Extensão da fábrica de jogos SharpLudus para desenvolvimento em J2ME

Heitor Vital, Marcus V. L. Machado, Geraldo F. S. Filho, Roberto S. M. Barros
Centro de Informática – Universidade Federal de Pernambuco (UFPE)
{hvc, mvlm, gfsf, roberto}@cin.ufpe.br

Resumo

Pesquisas de mercado revelam um forte crescimento do consumo de jogos móveis para os próximos anos. Em paralelo a esta tendência, existe o aumento dos custos de produção de novos jogos motivada pela evolução da complexidade dos mesmos, a necessidade de adaptação do jogo para diversos aparelhos, e a falta de reuso e automação no processo de desenvolvimento. No esforço para solucionar esta questão, faz-se necessário evoluir o processo de desenvolvimento de jogos móveis - integrando o desenvolvimento destes, com o conceito de fábricas de software. Neste sentido, o presente trabalho estende a fábrica SharpLudus para desenvolvimento de jogos móveis.

Palavras Chave: desenvolvimento de jogos móveis, fábrica de software, SharpLudus, Linguagens de domínio específico (DSL).

1. Introdução

O mercado mundial de jogos móveis gerou um faturamento aproximado de US$ 2.5 bilhões em 2006 e as projeções apontam para uma receita de US$ 7 bilhões no final desta década segundo [Tercek 2007]. Para atender a esta demanda de mercado, empresas estão investindo alto em inovação e criatividade para lançar novos produtos, oferecendo jogos mais elaborados ao seu mercado consumidor.

Por outro lado, assim como no desenvolvimento de jogos para PC e console, altos custos também atingem os jogos para dispositivos móveis. Entre os fatores para esta questão estão a pouca automação no processo de desenvolvimento e a necessidade de adaptação dos jogos para diversos modelos de aparelhos através do processo de porting [Sampaio 2004]. A Figura 1 apresenta a evolução dos custos de desenvolvimento de um jogo e a proporção do custo de porting para diferentes plataformas, conforme relatório de [Tercek 2007].

Os especialistas da área apontam como um caminho promissor a reutilização dos componentes de software, visto que o projeto e reprojeto de diferentes jogos partem quase sempre dos mesmos princípios e idéias básicos.

Diversas soluções sugiram com o passar dos anos com destaque para padrões de projeto e framework/ game engine (ou simplesmente engine) – pacote de software flexível e extensível que armazena as características comuns de um domínio de aplicação. O objetivo aqui é a construção de várias aplicações com pouco esforço, especificando apenas as particularidades de cada uma.

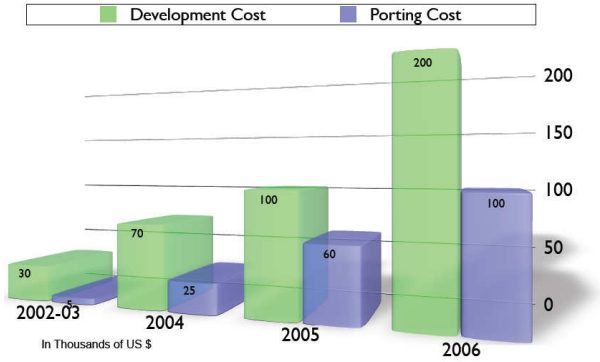

Figura 1 - Evolução do custo de desenvolvimento e porting de jogos para celular

Um passo adiante da produtividade adquirida após o advento dos engines é o conceito de fábricas de software [Greenfield 2004] que visam não somente o reuso das partes comuns, mas também a automatização do empacotamento das partes comuns com as partes específicas de um jogo, esta última geralmente definida utilizando linguagens específicas de domínio.

Neste sentido, uma solução que vem ganhando visibilidade na literatura é o SharpLudus [Furtado et al. 2006; Furtado e Santos 2006] - uma fábrica de software especializada em desenvolvimento de jogos para PC utilizando linguagem de programação C#.

O presente trabalho estende a fábrica SharpLudus para desenvolvimento de jogos móveis. Nele uma linha de produtos e sua arquitetura foram definidas, bem como foram implementados os artefatos relacionados à fábrica de software - design visual baseado em uma linguagem específica de domínio, validadores semânticos e geradores de código. Por fim, um estudo de caso foi realizado para ilustrar e validar a abordagem proposta.

2. Trabalhos Relacionados

Fábricas de software consistem em uma abordagem de desenvolvimento que busca aumentar a produtividade e reduzir o tempo de produção de software, onde os desenvolvedores não estão limitados a tratar apenas com conceitos de programação na criação de suas aplicações, como código-fonte. Assim, conforme o

domínio da aplicação sendo desenvolvido (jogos, e-business, telemedicina), o programador pode utilizar conceitos específicos daquele domínio, como "fase do jogo", "personagem principal" e "inimigo", de modo a criar visualmente diagramas que podem ser transformados facilmente no código-fonte final da aplicação.

