de tipos de dados em inglês, assim temos como principais valores: *int* - valores numérico inteiros; *uint* - valores numérico inteiros positivos; *string* – texto literal; *list* - uma lista ou, em linguagem computacional, um array de valores; *pt3D* - um ponto cartesiano num espaço tridimensional; *boolean* - é uma chave de valores booleanos: verdadeiro ou falso. Propriedades imperativas são aquelas propriedades marcadas pela simbologia ⓘ, que se tiverem seus valores modificados afetarão seus filhos na hierarquia. Com o ímpeto de fidelizar a abordagem hierárquica das classes da ontologia, a representação descritiva das mesmas vai ser dada via utilização de uma formatação tabular hierárquica. Uma das restrições presentes nas definições de classes da ontologia é o fato de que uma instância não pode ter outra instância da mesma classe como filha exceto se na definição da classe existir o símbolo ⓜ, indicando mutualidade.

## 6.6 Classes de níveis primários

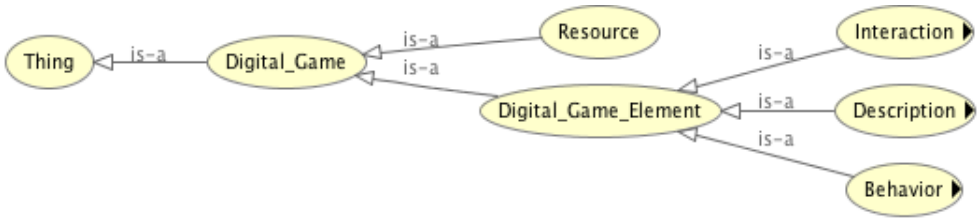


Figura 6.1: Classes de primeiro grau da hierarquia. “Thing” representa o ponto de partida de toda e qualquer ontologia.

ⓒ **Digital Game** Representa um jogo digital a partir do ponto de vista de um desenvolvedor, independente do grau de experiência. A composição de um jogo é obtida através do agrupamento de elementos.

ⓟ **id** $_{\text{(string)}}$ Um nome, único, que serve de referência entre as outras subclasses de *Digital Game Element*. Quando referenciado o id de um elemento deve ser precedido do símbolo sustenido (#).

ⓟ **name** $_{\text{(string)}}$ Nome textual que representa a instância da classe. Caso suprimido, nenhum texto é associado.

[2]Tabular é o espaçamento de aproximadamente 1cm entre uma palavra (ou espaçamento) e outra

ⓒ **Resource** Algum material externo à definição do jogo em si. Pode ser alguma malha 3D, XML, *script*, imagem 2D, som, entre outros.

ⓟ **src** (string) O caminho relativo de um material.

ⓟ **type** (string) Representa o tipo (classificação) do material. Valores existentes: sound, image, xml, lua, mesh.

ⓒ **Digital Game Element** O conjunto de todos os elementos que compõem um jogo digital. A alteração no valor das propriedades afetará, também, os elementos filhos.

ⓟ **input** (list) Todo e qualquer elemento pode possuir de 0 a N entradas (*inputs*) - essas são compostas pela referência a outros elementos. Essas entradas servem de base para a iniciação do elemento. Por exemplo, um elemento *"hit"* vai possuir uma entrada referenciando o elemento com o qual ele possuirá interação; entre outras palavras, o elemento necessita saber contra quem ele irá colidir.

ⓟ **childrens** (list) A lista de filhos que esse elemento possui. Um elemento é uma instância iterativa, ou seja, ela pode conter de forma recursiva, inúmeros elementos filhos - os mesmos serão afetados por trocas de estados de seus pais. Como exemplo, um elemento pai "level" possui um elemento "user"; se o elemento pai tivesse a propriedade *visible* = *false*, automaticamente seus filhos não seriam visíveis - apenas se a propriedades for imperativa (ⓘ).

ⓟ **filterExpression** (string) O elemento só será interpretado se a expressão retornar verdadeira. Os valores de variáveis são referentes a proriedade de um elemento, referenciado através do id do mesmo. Exemplo, uma expressão válida seria "#id.visible == true", ou seja, o elemento só sofreria algum tipo de interação ou troca de estado de essa expressão fosse válida.

ⓟⓘ **state** (string) Quando possuímos diversas instâncias de um mesmo elemento necessitamos saber qual delas que estão ativas e inativas. Por exemplo, se um jogo possui vários leveis (níveis), necessitamos saber qual é o level atual (o que tiver *state* = ACTIVE) e quais são os outros leveis que não estão ativos no momento (os que tiverem a propriedade *state* = INACTIVE). Os valores possíveis são: ACTIVE, informando o elemento que possui interação com os outros e com o meio; IDLE, informando que um elemento não sofrerá atualização de seu estado atual; INACTIVE, o elemento será ignorado.

ⓒ **Behavior** Elementos que representam os comportamentos dos elementos consigo mesmo ou entre os existentes. São pares de ação que geram determinadas reação em um elemento. Seus comportamentos possuem gatilhos dúbias, podendo ser dinâmicas (acionados por outros elementos) ou atônomas (acionados por si mesmo).

ⓒ **Description** Elementos que descrevem um jogo digital como um título. São os elementos que caracterizam um jogo de acordo com as características basicas de um título, são os responsáveis por classificar o jogo sendo pertencendo de certa categoria primária. Composto de 0 a N *behaviors* & *interactions*.

ⓒ **Interaction** Elementos interativos, são aqueles que possuem interação entre elementos de acordo com *behaviors* ou outras disturbios do meio. É um elemento que atua como um processador de *inputs* externos gerando saídas, sendo essas o processamento de outro elemento.

## 6.7 Classe "Behavior" e arvore hierárquica

ⓒ **AI** Interações e modificações de estado sem a interação do usuário ou de outro elemento. São ações autônomas que modificam o estado do próprio elemento. Possui como *input* o id de uma *ressource* que seja um script externo (até o momento o único suportado é lua).

ⓒ **EventListener** É um elemento que, ao receber uma notificação de um *Event Dispatcher* específico, troca os estados dos seus elementos filhos, que incialmente é INACTIVE para ACTIVE (durante o tempo de ação dos elementos). os ids dos elementos gatilhos (*Event Dispatcher*) são recebidos através da propriedade *input*.

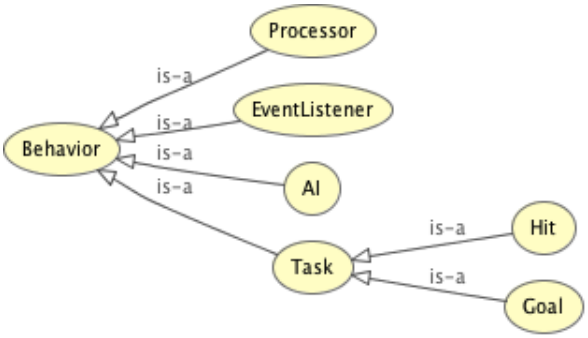


Figura 6.2: Ramo hierárquico filho do elemento **Behavior**.

ⓒ **Processor** É um elemento que quando ativado processa em um ciclo de N vezes (definidas na propriedade) seus elementos filhos - diferente do que acontece num listener normal, quando os elementos filhos são executados apenas uma vez.

ⓟ **times** (uint) Número de ciclos em que os elementos filhos são interpretados.

ⓒ **Task** Uma tarefa é computada somente quando um elemento cumprir a tarefa a si inferida. No momento da validação (computação), seus elementos filhos, que inicialmente tinham o *state* INACTIVE passam para ACTIVE (somente nesse momento, depois de executada a ação eles voltam para INACTIVE).

ⓒ **Goal** Se a propriedade *filterExpression* for verdadeira a tarefa é computada. Esse elemento serve para quando existem tarefas mais complexas que podem ser representadas através de expressões computacionais. Por exemplo, com ela podemos validar se a pontuação de um ou mais jogador atingiu o máximo; ou até mesmo podemos verificar se certo elemento está na posição X igual a 100.

ⓒ **Hit** Se o elemento pai deste elemento entrar em contato (colidir) com pelo menos um dos elementos que estão referenciado no *slot input* a tarefa é computada.

## 6.8 Classe "Description" e arvore hierárquica

ⓒ **Bound** São os limites do jogo, ou seja, espaço 3D no qual os elementos visuais de um jogo estão presentes. Não pertencem a esse grupo os elementos de GUI.

ⓟ **collidable** (boolean) Faz com que os limites do cenário atuem de forma a barrar objetos que intentem sair do limites.

ⓟ **from** (pt3D) Ponto de origem, do plano cartesiano que representa os limites do jogo.

ⓟ **to** (pt3D) Ponto final no espaço 3D. Esse ponto formaliza a existência do polígono, abstrato, que representa os limites do mundo 3D do game.

ⓒ **Display** É o recipiente que armazena as classes de visualização, representando os dados visuais do game.

ⓒ **Camera** É um observador que, dependendo do posicionamento, do alvo e das características, revela as imagens do mundo 3D. É um referencial visual. Um jogo pode estar repleto de *Cameras* - elas atuam da mesma maneira que uma camera de gravação de vídeo atua na vida real.

ⓟ **target** (string) Alvo, ou seja, o elemento que a câmera estará mirando.

ⓟ **position** (pt3D) Posição da câmera no espaço 3D.

ⓟ **type** (string) Categoria da câmera diante de seu comportamento. Os tipos variam entre "fixed" - sendo uma camera parada, "tracked" - uma câmera que acompanha o alvo, "free" - uma câmera móvel com total liberdade de posicionamento e "movable" - sendo uma câmera "free" com a possibilidade de rotacionar a câmera.

Figura 6.3: Ramo hierárquico filho do elemento **Description**.

ⓒ **Projection** É a representação - modo de visualização - de objetos tridimensionais em uma superfície planar (Câmera) de acordo com pontos de referência.

ⓟ **type** (string) O tipo classifica uma *Projection* de acordo com dada perspectiva 3D. Os tipos variam entre "isometric", "side", "top", "free", "third-person" e "first-person".

ⓒ **Level** É um nível de um jogo. Tudo que é jogo possui pelo menos um level. Essa classe atua como um recipiente de elementos de determinada fase de um jogo.

ⓒ **Map** Cada região visual do jogo possui um mapa, sendo limitado por um *Bound*.

ⓟ **type** (string) Essa propriedade classifica a classe diante do tipo de visualização do mapa, ou seja, como o mapa é revelado ao jogador. Os valores existentes são "visible", "masked" e "fog of war".

ⓒ **Dialog** Algum tipo de comunicação ou mensagem textual com representação visual, legível ao usuário. Pode ser uma mensagem, um diálogo, ou algum texto informativo. Um diálogo é composto por diversas mensagens (que inicialmente estão no estado INACTIVE).

ⓒ **Message** É conteúdo da informação (comunicação) apresentado de forma inteligível e compreensível, ou seja, é um texto explícito.

ⓟ **emissor** (string) O emissor, ou seja, o id de quem estará emitindo o texto. Se o slot for omitido a mensagem não possuirá emissor explícito.

ⓟ **receptor** (string) O id da entidade que recebe a mensagem. Se o slot for omitido a mensagem não possuirá emissor explícito.

ⓟ **text** (string) O texto, representação textual da mensagem.

ⓒⓜ **Object** Uma entidade visual existente no mundo do jogo. Atua como se fosse um *container* visual que interage com o meio.

ⓟⓘ **visible** (boolean) Indica se o elemento é visível ou não ao usuário final.

ⓟ **display** (string) O id da *Ressource* que representa esse objeto de forma visual. O *slot* é deixado em branco se a instância atua como repositório.

ⓟⓘ **position** (pt3D) Representa a posição do objeto dentro do mapa.

ⓟⓘ **size** (pt3D) *Bouding box* representando a dimensão do objeto no espaço 3D.

ⓒ **Dynamic Object** Objetos dinâmicos são aqueles que possuem sensores perceptivos capazes de reconhecer mudanças de estados e interagir com as mesmas.

ⓒ **Static Object** Objetos estáticos que não modificam suas propriedades ou comportamentos meio a interatividades de elementos. Mesmo se possuir algum elemento filho, todos os elementos estarão no estado IDLE. Diferentemente de um objeto dinâmico, que é afetado pela gravidade (quando presente), um objeto estático não reconhece e manipula com a presença de fatores externos.

ⓒ **Physics** Descreve a simulação física existente no jogo.

ⓟ **value** (boolean) Valor dizendo se o jogo possui ou não simulação física pré-definida

ⓒ **Score** Armazena informações relacionadas às pontuações do jogo.

ⓟ **value** (int) Valor da pontuação, podendo representar qualquer unidade de representação.

ⓒ **State** Descreve o estado atual de um certo elemento.

ⓟ **value** (string) Os valores de estado servem para descrever um estado de forma textual. Por exemplo, pode-se atribuir o valor de "defendendo" para uma instância de *Entity* e assim pode-se criar regras de gameplay (se um id está atacando e o outro não está defendendo soma ponto no score, caso contrário não faça nada).

