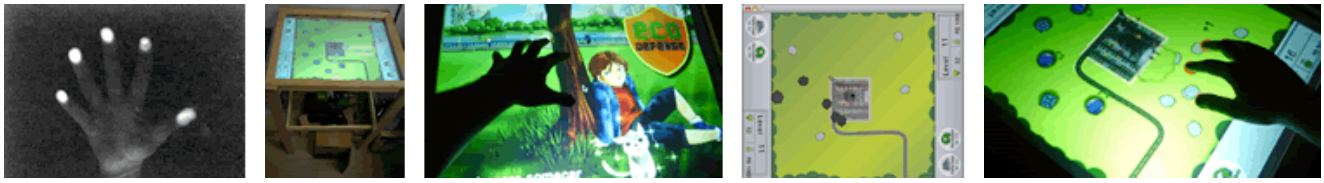
Construção de um jogo multiusuário para uma superfície de projeção multitoque

Pedro G. Brandão Saulo C. R. Braga Carla D. Castanho Ricardo P. Jacobi
Departamento de Ciência da Computação, Universidade de Brasília, Brasil



(a) (b) (c) (d) (e)

Figura 1: *Jogo* Eco Defense *desenvolvido para uma mesa multitoque.*

Resumo

Este artigo tem como principal objetivo apresentar o jogo eletrônico, denominado *Eco Defense*, construído para uma superfície horizontal sensível a múltiplos toques. Assim, este trabalho aborda um paradigma de interação mais natural entre jogador e o sistema utilizando o toque das mãos. Considerando a complexidade e a dinamicidade inerentes do software de um jogo eletrônico, utilizou-se uma arquitetura de software baseada nos conceitos de engenharia de software já estabelecidos, como sistema de componentes e padrões de projeto. Devido à ausência de soluções multitoque acessíveis no mercado, o trabalho envolveu, também, a construção da mesa multitoque com dispositivos e materiais de baixo custo para ser utilizada como interface para jogo.

Keywords: jogo eletrônico, interfaces naturais, multitoque, arquitetura de software, sistema de componentes

Author's Contact:

{pedro, saulo}@beholdstudios.com.br
{carlacastanho, rjacobi}@cic.unb.br

1 Introdução

Para o homem, interagir com objetos sobre uma mesa é considerado bastante natural. Fazendo uma analogia ao computador, uma interface gráfica com a possibilidade de arrastar e manipular documentos e programas com os próprios dedos torna-se mais natural. A naturalidade do toque também pode ser empregada em jogos eletrônicos. Sabe-se que, a manipulação de elementos e objetos em um jogo tradicional de tabuleiro torna o mesmo significativamente interessante e envolvente. Da mesma forma, a possibilidade de interagir com um jogo eletrônico através do toque em uma superfície, traz um novo conceito de jogabilidade e interatividade.

É possível encontrar diversos trabalhos, [Corso et al. 2008], [Bomark 2007], [Shen et al. 2004], [spa 2009], [ver 2009], cujo foco é o estudo de interfaces de toque em sistemas multimídia interativos. Nestes trabalhos, jogos tradicionais como *Tangram* e *Game of Life*, por exemplo, foram desenvolvidos para superfícies multitoque com o objetivo de promover uma interação mais natural do jogador com o jogo.

Neste contexto, propõe-se um jogo eletrônico, denominado *Eco Defense*, cujo mecanismo de *input* é uma superfície horizontal sensível a múltiplos toques, onde diversos usuários podem simultaneamente manipular objetos virtuais.

Um dos desafios que envolvem o emprego de interfaces naturais, como o toque por exemplo, é a obtenção do equipamento que viabiliza o projeto. Devido a ausência de soluções prontas que atendam os requisitos técnicos e econômicos, este trabalho envolve, também, a construção da mesa multitoque a ser utilizada pelo jogo proposto. Esta proposta emprega técnicas de visão computacional para o reconhecimento dos toques realizados sobre a mesa.

Além disso, o desenvolvimento de um jogo eletrônico requer o entendimento e a aplicação dos conceitos de Engenharia de *Software* dentro deste cenário específico. O alto grau de interação e comunicação das entidades do jogo entre si e com o *hardware* influenciam na complexidade destes projetos. Dessa forma, o desenvolvimento do jogo proposto neste trabalho fundamenta-se nos padrões e conceitos de engenharia de *software* já estabelecidos, tais como arquitetura de sistema, camadas de abstração, reusabilidade de código e padrões de projeto [Doherty 2003].

