Um Caso Prático de Reabilitação de um Editor de Jogos Móveis Utilizando o Paradigma Model-View-Controller

Bruno Miguel Cardoso
Departamento de Informática
Universidade de Évora
Rua Romão Ramalho 59
7000-671 ÉVORA, PORTUGAL
+351 266 745373
b.m.pinto.cardoso@gmail.com

Teresa Romão
CITI, Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica, PORTUGAL
+351 212948536
tir@di.fct.unl.pt

Sumário

O presente artigo pretende expor um caso prático relacionado com o processo de reabilitação de uma aplicação informática, cuja utilização havia sido descontinuada devido a problemas, generalizados e graves, na arquitectura do software e na interface com o utilizador. A referida aplicação, designada por 2D Game Editor, tinha como objectivo proporcionar-se como um canal de comunicação de trabalho e informação entre os membros de uma equipa de produção de videojogos para telemóvel. Para este fim, o programa pretendia oferecer uma interface de utilizador amigável através da qual o utilizador, o Game Designer, pudesse definir certos parâmetros e variáveis que eram, depois de trabalhados, passados para os profissionais responsáveis pela programação final do dito jogo. A reabilitação desta aplicação foi atingida com recurso a técnicas de engenharia do software, mais precisamente aplicando um padrão de arquitectura especifico – o Model-View-Controller – e através do redesenho da interface respeitando os princípios da usabilidade. Assim, este artigo descreve o processo de selecção do padrão de arquitectura; a sua aplicação ao caso particular do 2D Game Editor;, explora os benefícios que daqui advieram para a tarefa de redesenhar a interface de utilizador e comprova o sucesso desta intervenção de reabilitação descrevendo alguns testes de utilizador que foram realizados.

Palavras-Chave

Usabilidade, Interface de utilizador, padrão de arquitectura Model-View-Controller, produção industrial de videojogos, engenharia de software.

1. INTRODUÇÃO

Num ambiente de produção industrial de videojogos para telemóvel é comum a atribuição de projectos a equipas que devem dar resposta a requisitos de natureza vária. Desde o som, às artes gráficas e à programação, a chave para o bom desempenho passa, obrigatoriamente, pela multidisciplinaridade. Não é sensato esperar que um conjunto de trabalhadores de uma determinada área profissional seja profícuo em todas aquelas que são exigidas pelo processo de produção de um jogo, pelo que uma equipa de constituição o mais heterogénea possível tem, à partida, mais hipóteses de ter sucesso e atingir mesmo alguma auto-suficiência em termos de requisitos de recursos humanos. Para potenciar o bom desempenho de uma equipa com estas características é necessário que a mesma assuma um comportamento holístico, ou seja, que actue como um todo no ambiente de produção industrial, completando com eficiência os objectivos que motivaram a sua criação. Ora, para que se atinja uma coerência deste nível, quase orgânico, as redes de comunicação internas da equipa devem estar o mais optimizadas possível e é neste ponto que a questão da heterogeneidade se pode impor como um obstáculo.

É evidente que pessoas com formação em áreas profissionais distintas possuem também linguagens e termos técnicos próprios e isto, a par da subjectividade inerente à linguagem natural, potencia que a utilização exclusiva de vias de comunicação convencionais, como a expressão oral ou escrita, possa contribuir para a introdução de equívocos na transmissão da informação entre os membros da equipa. Do ponto de vista do objectivo, o problema mais grave é que a acumulação destes equívocos propiciam o afastamento entre as características do produto final e as que foram originalmente pedidas.

Esta vulnerabilidade derivada da multidisciplinaridade complementar dentro de uma equipa, constitui a motivação para o desenvolvimento de soluções que permitam reduzir ao mínimo a transmissão de informações e requisitos por meios mais convencionais.

