Criação de Jogos Eletrônicos com Desenvolvimento de Jogos Orientado a Modelos

Jayme G. M. Calixto[1], **Thiago J. Bittar**[12]

[1]Departamento de Ciência da Computação
Universidade Federal de Goiás - Campus Catalão (UFG)
Catalão – GO – Brasil

[2]Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)
São Carlos – SP – Brasil

{jaymecalixto,thiagojabur} @gmail.com

Abstract. *One major concern in the video game industry is dealing the increasing demand and complexity of newer games. The methods of development currently in use have little efficiency in the area of development process automation, requiring the presence of a human programmer performing repetitive and bothersome mechanical tasks. This paper describes the features of Model-driven Game Development and its advantages over the development models currently in use.*

Resumo. *Uma das grandes preocupações da indústria de videogames é com o crescente aumento na demanda e na complexidade dos novos títulos. Os sistemas atualmente em uso não são muito eficientes no sentido de automação do processo, exigindo a presença de um programador humano até para a execução de tarefas repetitivas e mecânicas. Este artigo descreve as características do desenvolvimento de jogos orientado a modelos e investiga suas vantagens sobre os modelos de desenvolvimento atualmente em uso.*

1. Introdução

Jogos eletrônicos compreendem uma das indústrias mais lucrativas do mundo. Foram responsáveis por movimentar, em 2009, mais de dez bilhões de dólares.[1] Quantidade invejável até por indústrias altamente lucrativas, como a indústria do cinema.

Essa indústria, porém, não é cercada apenas por sucessos. Vários desafios computacionais apareceram primeiro no mundo dos videogames, em que tecnologias de ponta são largamente aplicadas e o hardware é levado ao seu limite em uma frequência relativamente alta. Além dos desafios computacionais, o processo de criação de um jogo, do ponto de vista industrial, também recebe uma atenção especial. Com prazos cada vez mais curtos, maior complexidade esperada do software e a crescente demanda por novos títulos do mercado consumidor, um método de automação de todo ou grande parte do processo de concepção de jogos eletrônicos é altamente desejado.

Alguns estudos apontam que o atual paradigma de desenvolvimento está próximo de seu fim e que um novo paradigma se faz necessário para apoiar o próximo salto na

[1]Fonte: http://www.theesa.com/facts/index.asp acessado em 08 de set. 2010

tecnologia de desenvolvimento de software. Por exemplo, apesar de softwares motores de jogos agregarem os benefícios da engenharia de software e da orientação a objetos na automação do desenvolvimento de jogos, o nível de abstração fornecido por eles poderia ser menos complexo com o uso de modelos visuais e melhor integração com o processo de desenvolvimento [Furtado and Santos 2006].

Este artigo está organizado da seguinte forma: a seção 2 apresenta uma visão geral sobre as tecnologias atualmente empregadas na concepção de um jogo eletrônico, suas características, vantagens e desvantagens. A seção 3 relaciona a Orientação a Modelos como ferramenta alternativa no desenvolvimento de jogos, com a subseção 3.1 tratando do conceito de Linguagem de Domínio Específico, a subseção 3.2 descrevendo a Orientação a Modelos no ambiente de criação de jogos eletrônicos e a subseção 3.3 traz uma definição de meta-modelo bem como o detalhamento deste tipo de modelo criado para a elaboração deste artigo. A seção 4 lista os trabalhos futuros relacionados à metodologia Orientada a Modelos e a criação de um meta-modelo auxiliar no desenvolvimento de jogos.

2. Tecnologias e Ferramentas Atuais

Várias ferramentas foram criadas com o intuito de facilitar e reduzir custos na criação de um jogo eletrônico. São elas as APIs (Application Program Interfaces) Gráficas ou Multimídias, os motores de jogo e ferramentas visuais. Nesta seção são descritas as principais características de cada uma, bem como suas vantagens e desvantagens.