2 Trabalhos correlatos

Os resultados publicados por [Bomark 2007] indicam que com o avanço da performance dos computadores pessoais e a disponibilidade de câmeras eficientes, é possível realizar processamento de imagens em tempo real para a construção de interfaces multitoque, e ainda preservar poder computacional para aplicações gráficas intensas, como jogos e interfaces de usuário atrativas.

Um projeto desenvolvido na *Lulea University of Technology*, na Suíça, propõe a construção de uma mesa multitoque com uma superfície de acrílico de $6mm$ para refletir uma luz infravermelha, através de um fenômeno físico denominado reflexão interna total frustrada (FTIR—*Frustrated Total Internal Reflection*) [Bomark 2007]. Quando o usuário pressiona seu dedo contra a superfície de acrílico, a reflexão total da luz dentro da superfície é quebrada naquele ponto, iluminando o dedo. Desta forma, uma câmera que filma a superfície é capaz de detectar a luz infravermelha naquele ponto e assim um *software* pode processar a imagem a procura destes pontos acesos.

O *Music Technology Group*, na *Universitat Pompeu Fabra* em Barcelona, propôs uma aplicação de uma mesa multitoque que consiste em um instrumento musical inovador, denominado *reacTable* [Jordà et al. 2005]. Seu principal componente é uma superfície tangível. A *reacTable* trabalha principalmente com marcadores fiduciais conhecidos pelo sistema. À medida que objetos são movimentados na superfície, ocorre um *feedback* sonoro e visual, de acordo com o tipo de fiducial. Certos tipos de objetos interagem entre si, sendo possível que a rotação ou a distância influencie seus efeitos sonoros.

Uma das principais formas de interação homem-máquina da atualidade é a utilizada em jogos eletrônicos. Hoje, diversos componentes fazem parte do computador ou *console* a fim de maximizar a performance dos jogos eletrônicos e melhorar sua jogabilidade. Um elemento importante no avanço da jogabilidade destes dispositivos é a inovação nos mecanismos de *input*. Diversos trabalhos de pesquisa e produtos comerciais inovam neste conceito, com multitoque, acelerômetros e câmeras de vídeo. De acordo com pesquisadores da *Brown University* [Katzourin et al. 2006], muitos dos jogadores se surpreenderam com a facilidade de aprendizado e naturalidade destes novos mecanismos de interação.

Estudantes do Centro de aplicações em Realidade Virtual da *Iowa State University*, nos Estados Unidos, desenvolveram uma mesa multitoque e um jogo multijogador interativo no âmbito do projeto denominado *Sparsh UI* [spa 2009]. Trata-se de jogo em

que dois times, situados em lados opostos da mesa, devem simultaneamente, através de toques sobre a superfície, atacar as bases do time oposto ou defender suas próprias instalações virtuais. Este tipo de trabalho põe em prática a discussão sobre multiusuários em superfícies multitoque, trazendo um novo paradigma de interação com as tecnologias.

Os trabalhos e projetos descritos nesta seção demonstram o direcionamento de esforços no desenvolvimento de jogos eletrônicos que incluam uma interação mais natural e envolvente para o usuário.

3 Reconhecimento de toques

O estudo e a pesquisa sobre a utilização de jogos em mesas multitoque demanda conhecimentos a respeito deste novo paradigma, em particular as técnicas voltadas para reconhecimento de toques através da captura de imagens. Uma das soluções existentes para reconhecimento de toques em uma superfície, utiliza um computador dotado de visão computacional com a habilidade de interpretar o posicionamento dos dedos. Uma câmera conectada ao computador captura imagens dos dedos dos usuários sobre a superfície multitoque. Através de algoritmos de processamento de imagens, o posicionamento de cada dedo e seus movimentos são calculados e transmitidos para a aplicação.