ⓒ **Time** Contador de tempo, funciona como se fosse um cronômetro.

- ⓟ **type** (string) Informa o tipo de contador. Como valores temos "free" representando um contador que não possui limite de contagem, "limited", contendo uma informação de tempo limite e "turn" que faz o cronômetro atuar em ciclos repetitivos - quando acaba um ciclo começa outro.
- ⓟ **state** (string) É o estado atual do contador, sendo que ele pode estar parado (STOPPED), contando tempo (RUNNING) ou já ter acabado (DONE).
- ⓟ **current** (uint) Marcação atual.
- ⓟ **end** (uint) Valor limite.

ⓒ **User** É a representação abstrata de um usuário presente no jogo.

- ⓟ **controlls** (list) É uma lista de ids de entidades (uma instância da classe Object) que o jogador controla. Representa o controle do usuário sob os objetos presentes no jogo.

ⓒ **User Interface** Interface gráfica do jogo, sendo composta por telas, botões, imagens e todas as outras representações planar, no ambiente 3D, que possuem a finalidade de informar o usuário final.

- ⓟ **display** (string) Id da Ressource que representa essa interface. Pode ser uma imagem, uma primitiva 2D ou um objeto 3D; porém, caso não especificado, o elemento possui uma representação transparente (podendo servir como um container).
- ⓟ **position** (pt3D) Posição do elemento no espaço 3D. O ponto de registro (ponto de origem, referência pela qual será posicionado) localiza-se na posição 0, nas três coordenadas cartesianas, do seu tamanho (propriedade *size*).

ⓒ **Visual Component** É uma especialização da interface gráfica com formatos pré-definidos.

- ⓟ **type** (string) Meio a especificação de tipo pode-se inferir como será a representação visual e o comportamento do elemento. Os valores descritos são "textField", representando um campo de texto; "inputTextField", sendo um campo de texto o qual o usuário pode inserir algum texto com o uso do teclado; "line" é uma primitiva gráfica 2D figurada meio ao conceito de linha; "image" sendo uma imagem 2D; "button" é um botão de se clicado muda o estado de seus filhos de INACTIVE para ACTIVE.
- ⓟ **value** (list) É um valor ou uma lista de valores a ser interpretada pela classe e figurada meio a representação visual da instância.

ⓒⓜ **Screen** É a representação de uma tela, o contexto gráfico base para a representação visual de algum elemento. Como exemplo do mundo real se fosse igualado um Digital Game a uma pintura, a Screen seria a tela, pois isso indica que a pintura pode até existir sem a tela (na mente do pintor, por exemplo), porém a pintura não poderá ser vista até ser posta em uma tela.

## 6.9 Classe "Interaction" e arvore hierárquica

ⓒ **Action** É uma ação inferida sob algum elemento. Essa ação modifica o estado e/ou as características do elemento alvo das ações.

- ⓟ **source** (string) O id do elemento que irá inferir a ação. Caso omitido tal propriedade, a fonte da ação será o pai deste elemento.
- ⓟ **target** (string) O id do elemento alvo que sofrerá a ação.

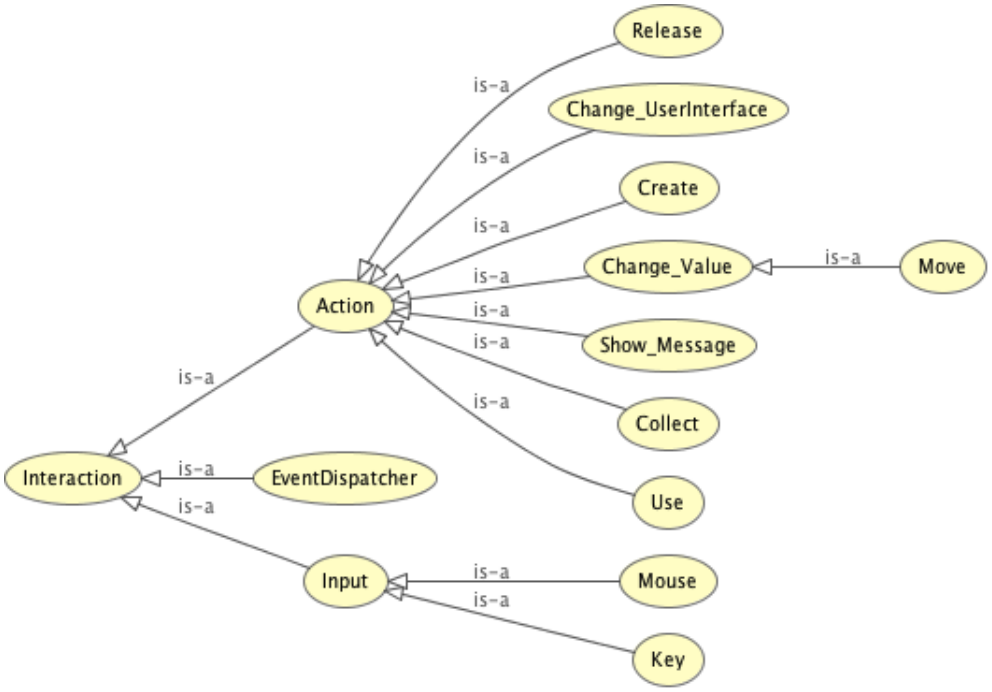


Figura 6.4: Ramo hierárquico filho do elemento **Interaction**.

- ⓒ **Change Screen** Esta classe tem como objetivo a mudança da tela atual para outra. Ou seja, não será mais vista a tela atual, mas sim a tela alvo. Um bom exemplo é a troca de telas de um jogo - quando está no menu principal e se escolhe alguma outra seção.
- ⓒ **Change Value** Modifica o valor de qualquer propriedade de um elemento.
  - ⓟ **property** (string) A proriedade que deseja-se alterar o valor.
  - ⓟ **type** (string) Tipo de operação que será computada, sendo que exitem "change" que troca o valor atual pelo novo valor, "add" que adiciona o novo valor'ao valor antigo da propriedade e "multiply" que multiplica o valor da propriedade alvo pelo valor desejado.
  - ⓟ **value** (string ou int) O valor que deseja-se computar.
  - ⓒ **Move** Especialização de Change Value, essa classe movimenta a entidade alvo.
    - ⓟ **position** (pt3D) Ponto 3D para o qual o elemento será reposicionado.
    - ⓟ **ammount** (int) Valor que será adicionado a posição atual. Serve para movientar o objeto alvo.
    - ⓟ **axis** (list) Eixos que serão afetados pela ação. Os valores aceitos são "x", "y" e "z".
    - ⓟ **invert** (boolean) Se verdadeira, inverte o movimento do elemento.
- ⓒ **Collect** Armazena elementos.
- ⓒ **Create** Cria elementos dinamicamente.
- ⓒ **Release** Libera elementos armazenados.
- ⓒ **Show Message** Mostra uma mensagem de diálogo.

ⓒ **EventDispatcher** Dispara eventos (avisos) que são reconhecidos pelos Listeners e executados.

ⓒ **Input** Representa as operações de input (entradas) feitas pelo usuário.

- ⓒ **Key** Captura as entradas obtidas através do teclado
  - ⓟ **code** (string) Representa o keycode (código numérico) de uma tecla do teclado.

ⓒ **Mouse** Captura as entradas de dados obtidas pela interação do usuário com o mouse.

ⓟ **button** (string) Indica o botão pressionado.

## 6.10 Conclusão

A existência de um modelo capaz de atuar como um referencial a definições do que são e quais são os elementos presentes na produção de jogos digitais, serve como uma maneira de amenizar o processo burocratizado e a necessidade de um conhecimento técnico mais abrangente sobre o conhecimento ligado ao processo de desenvolvimento de jogos digitais. Impelido a esse desafio, criou-se a DIGEO, uma ontologia que serve como base a um conhecimento padrão, estável, agregado e formal, necessário para a construção de um modelo que impulsione o desenvolvimento, teórico e prático frente a uma "receita" formal, porém com palavras e definições de fácil compreensão[3].

Desse modo, novos conceitos e abordagens são relacionadas ao entendimento de elementos de jogos que estarão presentes nas etapas de *game design* de um jogo digital, cujos usuários utilizem a ontologia proposta. Com o conhecimento compartilhado por diversas pessoas, por natureza, a utilização de forma colaborativa (para sua evolução e modificação) pela comunidade é a razão do uso que permite a ontologia desempenhar o seu papel — possibilitando acesso ao conhecimento via o *website*[4] da DIGEO.

[3]É por esse motivo que muitas vezes confunde-se o formal e o informal. A ontologia lança mão de definições informais baseando-se em estudos técnicos e formais.

[4]Acessível via url `http://www.filipesilvestrim.com/digeo/`

# 7 IMPLEMENTAÇÃO

A fim de comprovar a aplicabilidade prática da DIGEO propõe-se a implementação de um motor de jogo, que utiliza todos os elementos e relacionamentos da ontologia, para o desenvolvimento de jogos Desktop através da linguagem C++. A implementação do motor baseia-se na aplicação de um modelo (Figura 7.1) de implementação que propõe a descrição de jogos digitais a partir de uma linguagem de alto nível que descreva instâncias e relacionamentos de elementos de jogos digitais com fins à implementação da teoria de elementos. Com o intuito de fomentar o uso do modelo como uma proposta sólida ao desenvolvimento de jogos, oriundos de conceitos abstratos de *game design*, reconhece-se de forma explícita as classes da DIGEO. Isso acontece pelo fato de que de ontologias trabalham num paradigma muito similar ao OOP.

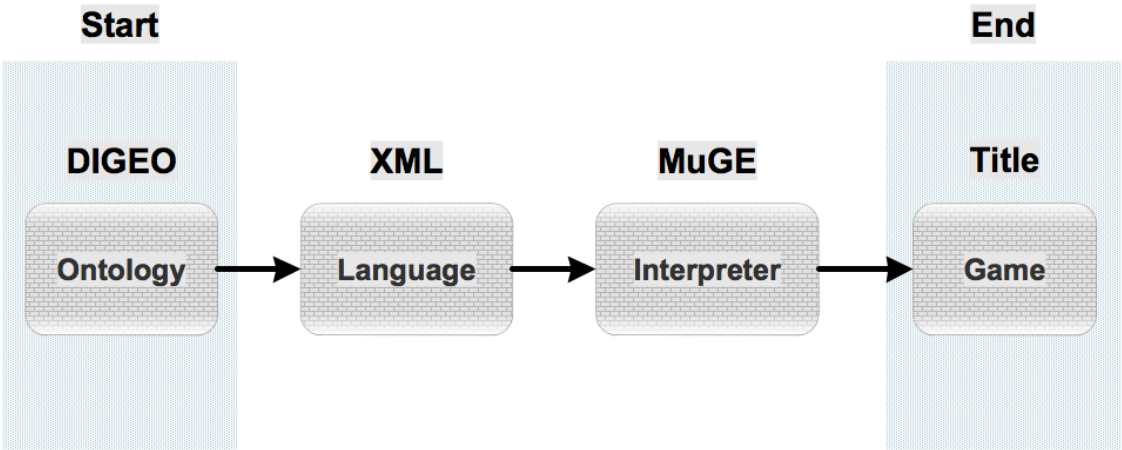


Figura 7.1: Modelo de desenvolvimento proposto para a criação de um título por meio da DIGEO.

O modelo aborda uma proposta de desenvolvimento focada num fluxo de trabalho que a cada etapa é um processo que serve de base para o próximo. A construção do modelo requer a implementação de todas as etapas de forma que a ontologia DIGEO, já proposta no capítulo 6 como base, ou etapa inicial, para o desenvolvimento do modelo. O modelo é desenvolvido consoante a um fluxo de dependências. O início (*start*) é dado a partir de uma ontologia, nesse caso a DIGEO, que, por sua vez, precisa ser transcrita para uma formato computacionalmente válido. Na questão, opta-se pelo uso do XML, uma linguagem textual e de marcação. Criada a linguagem, necessita-se de um programa interpretador capaz de traduzir tudo que está

expresso nessa definição para uma sintaxe e semântica interpretável pelo MuGE. Por fim, pelo uso desse interpretador obtém-se o título do jogo digital, definido através do XML e de modificações na etapa de interpretação.

## 7.1 A linguagem

A *DIGEO Language* é uma linguagem de marcação, alicerçada em XML, para representação de elementos de jogos digitais. Ela é utilizada principalmente para esboçar, declarativamente, um jogo digital voltado a instâncias da ontologia DIGEO. O XML é uma linguagem de marcação proposta pela W3C [83] consistindo numa série de regras para codificação de documentos eletrônicos — uma linguagem muito semelhante ao HTML — através de uma linguagem formal e legível para compreensão humana, o XML é um agrupado de *tags* (marcadores da sintaxe) personalizados que descrevem os dados (objetos) de certo domínio de estudo, como neste caso as classes da DIGEO.