Assim, no caso particular do ambiente de produção de jogos para telemóvel da empresa YDreams Software, Lda., a concretização de uma destas soluções passou pela criação de uma aplicação denominada "*2D Game Editor*". Este programa foi concebido para se estabelecer como um meio optimizado de comunicação, permitindo a transmissão de material de trabalho e informação entre o *Game Designer* e a equipa responsável pela implementação final de um jogo. O cargo designado por *Game Designer* não é, necessariamente, ocupado por uma pessoa com competências técnicas profundas em ciências da computação, dado que as suas responsabilidades no processo de produção do jogo se reportam ao domínio da concepção original, da gestão do desenvolvimento – em termos de manutenção da coerência entre o conceito inicial e o produto final – e de afinação de alguns parâmetros que influenciam a experiência de jogo do utilizador. Por seu lado, as competências do pessoal que integra a equipa de programação também não coincidem, necessariamente, com as que são exigidas para o desempenho do cargo de *Game Designer*. Numa tentativa de aproximar e promover o entendimento e a coesão entre estes dois intervenientes distintos, que desempenham um papel fulcral na produção de um jogo para telemóvel, o "*2D Game Editor*" visa fornecer ao *Game Designer* um meio de manipular configurações e informação já dentro da implementação concreta do jogo – tarefa que, anteriormente, pertencia ao domínio exclusivo da equipa de programação. Claro está que, devido à formação não obrigatoriamente tecnológica do *Game Designer*, é preciso que esta manipulação decorra através de uma interface amigável, de fácil compreensão e utilização, que oculte todos os detalhes que estejam relacionados com questões de natureza tecnológica e que saiam fora das competências deste profissional.

Após a conclusão da 1ª versão do "*2D Game Editor*" foram surgindo problemas graves relacionados com a arquitectura do software e com a interface com o utilizador que se traduziram, em última análise, no seu abandono pelo utilizador final (*Game Designer*) e na relutância em aceitar de trabalhos de alteração do código por parte dos membros da equipa de programação. Deixa-se aqui a nota, nesta altura o programa continuava a ser usado, embora não directamente pelo Game Designer: este solicitava ao programador responsável pela implementação do "*2D Game Editor*" que levasse a cabo as tarefas que precisava – gorando-se assim o objectivo ultimo da aplicação.

Mais detalhadamente, desde a sua implementação original que o desenvolvimento desta aplicação não seguiu linhas de produção bem definidas. Não houve, desde o início, uma normalização da arquitectura, tendo ficado ao critério do programador a estruturação do código. Além disso, quando a necessidade a tal obrigava, o *Game Designer* requisitava a integração de novas funcionalidades no plano geral da aplicação, ou de outras alterações mais ou menos urgentes. Claramente, estas intervenções "*ad-hoc*" no código também contribuíram para o problema da sua desestruturação.

Por seu lado, o *Game Designer* acabou por se debater, após a requisição de algumas destas alterações "*ad-hoc*" no programa, com uma interface complexa e pouco clara, onde o caminho a seguir para atingir determinados objectivos nem sempre era intuitivo.

Era, pois, necessário reabilitar a aplicação para que as duas entidades que mais contacto tinham com ela, o *Game Designer* e a equipa de implementação, pudessem voltar a tirar partido do seu potencial. Estes objectivos foram conseguidos, numa primeira etapa, pela conformação estrutural do código aos preceitos de um padrão de arquitectura, o *Model-View-Controller*. Posteriormente, uma vez facilitado o trabalho de intervenção no código através desta padronização, a solução para o problema do utilizador final passou pelo estudo cuidado e pela reconstrução da interface com o utilizador.

O presente artigo visa, então, evidenciar os benefícios da adopção de um padrão de arquitectura para o código, na medida em que facilitou o processo de manipulação da interface que, por sua vez, veio tornar de novo possível a reintegração do "*2D Game Editor*" no processo de produção de videojogos na empresa.