A biblioteca utilizada neste trabalho para o processamento das imagens capturadas através da câmera é a *Touchlib* [tou 2009]. Foi desenvolvida especificamente para a criação de superfícies com interação multitoque. Seu funcionamento consiste na aplicação de filtros de forma customizável que permite uma maior adequação às necessidade do projeto. Para a execução destes filtros, a *Touchlib* faz uso extensivo da *OpenCV* (*Open Source Computer Vision Library*) [ope 2009], uma biblioteca multiplataforma para desenvolvimento de aplicativos de processamento de imagens.

A *Touchlib* atua de acordo com o paradigma servidor–cliente, sendo que esta faz o papel de servidor e a aplicação faz o papel do cliente. A comunicação com este *software* é realizada usando o protocolo TUIO (*Tangible User Interface System*) [Kaltenbrunner et al. 2005], que foi desenvolvido especificamente para a transmissão do estado de objetos tangíveis e eventos de toque em uma superfície. Este opera sobre a camada UDP (*User Datagram Protocol*) de transporte, permitindo a divisão de aplicativos em mais de um computador.

4 Arquitetura de *software* aplicada a jogos eletrônicos

Atualmente, os jogos eletrônicos estão atingindo um alto grau de sofisticação e, como consequência, os *softwares* necessários para sua implementação têm se tornado cada vez mais complexos. As equipes de programação de jogos estão se tornando maiores e seus membros realizam tarefas cada vez mais especializadas. Além disso, de acordo com [Doherty 2003], a área de desenvolvimento de jogos está crescendo e amadurecendo ao ponto que as expectativas são similares às outras áreas de desenvolvimento de software. Assim, é essencial que jogos tenham uma arquitetura bem-definida para auxiliar, ou até mesmo viabilizar, o processo de desenvolvimento, manutenção e eventual evolução.

A arquitetura de um jogo deve contemplar a execução de diversas tarefas, tais como receber *input* do jogador, verificar colisão entre objetos, atualizar entidades do jogo, renderizar texturas e executar simulações. Embora tratem de aspectos diferentes do jogo, muitas destas tarefas devem ser executadas a cada *loop* iterativo. Esta arquitetura de *software* deve ser projetada de modo a evitar o forte acoplamento entre estas tarefas distintas, uma vez que reduz a dependência entre diferentes módulos do sistema, permitindo assim um maior reuso e modificações menos onerosas no processo de desenvolvimento.

4.1 Sistema de componentes

Conforme sugerido por [Folmer 2007], uma entidade de um jogo eletrônico é formada por diversos componentes distintos, cada qual com um comportamento específico. Estes componentes podem ser reutilizados por diferentes entidades, evitando assim o replicamento do código de forma desnecessária. Sabe-se que um sistema orientado a objetos não obriga a utilização de hierarquia de classes para estruturar as entidades do jogo. Logo, é possível utilizar uma alternativa que permite uma maior dinâmica da estrutura ainda em tempo de execução. Uma arquitetura orientada a componentes é uma solução que flexibiliza os comportamentos das entidades do jogo através de uma agregação de componentes independentes em uma classe única.

De acordo com [Rabin 2005], uma arquitetura de jogo baseada inteiramente em hierarquia de classes possui algumas limitações. O primeiro problema é o forte acoplamento entre classes. Um *software* com um acoplamento forte é indicativo de ser pouco modularizado e complexo de ser modificado. A herança cria o maior acoplamento possível entre duas classes, pois a classe derivada possui conhecimento sobre as estruturas públicas e protegidas da classe pai. Ou seja, mudanças na classe pai frequentemente representam mudanças na classe derivada.

Outra limitação, exposta em [Rabin 2005], é a falta de flexibilidade da hierarquia de classes. Em um sistema complexo, como um jogo eletrônico, podem existir situações difícies de modelar através de hierarquia. Existem características comuns entre entidades de jogo que não possuem herança comum, logo isto leva a uma replicação de código e comportamento semelhantes, prejudicando a manutenção desta estrutura.

Por fim, a estrutura imposta por hierarquias é estática em linguagens comuns como C++ e JAVA. Durante a execução não é possível alterar a herança de classes. Em um *software* comum isto não costuma ser um limitante, mas em um jogo eletrônico muitas vezes entidades alteram drasticamente seu comportamento em tempo de execução.