2.1. APIs Gráficas

As APIs Gráficas como o **Microsoft DirectX**[2] e o **OpenGL**[3] são bibliotecas que podem ser usadas para acessar diretamente o hardware da máquina (processador gráfico, placas de som, dispositivos de entrada). Essas APIs são úteis para criar jogos com um nível alto em performance e por admitir a portabilidade de jogos de computador entre dispositivos produzidos por diferentes empresas. Assim, uma API Gráfica fornece uma interface padrão de manipulação de hardware e o programador não precisa se preocupar com peculiaridades da estrutura em baixo nível de cada possível dispositivo alvo.

Por fornecer um grau razoável de abstração e bons níveis de otimização do produto final, são largamente usadas nos dias de hoje e provavelmente vão durar por muito tempo, sendo usadas tanto direta, com o programador literalmente realizando chamadas às suas funções, ou indiretamente, como quando uma outra ferramenta de criação de jogos é escrita sobre uma API.

Porém, tudo o que as APIs fornecem são os meios de acessar o hardware. O nível de abstração ainda é muito aquém do desejado pelos programadores, caso as APIs multimídia sejam o único mecanismo de abstração usado. Além disso, o desenvolvimento de jogos usando essas APIs só pode ser realizado através de códigos, e não visualmente. Isso impede a automação e diminui a produtividade, já que atividades simples são obrigadas a serem realizadas com vários "copiar e colar" seguidos, por exemplo [Walbourn].

Outro ponto negativo é que todo o código da lógica do jogo é "mesclado" às chamadas a funções da API, diminuindo a legibilidade e dificultando a modularidade.

[2]Microsoft DirectX `http://www.microsoft.com/games/en-us/aboutgfw/pages/directx.aspx` acessado em 12 de out. 2010

[3]OpenGL `http://www.opengl.org/` acessado em 12 de out. 2010

2.2. Ferramentas Visuais

Ferramentas visuais para a criação de jogos foram concebidas com a intenção de simplificar o processo de criação de jogos e torná-lo mais acessível a um número maior de usuários. Essas ferramentas logo se tornaram bastante populares, principalmente entre *hobbistas* e desenvolvedores independentes. Com o objetivo de criar jogos completos com pouca ou nenhuma programação, muitas vezes com apenas cliques do mouse, essas ferramentas auxiliam o usuário com interfaces gráficas intuitivas para a criação de animações, definição de comportamento de entidades, o fluxo do jogo como um todo e a adição de áudio, menus, textos informativos e outros recursos inerentes ao *software*.

Uma ferramenta visual de criação de jogos pode ser tanto genérica ou com o foco em um gênero específico de jogo, como RPGs (*Role-playing Game*) ou *adventures*. Ferramentas populares incluem o *RPG Maker*[4], voltado para a criação de RPGs no estilo de clássicos do Super Nintendo como *Final Fantasy VI*[5] ou *Breath of Fire*[6] e o *Ren'py*[7], ferramenta de código aberto que auxilia a criação de *visual novels* como a série *Phoenix Wright: Ace Attorney*[8].

Mas, embora a ideia de criar jogos completos com apenas alguns cliques pareça muito interessante, as possibilidades são bastante limitadas. Alguns tipos de jogos podem ser feitos, mas características como a inovação dos jogos nem sempre são fáceis (ou possíveis) de se atingir com essas ferramentas. Apesar de extremamente populares entre iniciantes, desenvolvedores amadores e entusiastas, ferramentas visuais nunca foram adotadas em grande escala pela indústria.

Algumas ferramentas, como o *RPG Maker*, tentam contornar esses problemas com o uso de uma linguagem de *scripts*, permitindo uma melhor flexibilidade do jogo criado ao custo de programação numa linguagem de alto nível. Entretanto, o uso de uma linguagem de programação força o usuário a aprender uma linguagem nova (muitas vezes a sua primeira linguagem) e a ter alguma habilidade com programação de computadores, divergindo da premissa da ferramenta de ser um ambiente de "programação visual" ou "voltado para leigos".

Nem todos os usuários estão dispostos a isso e muitos continuam usando as ferramentas apenas no "modo visual". Enquanto os que realmente aprendem a programar preferem os benefícios de uma linguagem realmente orientada a objetos e mais poderosa com o auxílio de um ambiente de desenvolvimento (IDE) robusto, ferramentas de depuração e alto nível de flexibilidade em vez de usar a "tentativa e erro" com linguagens de *script* em ambientes que não foram concebidos como um ambiente de programação [Furtado and Santos 2006].