O artigo encontra-se estruturado da seguinte forma: na secção 2 apresentam-se, de forma não exaustiva, alguns trabalhos relacionados com os tópicos abordados; na secção 3 dá-se um enquadramento da situação encontrada na empresa e dos critérios que levaram à selecção do padrão de arquitectura para o "*2D Game Editor*", e uma descrição das vantagens que advieram da nova arquitectação para os trabalhos de manipulação da interface e, por último, nas restantes secções são referidas as principais conclusões encontradas e algumas constatações sobre eventuais futuros desenvolvimentos.

2. TRABALHO RELACIONADO

Existe, no âmbito da engenharia do software, um conceito que teve particular importância na reabilitação do "*2D Game Editor*": o padrão. Entenda-se aqui este conceito, enquanto aplicável ao âmbito da engenharia do software, como uma solução padronizada, testada em contexto real, para problemas frequentes no desenvolvimento de software. Regra geral, um padrão não é um plano acabado e pronto a traduzir para código, mas sim uma descrição, um conjunto de normas vocacionadas para resolver uma determinada situação particular.

Dado que os problemas encontrados na aplicação se reportavam, maioritariamente, a problemas na arquitectura do software, procedeu-se ao estudo de vários padrões de arquitectura. Existem actualmente muitos padrões de arquitectura, porém, devido à orientação a sistemas de forte componente interactiva, para a escolha final sobressaíram dois:

- Presentation-Abstraction-Control – De acordo com [Buschmann96] este padrão define uma estrutura para sistemas de software interactivos sob a forma de uma hierarquia de agentes inter-cooperativos. Cada agente está responsável por um aspecto específico da

funcionalidade e é composto sempre por três componentes: *presentation*, *abstraction* e *control*. O tratamento do *input* e do *output* é combinado num só componente (*presentation*). Esta subdivisão separa os aspectos de interacção utilizador-computador de cada agente, do seu funcionamento nuclear e da sua comunicação com os outros agentes. Este modelo foi sugerido por [Coutaz87].

- Model-View-Controller – Segundo [Buschmann96], procura separar a implementação de uma aplicação interactiva em três componentes, por responsabilidades: o modelo é responsável pelos dados e pelos métodos de manipulação destes; a vista disponibiliza a informação contida no modelo que lhe corresponde e o controlador, por seu lado, reage a *input* do utilizador. O par vista-controlador forma a interface com o utilizador da aplicação. Este modelo foi sugerido no ambiente de programação do *SmallTalk* [Krasner88].

Tendo em conta as condicionantes do problema em mãos e as características exclusivas e os benefícios que adviriam da adopção de cada um destes paradigmas procedeu-se à selecção do modelo a utilizar conforme é descrito mais à frente neste artigo.

Por outro lado, dado que os problemas mais graves que a aplicação apresentava se verificavam ao nível da interface com o utilizador, foi necessário outro estudo, este vocacionado para a sua usabilidade. Seguiu-se um processo de desenho iterativo da interface do "*2D Game Editor*" [Nielsen93a], que envolveu o consecutivo melhoramento da sua interface com base em testes de utilizador, o que manteve o desenho centrado neste e nos seus requisitos.

No sentido de promover a usabilidade do sistema foram considerados os princípios de usabilidade descritos em [Dix03] e seguidas as heurísticas de usabilidade de Nielsen [Nielsen93b], que procuram sintetizar dez princípios gerais de usabilidade para uma interface de utilizador.

3. DESCRIÇÃO DO TRABALHO

O "*2D Game Editor*", desenvolvido para o ambiente Microsoft .Net e implementado na linguagem C#, foi concebido para se estabelecer como um meio optimizado de comunicação, permitindo a transmissão de material de trabalho e informação entre o *Game Designer* e a equipa responsável pela implementação final de um jogo móvel (ver a figura 1).

A título de exemplo, dado que o programa dispõe de muitas outras funcionalidades, é possível usar exclusivamente o "*2D Game Editor*" para efectuar o corte, a selecção e a ordenação das imagens estáticas – os fotogramas – que irão compor uma animação no jogo final.