Um sistema de componentes utiliza composição para estruturar as entidades do jogo. Ao invés de cada entidade do jogo possuir uma classe distinta, apenas uma classe é criada com o objetivo de simbolizar todas as entidades do jogo. Esta classe pode, por exemplo, ser denominada de `GameEntity`. Esta classe única contém diversos componentes que juntos compõem a estrutura de uma entidade.

Pelo fato de cada componente ser independente dos demais, é possível que a entidade do jogo seja composta por qualquer combinação entre som, física, IA, GUI, gráfico, rede ou componentes de lógica de jogo. Nesta arquitetura de sistema, a entidade `GameEntity` não precisa conhecer a estrutura específica de cada componente, pois ela é apenas uma composição entre estas classes. Assim, o código mantém um acoplamento baixo, permitindo sua manutenção e modificação com mais facilidade.

5 *Game design* do *Eco Defense*

O jogo desenvolvido neste trabalho, denominado *Eco Defense*, tem como objetivo utilizar-se de uma mesa multitoque e trazer como conteúdo uma temática relacionada ao combate dos problemas do meio-ambiente. Durante o jogo, seus participantes deverão colaborativamente destruir a poluição gerada por uma fábrica no centro da tela. Os jogadores devem pressionar seus dedos contra estes poluentes, a fim de destruí-los, antes que atinjam e destruam a floresta ao redor da fábrica.

A destruição dos poluentes gera uma pontuação na forma de dinheiro virtual para os jogadores. Este dinheiro pode ser utilizado para construir torres que servem para auxiliar na eliminação dos poluentes. Os usuários tomam decisões em conjunto de como investir o dinheiro arrecadado pela destruição dos poluentes na construção de novas torres. À medida que o jogo avança, mais poluentes aparecem da fábrica e mais torres podem ser construídas.

Quanto mais jogadores entram no jogo, maiores são as chances de se atingir um melhor resultado. Com mais dedos disponíveis, é possível destruir mais poluentes ao mesmo tempo e obter mais dinheiro para investimento. Assim o jogo promove a participação de vários usuários colaborativamente.

A interface do jogo *Eco Defense* deve atender aos requisitos da mesa multitoque, sendo distribuída em torno da superfície de projeção. Desta maneira todos os usuários podem vivenciar a mesma experiência. A interface é replicada em lados opostos da

projeção, e contém os valores referentes aos pontos de vida da floresta, dinheiro arrecadado e o nível de dificuldade atingido pelos jogadores.

6 Construção da mesa multitoque

Através do estudo dos diversos trabalhos correlatos e utilizando-se do aprendizado dado pelas soluções previamente testadas, foi possível construir uma mesa multitoque a baixo custo com a utilização de componentes e equipamentos facilmente encontrados no mercado. A estrutura da mesa está ilustrada nas Figuras 1(b) e 2.

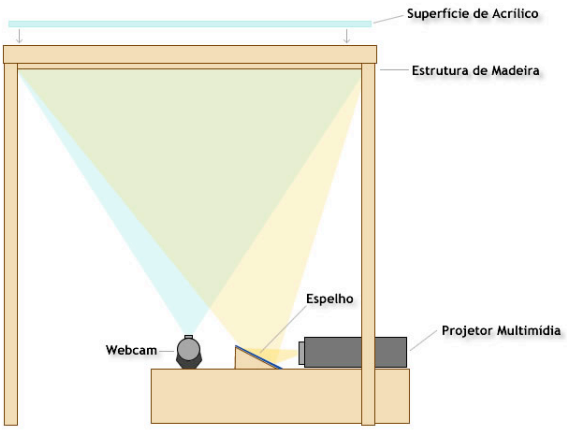


Figura 2: *Estrutura da mesa multitoque.*

Sua estrutura é constituída de madeira que dá suporte à superfície e se posiciona a uma altura aproximada de 1 metro, para que os usuários possam interagir ainda em pé. A superfície é constituída de uma placa de acrílico de espessura de $5mm$ e dimensões aproximadas de $65x45cm$. A espessura foi determinada para garantir a sustentação da superfície mesmo quando pressionada por vários usuários.

A projeção da imagem é obtida através de um projetor XVGA de resolução padrão $1024x768$ *pixels*, a uma distância aproximada de $100cm$ da superfície com auxílio de um espelho. Abaixo da superfície, utiliza-se uma película de papel vegetal para receber a imagem gerada pelo projetor. Esta superfície é essencial, também, para a iluminação correta dos dedos sobre a superfície. Sem a película, toda a mão do usuário seria iluminada igualmente. Com a película, os dedos ficam melhor iluminados se comparados à palma da mão.

