Universidade de Évora
Honesto Estudo com Longa Experiência Misturado
Departamento de Informática

.: 2D Game Editor On-line :.

A Edição de Videojogos em Contexto Web

Bruno Miguel de Lemos Ribeiro Pinto Cardoso
Aluno de Mestrado nº 17567

Évora, Abril de 2010

Orientadora:
Professora Doutora Teresa Romão

Universidade de Évora
Honesto Estudo com Longa Experiência Misturado
Departamento de Informática

.: 2D Game Editor On-line :.

A Edição de Videojogos em Contexto Web

Bruno Miguel de Lemos Ribeiro Pinto Cardoso
Aluno de Mestrado nº 17567

Évora, Abril de 2010

Orientadora:
Professora Doutora Teresa Romão

Agradecimentos

Agradeço:

- À Professora Doutora Teresa Romão, pela orientação e revisão do presente trabalho.
- Ao professor José Carlos Danado, pelas ajudas com a revisão bibliográfica.
- À YDreams, pela oportunidade que me deu, em 2007, para ali realizar o meu estágio curricular.
- Ao Tiago Carita, da empresa Blue Shark Studios, pelas opiniões e comentários sempre pertinentes.
- Aos meus pais, António Cardoso e Izilda Cardoso, pelos incentivos, apoio e opiniões construtivas.
- A Maria Carvalho, pelo apoio e paciência.

Tabela de Conteúdos

Lista de Figuras

Lista de Tabelas

Resumo

Descreve-se, no presente trabalho, os esforços envidados no sentido de criar uma solução informática generalista, para os problemas mais recorrentes do processo de produção de videojogos 2D, baseados em sprites, a correr em plataformas móveis. O sistema desenvolvido é uma aplicação *web* que está inserida no paradigma *cloud-computing,* usufruindo, portanto, de todas as vantagens em termos de acessibilidade, segurança da informação e manutenção que este paradigma oferece actualmente.

Além das questões funcionais, a aplicação é ainda explorada do ponto de vista da arquitectura da implementação, com vista a garantir um sistema com implementação escalável, adaptável e de fácil manutenção.

Propõe-se ainda um algoritmo que foi desenvolvido para resolver o problema de obter uma distribuição espacial optimizada de várias áreas rectangulares, sem sobreposições nem restrições a nível das dimensões, quer do arranjo final, quer das áreas arranjadas.

2D Game Editor On-line
Video Game Edition in Web Context

Abstract

This document describes the efforts taken to create a generic computing solution for the most recurrent problems found in the production of two dimensional, sprite-based videogames, running on mobile platforms. The developed system is a web application that fits within the scope of the recent cloud-computing paradigm and, therefore, enjoys all of it's advantages in terms of data safety, accessibility and application maintainability.

In addition to the functional issues, the system is also studied in terms of its internal software architecture, since it was planned and implemented in the perspective of attaining an easy to maintain application, that is both scalable and adaptable.

Furthermore, it is also proposed an algorithm that aims to find an optimized solution to the space distribution problem of several rectangular areas, with no overlapping and no dimensinal restrictions, neither on the final arrangement nor on the arranged areas.

Capítulo 1. Introdução

1.1. Contexto Geral

Em crescente actividade desde há cerca de trinta anos, o mercado do entretenimento electrónico - do qual os jogos de vídeo são os representantes mais carismáticos - movimentou, apenas no ano de 2007, 41 biliões de dólares no mundo inteiro, ultrapassando mesmo os valores apresentados pela indústria cinematográfica para igual período (Alves, 2008). A longa sequência de casos de sucesso que culminou com esta conquista provou, de forma indisputável, a influência que a informática pode ter dentro de um dos mercados mais prolíferos da sociedade contemporânea globalizada. Esta vitalidade não passa ao lado dos mercados europeus, nem mesmo dos portugueses, visto que Portugal foi o país que teve o maior crescimento no mercado dos videojogos, segundo um balanço da *Media Control GfK International* sobre o crescimento do mercado dos videojogos na Europa, relativamente ao primeiro semestre de 2009 (Arnaud, 2009).