Num passo final, o "*2D Game Editor*" produz um ficheiro em formato binário que vai ser lido por uma aplicação cliente utilizada pela equipa de programação, implementada em Java, e que contém o conjunto de todas as informações que tenham sido manipuladas pelo *Game Designer* (veja-se a figura 1).

Figura 1. O nicho do *"2D Game Editor"* na linha de produção de um jogo.

A adopção de uma mecânica de comunicação – do género da que é concretizada pelo "*2D Game Editor*" – para acelerar e fidelizar a produção de um jogo reveste-se de grande relevância, uma vez que possibilita ao *Game Designer* a manipulação directa da informação que será utilizada na versão final do jogo. Previnem-se assim situações em que a implementação do jogo se afasta em demasia dos critérios definidos inicialmente, obrigando o desenvolvimento a retroceder para etapas anteriores. Consequentemente, é também minimizada a discrepância – já de si inevitável – entre as características do jogo final e aquelas que foram projectadas na fase de planeamento, obtendo-se assim um resultado mais fiel.

3.1 Análise à estruturação encontrada

A programação do sistema, na altura do seu abandono pelo *Game Designer*, tinha já sido submetida a várias intervenções. A implementação original era reconhecível por uma abordagem mais ou menos sistemática no tocante ao fluxo de controlo e por alguns detalhes menos evidentes, como uma certa regularidade nas declarações de variáveis e de funções. Por outras palavras, vigorava uma aproximação estrutural, embora difusa, a um padrão: havia módulos unicamente dedicados a manter a informação e outros dedicados apenas à visualização.

Porém, com a adição de alterações mais recentes no código, essa padronização foi sendo gradualmente negligenciada em favor de antecipar a apresentação do resultado requerido.

Esbateram-se, assim, os resultados da tentativa inicial de seguir um padrão no desenvolvimento do programa: uma vez que este já não dominava os aspectos essenciais do código dificultava-se a tarefa de extrapolar, por mera leitura, uma abordagem coerente para manipular o código.

Impunha-se, dado que o programa estava em contínuo desenvolvimento e utilização, a adopção de um conjunto de regras que normalizasse a arquitectura do "*2D Game Editor*" e proporcionasse aos programadores, em última instância, alguma familiarização no âmbito dos detalhes do código.

Está, então, aqui patente a motivação para a procura de um padrão de arquitectura, por definição já testado e provado em contexto real.

3.2 Selecção do padrão

A selecção de um padrão para a arquitectura de qualquer sistema é um processo que não deve ser encarado de ânimo leve, já que uma escolha inadequada pode acabar

por trazer mais problemas ao programa, quer a nível de desempenho, quer de legibilidade do código.

No caso apresentado, o do "*2D Game Editor*", os factores que tiveram maior peso na decisão relacionavam-se essencialmente com considerações práticas. Procuravam-se, então, soluções para problemas de programação – sendo necessário um padrão que formalizasse a arquitectura da aplicação – e para problemas na interface de utilizador – ou seja, o padrão escolhido deveria acautelar sempre uma forte componente dedicada à interactividade. Havia ainda que levar em conta que o programa já estava em utilização, ainda que não pelo utilizador desejado, e havia que salvaguardar o modo de interacção mantendo-o, numa primeira fase, inalterado. Deste modo, dentre as características que o padrão tinha de cumprir, destacava-se uma de grande relevo: a flexibilidade.

Pelos objectivos expostos, e à primeira vista, qualquer um dos padrões orientados a aplicações altamente interactivas, o *Presentation-Abstraction-Control* e o *Model-View-Controller*, tinham o potencial de ser uma solução. No entanto, o *Presentation-Abstraction-Control* possui um tecido arquitectónico bastante rígido, obrigatoriamente composto por agentes, que por sua vez são tripartidos nos três componentes que nomeiam o padrão. Ao utilizá-lo estar-se-ia a tentar adequar a realidade do código e das funcionalidades já implementados a um conceito teórico. Seria de prever, pela natureza restringente do padrão *Presentation-Abstraction-Control* e pela implementação pouco estruturada que o "*2D Game Editor*" apresentava, que as alterações ao código – logo, os riscos de haver problemas – seriam bastantes extensas. Resumindo, este padrão gora desta forma o importante objectivo da flexibilidade.