Foi utilizado um sistema de iluminação direta, assim, a estrutura serve de apoio para posicionar o acrílico a uma distância adequada de duas fontes de luz infravermelhas posicionados abaixo da mesa junto ao projetor. A iluminação é oriunda de 96 LEDs IR montados sobre duas placas de circuito. Estes LEDs são ligados a uma fonte ATX de 12V e organizados em 12 circuitos paralelos de 8 LEDs. Os LEDs infravermelhos utilizados possuem uma voltagem (*forward voltage*) de $1,4V$ e necessitam de uma corrente aproximada de $20mA$. Para tal, utilizou-se um resistor de $39/ohm$ em cada circuito paralelo.

Por fim, com o objetivo de reduzir os gastos, utilizou-se uma câmera comum, *Microsoft VX-1000*. Esta câmera possui características adequadas de resolução (320x240 *pixels*) e *frame rate* ($60Hz$), e permite a remoção do filtro de luz infravermelho de fábrica. Foram incluídos três filtros de luz visível dentro da câmera, que melhoram o contraste da imagem com relação a luz infravermelha. Como filtro, é utilizado filme fotográfico revelado exposto à luz visível que possuem esta característica de filtragem, fornecendo resultado semelhante à figura 1(a).

7 Implementação do *software* interativo

Neste trabalho, em toda implementação do jogo (cujo resultado pode ser visto nas figuras 1(c), 1(d) e 1(e)), foi utilizada a linguagem de programação C++. De acordo com [Rabin 2005], esta linguagem é amplamente usada nos jogos comerciais, sendo a mais adotada para tal finalidade. Utilizou-se também uma biblioteca multiplataforma denominada SDL (*Simple DirectMedia Layer*) [sdl 2009]. Uma de suas vantagens é oferecer uma fina camada entre o *hardware* e *software* para prover serviços multimídia.

Com relação à parte gráfica, a SDL foi utilizado apenas para disponibilizar o acesso direto ao ambiente *OpenGL* (*Open Graphics Library*). Para carregar imagens em diversos formatos foi utilizada a pequena biblioteca *SOIL* (*Simple OpenGL Image Library*) [soi 2009], que carrega imagens diretamente em texturas *OpenGL*. Para carregar fontes *TrueType* e gerar texturas com texto foi usada a FTGL (*Free TrueType OpenGL Library*) [ftg 2009], que por sua vez usa a biblioteca *FreeType*.

Para a parte de áudio, a abstração da SDL fornece acesso direto ao *stream* de áudio. Assim, foi necessário a utilização do SDL_mixer, um *plugin* para a SDL que permite a execução de diversas faixas simultaneamente. Por fim, para o *input* foi utilizada a *Touchlib*. Do ponto de vista do *software*, este opera de acordo com o protocolo TUIO, responsável pela interface entre *Touchlib* e a aplicação do jogo.

7.1 Visão geral da arquitetura

A arquitetura do jogo é composta por uma controladora principal `Engine` que gerencia os subsistemas e as entidades de jogo. A classe `Engine` é intrinsecamente bastante simples. Ela gerencia as fases de inicialização, execução e encerramento do sistema.

Durante a inicialização, instancia todos os subsistemas e, durante a rotina de encerramento, finaliza cada um na ordem inversa. A fase de execução consiste em um *loop* infinito que atualiza cada subsistema e todos os `GameObjects`. Este *loop* somente é encerrado quando um dos subsistemas solicita término de execução.

7.2 Subsistemas

A arquitetura implementada foi dividida em quatro subsistemas: Gráfico, *Input*, Som e Eventos. Os subsistemas acessam uma controladora de mensagens, denominada `EventManager` do subsistema Eventos, para se comunicar e receber eventos de outros subsistemas.

A arquitetura do jogo implementado seguiu o padrão de sistema de componentes. Assim, além de uma controladora principal, todo subsistema possui famílias de componentes que agregam comportamento às entidades do jogo. Os componentes possuem nomes com prefixo `GOC`, como em `GOCRenderTexture`, componente do subsistema gráfico.