Esta é a proposta do SharpLudus defendida em uma dissertação de mestrado e aceita pela comunidade acadêmica em [Furtado et al. 2006; Furtado e Santos 2006]. O principal objetivo é permitir a criação mais rápida e produtiva de jogos de computador do tipo adventure 2D [Answers.com 2007], gênero também bastante popular no mercado de jogos para celular, em que um personagem principal explora um mundo coletando e usando itens, combatendo e fugindo de inimigos.

Um jogo no Sharpludus é descrito através da Game Modeling Language (SLGML). Esta DSL nada mais é do que uma linguagem visual para que o programador modele em alto nível todo o fluxo de execução de um jogo.

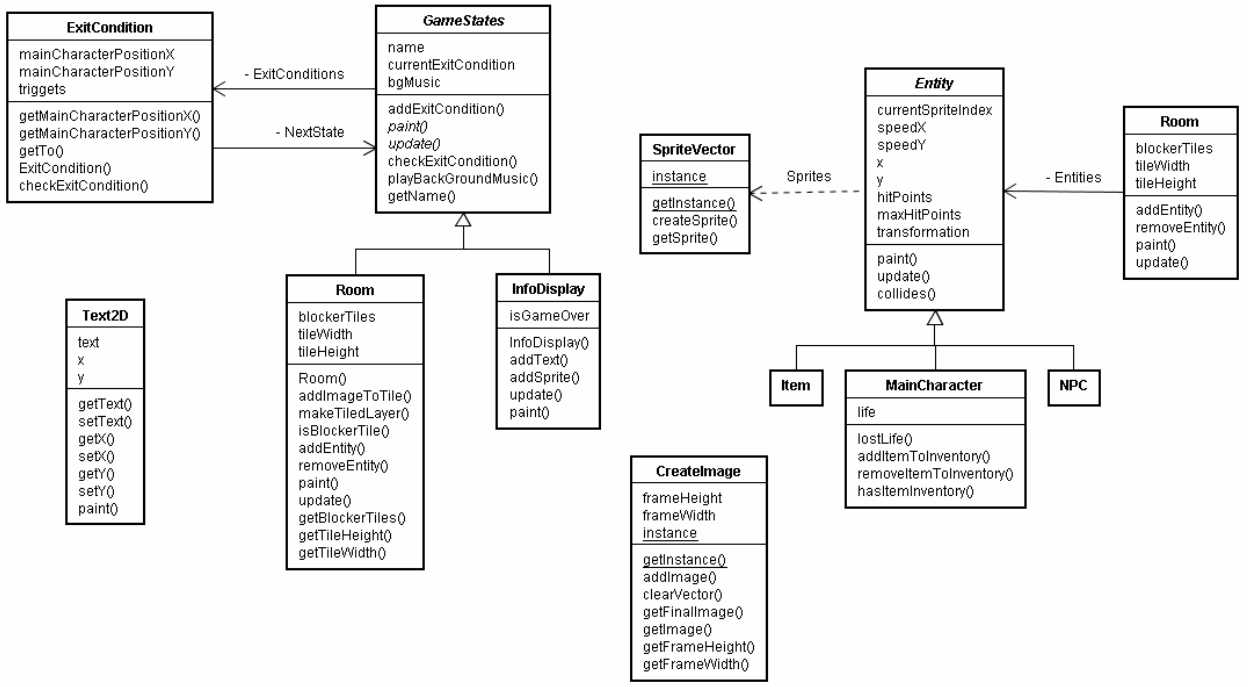

Figura 2 - Sharpludus Mobile Engine

3. SharpLudus Mobile

Nesta seção, serão apresentados alguns detalhes de implementação do SharpLudus e as modificações feitas nesta fábrica a fim de estendê-la para o contexto de jogos móveis. A primeira parte abordará o engine móvel desenvolvido para o projeto e em seguida apresentaremos as alterações feitas no gerador de código.

3.1. Engine J2ME

A game engine usada pela fábrica de software SharpLudus é uma extensão da game engine disponibilizada pelo DigiPen Institute of Technology [Digipen.edu 2007], desenvolvida em C# e que utiliza a API multimídia do DirectX.