A fim de padronizar e criar uma fidelidade entre os dados, a sintaxe da linguagem baseia-se nas classes e propriedades da DIGEO. Isso é possível devido à existência do *schema*[1] OWL da ontologia. Com o auxílio de tais definições, a sintaxe desta linguagem é uma extensão da representação XML do *schema* OWL da ontologia. Além das definições da DIGEO, foram criadas algumas definições extras (seção 7.1.1) a fim de criar uma estrutura e um fluxo computacional mais conciso.

Essa linguagem é apenas um guia de representação, ou seja, ela não é uma linguagem oficialmente documentada ou mantida por alguma organização, isto é, decisões de como a linguagem será interpretada ou qual será o fluxo lógico depende inteiramente da interpretação feita por dado interpretador. Tal fato acontece uma vez que se deseja criar uma boa prática de uso que não seja restrita a alguma plataforma ou interpretador. Dá-se, assim, liberdade a programadores para criarem interpretadores que manipulem de diversas formas a linguagem: valorizando visualização, performance, *gameplay*, *script*, entre outros.

### 7.1.1 Elementos Extras

Elementos extras são regras que complementam a forma com que o interpretador, independente se é o interpretador do MuGE ou outro interpretador externo, irá manipular os dados, sob um ponto de vista computacional da ontologia.

**Aninhamentos** Se existirem *tags* aninhadas determina-se que as *tags* filhas estão necessariamente conectadas com seus pais. Isso significa que a *tag* filha possuirá como primeiro índice da sua lista de valores do parâmetro *input* o elemento pai. Conforme no exemplo abaixo (listagem 7.1) a *tag* **hit** é filha da *tag* **entity**, então, **hit** receberá implicitamente como primeiro valor de input `#entityP1`. Assim, o input de **hit** seria compreendido pelo interpretador como `input="#entityP1,#room"`

Listing 7.1: Exemplo no qual quando o elemento sofre ação de hit a entidade target de move irá se movimentar

[1]Documentos XML que expressam um vocabulário compartilhado a fim de permitir que programas realizem a ação restritiva de regras feito por pessoas.

```
1 <entity id="entityP1">
2     <hit id="hitPlayer1Bounds" input="#room">
3         <move target="#entityP1" invert="true" />
4     </hit>
5 </entity>
```

**Configuração** Usualmente existe ordem de prioridades para execução de certos códigos em teoria da programação. O mesmo acontece com os elementos de um jogo, uma vez que se necessita, por exemplo, configurar as câmeras e o tipo de projeção de certa fase de um jogo antes que outros elementos da fase sejam executados. Visto o grau de importância de tal característica criou-se a *tag* `<config/>` com o propósito de representar prioridade de execução diante de outras *tags* no mesmo grau hierárquico (*tags* irmãs, ou seja, filhas do mesmo pai). Conforme a listagem 7.2, somente depois de processar a `config`, o interpretador (que por padrão percorre os elementos de forma seqüencial) irá percorrer os outros elementos filhos de level.

Listing 7.2: Exemplo de prioridade de interpretação de código

```
1 <level id="mainLevel" input="#mainScreen, #mainDisplay">
2     <entity id="entityP1">
3         <!-- Inner code -->
4     </entity>
5
6     <entity id="entityP2">
7         <!-- Inner code -->
8     </entity>
9
10    <config>
11        <boundaries id="room" from="0,0,0" to="100,100,0"/>
12    </config>
13 </level>
```

**Grupos** A fim de criar uma maior legibilidade do código criado, uma vez que podem existir muitas *tags* dentro de um mesmo elemento, criou-se a *tag* `<group/>`, a fim de ser uma representação visual. No exemplo baixo as *tags* `group` servem apenas para diferenciar visualmente grupos de elementos.

Listing 7.3: Agrupamentos meramente visuais que não perturbam o sistema do interpretador

```
1 <level id="mainLevel" input="#mainScreen, #mainDisplay">
2     <config>
3         <boundaries id="room" from="0,0,0" to="100,100,0"/>
4
5         <group id="rules">
6             <!-- Inner code -->
7         </group>
8
9         <group id="users">
10            <!-- Inner code -->
```

```
11          </group>
12      </config>
13
14      <group id="elements">
15          <staticObject id="sensorGoalP1" position="0,0,0" size="↩
                10,10,1" display="protoBox">
16              <!-- Inner code -->
17          </staticObject>
18      </group>
19  </level>
```

**Carregamento** Muitas vezes utiliza-se muitos recursos externos (som, 3D, imagens, etc) que necessitam ser carregados antes mesmo que o jogo seja iniciado. Para isso, criou-se a *tag* `<load/>`. No exemplo abaixo, os recursos serão carregados antes mesmo que o jogo comece a rodar.

Listing 7.4: Exemplo de acesso a recursos externos

```
1 <Digital_Game>
2     <load>
3         <ressource id="logo" type="image" src="./res/logo.png" /↩
              >
4         <ressource id="mainSound" type="sound" src="./res/sound.↩
              mp3" />
5     </load>
6     <level id="mainLevel" input="#mainScreen, #mainDisplay">
7         <!-- Inner code -->
8     </level>
9 </Digital_Game>
```

**Scripts** Aspirando-se o suporte à linguagens de *scripts* externas surge a definição da *tag* `<script/>`. Abaixo segue um exemplo de como seria o comportamento deste recurso se implementado.

Listing 7.5: Linguagem de script aninhada à descrição do jogo

```
1 <listener id="winCondition" input="#collisionGoal1, #↩
      collisionGoal2">
2     <script language="python">
3         <![CDATA[
4             def processor(inputList):
5                 #Put the code over here
6                 return;
7         ]]>
8     </script>
9 </listener>
```

**Arquivos externos** Para a descrição de protótipos ou jogos com poucos elementos, a atual estrutura é satisfatoriamente aplicável, porém se nota certo desconforto com a criação de jogos complexos pelo motivo da existência de arquivos com muitas linhas de código, além da mistura entre *tags*. Para isso, aviva-se a idéia de criar uma *tag* `<include/>` que tenha o papel de importar pedaços

de código (como exemplifica a listagem 7.6), definidos em outros arquivos, de forma organizada e sucinta. Tais elementos, presentes nos arquivos carregados possuem a propriedade `state` implicitamente no valor **IDLE**, ou seja, estes códigos não serão interpretados sem uma inferência do desenvolvedor — seja via a linguagem XML ou via ação direta do interpretador.

Listing 7.6: Diversos arquivos definindo um jogo de forma mais organizada visualmente

```
1 <Digital_Game>
2     <load>
3         <!-- Inner code -->
4     </load>
5
6     <include id="level1" src="./levels/level1.xml" />
7     <include id="level2" src="./levels/level2.xml" />
8     <include id="level3" src="./levels/level3.xml" />
9 </Digital_Game>
```

Uma questão que deve ficar clara é que nem todas definições da linguagem estarão sempre presentes. Cada interpretador dará suporte a determinadas *tags*, elementos e definições da linguagem. Não se exige o suporte completo da linguagem, tanto que não se tem controle sobre os interpretadores criados.

## 7.2 Multi-Genre Game Engine

Tratando-se do quesito identificação deste motor escolheu-se o nome MuGE como sendo acrônimo de "*Multi-Genre Game Engine*". A escolha deu-se diante da visível possibilidade de serem criados jogos independentes de gênero voltado ao uso da ontologia DIGEO. Posto isso, a mesma tem como propósito proporcionar tanto a programadores, prototipadores e projetistas de jogos, quanto a iniciantes na criação de jogos digitais, um ambiente de desenvolvimento de jogos amigável de acordo com o uso da *DIGEO Language* e da proposta da DIGEO em si.

Como premissas gerais do motor, conta-se com uma proposta muito clara: "O MuGE proporciona um modelo de desenvolvimento RAD[2] e flexível, por intermédio da *DIGEO Language*, para novos desenvolvedores de jogos digitais". Apesar da asserção enfatizar o público de desenvolvedores iniciantes, o motor oferece ainda o acesso direto à programação de baixo nível, utilizando a biblioteca de código da mesma. Essa possibilidade é muito interessante numa abordagem mais mercadológica, quando se necessitam criar jogos profissionais e se possui pouco tempo de desenvolvimento. Nesse caso, utilizar-se-ia uma combinação da *DIGEO Language* para criar a estrutura básica do jogo — como se fosse um exoesqueleto — com a programação baixo nível feita em C++, lançando mão do uso do motor MuGE para que, assim, sejam criados efeitos e os comportamentos customizados (mais complexos) aos título a serem desenvolvidos.

A *engine* utiliza uma arquitetura computacional que tem como inspiração e base padrões de arquiteturas já propostas no meio [29; 3; 54]. Todas elas possuem por

[2]Modelo de processo de desenvolvimento de software interativo e incremental que enfatiza um ciclo de desenvolvimento extremamente curto

preemptivas de que todos os jogos lançam mão de um mesmo estilo arquitetural, sendo formadas assim, por componentes reutilizáveis. Com isso, criou-se uma estrutura composta por quatro partes principais: **o core**, ou núcleo, (seção 7.2.1), sendo a parte responsável por tratar de todas as tarefas básicas [84] de uma *game engine*; **o conjunto de elementos da DIGEO** (seção 7.2.2),sendo a implementação funcional das classes existentes na ontologia; **os *plugins*** (seção 7.2.3) que são camadas de abstração que possibilita a utilização de bibliotecas externas a fim complementar abstrações (visualização, física, UI, etc) que não são implementadas pelo motor e, por ultimo, **o interpretador** XML da *DIGEO Language* (seção 7.2.4).

### 7.2.1 Núcleo

O core principal, ou *núcleo*, é o ponto de partida (entrada) para a aplicação (título a ser desenvolvido) que utiliza o MuGE. Isso acontece uma vez que o core é responsável pelo funcionamento básico de um jogo. O core manipula a estrutura de dados em forma de *nodos*[3], desempenha as manipulações de eventos, cuida do ciclo de atualização e, por fim, é responsável pelo funcionamento das classes da DIGEO. O mesmo é a entidade funcional mais importante, como se fosse o coração de um humano, isto é, o motor. Ele é referenciado por todas as outras partes da *game engine*. Este core possui a responsabilidade principal, a inicialização e gerenciamento de todos os recursos existente no motor.

Devido ao uso de uma estrutura que segue um padrão arquitetural diferenciado, utilizando uma ontologia própria, elimina-se a necessidade do usuário final em se preocupar com questões de estrutura do jogo: como controles de relacionamentos arquiteturais entre manipuladores de jogabilidade, entradas, laço de jogo (*game loop*), manipulação de níveis, entre outros.

### 7.2.2 Elementos

Elementos são entidades (objetos de classes especializadas) referente a códigos responsáveis pela descrição e relacionamento, de forma específica de elementos da DIGEO com inúmeras especializações. Os mesmos são descritos como elementos acopláveis, permitindo a criação de interdependências que caracterizam um jogo como instância. Como a própria ontologia já infere, os elementos são pedaços compreensíveis da existência dos jogos digitais. Observando-se o emprego do modelo sob o qual o MuGE é descrito, os elementos são objetos concretos da computação (interpretação) dos elementos (*tags*) da *DIGEO Language*.

### 7.2.3 Plugins

Um *plugin*, no MuGE, consiste em um componente capaz de fornecer a funcionalidade (implementação) de certo tipo de serviço pré-definido por meio de promessas na composição do motor. Por componente, entende-se uma biblioteca ou bloco de código que serve de middleware[4] entre o motor em si e a tecnologia escolhida pelo usuário (programador que faz uso do motor). A infra-estrutura do MuGE é projetada a fim de facilitar o desenvolvimento de aplicativos (jogos) em união com

[3]Nodos são estruturas hierarquicas de relacionamento dos objetos das classes do motor.

[4]Um *middleware* é um aplicativo ou implementação que conecta aplicações e/ou componentes de software. Essa tecnologia permite a interoperabilidade entre arquiteturas e tecnologias com características, nativamente, incompatíveis.

diversos *plugins* já desenvolvidos; tal modelo proporciona um ambiente de desenvolvimento transparente — sem a necessidade da *game engine* estar engessada a certas tecnologias que, por sua vez, não agradam a todos os desenvolvedores.

Faz-se necessário o uso de *plugins* quando se deseja fazer uso de componentes visuais, sonoros, inputs e física. Por padrão, o MuGE foi criado somente com um *plugin* externo que fornece a implementação de todos esses componentes. Tal *plugin* é o **IrrlichtPlugin**, uma implementação que utiliza a biblioteca Irrlicht [85] para fornecer todas os recursos básicos necessários. Caso seja de interesse, para o uso de alguma outra tecnologia deve-se construir plugins que acoplem e interajam com a estrutura de promessas de código fornecidos pelo motor. Tal abordagem proporciona, num futuro, a criação de repositórios de plugins que se voltem à distribuição de código, proporcionando maior agilidade e conforto à produção de jogos, visto que os usuários menos experiente poderiam desenvolver jogos lançando mão do uso de plugins que utilizem diversas tecnologias.