7.2.1 Gráfico

O subsistema gráfico possui uma controladora principal denominada `Graphics`. Esta é responsável pela renderização de todos os elementos visíveis na tela. Cada componente da família `GOCRender` se registra na controladora para serem exibidos na tela. A atualização do `Graphics` consiste em limpar o *buffer* de exibição e executar a atualização de todos os componentes registrados para este subsistema. Cada componente `GOCRender` registrado é desenhado neste *buffer*.

7.2.2 *Input*

O subsistema de *input* é controlado primariamente por uma classe denominada `Input`. A entrada do jogo tem duas fontes, uma é oriunda da SDL e trata de convencionais eventos de teclado e mouse, e outra oriunda da *Touchlib*, que trata eventos transmitidos por meio do protocolo TUIO. Um cliente fornecido em C++, denominado `TuioClient`, se associa a uma porta UDP pré-estabelecida, escutando os eventos que nela chegarem. Esta classe é executada em outra *thread*, permitindo a escuta dos eventos de forma paralela. Assim, a classe *Input* recebe os eventos do `TuioClient` e os repassa para a gerenciadora de eventos do jogo.

Qualquer entidade do jogo interessada em receber eventos de *input* deverá registrar-se como receptora do tipo de evento `FingerInputEvent`. Cada evento possui posição e um ID, referente à "instância do dedo" que provocou aquele evento. Desta forma, é possível acompanhar o movimento ao longo da superfície de um único dedo com facilidade.

7.2.3 Áudio

O subsistema responsável pelo áudio é controlado pela classe `Sound`. Durante a inicialização, é aberta uma instância do *SDL_mixer*. Como a execução do som é delegada ao *SDL_mixer*, não é necessário nenhum passo de atualização. Além disso, compõem este subsistema componentes da família `GOCSound`, que são responsáveis por carregar e reproduzir as faixas de áudio.

7.2.4 Eventos

A comunicação entre objetos é essencial ao sistema de qualquer jogo. Existem casos em que o fluxo de informação é muito complexo e um mecanismo mais flexível deve ser adotado. Para o caso em que várias entidades estão interessadas na notificação da ocorrência de determinado tipo de evento, ou quando se deseja enviar um evento mas não se sabe previamente quem deve recebê-lo, utiliza-se um sistema de mensagens.

Existem diversos tipos de evento. Para cada tipo, cria-se uma classe que herda de `Events`, que pode conter todas as estruturas de dados necessárias. O `EventManager`, única classe *singleton* [Gamma et al. 1995], é responsável pelo gerenciamento dos eventos. Objetos que desejam receber certo tipo de evento devem se registrar com esta classe, informando também o tipo de evento pelo qual se interessa. Objetos que desejam enviar um evento bastam especificar o evento como parâmetro na chamada de `SendEvent(Event)` do `EventManager`. Em seu funcionamento interno, a classe `EventManager` utiliza a estrutura de dados `std::map`, juntamente com `boost::signals`. O envio de eventos pode ser instantâneo ou pode ser atrasado (*delayed*). Neste segundo caso, o evento é enviado apenas no início do próximo *update*.

7.3 *Game Objects* e *Game Object Components*

Um *Game Object* representa uma entidade que pode existir no mundo do jogo. Consiste de uma identificação, um *Transform* (representando a posição, orientação e escala do objeto), e um conjunto de GOCs (*Game Object Components*)[Stoy 2006]. Essencialmente qualquer entidade do jogo é um *game object*, inclusive entidades não visíveis. Em geral, deseja-se sempre ter componentes associados a um *game object*, caso contrário este não terá comportamento algum.

O objetivo de cada GOC é oferecer interfaces simples e funcionalidades muito específicas, permitindo que os *game objects* possam fazer uso de diversos GOCs de forma flexível. Um GOC abstrato descreve uma interface comum que permite com que cada *Game Object* possa gerenciar seus próprios GOCs. Componentes são organizados de acordo com suas funcionalides. Estes grupos são denominados famílias. Cada componente obrigatoriamente pertence a uma família, que define a interface comum aqueles tipos de componentes.