---

[4]RPG Maker `http://www.rpgmakerweb.com/` acessado em 12 de out. 2010

[5]Final Fantasy VI `http://na.square-enix.com/games/anthology/FFVI/world.html` acessado em 12 de out. 2010

[6]Breath of Fire `http://www.capcom.co.jp/newproducts/consumer/gbabof1/index.html` (em japonês) acessado em 12 de out. 2010

[7]Ren'py `http://www.renpy.org/` acessado em 12 de out. 2010

[8]Phoenix Wright: Ace Attorney `http://www.capcom.com/phoenixwright/` acessado em 12 de out. 2010

2.3. Motores de Jogo

Os motores de jogo surgiram como o resultado da aplicação de conceitos de engenharia de software para o desenvolvimento de jogos de computador. Um motor de jogo pode ser visto como uma API reusável que agrega características comuns ao desenvolvimento de jogos e apresenta uma interface programática através da qual o comportamento do jogo pode ser especificado. Em outras palavras, os desenvolvedores podem focar seus esforços em características mais orientadas aos jogos, como a lógica, inteligência artificial e recursos artísticos [Furtado and Santos 2006].

Diferente das APIs multimídia, em que suas funções são chamadas diretamente pelo código do jogo, com o uso de um motor de jogo essas chamadas são encapsuladas e tratadas automaticamente, deixando o desenvolvedor livre de detalhes técnicos de baixo nível e até do nível das APIs. De fato, a maioria dos motores de jogo são construídos em cima de uma API multimídia, oferecendo as mesmas características em questão de portabilidade. Motores de jogo populares são o OGRE[9], Crystal Space[10], Unreal Engine[11] e Gamebryo[12].

Similar às ferramentas visuais, os motores de jogo podem ser tanto genéricos quanto voltados para um gênero de jogos específico. Porém, buscando uma melhor eficiência, até os motores genéricos limitam sua área de domínio a um subconjunto de possíveis gêneros de jogos de computador (por exemplo, um motor de jogo 3D tem muitas características específicas e diferentes de um motor para jogos 2D isométricos). Assim sendo, a maior vantagem de usar um motor de jogo é que, se for construído em uma arquitetura modular, ele pode ser reutilizado para criar vários jogos diferentes entre si, que só iriam usar os módulos necessários do motor de jogo. O motor proprietário Unreal Engine é um exemplo de motor de jogo modular. Concebido inicialmente para o desenvolvimento de *First-person Shooters*, especificamente o jogo *Unreal*[13], vários jogos comerciais de outros gêneros foram criados com o seu auxílio, como *Harry Potter*[14] e *Batman: Arkham Asylum*[15], respectivamente jogos de plataforma e ação.

Os motores de jogo atualmente representam o estado da arte no desenvolvimento de jogos. Fornecem mais abstração, encapsulamento e reuso, possibilitando que a indústria de jogos atinja um nível elevado de produtividade [Clua and Bittencourt 2005]. Entretanto, a tecnologia tem algumas desvantagens.

Primeiramente, graças à complexidade inerente dos motores de jogo, a curva de aprendizado é bastante alta. A necessidade de conhecimentos da arquitetura da ferramenta, interação dela com a API e peculiaridades da programação tornam o seu uso uma

[9]OGRE `http://www.ogre3d.org/` acessado em 12 de out. 2010

[10]Crystal Space `http://www.crystalspace3d.org/main/Main_Page` acessado em 12 de out. 2010

[11]Unreal Engine `http://www.unreal.com/` acessado em 12 de out. 2010

[12]Gamebryo `http://www.emergent.net/Products/Gamebryo/` acessado em 12 de out. 2010

[13]Unreal `http://www.unrealtournament.com/us/index.html` acessado em 12 de out. 2010

[14]Harry Potter `http://www.ea.com/uk/game/harry-potter-philosophers-stone` acessado em 12 de out. 2010

[15]Batman: Arkham Asylum `http://www.batmanarkhamasylum.com/start` acessado em 12 de out. 2010

experiência pouco intuitiva ao primeiro contato.