### 7.2.4 Interpretador

A definição de uma linguagem de marcação deve-se ao uso da *DIGEO Language* que, por sua vez, necessita de um interpretador para processar todas as definições representadas no arquivo em objetos das classes da MuGE, isto é, algum jogo cuja estrutura foi definida em um arquivo XML. O uso da linguagem proporciona um padrão de especificação de elementos de forma externa à programação de baixo nível[5] do título. Entretanto, a fim de proporcionar uma fidelidade de comportamento teórico, especificação sintática, e prático, o comportamento dos elementos presentes no título compilado, necessitou-se de um bom período de refinamento e testes após a programação do mesmo.

A principal desvantagem do interpretador é o tempo para efetuar o *parsing*[6] dos dados descritos pela linguagem, entretanto, diante das inúmeras vantagens já mencionadas e do fato de que o XML será analisado somente antes na iniciação do jogo não impactando no desempenho do jogo em si, tal desvantagem é minimizada.

Para implementar a funcionalidade no MuGE, utilizou-se a biblioteca *irrXML* [86], sendo uma tecnologia cross-plataforma e open-source. Esta biblioteca fornece serviços como efetuar leitura e escrita de arquivos e manipulação de elementos de forma a transcrever as especificações do XML a uma sintaxe compreensível em C++. O acesso aos dados é obtido via a utilização de funções auxiliares que atuam de forma a fornecer os valores através de um acesso semanticamente semelhante.

## 7.3 Aplicação

A partir da estrutura proposta pelo motor, existem três etapas (Figura 7.2), após gerada a idéia do título, sob um olhar analítico do uso da DIGEO. Na primeira etapa temos a preparação do ambiente de desenvolvimento em que se criam os códigos-base para carregar a especificação de jogo — conforme o uso da biblioteca MuGE, bem como são acoplados os plugins desejados na criação do título. Seqüencialmente à etapa anterior, temos a percepção/transcrição dos elementos presentes na idéia do jogo para um arquivo XML utilizando-se a *DIGEO Language*; ainda nessa etapa, junto a especificação cria-se o fluxo de comunicação dos elementos via a represen-

[5]Modo de programação mais próximo à uma linguagem de hardware.
[6]Processo de análise de um dado texto.

tação dessa linguagem. Por fim tem-se a etapa de customização, na qual se volta a focar o desenvolvimento em C++ em conjunto com o motor de jogo; esta etapa proporciona uma maior liberdade de customização e manipulação de recursos que não encontram-se presentes na *DIGEO Languege*.

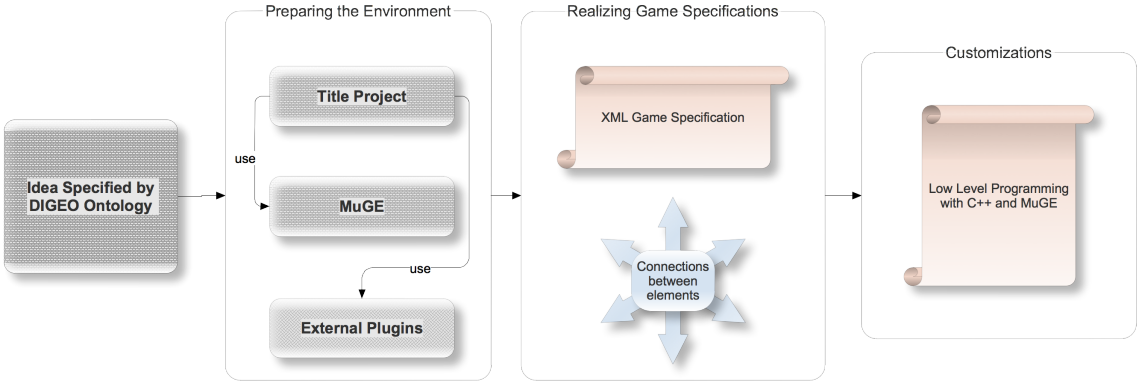


Figura 7.2: Fluxo de desenvolvimento de um título mediante a aplicação do modelo proposto.

A aplicação desse motor, num olhar singular sob a etapa de preparação do ambiente de desenvolvimento, é obtida a partir da criação de, pelo menos, três diferentes projetos, conforme pode ser verificado (Figura 7.3). O primeiro projeto, no exemplo chamado de *MuGE.framework*, é composto pelas classes e bibliotecas que contemplam as funcionalidades básicas de uma *game engine*. Após finalizado, o projeto gera uma biblioteca dinâmica em C++ (.DLL no Windows ou .Framework no Mac — como é o caso); e esta que, por sua vez, será utilizada em todos os projetos de jogos. Ainda neste projeto verifica-se a presença de inúmeras promessas de código, ou seja, são pedaços de código, classificados como plugins que possuem somente uma promessa de como eles irão funcionar. As implementações destes plugins, pedaços de código, são feitas em projetos externos; neste caso, o *IrrlichtPlugin.framework* desempenha o papel de criar o conteúdo (código C++) das promessas do projeto referenciado (*MuGE.framework*). Isso acontece, pois somente as promessas cujos códigos foram implementados poderão ser utilizados no projeto final (projeto do jogo). Por fim, temos o projeto do título em si; o projeto na figura enunciado *como GamePrototype.app* é um projeto C++ que inclui a biblioteca *MuGE.framework* para poder começar a desenvolver um título e a *IrrlichtPlugin.framework* para que seja possível utilizar as promessas implementadas pela mesma.

### 7.3.1 Peculiaridades do Desenvolvimento

Para o desenvolvimento de tal implementação optou-se pelo desenvolvimento em ambiente Mac OS X [87] utilizando a ferramenta XCode [88]. Sendo, o XCode, a principal ferramenta de desenvolvimento da Apple, funcionando apenas em ambientes Mac OS X, ela possui todos os *softwares* necessários para a criação, debug e otimização de aplicativos. De posse de uma integração entre um editor de códigos robusto e o compilador GCC [89], cria-se o ambiente perfeito para codificação de aplicativos C++. Além do ambiente de desenvolvimento, o fato da implementação da MuGE ser feita em C++ deriva da vontade de se desenvolver um motor que utilize a mesma linguagem, de alto desempenho, cujas grandes empresas de jogos utilizam para desenvolver seus títulos.

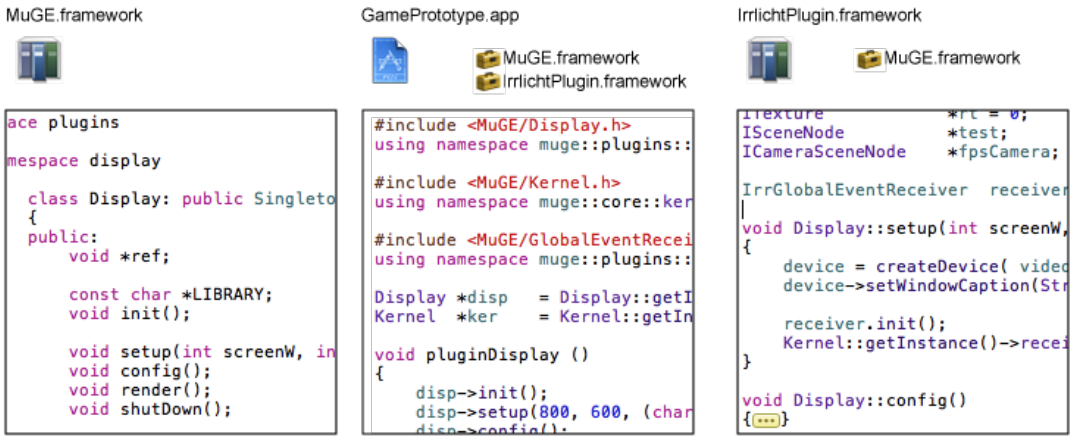


Figura 7.3: Meio a esse esboço evidencia-se a interdependência entre os códigos sendo que o MuGE não requer a instalação prévia de nenhuma tecnologia — esse papel é desempenhado pelos plugins.

A flexibilidade do código torna-se possível a partir da distribuição da bibliotecas — próprio motor e das implementações de plugins — sendo feitas mediante a distribuição por *frameworks* que são bibliotecas dinâmicas para plataformas Mac, similares às DLL's do Windows e constituídas de um código binário, resultado da compilação e de um conjunto de premissas (arquivos C++ com extensão ".h") num só arquivo fechado para *linkagem* (junção) e uso em diversos projetos. Além disso, o uso de XML define a linguagem proposta pela *DIGEO Language*, permitindo uma maior flexibilidade de desenvolvimento aliada a uma estrutura textual e estruturalmente compreensível.

## 7.4 Conclusão

O uso desse modelo propõe, necessariamente, um padrão de desenvolvimento que necessariamente deverá ser seguido, minimizando assim as chances de criação de jogos com estruturas deficientes. Tal fluxo ocorre devido à ontologia proposta na arquitetura do motor. Dessa forma, o modelo tem por objetivo minimizar os casos de fuga ao padrão proposto, o que poderia acarretar erros na hora do desenvolvimento de um título.

Uma implementação que atua sob um modelo de trabalho diferenciado e utilizando uma linguagem de *alto nível* (de mais fácil compreensão), bem como possibilitando também o acesso a camadas de baixo nível, evitando restrições de desenvolvimento, torna-se um diferencial quando se analisa os ciclos de desenvolvimento que são utilizados por empresas de desenvolvimento de jogos digitais. É por tal motivo que o MuGE é um motor com código proprietário, visando à utilização do mesmo como base para que haja o desenvolvimento de um plano de negócios — mercadologicamente útil à implementação.

Para que não se prejudique o domínio de conhecimento, bem como os estudos relacionados, a ontologia e a linguagem relacionada estão acessíveis publicamente soba licença MIT [90] que permite a liberdade para alterar e utilizar de forma não restritiva — o software e seu código, porém, a licença possui um único termo: "O aviso de copyright acima e esta permissão deverá ser incluído em todas as cópias ou

partes substanciais do Software.”, fornecendo assim liberdade de uso e segurança de que toda e qualquer produção que utilize tanto a *DIGEO* quanto a *DIGEO Language* serão feitas também sob a licença MIT.

Por fim, é importante ressaltar que esta implementação — MuGE — possui algumas restrições quanto ao *parse* de XML, que descrevem jogos com uma linguagem proposta. Isso acontece uma vez que o MuGE ainda não possui suporte a todos as especificações descritas pela *DIGEO Language*.

# 8 PONG: UMA APLICAÇÃO DA DIGEO E MUGE

Com o intento de exemplificar de forma prática e concreta, no uso da DIGEO e da MuGE apresenta-se o estudo de caso do *remake* (reconstrução) do jogo *Pong* [91], bem como os passos necessários para o desenvolvimento completo do título. Tal estudo de caso possui como finalidade testar a aplicabilidade da DIGEO, bem como as funcionalidades da MuGE. Este exemplo foi construído utilizando o *plugin* padrão (o *PluginIrrlicht*) do motor.

A fim de criar um jogo que fosse definido quase que completamente pela *DIGEO Language* escolheu-se o *Pong*, por ser um título sem muitas complexidades (necessidades de customizações extras) de domínio, ou seja, sem muitos conjuntos lógicos sofisticados. Não seria adequada, no momento, uma abordagem de um jogo mais complexo por duas razões: a implementação do MuGE não possui total suporte a todos os elementos da *DIGEO Language* e devido ao objetivo deste trabalho não focar o desenvolvimento de nenhum jogo digital mas sim criar a ontologia e provar sua aplicabilidade.

## 8.1 O título: Pong

A fim de ilustrar a criação de um jogo digital de conhecimento geral esta análise abordará o desenvolvimento do título *Pong DIGEO*, como sendo um *remake* do clássico *Pong* que tem por finalidade comprovar o uso e a eficiência do MuGE — motor desenvolvido.

Sendo um *remake* do jogo *Pong*, é necessário ambientar o jogo em que o mesmo se baseia. Pong (Figura 8.1) é um dos primeiros jogos digitais para plataforma Arcade o qual possui gráficos 2D simples e simula um jogo de tênis de mesa. O objetivo é derrotar o oponente por obter a maior pontuação. Este jogo digital foi pela Atari Incorporated [92] e lançado em 1972. O jogo inicia com dois jogadores e cada jogador representa um "retângulo"; os mesmos terão suas movimentações limitadas em uma só direção (imaginando que estarão dispostos sob uma superfície plana e a perspectiva de visualização do usuário será *top-view* (visão superior ou também conhecida como visão de helicóptero)).