Esta conjuntura sugere uma promessa de rentabilidade que desperta, naturalmente, o interesse generalizado de empresas e privados, e propicia um ambiente de mercado altamente competitivo e pleno de vitalidade. Como consequência, a actividade comercial nesta área torna-se muito exigente em vários aspectos, ficando o sucesso de um participante condicionado pela sua capacidade de adaptação e aplicação eficaz de recursos. Em simultâneo, abre-se no mercado margem, quer para a competição directa, quer para a colaboração estratégica entre grandes estúdios e pequenas empresas produtoras, no sentido de atingir objectivos mais ambiciosos.

Se se realizar uma análise deste mercado dos videojogos, na perspectiva do produto final, observa-se uma variabilidade imensa, tão grande quanto a variabilidade do próprio consumidor alvo, que pode ir desde o público geral até qualquer outro género de entidades com interesse na área do entretenimento. Naturalmente, este grau de disparidade impossibilita que se possa delimitar um conjunto objectivo de requisitos que, a serem observados, garantam o bom desempenho de um jogo específico no mercado. No fundo, há demasiadas variáveis em jogo para se poder antecipar, com algum grau de certeza, a resposta do mercado ao nível das vendas e da popularidade atingidas por determinado título.

Apesar da aparência caótica do mercado do entretenimento em geral, e dos videojogos em particular, existe alguma ordenação observável e que, embora não podendo ser considerada absoluta, impõe alguma consistência nas ofertas. Assim, de um modo muito abrangente, podemos referir que os jogos se inserem, na sua maioria, dentro de um género: *"acção"*, *"plataformas"*, *"primeira pessoa"*, *"puzzles"*, etc. Esta classificação contribui para a satisfação de um comprador na medida em que lhe permite adquirir um título que, estando enquadrado num dos géneros da sua preferência, possivelmente será mais do seu agrado. No entanto, frisa-se novamente que, mesmo dentro de um género existe demasiada variação para que se possa inferir, com certeza, que um título em particular agradará a um determinado jogador, mesmo que este último tenha manifesta preferência pelo

género a que pertence o jogo adquirido. Existem, aliás, certos casos em que jogos de vídeo com características pertencentes a mais do que um género, ou não enquadráveis em nenhum já existente, tiveram mais sucesso do que outros mais facilmente classificáveis, tendo conduzido, inclusivamente, à criação de novos géneros de jogo.

Existe uma classe de jogos em concreto, os jogos casuais que, pela sua facilidade de aprendizagem e utilização, se tornam particularmente apelativos a uma grande audiência, dado que estas características lhes permitem estabelecer-se como alternativa a passatempos mais tradicionais. Clarificando, pela sua simplicidade conceptual e pelo facto de não exigirem grandes esforços da parte do jogador para poderem ser utilizados, quer a nível cognitivo quer de tempo, qualificam-nos como uma actividade que pode ser realizada em momentos de espera, durante viagens, ou até em simultâneo com a execução de outras actividades que não exijam a atenção total do jogador.

Em adição, se se considerar este género de jogos a correr em plataformas altamente difundidas e com aceitação privilegiada no mercado – como é o caso dos telemóveis – pode-se ainda alargar mais o espectro dos potenciais consumidores. A título ilustrativo, pode-se referir que, em 2008, mais de 60% da população portuguesa dentro da faixa etária dos 10 aos 65 anos, era utilizadora de telemóvel, e que esta tendência se acentua quanto mais nos aproximamos da faixa dos 16 aos 24 anos, com uma taxa de utilização que ronda os 97% (Instituto Nacional de Estatística, 2009).

Interpretando os factos mencionados, pode-se afirmar que se tem um potencial cliente em, virtualmente, 97% da população portuguesa na faixa dos 16 aos 24 anos, ou em 60% da faixa mais alargada dos 10 aos 65 anos, e um possível ambiente de utilização em qualquer lugar onde um telemóvel possa ser usado. Assim, os jogos casuais, além de serem um produto facilmente escoado, têm ainda outros aspectos interessantes que justificam a atenção de que têm sido alvo. Refere-se, como exemplo, o reduzido esforço de produção quando comparado com títulos de outros géneros, desenvolvidos para plataformas mais poderosas. Esta facilidade traduz-se, em termos práticos para o produtor, numa menor exigência em termos orçamentais, quer ao nível de recursos humanos, quer materiais, quando comparado com a produção de títulos mais evoluídos que requerem também um perfil de utilizador/jogador mais dedicado e exigente.