O gerenciamento de componentes em *game objects* consiste em adicionar e obter um componente. Sendo assim, a estrutura de dados adotada é o `std::map` da STL (*Standard Template Library*), onde é feita a relação entre `ComponentIDs` e `Component*`. Para obter um componente de determinada família é usado o método `getGOC` especificando o ID da família desejada. Para adicionar um GOC, basta passar o ponteiro para o GOC que este será adicionado ao mapa de componentes.

8 Conclusão

O desenvolvimento de jogos eletrônicos tem se tornado, cada vez mais, uma área importante da engenharia de software. A evolução de sistemas interativos leva a um estudo aprofundado de diversas técnicas e conceitos de arquitetura, padrões e processos. Sem este estudo, os jogos complexos de hoje se tornariam inviáveis, pois alguns ultrapassam milhões de linhas de código, demandam equipes com dezenas de profissionais e algoritmos muito eficientes. [Rabin 2005]

Para acompanhar tal evolução, a pesquisa e o estudo de novas interações homem-máquina são essenciais. Através do estudo e uso de novas tecnologias se rediscute os paradigmas de interação entre jogadores e jogos eletrônicos. Assim, a imersão, o envolvimento e o entretenimento atingem outros patamares.

Neste trabalho foi apresentado o jogo eletrônico *Eco Defense*, desenvolvido para uma superfície multitoque. O resultado obtido demonstrou a possibilidade de inovação através da utilização de conceitos já estabelecidos por diversos estudos. Além disso, a abordagem de desenvolvimento de software adotada promove a discussão sobre a adequação dos processos de arquitetura de sistemas para jogos eletrônicos. Por fim, o trabalho demonstrou a possibilidade de construção de interfaces naturais a partir de elementos de baixo custo disponíveis no mercado.

Referências

BOMARK, P. 2007. *Visualization and Prototyping of a Multitouch Display Device*. Master's thesis, Luleå tekniska universitet.

CORSO, J., YE, G., BURSCHKA, D., AND HAGER, G. 2008. A practical paradigm and platform for video-based human-computer interaction. *Computer, IEEE Computer Society 41*, 5, 48–55.

DOHERTY, M. 2003. A software architecture for games. *University of the Pacific Department of Computer Science Research and Project Journal 1*, 1.

FOLMER, E. 2007. Component-based game development. *Lecture Notes in Computer Science 2007*, 66–73.

2009. FTGL: A font rendering library for OpenGL. `http://homepages.paradise.net.nz/henryj/code/`. Acessado em junho/2009.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns*. Addison Wesley.

JORDÀ, S., KALTENBRUNNER, M., GEIGER, G., AND BENCINA, R. 2005. The reactable*. In *Proceedings of the International Computer Music Conference (ICMC 2005)*.

KALTENBRUNNER, M., BOVERMANN, T., BENCINA, R., AND CONSTANZA, E. 2005. Tuio: A protocol for table-top tangible user interfaces. In *Proc. of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*.

KATZOURIN, M., IGNATOFF, D., QUIRK, L., JR., J. L., AND JENKINS, O. 2006. Swordplay: Innovating game development through vr. *Computer Graphics and Applications, IEEE 26*, 6, 15–19.

2009. OpenCV: Open Source Computer Vision. `http://opencvlibrary.sourceforge.net/`. Acessado em junho/2009.

RABIN, S., Ed. 2005. *Introduction to Game Development*. Charles River Media.

2009. SDL: Simple directmedia layer. `http://www.libsdl.org`. Acessado em junho/2009.

SHEN, C., VERNIER, F., FORLINES, C., AND RINGEL, M. 2004. Diamondspin: An extensible toolkit for around-the-table interaction. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, 167–174.

2009. SOIL: Simple OpenGL Image Library. `http://www.lonesock.net/soil.html`. Acessado em junho/2009.

2009. Sparsh UI: A multitouch api for any multitouch hardware / software system. `http://code.google.com/p/sparsh-ui/`. Acessado em junho/2009.

STOY, C. 2006. *Game Programming Gems 6*. Charles River Media, ch. Game Object Component System.

2009. Touchlib: a library for creating multi-touch interaction surfaces. `http://nuigroup.com/touchlib/`. Acessado em junho/2009.

2009. Verve: Veneral purpose agents. `http://verve-agents.sourceforge.net/`. Acessado em junho/2009.