Assim como o título clássico, o objetivo principal do *Pong DIGEO* é simular uma partida de tênis de mesa — com objetos geométricos retangulares representando as raquetes, a mesa e os limites do cenário. Deve-se tomar por evidência que a bola iniciará no centro do cenário com um vetor de lançamento (força e direção) pré-determinado. Ainda, ela provoca colisão com as paredes laterais, ou seja, os únicos locais de "fuga" do cenário são as arestas na posição relativa às costas de cada jogador. Por fim, o *gameplay* baseia-se na interatividade de um jogo de tênis entre a

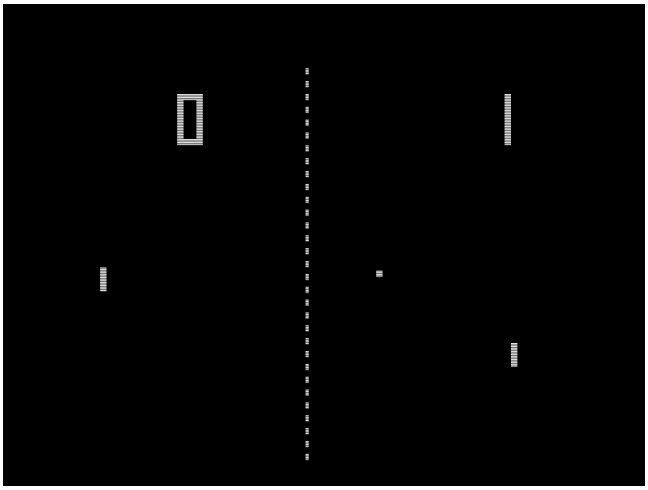

Figura 8.1: Captura de tela referente ao título Pong clássico.

bola e as raquetes. No caso de um jogador não conseguir rebater a bola e a mesma passar a sua posição, o jogador adversário marca um ponto a seu favor.

## 8.2 Descrição Ontológica

Toda e qualquer descrição ontológica de um jogo digital tem como fundamento a aplicação de forma lógica da DIGEO. Isso acontece pela listagem de todos as instâncias de classes presentes na ontologia, bem como seus relacionamentos e valores de propriedades. Além da aplicação das definições necessita-se da abstração das instâncias mediante a uma estrutura hierárquica. Para melhor visualizar a descrição ontológica do título, criou-se uma representação visual (Figura 8.2) frente ao uso de um diagrama *mind map*[1] cuja idéia central é o título em si e as palavras conectadas representam instâncias de classes da ontologia.

Por intermédio da DIGEO, descreve-se o título *Pong DIGEO* como um jogo *multiplayer* (multi-jogador) no qual dois jogadores controlam através do mesmo teclado as raquetes do jogo. O ambiente do jogo é limitado por uma região que representa uma quadra de tênis. O modo de visualização é através de uma câmera fixa posicionada na parte superior do ambiente, tendo seu foco na parte central do campo e proporcionando uma visão superior chamada *top-view*. A jogabilidade (*gameplay*) do jogo é definida pela ação das duas raquetes em rebater a bola, sendo que quando um dos dois jogadores não conseguir rebater a bola é computado um ponto ao oponente. Quando um dos jogadores fizer mais do que nove pontos determina-se que este jogador é o campeão. A partir da análise de jogabilidade, explorada anteriormente, evidencia-se a existência dos elementos ontológicos conforme apresentados na tabela 8.1.

[1]Mind Map é um estilo de diagrama que serve para representar palavras, idéias, tarefas e outros dados de forma a possuirem interconectividades. Toda diagramação é feita a cercando uma palavra ou idéia principal.

Tabela 8.1: Representação descritiva dos elementos presentes a descrição do título proposto.

| Nº de Instâncias | Descrição | Relacionamentos |
|---|---|---|
| 1 | elemento Digital Game representando o título. | Este elemento é o ponto de partida para interpretação do jogo |
| 2 | User representando os jogadore do jogo. | Se relacionam com dois objetos (a fim de representar os mesmos) por meio do input de teclado |
| 2 | Key com 1 Move cada um, para quando a tecla pressionada movimentar a raquete. | Quando pressionada a tecla que possui o mesmo código, o objeto relativo será movimentado |
| 2 | Dynamic Object representando as raquetes. | Possuem colisão com os limites da sala, sendo que se encostarem em uma extremidade serão forçadas a voltar para o perímetro da sala |
| 1 | Boundary limitando a quadra de jogo. | Descreve os limites do level ao qual ele pertence |
| 1 | Screen sendo a tela de jogo que conterá a representação do jogo em si. | É composta por outros elementos que a caracterizam de forma visual |
| 1 | Camera para visualizar os elementos do jogo. | Inferem na maneira com que o jogo será visualizado, dependendo de angulação da câmera, abertura de lente, movimetação, etc |
| 1 | Viewport que serve de suporte a câmera. | Faz com que a câmera se comporte de forma a obedecer as definições propostas pelo mesmo |
| 2 | Hit dos jogadores com o limite do cenário, não deixando que eles passem para fora do campo. | Os mesmos acionam um movimento inverso ao atual. |
| 1 | Hit da bola com os jogadores. | Isso faz com que cada vez que a bola tocar em alguma raquete, ela inverta seu movimento |
| 2 | Static Object atrás das raquetes representando o sensor do gol. | Cada objeto possui um Hit para detectar colisão e avisa o jogo meio a uma ação de Event Dispatcher |
| 1 | Listener que recebe as ações do Event Dispatcher. | Interpreta o evento recebido e atua de forma a trocar os estados dos seus elementos filhos |
| 1 | Change Screen que só acontece quando a propriedade filterExpression do Lister for verdadeira. | Faz com que a atual tela seja retirada e substituída pela requerente |
| 1 | Screen de fim de jogo. | Que quando visível para a receber todos os inputs do level |

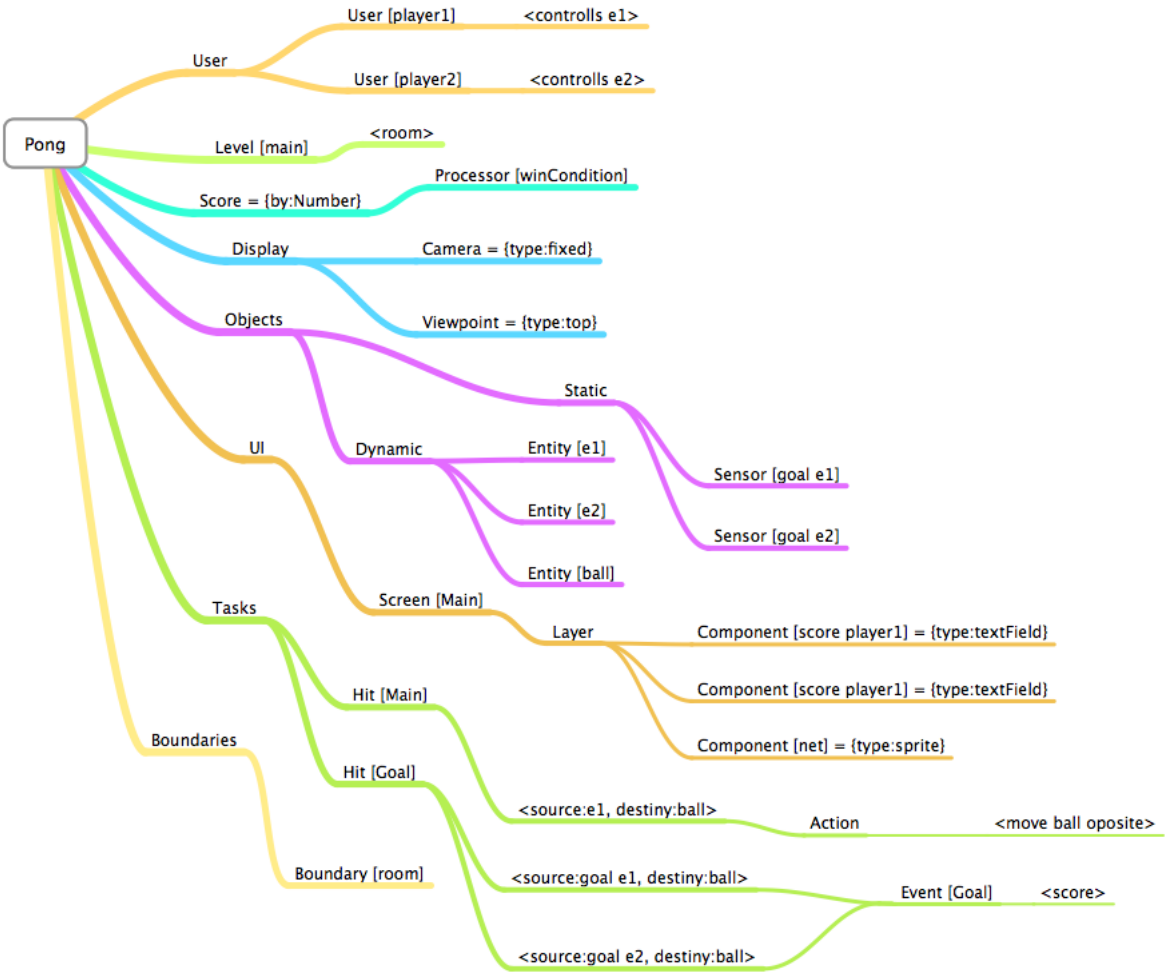


Figura 8.2: Diagrama *mind-map* que representa a estrutura de elementos ontológicos do título a ser produzido.

## 8.3 Implementação

Antes de definir por completo o jogo, é necessária a criação do projeto do título como um projeto C++ com as características já mencionadas no início do capítulo. Com isso, a fase de implementação inicia-se com a criação de um projeto de código C++ na IDE de preferência; nesse caso é utilizando o *XCode* cria-se uma *Cocoa Application* (Aplicativo Mac). Após as *frameworks* terem sido instaladas no sistema[2], elas devem ser referenciadas no projeto, conforme mostra a figura 8.3.

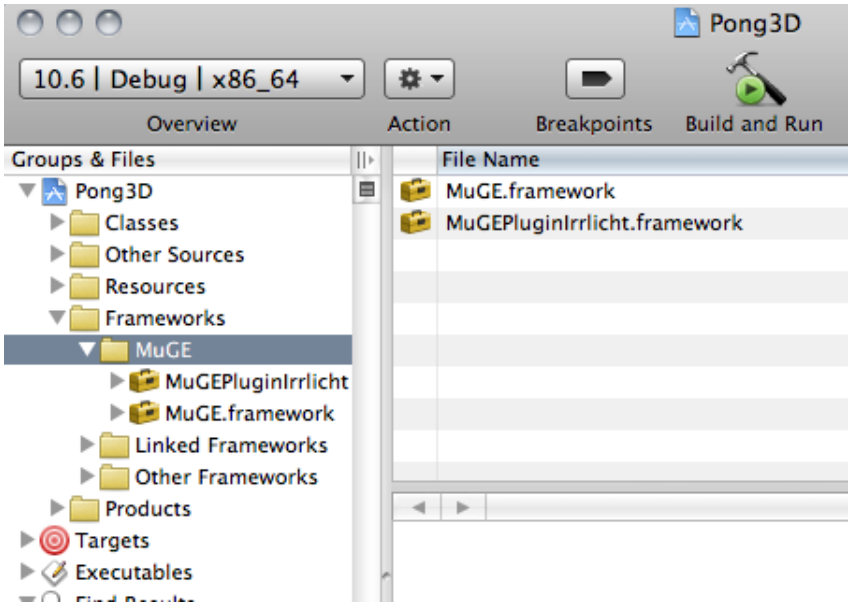


Figura 8.3: Uso prático do MuGE e de seu plugin padrão num projeto do XCode.

Uma vez com o ambiente configurado, inicia-se a programação necessária para fazer o motor, junto do plugin, funcionar a fim de produzir uma aplicação básica para a interpretação e execução do jogo descrito via *DIGEO Language*. A partir desta etapa, são necessárias poucas linhas de códigos para a obtenção do título. Conforme evidencia o pseudocódigo 8.1, para o desenvolvimento de um projeto, necessita-se somente inicializar o display (linhas 19, 20 e 21) e carregar o título em XML (linha 23).

Listing 8.1: A sample

```
1  #include <MuGE/Display.h>
2  #include <MuGE/Kernel.h>
3
4  using namespace muge::plugins::display;
5  using namespace muge::core::kernel;
6
7  Display *disp   = Display::getInstance();
8  Kernel  *ker    = Kernel::getInstance();
9
10 void display(float interpolation)
11 {
12     disp->render();
13 }
14
```

[2]No caso do MacOSX as *frameworks* devem ser instaladas no diretório `/Library/Frameworks`.

```
15 void update() {}
16
17 int main(int argc, char *argv[])
18 {
19     disp->init();
20     disp->setup(800, 600, (char *)"Pong DIGEO by Filipe Silvestrim - MuGE Test");
21     disp->config();
22
23     ker->loadGame("./pong.xml")
24     ker->gameloop(update, display);
25     disp->shutDown();
26     delete disp;
27
28     return 0;
29 }
```

## 8.4 Criando a especifica XML via *DIGEO Language*

A fim de criar um documento em formato XML que descreva o título serão detalhados (atendendo as especificações da *DIGEO Language*) os elementos e relação ontológicas presentes no *Pong DIGEO*. Para formalizar essas idéias será descrito o título atendendo a uma seqüência estrutural como a que segue:

1. Em primeira instância necessita-se, bem como faz-se explícito à base hierárquica da ontologia, a especificação do título de jogo digital.