Por exclusão de partes e dado que a sua especificação satisfazia em pleno os requerimentos apresentados, a escolha acabou por recair sobre o padrão de arquitectura *Model-View-Controller*.

Em [Buschmann96] refere-se que este padrão procura dividir uma aplicação interactiva, ou partes dela, em tríades de componentes (veja-se a figura 2):

- O modelo, que modela a lógica de negócio, contém a informação manipulada e, num reforço dos princípios do encapsulamento, providencia os métodos para operar sobre esta;
- A vista, que mostra a informação ao utilizador;
- O controlador, que gere o *input* do utilizador.

A consistência da informação entre o componente de mais baixo nível, o modelo, e o de mais alto, a vista, é garantida através de um sistema de propagação de alterações [Buschmann96].

Esta citada propagação de alterações traduz-se, em termos práticos, na protocolarização das comunicações entre os componentes salvaguardando-se assim a facilidade de integração de novos componentes, desde que estes estejam conformes ao *standard* vigente.

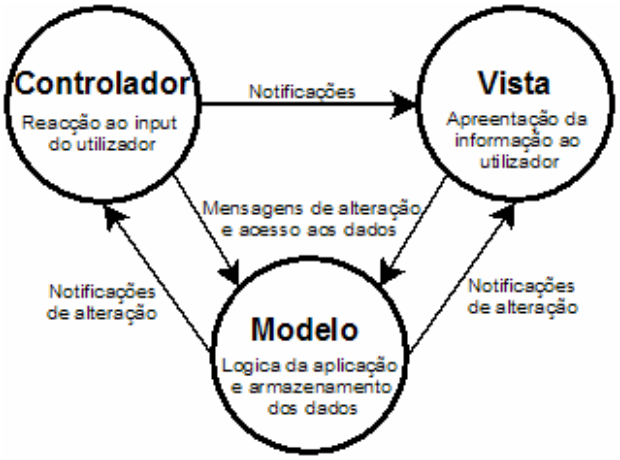

Figura 2. Diagrama da tríade basilar do padrão de arquitectura *Model-View-Controller*.

De particular interesse, o padrão Model-View-Controller preconiza a possibilidade de modificar os componentes vista e controlador sem que seja afectada a lógica da aplicação associada com os dados [Avgeriou05]. Na prática, isto significa que é possível alterar vários aspectos da interactividade da aplicação sem quaisquer preocupações com os dados.

De um ponto de vista mais relacionado com o problema que se procurava resolver, este padrão conta com especificações suficientemente explicitas a nível de distribuição de responsabilidades entre os componentes do sistema e irrestritas o bastante para permitir alguma flexibilidade na implementação concreta do código. Esta flexibilidade veio a revelar-se de suma importância dado que permitiu manter partes da aplicação com a estruturação inicial sem comprometer a integração das restantes partes que sofreram o processo de adequação da arquitectura às novas exigências.

3.3 O MVC e as metáforas na interface

Considerando que os utilizadores não sabem, de forma inata, como operar sobre um sistema, vem do ramo da psicologia cognitiva que quanto mais familiar for a interface e respectivo modo de interacção, menor é a curva de aprendizagem e a resistência à utilização manifestadas pelos utilizadores finais. As experiências mais naturais e imediatas para qualquer pessoa são aquelas do quotidiano e isto torna estranhas, logo à partida, as experiências que estejam limitadas a um dispositivo de *output* como o ecrã de um computador. Entenda-se aqui, então, o termo "metáfora" como a implementação em código de um conceito de manipulação ou visualização, ou seja a sua redução à realidade informática com as respectivas restrições. Segundo [Szabo95], a utilização de metáforas no desenho de interfaces visa facilitar a aprendizagem, a orientação e a formação e manutenção de um modelo mental da aplicação. Ao adoptar-se uma metáfora para reger o desenho de uma interface, ou partes desta, está-se na prática a tentar modelar a interacção da aplicação fazendo recurso a memórias empíricas que sejam familiares ao potencial utilizador.