Em segundo lugar, o uso de um motor de jogo envolve um alto investimento financeiro, como o custo de aquisição da ferramenta e treinamento dos desenvolvedores. E construir um motor de jogo também é uma tarefa extremamente dispendiosa, repleta de requisitos a serem satisfeitos e que consome bastante tempo.

3. Engenharia de Software Orientada a Modelos

No ambiente de desenvolvimento de jogos, como na maioria dos sistemas complexos, é comum o desenvolvimento de protótipos. Visando aproveitar os recursos já aplicados e acelerar o tempo necessário para o desenvolvimento de protótipos e, por consequência, dos jogos, surgiu a aplicação de conceitos da Engenharia Orientada a Modelos (*model-driven engineering*, MDE).

A MDE introduz uma mudança de paradigma, colocando os modelos como prioridade máxima no ciclo do software. Com isso, os modelos se tornam a base no desenvolvimento, manutenção e evolução do software. Com o foco nos modelos e não no código, um nível maior de abstração e automação é atingido. Assim surge um novo ramo na MDE, o desenvolvimento de jogos orientado a modelos (*model-driven game development*, MDGD).

Algumas vantagens preliminares do desenvolvimento através do MDGD são o melhor uso de artefatos de software, aumentado o reuso e a produtividade, reforço no uso de um motor de jogo e um aumento na portabilidade e interoperabilidade entre diferentes plataformas de jogos [Reyno and Cubel 2009].

Com a finalidade de acelerar ainda mais o desenvolvimento de jogos e tornar o desenvolvimento orientado a modelos ainda mais flexível, é possível utilizar meta-modelos. O meta-modelo para o desenvolvimento de jogos é abordado em detalhes na subseção 3.3.

3.1. Linguagem de Domínio Específico

O uso de MDGD é realizado, primariamente, por uma linguagem de domínio especifico (*domain-specific language*, DSL). Apesar de ser um termo relativamente novo, linguagens de domínio específico são bastante conhecidas e já estão presentes em diversas aplicações computacionais. Exemplos de DSLs incluem o SQL (*Structured Query Language*), para consultas a bancos de dados relacionais, a folha de estilos CSS (*Cascade Style Sheet*), para a formatação e layout de páginas web, e o UnrealScript, linguagem específica para a programação dentro do motor de jogo Unreal Engine.

O antônimo de uma linguagem de domínio específico são as linguagens de propósito geral, como C/C++ ou uma linguagem de modelagem de propósito geral, como o UML. As vantagens de se utilizar uma DSL são interessantes:

- Maior facilidade na resolução de problemas específicos aos casos em que a linguagem se aplica;
- Sintaxe (ou modelos, no caso de linguagens de modelagem) mais intuitiva, ocasionando uma curva de aprendizado pequena;
- DSL aumentam a qualidade, produtividade, confiabilidade, manutenção, portabilidade e reuso [Spinellis 2001].

3.2. Desenvolvimento de Jogos Orientado a Modelos

O desenvolvimento orientado a modelos já abrange diversas áreas e está começando a ser estudado e aplicado no desenvolvimento de jogos. Como visto anteriormente, os antigos paradigmas não conseguem acompanhar o crescimento exponencial de demanda e complexidade dos jogos, exigindo cada vez mais um novo método de desenvolvimento. O MDGD se encaixa nessas necessidades, oferecendo uma plataforma sólida para o desenvolvimento de jogos, um melhor aproveitamento dos recursos financeiros e humanos e um rápido desenvolvimento.

Reyno e Cubel (2009) realizaram um estudo preliminar a fim de determinar o quão eficiente é a implantação do MDGD no desenvolvimento de jogos. Dividiram o desenvolvimento de um jogo simples de plataforma em dois times. Um deles implementaria seguindo o MDGD enquanto o outro time desenvolvia o jogo através de métodos convencionais. É importante notar que, por ser um jogo simples, nenhum dos grupos usou um motor de jogo.