Listing 8.2: Código referente ao jogo em sí

```
1 <game id="pongDigeo" name="Pong DIGEO" />
```

2. Após definida a instância do jogo, observa-se a necessidade de uma descrição visual dos recursos que representam os objetos (raquetes e bola) do jogo. O MuGE possui uma propriedade especial na falta de recursos profissionais; nesse caso pode-se utilizar a *tag* `ressource` com o valor de `type` como sendo "proto" — representação visual de um polígono.

Listing 8.3: Carregamento de arquivos de recursos

```
1 <load>
2     <ressource id="dispBigBox" type="proto" input="box, 20"/>
3     <ressource id="dispLittleBox" type="proto" input="box, 10"/>
4 </load>
```

3. Antes de ser inicializada a execução de todos os elementos representados no código, evidencia-se prioridade quanto as configurações do jogo (o alicerce do título).

Listing 8.4: Código de configuração inicial do jogo

```
1 <config>
2     <display id="mainDisplay">
3         <camera id="mainCamera" type="fixed" position="50,50,100↩
              " target="50,50,0"/>
```

```
4         <viewpoint id="mainView" type="top" input="#mainCamera"/↩
              >
5     </display>
6
7     <userInterface>
8         <screen id="mainScreen">
9             <component id="scoreP1" type="textField" input="#↩
                  scoreP1.value" position="130,100,0"/>
10            <component id="scoreP2" type="textField" input="#↩
                  scoreP2.value" position="660,100,0"/>
11        </screen>
12        <screen id="endScreen">
13            <component type="textField" input="Player X Wins" ↩
                  position="300,300,0"/>
14        </screen>
15    </userInterface>
16 </config>
```

4. Mesmo sem a evidência sob a presença de leveis, determina-se que todo e qualquer jogo tenha, pelo menos um level principal. Conforme representado na listagem 8.5, o título *Pong DIGEO* possui apenas o level principal, sendo que sua visualização é obtida mediante a sua existência `input` na tela principal e descrita visualmente por intermédio do `display`.

Listing 8.5: Descrição do level principal

```
1 <level id="mainLevel" input="#mainScreen, #mainDisplay" />
```

5. A necessidade de se priorizar a *tag* de configuração não é mérito somente da *tag* raiz (do jogo em si), mas sim, é acessível a qualquer outro elemento (*tag*). Neste caso, defini-se como elementos de configuração do level todos aqueles presentes nesta e na próxima etapa. Inicialmente deve-se limitar a área de ação dos elementos do jogo quanto ao espaço geométrico do mesmo (definindo-se o `bound`). Ainda nesta etapa são definidas as regras principais do jogo.

Listing 8.6: Explicitação das regras e do limite do jogo

```
1 <bound id="room" from="0,0,0" to="100,100,0" collision="true"/>
2
3 <group id="rules">
4     <score id="scoreP1" type="int" input="0"/>
5     <score id="scoreP2" type="int" input="0"/>
6
7     <listener id="winCondition" input="#collisionGoal1, #↩
          collisionGoal2" filterExpression="#scoreP1.value > 9 || #↩
          scoreP2.value > 9">
8         <changeUserInterface input="#endScreen" />
9     </listener>
10 </group>
```

6. Nesta etapa definem-se os elementos referente aos usuários, sendo que os mesmos irão controlar entidades presentes no jogo atendendo ao uso do teclado, no caso, o *player 1* é responsável pelas teclas W e S do teclado e o *player 2* pelas é responsável pelas teclas UP e DOWN. Estas interatividades resultam na ação de mover as entidades alvo da ação.

Listing 8.7: Definindo os usuários do jogo

```
1 <group id="users">
2     <user id="player1" input="#entityP1">
3         <input id="inptP1">
4             <key id="W" code="17"> <move target="#entityP1" axis↩
                ="y" ammount="1"/> </key>
5             <key id="S" code="31"> <move target="#entityP1" axis↩
                ="y" ammount="-1"/> </key>
6         </input>
7     </user>
8     <user id="player2" input="#entityP2">
9         <input id="inptP2">
10            <key id="UP" code="103" > <move target="#entityP2" ↩
                axis="y" ammount="1"/> </key>
11            <key id="DOWN" code="108" > <move target="#entityP2"↩
                 axis="y" ammount="1"/> </key>
12        </input>
13    </user>
14 </group>
```

7. Como última etapa, devem ser definidos os elementos visuais e interativos do título — responsáveis pela interatividade e pelas ações que representam a jogabilidade do mesmo.

Listing 8.8: Definindo os elementos de *gameplay*

```
1 <group id="elements">
2     <staticObject id="sensorGoalP1" position="0,0,0" size="↩
        10,10,1">
3         <hit id="hitGoal1" input="#entityBall">
4             <changeValue id="scorePLayer2" input="#scoreP1" ↩
                property="value" type="add" value="1" />
5             <eventDispatcher id="collisionGoal1" />
6         </hit>
7     </staticObject>
8     <staticObject id="sensorGoalP2" position="0,0,0" size="↩
        10,10,1">
9         <hit id="hitGoal2" input="#entityBall">
10            <changeValue id="scorePLayer2" input="#scoreP1" ↩
                property="value" type="add" value="1"/>
11            <eventDispatcher id="collisionGoal1" />
12        </hit>
13    </staticObject>
14
15    <entity id="entityP1" display="#dispBigBox" />
16    <entity id="entityP2" display="#dispBigBox" />
17    <entity id="entityBall" display="#dispLittleBox">
```

```
18          <hit id="hitBallPlayers" input="#entityP1, #entityP2, #↩
                room">
19              <move invert="true" />
20          </hit>
21      </entity>
22 </group>
```

Fazendo uso dessa descrição mais tecnológica constatou-se a formalização do título proposto por meio da *DIGEO Language*. Por fim, para não poluir de forma visual este capítulo, o arquivo com a descrição, completa e linear, do título *Pong DIGEO* está disponível como anexo na seção A.2.

## 8.5 Resultados obtidos

Conforme evidenciado ao decorrer deste capítulo, registrou-se a existência de diversos elementos ontológicos capazes de definir o título *Pong DIGEO*. Como resultado positivo na implementação do modelo de desenvolvimento que abrange a *DIGEO Language* e o motor MuGE, possibilitou-se a interpretação da definição XML do *Pong DIGEO* (anexo A.2) como forma de validar e registrar o desenvolvimento do título de jogo proposto de forma que o mesmo atende a todas as especificações descritas no arquivo XML. Assim sendo, obteve-se, como resultado dessa implementação, o título de jogo digital *Pong DIGEO* é uma instância de jogo funcional e interativa, conforme pode ser evidenciado na figura 8.4. A fim de validar a interpretação dos elementos ontológicos destaca-se, na figura 8.4, a existência de seis elementos visuais: *1*) **component**, representação visual do tipo textField representando o valor do score atual do player 1; *2*) **dynamic entity**, sendo o objeto responsável com o qual o usuário interagir para rebater a bola; *3*) **bound**, é o limite do jogo, ou seja, os pontos cartesianos que limitam os elementos presentes ao título; *4*) **component**, representação visual do tipo textField representando o valor do score atual do player 2; *5*) **dynamic entity**, sendo o objeto responsável com o qual o oponente irá interagir a fim de rebater a bola e pontuar sob o outro usuário; *6*) **dynamic entity**, sendo a "bola" do jogo, ou elemento responsável por atuar como sendo interativo, sofrendo todas as ações do sistema de jogo e dos outros elementos. Visto isso, comprova-se a existência de alguns (somente os visuais) dos elementos ontológicos de forma fiel às suas descrições no arquivo XML.

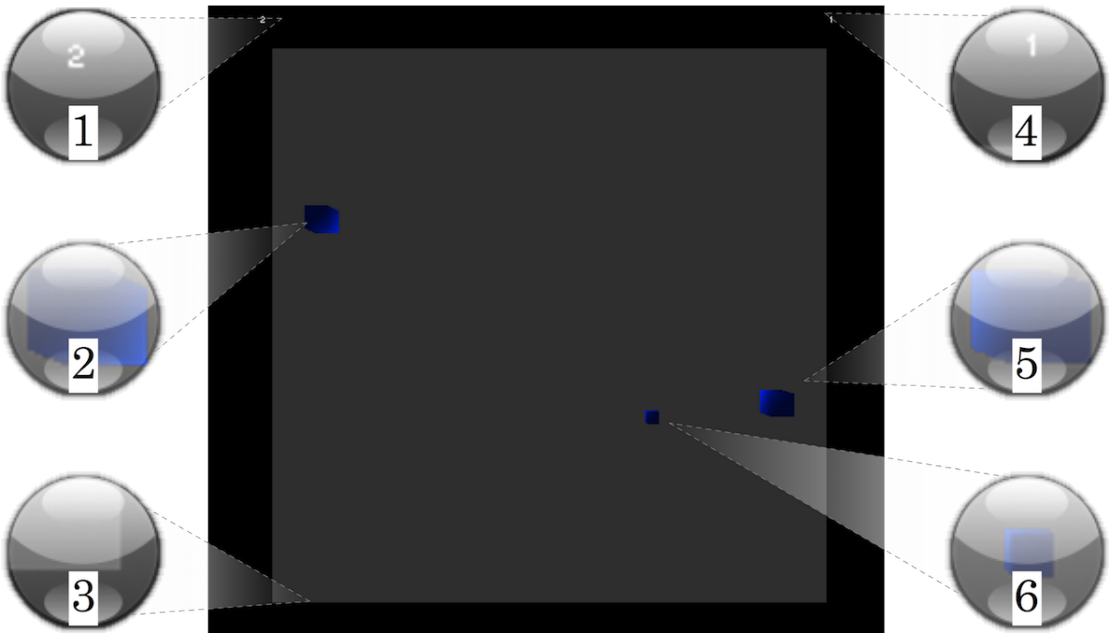


Figura 8.4: Título do jogo digital obtido mediante a interpretação da definição em XML do *Pong DIGEO*.

# 9 CONSIDERAÇÕES FINAIS

Ao término deste trabalho, destaca-se a criação da primeira versão da DIGEO como uma ontologia de jogos digitais. O trabalho principal (DIGEO e *DIGEO Language*) torna-se público, sob a licença MIT (seção 7.4) e tem o acesso público da versão mais recente via sua URL[1] de publicação. Dedicaram-se diversas considerações a respeito das ontologias de jogos digitais já existentes e comprovou-se de forma efetiva a implementação do *Pong DIGEO*, a diferença da DIGEO quanto aos diversos trabalhos relacionados, isso acontece num momento em que se propõem a descrição de jogos digitais a partir de um ponto de vista diferenciado — o desenvolvimento — e com uma implementação formal e válida.

Algumas considerações sobre a concepção do que são *elementos* registraram-se ao longo do volume para que houvesse uma perfeita assimilação do significado do termo. O desenvolvimento do modelo associado à implementação do MuGE faz-se presente de forma a servir de base para implementar apenas os requisitos necessários à existência do protótipo do jogo *Pong DIGEO* proposto. Isso acontece uma vez que o objetivo principal da monografia é a criação da ontologia; ainda é importante ressaltar a importância da liberação do conhecimento, diga-se *DIGEO* e *DIGEO Language* diante do motor que é uma solução proprietária, uma vez que o propósito futuro de seu uso é a aplicação do mesmo sob um modelo de negócios visando atacar uma fatia de mercado ainda em ascensão: desenvolvimento de jogos profissionais via RAD.

Para a efetivação da pesquisa, foi necessária uma profunda revisão e definição acerca da DIGEO, bem como o estudo de caso de sua aplicação. Assim, conclui-se que a criação de uma ontologia que descreva jogos digitais sob uma abordagem diferenciada teve sucesso quanto ao seus propósitos: criando uma nova forma de ensino-aprendizagem num contexto de educação digital e ainda proporcionando a descrição e o desenvolvimento de jogos mediante o uso de um conhecimento específico que, ao mesmo tempo, generaliza, formaliza e solidifica a base e as combinações das definições deste conhecimento. Então, com os resultados obtidos com a *DIGEO* e o *MuGE*, demonstra-se a possibilidade do desenvolvimento de uma implementação/estrutura prática que interprete essa ontologia e produza títulos amadores e até profissionais.