Exemplificando, no cumprimento de determinada funcionalidade, o "*2D Game Editor*" requeria que se efectuasse o corte de várias partes de imagens, fotogramas, e se procedesse numa segunda fase à montagem destes cortes em animações sequenciais. Uma possível metáfora que transponha este processo para a realidade de uma interface de utilizador seria a da montagem de um puzzle, onde as peças são exibidas isoladamente antes de serem dispostas na ordem final.

O trabalho envolvido na implementação deste conceito foi facilitado, na medida em que a arquitectura da aplicação já se encontrava modularizada, nesta fase, sob as normas do padrão *Model-View-Controller*. Esta padronização permitiu a separação total, na fase de planeamento, dos raciocínios necessários à concepção da mecânica dos fluxos de controlo e de transmissão da informação. Clarificando, ao planear os detalhes específicos do componente modelo a usar, foi possível definir com precisão a informação que era necessário guardar, bem como as possíveis manipulações de que esta seria alvo sem ter presentes quaisquer outras preocupações que escapassem ao domínio da manutenção e gestão de baixo nível desses dados. De igual modo, aquando da definição da vista e do controlador, foi possível estudar a forma mais amigável de disponibilizar a informação ao utilizador e de processar o *input*.

Esta quase total separação de considerações não se limitou às fases iniciais de planeamento e concepção, tendo-se alargado mesmo às de implementação e planeamento de aspectos de mais baixo nível, como a gestão da memória do telemóvel necessária para processar as imagens binarizadas que foram manipuladas pelo "*2D Game Editor*".

É uma clara vantagem que, ao apartar-se o desenvolvimento do software em momentos distintos, está-se a salvaguardar que a atenção do programador fique centrada na tarefa em mãos, sem preocupações muito prementes em áreas mais globais como a integração coerente das diversas componentes do software e da transmissão da informação entre estas. Como reforço desta modularização do código segundo as responsabilidades, foram definidos standards para a manipulação e transmissão da informação entre componentes. Garante-se assim que todas as comunicações efectuadas entre os modelos, vistas e controladores implementados seguem o mesmo arquétipo, o que facilita futuras intervenções a nível do código do programa.

3.4 O MVC e o fluxo de controlo

Em paralelo com os problemas associados a uma interface que não permitia tirar partido dos conhecimentos empíricos do *Game Designer* e que foram resolvidos com a já referida inclusão das metáforas na interface, havia problemas de outra ordem e com resolução menos imediata. Havia certas funcionalidades, como o processo de exportação da informação trabalhada para formato binário, que só eram conseguidas quando se concluía uma série de acções, algumas sequenciais, outras – tipicamente relacionadas com a selecção de um conjunto de opções – que eram executadas em paralelo. A confusão que estes procedimentos geravam era devida, sobretudo, a exigências desnecessárias sobre a memória cognitiva do utilizador. Explicitando, era necessário que este tivesse presente, num mesmo instante, opções que já estavam tomadas e informações de natureza distinta, não directamente relacionáveis. A solução para o caso geral destas dificuldades passava pela cisão dos processos em passos distintos, no fundo, a implementação de um "guia" ou "*wizard*".