Para fazer com que o SharpLudus pudesse gerar código para dispositivos móveis, um novo motor foi implementado. Na Figura 2 é possível visualizar a arquitetura resultante. As respectivas classes são responsáveis por:

- MIDletController: Esta classe implementa o ciclo de vida de uma aplicação J2ME e contem o controle do loop do jogo.
- Game: Classe principal do jogo. É responsável por atualizar e desenhar todos os componentes do jogo.
- AudioComponents: Controla os efeitos sonoros do jogo.
- GameStates: Implementa o fluxo do jogo. Cada GameState possui uma ou mais condições de saída, que ligam um GameState a outro. É especializado pelos conceitos InfoDisplay e Room. O primeiro é usado em telas que exibem informações básicas na tela, já o segundo é utilizado em telas que representam o espaço pelo qual as entidades se movimentam.
- Event: São regras que determinam o que deve acontecer no jogo. Um evento é formado por um conjunto de gatilhos (Trigger) e reações (Reaction).

- CreateImage: Classe responsável pela manipulação das imagens do jogo.
- Entity: Classe abstrata que define todos os objetos do jogo que interagem entre si. Possui três especializações: MainCharacter, non-playable character (NPC) e Item.
- SpriteVector: é responsável por armazenar todos os sprites do jogo.
- Text2D: indica onde e que texto devem ser pintado na tela.

A Tabela 1 contém um resumo da comparação entre as games engines do SharpLudus e do SharpLudus Mobile:

Tabela 1 - Comparação entre game engines

	Engine C#	Engine J2ME
Número de classes	45	31
Linhas de código	2340	~2000

3.2. Gerador de Código

O novo gerador desenvolvido continua a receber como entrada um arquivo SGML e é escrito na mesma linguagem de script que o gerador original.

Mesmo com as mudanças no gerador, a estrutura final do arquivo gerado é bastante semelhante ao arquivo gerado inicialmente para C#. Entretanto isso não significa que a codificação do gerador seja semelhante, como podemos constatar nas listagens Listagem 1 e Listagem 2, a qual também define um trecho do código responsável pela geração das classes responsáveis pela animação de imagens de um jogo (sprites).

```
public static class Sprites {
 <#
 foreach(Sprite sprite in this.SharpLudusGame.GameSprites) {
   string spriteName = sprite.Name.Trim().Replace(" ","");
 #>
   public static Sprite <#=spriteName#> {
      get {
         Sprite result = new Sprite();
          result.Name = "<#=spriteName#>";
          result.Loop = <#=sprite.Loop.ToString().ToLower()#>;
          ...
```

Listagem 1 - Script gerador da classe sprite em C#

```
static {
 CreateImage createImage = CreateImage.getInstance();
 SpriteVector spriteVector = SpriteVector.getInstance();
 Sprite result;
<#   ...
 foreach(Sprite sprite in this.SharpLudusGame.GameSprites) {
  string spriteName = sprite.Name.Replace(" ","_").ToUpper();
  foreach(Frame frame in sprite.Frames) {
    string frameName = frame.Name.Trim().Replace(" ","");
#>
    createImage.addImage(
            "<#=getPNGString(frame.Picture.FilePath)#>");
<#}#>
  result = new Sprite(createImage.getFinalImage(),
 createImage.getFrameWidth(),
 createImage.getFrameHeight());
 spriteVector.createSprite(result);
 ...
```

Listagem 2 - Script gerador da classe sprite em J2ME

Assim como no gerador original, o novo gerador também gera várias classes em um único arquivo. As classes geradas são:

- Resources: Responsável por toda a inicialização dos elementos de áudio do jogo e todos os sprites.
- Uma classe para cada Entity especificada pelo game designer. Tais classes são extensões das classes Item, MainCharacter ou NPC.
- EntityInstances: Responsável pela geração de todos as instâncias das entidades. Tal classe inicializa seus objetos estáticos através de blocos estáticos.
- States: responsável por prover informação sobre os objetos de screen, room e information display. Assim como EntityInstances, inicializa seus objetos estáticos através de blocos estáticos.
- GameEvents: nela as configurações do jogo, como tamanho do mundo, são inicializadas e os eventos são registrados.

Por último, para uma melhor utilização da versão do SharpLudus Mobile, é importante levar em consideração algumas restrições quando se modela um jogo que poderá ser usado com o gerador desenvolvido por esse trabalho. São elas:

- Os frames das animações devem ter o mesmo tamanho. Esta restrição é necessária uma vez que a classe Sprite de MIDP 2.0 utilizada na game engine exige que os frames tenham o mesmo tamanho.
- Os arquivos de recursos (imagens e sons) devem ter nomes diferentes, isto ocorre porque todos os arquivos de recursos são salvos no mesmo diretório, o que não acontece com o SharpLudus original.
- As teclas devem ser mapeadas da seguinte forma:
    - LeftArrow é equivalente a tecla 4 ou a tecla direcional esquerda;
    - RightArrow é equivalente a tecla 6 ou a tecla direcional direita;
    - UpArrow é equivalente a tecla 2 ou a tecla direcional superior;
    - DownArrow é equivalente a tecla 8 ou a tecla direcional inferior;
    - Return é equivalente a tecla 5 ou a tecla direcional central.