Visto todas as discussões referente à *DIGEO*, à *DIGEO Language* e ao *MuGE*, entende-se a aplicabilidade da ontologia e da linguagem resultando na criação de um modelo de desenvolvimento com um bom nível de abstração que simplifica o processo de prototipagem, criação e desenvolvimento de jogos digitais, modelo que

[1] http://www.filipesilvestrim.com/digeo/

é explicitado no capítulo 7 e comprovado mediante a implementação do jogo *Pong DIGEO* (capítulo 8).

Em relação aos objetivos do trabalho, estes foram plenamente atendidos, pois a DIGEO concretiza a proposta ao criar uma ontologia de jogos digitais que impõe uma abordagem informal (com uma base teórica) e aplicável ao mundo real, proporcionando a descrição de títulos existentes e a implementação/criação de novos jogos digitais; a estruturação e o desenvolvimento de um jogo por intermédio do uso da *DIGEO Language* e do motor *MuGE*. A par disso, pode-se inferir que o desenvolvimento da ontologia foi atendido de acordo com as expectativas iniciais do formando.

A implementação da linguagem, especificada na seção 7.1, foi implementada a fim de suprir as necessidades de transcrever a ontologia a uma linguagem que fosse computacionalmente interpretável. Referente às limitações do MuGE, destaca-se o suporte, na totalidade voltado aos elementos da ontologia proposta. Isto acontece uma vez que essa implementação visa, de forma tópica a esta monografia, a validação do uso do conhecimento formal proposto; porém destaca-se o interesse futuro em continuar a implementação em sua totalidade a fim de criar uma aplicação com fins mercadológicos.

Uma vez que a ontologia e a linguagem em XML possuem a licença de distribuição MIT, essas, bem como foi o caso do MuGE, proporcionam o desenvolvimento de motores de jogos e interpretadores que utilizem as mesmas. Tal abordagem serve de livre acesso e uso do conhecimento de forma a serem interpretados por aplicativos *open source* (com código livre) ou proprietários. É por esse motivo que se proporcionou a criação do MuGE com uma licença proprietária, objetivando a criação de um modelo de trabalho visando à alta competitividade, na atual conjuntura, de micro e pequenas empresas de jogos.

Ainda, quanto a *DIGEO Language*, evidencia-se como trabalhos futuros proporcionar um suporte oficial, tornando-a uma linguagem baseada em XML descrita através de especificações e de um quadro diretor que valide a aplicação e as tendências dos métodos de desenvolvimento de jogos e da DIGEO. Também neste escopo, nota-se que a configuração do descritivo do jogo deve ser feita manualmente, ou seja, alterando diretamente o arquivo XML. Para usuários com conhecimento de programação a esse nível, isto não seria problema, porém, a abordagem torna-se um problema quando considerada a possibilidade de designers ou usuários iniciantes estruturarem/configurarem seus próprios jogos. Visto isso, é aconselhável a criação futura de um editor gráfico que auxilie tal processo de criação e configuração do arquivo. Feito isso, a utilidade é maior para desvincular a responsabilidade de criação do arquivo a um desenvolvedor. Tal abordagem é interessante no âmbito de abranger a pessoas não familiarizadas com o desenvolvimento de jogos digitais num ambiente menos teórico ou ilegível (eliminando a necessidade de conhecimentos de programação).

Essa pesquisa não se esgota aqui, pelo contrário, pretende-se dar continuidade nos estudos e em novas versões da DIGEO visando a uma abordagem mais genérica e mais acessível, bem como a introdução de novos elementos, se necessário. Pretende-se ainda, continuar as pesquisas quanto à DIGEO e aos "elementos" presentes nos jogos digitais com vistas à publicação de artigos em revistas.

Por fim, a utilização da referida ontologia começa a ter certa visibilidade, quanto ao público dos desenvolvedores e *game designers* uma vez que surjam novos exemplos

de títulos descritos, bem como a criação de interpretadores gratuitos. Para contemplar tal desejo, percebe-se a necessidade de, quando lançado o MuGE ao mercado, a existência de duas distribuições: uma Indie sendo gratuita a desenvolvedores de jogos independentes e a outra paga quanto ao seu uso em empresas.

# A ANEXOS

Este anexo tem como propósito complementar as definições da DIGEO, bem como suas aplicações, mencionadas ao longo do texto.

## A.1 Repostas das Questões de Competência da DIGEO

Bem como o título já menciona, esta seção tem como objetivo responder de forma clara e objetiva as perguntas fomentadas para a criação da DIGEO.

**Qual é a composição de um jogo?** Um jogo digital é composto por vários elementos (sendo pedaços, compreensíveis, que caracterizam um jogo) e recursos externos, como modelos 3D, som, efeitos especiais, etc.

**Como um jogo é desenvolvido?** Um jogo é desenvolvido meio ao processo de, num contexto, adicionar elementos da DIGEO e criar uma comunicação entre os mesmos, ou seja, fazê-los interagirem entre si.

**Quais as relações entre os elementos dentro de um jogo?** Os mesmos possibilitam o reconhecimento da composição de jogos digitais de forma global sob um olhar de desenvolvimento.

**Quais são definições de elementos e instâncias dos mesmos?** Um elemento é uma classe, ou seja, uma estrutura conceitual que representa uma instância — elemento concreto de um jogo.

**O que leva esta especificação a não ser apenas uma taxonomia?** Além de uma taxonomia, esta ontologia é composta de especificações, definições, de um conceito — elementos que compõem os jogos digitais sob uma perspectiva de desenvolvimento, bem como suas relações entre as classes da ontologia.

**O que leva esta especificação a não ser apenas um agrupamento de definições?** Bem como mencionado acima, esses agrupamentos de classe são dispostos em forma hierárquica, com definições de propriedades e restrições de valores.

**Quais os relacionamentos existentes?** Além do relacionamento hierárquico (como no caso, uma classe ser filha de outra, e por sua vez herdar suas definições) existem relações entre classes (por exemplo, um jogo possui vários elementos).

**Qual a diferença entre um elemento e um componente?** Elementos e Componentes é uma terminologia com significados semelhantes, porém diferenciam

em quesitos de aplicabilidade; no caso elemento é uma representação conceitual, já um componente é uma representação concreta do conceito "elemento", ou seja, a implementação de um elemento, seja ela via programação, ou até mesmo visual, resulta em um componente.

**Qual será a cobertura do domínio?** O domínio da DIGEO limita-se a descrição e criação de jogos digitais através do conceito de elementos. Não pertence ao domínio a descrição de um jogo meio a outros pontos de vista, bem como a descrição de jogos com o uso de conceitos que excedam as definições propostas pela ontologia.

**Por que utilizar a ontologia?** Para facilitar a criação e o entendimento sobre o desenvolvimento de jogos digitais e também como sendo uma linguagem padrão de *gamedesign* para o desenvolvimento de jogos. Além do uso teórico infere-se o uso da ontologia a um modelo prático e mercadológico, proporcionando um ciclo curto de desenvolvimento.

**Qual a aplicabilidade da ontologia?** Descrever jogos digitais, sob o ponto de vista de desenvolvimento ou criar novos jogos através do conceito de elementos.

**Para que tipo de perguntas a ontologia deve providenciar as respostas?** "Quais os elementos presentes neste jogo digital?", "Quais as relações existentes entre os elementos do jogo?", "De que esse jogo é composto?", "Este jogo é muito complexo, ou seja, ele possui muitos elementos e muitas conexões entre os mesmos?", "Existe um dicionário de palavras e definições sobre o desenvolvimento de jogos digitais?", "Qual a nomenclatura existente no desenvolvimento de jogos?", "Um leigo consegue desenvolver um jogo através do modelo teórico da ontologia?".

## A.2 Estrutura XML do *Pong DIGEO*

A fim de visualizar de forma concisa a aplicação da *DIGEO Language* para a descrição do título chamando de *Pong DIGEO* evidencia-se o código da listagem abaixo.

Listing A.1: Código XML do *Pong DIGEO* mediante o uso da *DIGEO Language*

```
1  <?xml version="1.0"?>
2
3  <game id="pongDigeo" name="Pong DIGEO">
4      <load>
5          <ressource id="dispBigBox" type="proto" input="box, 20"/>
6          <ressource id="dispLittleBox" type="proto" input="box, 10"/>
7      </load>
8
9      <config>
10         <display id="mainDisplay">
11             <camera id="mainCamera" type="fixed" position="50,50,100" target="50,50,0"↩
                   />
12             <viewpoint id="mainView" type="top" input="#mainCamera"/>
13         </display>
14
15         <userInterface>
16             <screen id="mainScreen">
17                 <component id="scoreP1" type="textField" input="#scoreP1.value" ↩
                       position="130,100,0"/>
```

```
18                <component id="scoreP2" type="textField" input="#scoreP2.value" ↩
                      position="660,100,0"/>
19            </screen>
20            <screen id="endScreen">
21                <component type="textField" input="Player X Wins" position="300,300,0"↩
                      />
22            </screen>
23        </userInterface>
24    </config>
25
26    <level id="mainLevel" input="#mainScreen, #mainDisplay">
27        <config>
28            <bound id="room" from="0,0,0" to="100,100,0" collision="true"/>
29
30            <group id="rules">
31                <score id="scoreP1" type="int" input="0"/>
32                <score id="scoreP2" type="int" input="0"/>
33
34                <listener id="winCondition" input="#collisionGoal1, #collisionGoal2" ↩
                      filterExpression="#scoreP1.value > 9 || #scoreP2.value > 9">
35                    <changeUserInterface input="#endScreen" />
36                </listener>
37            </group>
38
39            <group id="users">
40                <user id="player1" input="#entityP1">
41                    <input id="inptP1">
42                        <key id="W" code="17"> <move target="#entityP1" axis="y" ↩
                              ammount="1"/> </key>
43                        <key id="S" code="31"> <move target="#entityP1" axis="y" ↩
                              ammount="-1"/> </key>
44                    </input>
45                </user>
46                <user id="player2" input="#entityP2">
47                    <input id="inptP2">
48                        <key id="UP" code="103" > <move target="#entityP2" axis="y" ↩
                              ammount="1"/> </key>
49                        <key id="DOWN" code="108" > <move target="#entityP2" axis="y" ↩
                              ammount="1"/> </key>
50                    </input>
51                </user>
52            </group>
53        </config>
54
55        <group id="elements">
56            <staticObject id="sensorGoalP1" position="0,0,0" size="10,10,1">
57                <hit id="hitGoal1" input="#entityBall">
58                    <changeValue id="scorePLayer2" input="#scoreP1" property="value" ↩
                          type="add" value="1" />
59                    <eventDispatcher id="collisionGoal1" />
60                </hit>
61            </staticObject>
62            <staticObject id="sensorGoalP2" position="0,0,0" size="10,10,1">
63                <hit id="hitGoal2" input="#entityBall">
64                    <changeValue id="scorePLayer2" input="#scoreP1" property="value" ↩
                          type="add" value="1"/>
65                    <eventDispatcher id="collisionGoal1" />
66                </hit>
67            </staticObject>
68
69            <entity id="entityP1" display="#dispBigBox" />
70            <entity id="entityP2" display="#dispBigBox" />
71            <entity id="entityBall" display="#dispLittleBox">
72                <hit id="hitBallPlayers" input="#entityP1, #entityP2, #room">
73                    <move invert="true" />
74                </hit>
75            </entity>
76        </group>
77
78    </level>
79 </game>
```

# REFERÊNCIAS


[1] ALMEIDA, M. B.; BAX, M. P. Uma visão geral sobre ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção. *Ci*, v. 32, p. 7–20, 2003. 12, 21

[2] BATES, B. *Game Design ,Äì The Art & Business of Creating Games.* [S.l.]: Roseville: Prima Tech, 2001. 300 p. 15, 56

[3] LEWIS, M.; JACOBSON, J. Game Engines in Scientific Research. *Communications of the ACM*, v. 45, p. 27–31, 2002. 15, 71

[4] LANGUAGE, A. P. P. Why and for who did we write this pattern language. *Language*, p. 1–36, 2000. 15, 46

[5] LUNDGREN, S. Game Design Patterns. *Design.* 15, 46, 56

[6] TUROCY, T. L. Game Theory. *October*, p. 1–39, 2002. 18

[7] STUDER, R. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, v. 25, p. 161–197, March 1998. ISSN 0169023X. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0169023X97000566>. 18, 19, 20

[8] GASEVIC, D.; DJURIC, D.; DEVEDCZIC, V. *Model Driven Architecture and Ontology Development.* [S.l.]: Springer, 2006. ISBN 3-540-32180-2. 18, 22, 23, 55

[9] GENESERETH M. R.; NILSSON, N. J. *Logical Foundations of Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann Publishers, 1987. 18

[10] GRUBER, T. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, v. 43, p. 907–928, November 1995. ISSN 10715819. Disponível em: <http://linkinghub.elsevier.com/retrieve/doi/10.1006/ijhc.1995.1081>. 18, 19