Regra geral, a implementação destes guias foi conseguida através da inclusão de uma nova tríade *Model-View-Controller* em que o componente modelo se limitava a guardar a meta-informação que era recolhida nos sucessivos formulários que compunham o guia. Caracteristicamente, a adição de novas tríades a uma aplicação padronizada em *Model-View-Controller* veio a revelar-se um processo simples e rápido. Devido à modularidade e encapsulamento das responsabilidades e dados inerentes a este padrão. Cada tríade simples do *Model-View-Controller* age, potencialmente, como uma pequena aplicação "per se" e, portanto, com *input* e *output* bem definidos. Tendo em conta esta padronização do *input* e *output*, corolário da aplicação do *Model-View-Controller* à arquitectura do sistema, é fácil encontrar o nicho de inclusão ideal para novas tríades, com funções bem definidas.

Outra possibilidade para orientar o utilizador em direcção à conclusão de determinado procedimento é, assim que se detecte que este procedimento foi iniciado, restringir as possibilidades de interacção com a aplicação, mas sem afastar a atenção do utilizador do ambiente de trabalho da aplicação, como aconteceria se fossem utilizados *wizards*. Esta restrição consegue-se, por exemplo, através da desactivação de certas *widgets*, como botões ou comboboxes, com funcionalidades de interesse divergente e não relacionado com a conclusão do processo iniciado. A maioria destas alterações, num ambiente padronizado segundo o *Model-View-Controller*, requer apenas que a atenção do programador se foque unicamente sobre os componentes da vista e do controlador. Assim que é dado início ao processo, o controlador notifica imediatamente a vista para desactivar os controlos. Por seu lado, quando o passo final do processo fica concluído, o mesmo controlador torna a notificar a vista, agora com o objectivo oposto, devolvendo assim a interface ao seu estado original. Todas as alterações efectuadas ao modelo ficam unicamente relacionadas com o instante da utilização, isto é, são desencadeadas quando o utilizador efectua determinada acção e podem ser, deste modo, planeadas em separado das acções que definimos para restringir a interactividade.

3.5 Testes de utilizador

O processo de desenho iterativo seguido ditou que ao longo do desenvolvimento dos trabalhos de reabilitação no "*2D Game Editor*", houvesse vários períodos de teste e reorientação dos desenvolvimentos. Lembre-se aqui que este software era para utilização interna da empresa pelo

que tanto o seu âmbito de utilização, resultados esperados e mesmo o próprio utilizador final estavam, à partida, bem identificados. Foi com base nessa premissa que se planearam sessões de testes com especial enfoque sobre a vertente da usabilidade. Assim, o programa foi sujeito a avaliações pela equipa de testes da empresa, mais vocacionada para medir a eficiência e a fiabilidade do programa, e pelo utilizador final, onde se deu especial atenção às opiniões subjectivas e à relação desenvolvida com a ferramenta.

Em termos de eficiência e fiabilidade, verificou-se que ocorriam menos erros na execução de processos mais complexos, vindo isto a provar a eficácia das restrições da interacção quer através de mecanismos de wizard que guiam o utilizador expondo as escolhas de forma simples, quer através de outros modos como a desactivação de controlos ou a actualização de outros aspectos da interface, como por exemplo, caixas de texto com ajuda contextual.

Ao primeiro contacto, a equipa de testes foi capaz de operar o sistema com razoável desempenho, tendo-lhe sido unicamente explicados o âmbito de utilização do programa ao nível de *input* e *output*, e facultada a informação mínima, em formato textual, sobre o modo de funcionamento da aplicação, sem praticamente nenhumas indicações verbais. Outro aspecto também indiciador deste sucesso foi a quantidade de informação sobre o modo de operação que as equipas de testes, e outros utilizadores ocasionais, conseguiram apreender. À maioria das questões colocadas bastava responder uma vez, ou seja, a utilização era facilmente compreendida e integrada no quadro geral da interactividade com as restantes funcionalidades da aplicação.

No capítulo da relação do utilizador final com o software, este sempre mereceu críticas positivas e alguns reparos que, quando devidamente trabalhados, conduziram a melhorias significativas na interface com o utilizador.