A implementação manual levou uma semana enquanto o desenvolvimento em MDGD levou algumas horas, tendo noventa e três por cento do código gerado automaticamente. Os resultados são bastante atraentes para o mercado de desenvolvimento de jogos, que sempre tem que lidar com prazos apertados e, não raro, recursos financeiros limitados [Reyno and Cubel 2009].

3.3. Meta-modelo

Um meta-modelo é, de forma geral, um modelo criado para o desenvolvimento de modelos específicos ou DSLs. Trata-se da elaboração de regras, métodos e conceitos que serão usados no modelo da atividade específica desejada [Ernst 2002]. O nível de detalhamento do meta-modelo deve englobar todas as características genéricas de um jogo. A figura 1 representa um meta-modelo desenvolvido visando a aplicação no ambiente de desenvolvimento de jogos com a ferramenta *Eclipse Modeling Framework*[16]. Este meta-modelo foi confeccionado pelos autores do presente artigo a fim de facilitar a compreensão do ambiente de modelagem e suas características.

A classe *Game* agrega todas as principais características do jogo, como o nome (*Name*) e seus objetivos (*Objectives)*. Os enumeradores *Players*, *GameLicense* e *CameraType* definem, respectivamente, todos os valores possíveis para a quantidade de jogadores, o tipo de licença do jogo e o seu estilo de câmera.

A classe *Ranking* armazena os dados das maiores pontuações dos jogadores. É uma classe opcional, já que nem todo jogo tem um *ranking*. A seguir, *Platform* define as limitações técnicas das plataformas em que o jogo será compatível. O atributo *SystemSpecs* define detalhes técnicos, como quantidade de memória disponível ou a velocidade do processador naquela plataforma enquanto o campo *Name* armazena o nome da plataforma, como *PC* ou *Nintendo DS*, por exemplo.

PlayerAccount é uma classe opcional cuja existência definirá ou não se o jogo necessita de cadastro por parte do jogador, armazenando seus dados básicos como nome de usuário (*Username*) e senha (*Password*). Por fim, a classe *GameCharacter* agrupa todos

[16]Eclipse Modeling Framework `http://www.eclipse.org/modeling/emf/` acessado em 12 de out. 2010

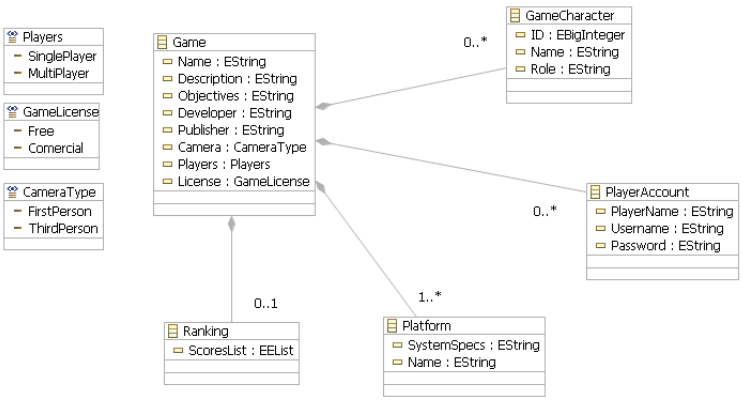


Figura 1. Meta-modelo para o desenvolvimento de jogos.

os personagens do jogo, diferenciando-os por um identificador único (*ID*) e definindo seus papéis (se é um personagem do jogador ou um inimigo, por exemplo) através do campo *role*.

A fim de garantir que a modelagem esteja correta, é possível gerar um arquivo .xmi (de *XML Metadata Interchange* ou Troca de Metadados com XML), que tem como objetivo fornecer um meio de fácil acesso a informações entre modelos UML e MOF (*Meta Object Facility*) uma linguagem voltada à definição de meta-modelos.