[11] REZENDE, S. O. *Sistemas Inteligentes - Fundamentos e Aplicações.* [S.l.]: Manole, 2003. 550 p. 19

[12] WIKIPéDIA, A. E. L. *Ontologia.* Disponível em: <http://pt.wikipedia.org/wiki/Ontologia/>. 19

[13] GUARINO, N. Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human-Computer Studies*, v. 43, p. 625,Äì640, 1995. 19, 20

[14] BORST, W. N. PhD Thesis, *Construction of Engineering Ontologies.* 1997. 19

[15] MIZOGUCHI, R. Tutorial on ontological engineering. *New Generation Computing*, v. 21, p. 363–364, December 2003. ISSN 0288-3635. Disponível em: <http://www.springerlink.com/index/10.1007/BF03037310>. 19

[16] HENDLER, J. Agents and the Semantic Web. *IEEE Intelligent Systems*, v. 26, p. 30–37, 2001. 19

[17] KALFOGLOU, Y. *Exploring ontologies.* Singapore: [s.n.], 2001. 863,Äì887 p. 19

[18] AI-TR, T. R.; MIZOGUCHI, R.; RESEARCH, I. Towards Ontology Engineering. *Technical Report AI -TR-96-1*, p. 1–10, 1996. 19

[19] GRUBER, T. R. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, v. 5, p. 199–220, 1993. 20

[20] NOY, N. F.; MCGUINNESS, D. L. Ontology Development 101: A Guide to Creating Your First Ontology. *Development*, p. 1–25, 2000. 20, 22

[21] ANDAROODI, E. et al. Ontology-based shape-grammar schema for¬†classification of¬†caravanserais: a¬†specific corpus of¬†Iranian Safavid and¬†Ghajar open, on-route samples. *Journal of Cultural Heritage*, v. 7, p. 312–328, October 2006. ISSN 12962074. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S1296207406000793>. 20

[22] ALLEMANG, D.; HENDLER, J. *Semantic Web for the Working Ontologist.* [S.l.]: Denise E. M. Penrose, 2008. 23

[23] INFORMATICS, S. M. *PROTÉGÈ. The Protégé Ontology Editor and Knowledge Acquisition System.* Disponível em: <http://protege.stanford.edu/>. 23

[24] INFORMATICS, S. M. *Protégé-Frames.* 2009. Disponível em: <http://protege.stanford.edu/overview/protege-frames.html>. 23

[25] INFORMATICS, S. M. *Protégé-OWL.* 2009. Disponível em: <http://protege.stanford.edu/overview/protege-owl.html>. 23

[26] POSITIVO, G. *Dicionário Aurélio.* 2009. Disponível em: <http://www.dicionariodoaurelio.com/>. 25

[27] BEALE, T.; THOUGHT, D.; PTY, I. Archetypes: Constraint-based Domain Models for Future-proof Information Systems. *OOPSLA 2002 workshop on behavioural semantics.*, p. 1–18, 2002. 25

[28] WHITELAW, M.; GUGLIELMETTI, M.; INNOCENT, T. Strange ontologies in digital culture. *Computers in Entertainment*, v. 7, p. 1, February 2009. ISSN 15443574. Disponível em: <http://portal.acm.org/citation.cfm?doid=1486508.1486512>. 26

[29] GELATTI, G. A Framework for Building Engines for Games and Simulations. *Building*, p. 2–4. 26, 71

[30] ZAGAL, J. P. M. M. F.-V. C. H. B. L. N. *Towards an Ontological Language for Game Analysis.* Atlanta, GA: [s.n.], 2005. 26

[31] LING, Y. et al. A Fuzzy Ontology and Its Application to Computer Games. *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*, Ieee, v. 1, p. 442–446, 2007. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4406428>. 26

[32] CHAN, J. T. C.; YUEN, W. Y. F. Digital game ontology: Semantic web approach on enhancing game studies. *2008 9th International Conference on Computer-Aided Industrial Design and Conceptual Design*, Ieee, p. 425–429, November 2008. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4730603>. 26, 37

[33] M., R. Y. A. S. S. The Music Ontology. *International Conference on Music Information Retrieval*, ISMIR, 2007. 26

[34] GONZáLEZ, J. L. *Ontologia de Videojuegos.* 2009. Disponível em: <http://lsi.ugr.es/joseluisgs/videojuegos/>. 27, 38

[35] MACHINERY, A. F. C. *ACM.* 2009. Disponível em: <http://www.acm.org>. 27

[36] ENGINEERS, I. O. E.; ELECTRONICS. *IEEE.* 2009. Disponível em: <http://www.ieee.org/>. 27

[37] ONTOLOGY, D. G. *DGO. Digital Game Ontology Wiki. 2008.* 2009. Disponível em: <http://myweb.polyu.edu.hk/-06722070g/wiki/index.php?title=Main_Page>. 27, 38

[38] ONTOLOGY, D. G. *DGO. Digital Game Ontology Search.* 2009. Disponível em: <links em cache disponíveis em http://search.yahoo.com/search?p=%22Digital+Game+Ontology+Wiki%22&ei=UTF-8&dups=1&fr=sfp&xargs=0&pstart=1&b=1&xa=FPDgg8ofy.4h3CtLdWV9bA–,1251751867>. 27

[39] PRENSKY, M. Digital game-based learning. *Comput. Entertain*, v. 1, p. 21–21, 2003. 28

[40] BERGER, A. A. *Video Games A Popular Culture Phenomenon.* [S.l.]: TRANSACTION PUB, 2002. 28

[41] WHITAKER, M. *Cursos de Games no Brasil.* 2008. 28

[42] ASSESPRO. *Desistências nos cursos de jogos Digitais.* 2009. Disponível em: <http://blog.assespro.org.br/2009/04/08/mec-divulga-estudo-de-ofertas-de-cursos-nas-areas-de-informatica/>. 28

[43] HUNICKE, R.; LEBLANC, M.; ZUBEK, R. MDA: A Formal Approach to Game Design and Game Research. *Discovery.* 28

[44] KOSTER, R. *A Theory of Fun for Game Design*. [S.l.]: Paraglyph Press, 2004. 29

[45] COSTIKYAN, G. I Have No Words & I Must Design: Toward a Critical Vocabulary for Games. *Computer*, p. 9–33, 2002. 29, 56

[46] GEAR, S. G. et al. *Ultimate Game Design: Building Game Worlds*. [S.l.]: McGraw-Hill/Osborne, 2003. 29

[47] III, R. R. *Game Design: Theory & Practice*. [S.l.]: Wordware Publishing, 2001. 29

[48] SCHELL, J. *The Art of Game Design*. [S.l.]: Elsevier, 2008. 29

[49] ROLLINGS, A.; ADAMS, E. *Andrew Rollings and Ernest Adams on Game Design*. [S.l.]: New Riders Games, 2003. 29, 47

[50] MAN, T. F.; SWAIN, C.; HOFF, S. S. *Game Design Workshop*. [S.l.]: Elsevier, 2008. 30

[51] WOLF, M. J. P. *The Medium of the Video Game*. [S.l.]: University of Texas Press, 2002. 30

[52] APPERLEY, T. H. Genre and game studies: Toward a critical approach to video game genres. *Simulation & Gaming*, v. 37, p. 6–23, 2006. ISSN 1046-8781. Disponível em: <http://sag.sagepub.com/cgi/doi/10.1177/1046878105282278>. 30, 46

[53] NOVAK, J. *Game Development Essentials: An Introduction*. [S.l.]: Paperback, 2007. 30, 47, 49

[54] ROLLINGS, A.; MORRIS, D. *Game Architecture and Design: A New Edition*. [S.l.]: New Riders, 2003. 30, 46, 47, 71

[55] SALEN, K. *Rules of Play: Game Design Fundamentals*. [S.l.]: The MIT Press, 2003. 30

[56] BATES, B. *Game Design, Second Edition*. [S.l.]: Stacy L.Hiquet, 2004. ISBN 1-59200-493-8. 31, 56

[57] CONTRERAS, J. A. A. Monografia, *Sdk gameplay*. 2008. 32

[58] LUIZ, C. J.; MARTIN, L. E. G. a. *Um modelo de Interações para Definição de Regras de Jogo*. São Paulo: [s.n.], 2005. 33, 44

[59] COSTI, D. A. T. Monografia, *Scald - Um Motor Genérico de Role-Playing Games Computadorizados Con gurável por Scripts*. 2009. 33

[60] SILVA, A. A. B. D. Monografia, *Arquitetura Reusável Baseada em Mecânica de Jogo para Jogos Casuais*. 2009. 34

[61] MANFROI, H. S. Monografia, *FACHwerk: Um Framework para Desenvolvimento de Jogos Competitivos*. 2008. 35

[62] OGE. *Open Game Engine*. 2009. Disponível em: <http://sourceforge.net/apps/mediawiki/oge/>. 36

[63] ODE. *Open Dynamics Engine*. 2009. Disponível em: <http://www.ode.org/>. 36

[64] OGRE. *Open Source 3D Graphics Engine*. 2009. Disponível em: <http://www.ogre3d.org/>. 36

[65] MATEAS, M.; HOCHHALTER, B.; LICHTI, N. Towards an Ontological Language for Game Analysis. *Technology*. 36

[66] HORRIDGE, M. et al. *A Practical Guide To Building OWL Ontologies Using Protége 4 and CO-ODE Tools*. [S.l.]: The University Of Manchester, 2009. 0–108 p. 38, 55

[67] ENTERBRAIN. *RPG Maker XP*. 2009. Disponível em: <http://tkool.jp/products/rpgxp/eng/>. 38

[68] YOYOGAMES. *Game Maker*. Disponível em: <http://www.yoyogames.com/make>. 39

[69] MICROSOFT. *Popfly*. 2009. Disponível em: <http://popfly.ms/>. 40

[70] GAUDREAULT, G. *Sploder*. 2009. Disponível em: <http://www.sploder.com/>. 40

[71] ZIPZAPPLAY. *PlayCrafter*. 2009. Disponível em: <http://www.playcrafter.com/>. 40

[72] ADOBE. *Flash Player*. 2009. Disponível em: <http://www.adobe.com/products/flashplayer/>. 40

[73] SOFTWARE, S. *Power Game Factory*. 2009. Disponível em: <http://www.sawbladesoftware.com/>. 41

[74] UNITY. *Unity 3D*. 2009. Disponível em: <http://unity3d.com/>. 42

[75] GAMES, G. *Torque Game Builder*. 2009. Disponível em: <http://www.garagegames.com/products/torque-2d>. 43

[76] CRAWFORD, C. *Chris Crawford on Game Design*. [S.l.]: New Riders Publishing, 2003. 46

[77] WIKIPEDIA. *Super Mario World*. 2009. Disponível em: <http://en.wikipedia.org/wiki/Super_Mario_World>. 48

[78] MALABY, T. M. Beyond Play: A New Approach to Games. *Games & Culture*, 2006. Disponível em: <http://ssrn.com/abstract=922456>. 56

[79] PEDERSEN, R. E. *Game Design Foundations*. [S.l.]: Wordware Publishing, 2003. 56

[80] BREYER, F. B. et al. Pesquisa de Jogos Similares como Fonte de Conhecimento para Construção de uma Ontologia de Jogos de Simulação Casuais. p. 1–9, 2009. 57

[81] FERREIRA, R. et al. Emprego da Caixa Morfológica como Técnica de Seleção de Alternativas no Processo de Construção de Advergames. p. 1–6, 2009. 57

[82] MALCHER, F.; NEVES, A. M. M.; aO, L. F. Aplicação do Game Ontology Project no Processo de Análise de Similares para Design de Jogos. *Interface*, p. 1–9, 2009. 57

[83] W3C. *World Wide Web Consortium*. 2009. Disponível em: <http://www.w3.org/>. 68

[84] MENEZES, J. a. R. D. B. Monografia, *Um Framework para Criação de Jogos Computadorizados Multiplataforma*. 2004. 72

[85] IRRLICHT. *Irrlicht*. 2009. Disponível em: <http://irrlicht.sourceforge.net/>. 73

[86] AMBIERRA. *IrrXML*. 2009. Disponível em: <http://www.ambiera.com/irrxml/>. 73

[87] APPLE. *Mac OS X*. 2009. Disponível em: <http://www.apple.com/macosx/>. 74

[88] APPLE. *XCode*. 2009. Disponível em: <http://developer.apple.com/TOOLS/Xcode/>. 74

[89] GNU. *GCC*. 2009. Disponível em: <http://gcc.gnu.org/>. 74

[90] INITIATIVE, O. S. *MIT License*. 2009. Disponível em: <http://www.opensource.org/licenses/mit-license.php>. 75

[91] WIKIPEDIA. *Pong*. 2009. Disponível em: <http://en.wikipedia.org/wiki/Pong>. 77

[92] ATARI. *Atari*. 2009. Disponível em: <http://www.atari.com/>. 77