4. CONCLUSÕES E TRABALHO FUTURO

O processo de conformação arquitectural da programação do "*2D Game Editor*" aos cânones do padrão de *Model-View-Controller* veio permitir a reabilitação desta ferramenta. A modularização empreendida veio melhorar significativamente a modificabilidade da aplicação, facilitando a reconstrução da interface do sistema e as futuras alterações que venham a ser requeridas. O processo de implementação de código torna-se mais fácil. Nomeadamente, do ponto de vista específico das modificações na interface e na interacção com o utilizador, o padrão de arquitectura *Model-View-Controller* prima pela simplicidade com que expõe ao programador as componentes da aplicação que requerem atenção. Intervenções no sentido de introduzir alterações de cariz puramente estético na interface, apenas exigem que o componente da vista sofra alterações; outras que requeiram manipulações mais complexas, eventualmente por reacção a eventos, demandam que as alterações sejam executadas nos componentes da vista, controlador e, em alterações mais profundas na interactividade, no modelo.

Adicionalmente, tanto a vista, o modelo ou o controlador de uma determinada tríade podem ser directamente alterados sem que seja necessário modificar também os restantes componentes, bastando para o efeito que seja respeitado o standard em vigência nas intercomunicações. Tudo isto permite centrar a atenção do programador nas necessidades do utilizador, assumindo-se estas como as linhas orientadoras do desenvolvimento do produto e alcançando-se um resultado final mais congruente face às expectativas.

As alterações operadas a nível da interface do "*2D Game Editor*" mereceram, da parte do *Game Designer*, as críticas mais favoráveis, estimando-se um aumento quer na produtividade, quer na qualidade da experiência de utilização. Esta aplicação é um projecto em contínuo desenvolvimento dado que os requisitos actuais não serão os mesmos que serão impostos no futuro. Como se retira da leitura do artigo, visto que a programação da aplicação ficou assente numa base altamente modificável e previsível, rentabiliza-se o trabalho realizado no esforço constante de mantê-la actualizada.

5. AGRADECIMENTOS

Os autores gostariam de agradecer à YDreams (http://www.ydreams.com) por todo o apoio prestado no desenvolvimento do trabalho apresentado neste artigo. Estendemos ainda os nossos agradecimentos a Miguel Almeida, gestor de projectos na YDreams, pela orientação do trabalho; Vítor Duarte pela primeira implementação do *2D Game Editor* e a Tiago Carita, o *Game Designer*.

6. REFERÊNCIAS

[Avgeriou05] Avgeriou, P. and Zdun, U., Architectural Patterns Revisited – A Pattern Language. In Proceedings of 10th European Conference on Pattern Languages of Programs (EuroPlop 2005),Irsee, Germany, July, 2005, pp. 1-39. (http://www.infosys.tuwien.ac.at/Staff/zdun/publications/ArchPatterns.pdf)

[Buschmann96] Buschmann, F. et al., Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons Ltd, West Sussex, England, 1996.

[Coutaz87] Coutaz, J., PAC, an object oriented model for dialog design. In H. J. Bullinger and B. Shackel, editors, Human- Computer Interaction – Interact'87, North Holland, Amsterdam, 1987, pp. 431-436.

[Krasner88] Krasner, G. E. and Pope, T., A cookbook for using the model-view-controller user interface paradigm in SmallTalk. JOOP, 1(3), August, 1988, pp. 26-49.

[Nielsen93a] Iterative User Interface Design. IEEE Computer Vol. 26, N. 11, November, 1993, pp. 32-41.

[Nielsen93b] Nielsen, J., Usability Engineering. Boston, MA: Academic Press, 1993.

[Dix03] Dix, A., Finlay, J., Abowd, G. and Beale, R., Human-Computer Interaction, London, Prentice-Hall, 2003.

[Szabo95] Szabo, K., Metaphors and the user interface retrieved from www.katalinszabo.com/metaphor.htm.