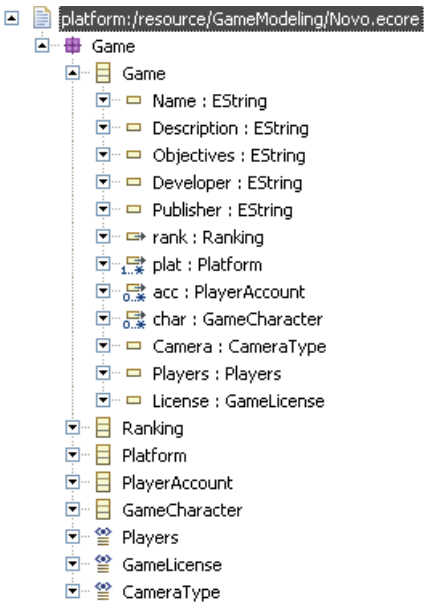
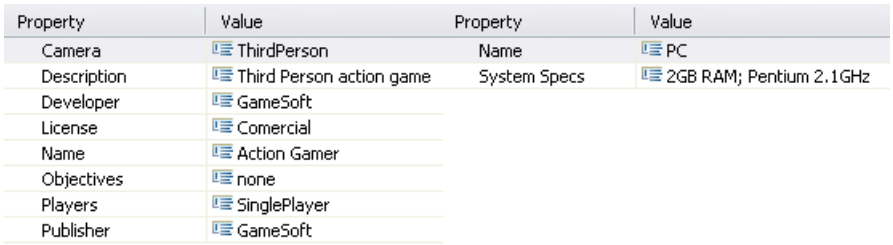


Figura 2. XMI gerado a partir do modelo na Figura 1.

O arquivo .xmi gerado conta com todos as classes e atributos definidos no meta-modelo e, por fim, é possível inserir dados de exemplo para popular o modelo. Na Figura 3 os dados inseridos nas classes *Game* e *Platform* são exibidos.



Property	Value	Property	Value
Camera	ThirdPerson	Name	PC
Description	Third Person action game	System Specs	2GB RAM; Pentium 2.1GHz
Developer	GameSoft		
License	Comercial		
Name	Action Gamer		
Objectives	none		
Players	SinglePlayer		
Publisher	GameSoft		

Figura 3. Dados inseridos nas classes *Game* e *Platform*.

4. Trabalhos Futuros

Mais estudos são necessários a fim de averiguar o ganho em produtividade do MDGD, bem como aliar o método de desenvolvimento com o uso de motores de jogo, realizando, no lugar de um resultado totalmente novo, uma aliança entre o modelo de desenvolvimento atualmente em uso e o MDGD. O benefício dessa aliança no lugar de uma substituição total é a retrocompatibilidade, além de unir todas as vantagens de portabilidade e performance dos motores de jogo ao rápido desenvolvimento da metodologia orientada a modelos.

Mas não só a indústria se beneficiaria do desenvolvimento orientado a modelos. Por conta da facilidade e intuitividade que o MDGD pode oferecer, entusiastas e desenvolvedores independentes podem usar este método no lugar das ferramentas visuais. A curva de aprendizado é ligeiramente maior, o que talvez afaste boa parte dos usuários casuais, mas desenvolvedores independentes que obtêm lucro através de pequenos jogos para celular ou até consoles[17], por exemplo, conseguiriam terminar um jogo em muito menos tempo do que programando sozinho ou em times pequenos.

Referências

Clua, E. W. G. e Bittencourt, J. R. (2005). Desenvolvimento de jogos 3d: Concepção, design e programação. *Anais da XXIV Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação*, pages 1313–1356.

Ernst, J. (2002). What is metamodeling? `http://infogrid.org/wiki/Reference/WhatIsMetaModeling` acessado em 11 de set. 2010.

Furtado, A. W. B. e Santos, A. L. M. (2006). Using domain-specific modeling towards computer games development industrialization. *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling*.

Reyno, E. M. e Cubel, J. . C. (2009). Automatic prototyping in model-driven game development. *ACM Comput. Entertain.*, 7,2:9.

Spinellis, D. (2001). Notable design patterns for domain specific languages. *Journal of Systems and Software*, 56(1):91–99.

Walbourn, C. Graphics apis in windows. `http://msdn.microsoft.com/en-us/library/ee417756(VS.85).aspx` acessado em 27 de ago. 2010.

[17]Fonte: `http://www.npr.org/templates/story/story.php?storyId=94025221` acessado em 12 de out. 2010