4. Estudo de Caso

Esta seção mostra um exemplo do que pode ser feito com as modificações no SharpLudus.

Mesmo que o gerador e a engine tenham mudado consideravelmente durante a extensão do SharpLudus, a modelagem do jogo continua de forma semelhante à versão original. Contudo existe a necessidade de seguir algumas restrições: na versão móvel, o tamanho da tela do dispositivo geralmente é menor que o tamanho da tela do jogo. Desta forma, foi necessário implementar o conceito de janela de visão, onde a área de pintura depende do tamanho da tela do celular.

Depois que a modelagem do jogo é feita, basta copiar a classe gerada (Resources.java) para a raiz dos sources da game engine em J2ME. A partir desse passo é possível rodar o jogo como um projeto normal em J2ME. A Figura 3 mostra a tela do jogo sendo executado.

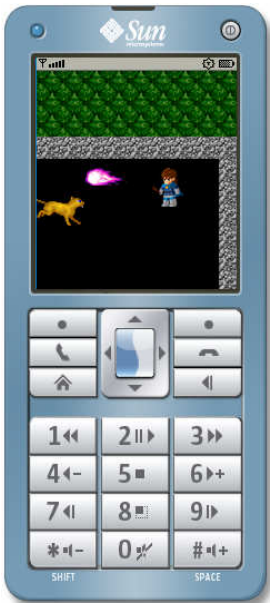

Figura 3 - Imagem do Jogo

5. Conclusões e trabalhos futuros

Este trabalho apresentou o início de um processo de produção de jogos para dispositivos móveis seguindo a metodologia definida no SharpLudus, visando a integração do desenvolvimento de jogos para celular com o conceito de fábricas de software. Para isso, o gerador de código e o motor de jogos foram remodelados e reimplementados para que a fábrica SharpLudus produzisse código para a plataforma J2ME.

É importante ressaltar que estes foram os resultados parciais de uma pesquisa em andamento e que há ainda muito que ser feito, como:

- Suporte à edição do código gerado. Funcionalidade importante para ajustes do jogo.
- Suporte ao processo de porting, tão importante no contexto móvel. Neste caso seria necessário uma grande mudança na arquitetura do SharpLudus, uma vez que seria necessário ter uma mesma lógica de jogo (eventos), mas com tamanhos de mundo, tela, imagens e sprites diferentes.
- Integração com ambientes de desenvolvimento para J2ME, como Eclipse ou Netbeans.
- Aperfeiçoamento da interface gráfica.
- Otimização da game engine.
- Separação das definições de imagem e sprite.
- Integração com ferramentas de compactação de resources (sons, imagens).
- Análise de desempenho do código gerado.

6. Referências

TERCEK, R., 2007. *First Decade of Mobile Games* [online] GDC Mobile, March, 2007, San Francisco. Disponível em: http://ww.roberttercek.com/resources/Decade+of+Mobile+Games+Part+1+GDCM07+Tercek.pdf [Acessado em 2007-04-20].

SAMPAIO, P., DAMASCENO, A., SAMPAIO, I., ALVES, V., RAMALHO, G. E BORBA, P., 2004. Portando Jogos em J2ME: Desafios, Estudo de Caso, e Diretrizes. *In: 3rd Brazilian Workshop on Games and Digital Entertainment*, Outubro 2004 Curitiba.

GREENFIELD, J., SHORT, K. ET AL, 2004. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. John Wiley & Sons.

FURTADO, A., SANTOS, A., PANIGASSI, R. E BECERRA, J., 2006. Specifying a Software Factory for 2D Adventure Computer Games Development. *In: Proceedings of 2nd Software Engineering Conference in Russia* (SEC(R)2006).

FURTADO, A. E SANTOS, A., 2006. Applying Domain-Specific Modeling to Game Development with the Microsoft DSL Tools. Tutorial (in English). *In: 3rd Brazilian Symposium on Computer Games and Digital Entertainment* (SBGames2006).

ANSWERS.COM, 2007. *Action-adventure game* [online]. Disponível em: http://www.answers.com/topic/action-adventure-game/ [Acessado em 2007-05-09].

DIGIPEN.EDU, 2007. *MSDN Webcast Archive - Video Game Development: Learn to Write C# the Fun Way* [online]. Disponível em: http://www.digipen.edu/webcast [Acessado em 2007-05-09].