As razões apresentadas justificam, então, o facto de este ter sido o género de jogos cujo mercado mais cresceu nos últimos cinco anos. Embora se tenham registado recentemente algumas constrições motivadas pelo surgimento constante de novos dispositivos com características inovadoras, e pela adopção de novos modelos de distribuição, como é o caso da *App Store* para o *iPhone*, o comércio desta linha de produtos continua a ser lucrativo (Alves, 2008).

Dentro da produção de jogos casuais, destacam-se os jogos 2D, com entidades de jogo baseadas em sprites, que continuam a ser uma opção muito interessante, em especial pela menor exigência de capacidade de processamento da parte das plataformas onde correm e pela relativa facilidade de implementação.

Tendo em atenção esta proliferação do mercado dos videojogos em anos recentes, é natural que surjam vários interessados na sua produção, desde profissionais, geralmente inseridos em contexto industrial, aos amadores sem fins lucrativos.

1.2. Descrição e Contexto

Independentemente de quem o tenha produzido, para que um jogo seja bem aceite pelo público, terá de corresponder a expectativas cada vez mais elevadas a nível de qualidade, desempenho e potencial de entretenimento (sendo este último um critério muito subjectivo). Estas exigências motivam o recurso, da parte dos produtores, a várias soluções para resolver os inúmeros problemas com que se deparam, quer a nível tecnológico, quer de gestão de equipas (no caso da produção empresarial).

Dado que existe uma oferta muito alargada de recursos informáticos hoje em dia, é necessário proceder-se a uma escolha criteriosa das tecnologias e ferramentas que serão utilizadas na implementação. Esta escolha reveste-se de importância estratégica, na medida em que a utilização de recursos cujas características estejam alinhadas com os objectivos dos projectos em desenvolvimento, pode levar a uma poupança nos tempos de implementação, bem como ao aumento da eficiência produtiva da equipa.

A título de exemplo desta observação, refere-se que uma das questões de natureza tecnológica mais comuns na produção de jogos para telemóvel, é a da portabilidade, ou seja, conseguir que a mesma implementação de um jogo corra em várias plataformas, com modelos e arquitecturas diferentes, sem necessidade de alterar o código já escrito. Então, a ilustrar o tipo de contribuição que podem ter as características específicas das ferramentas informáticas, como uma linguagem de programação, para atingir os objectivos de determinado projecto, destaca-se o caso particular do Java que, pela elevada portabilidade – apanágio desta linguagem –, permite minimizar o esforço de programação para garantir um jogo também portável. Refere-se que um dos lemas desta tecnologia é «write once, run everywhere», – portanto, «escrever uma vez, correr em qualquer sítio» – e isto traduz precisamente a preocupação com a problemática da portabilidade das aplicações. Assim, e desprezando as eventuais discrepâncias que possam existir entre versões e implementações da máquina virtual de Java, para que determinado dispositivo possa correr um jogo programado nesta linguagem, basta que tenha suporte para a referida máquina virtual, a JVM.

No entanto, as questões meramente tecnológicas são apenas parte do conjunto de problemas que uma empresa de produção de videojogos tem de enfrentar para conseguir dar resposta às exigências do mercado. A questão é que, independentemente do género de jogo onde se insira, da plataforma a que se destine ou das tecnologias escolhidas para a implementação, a produção de um jogo que se enquadre nos padrões de qualidade actuais exige a conjugação do trabalho de profissionais de áreas muito distintas, como é o caso da composição artística – gráfica e sonora –, da escrita criativa de guiões e da programação do produto final.

Em termos humanos, para que a equipa consiga ter um desempenho conjunto, é de todo o interesse que haja uma pessoa que se responsabilize inteiramente pelo conceito do jogo - o *Game Designer*. Embora este profissional não pertença obrigatoriamente a nenhum dos domínios profissionais referidos anteriormente, sejam artísticos ou tecnológicos, cabe-lhe a definição do conceito inicial do jogo e a tomada das decisões, em todas as vertentes da fase de implementação, que mantenham as características do produto alinhadas com as do conceito inicial. Resumindo, é sobre o *Game Designer*, o arquitecto geral do jogo, que é colocada a responsabilidade de garantir o bom desempenho deste e a fidelidade com que este implementa o conceito. Esta pode ser uma questão complexa, dado que um jogo percorre um longo caminho desde a fase de conceptualização até ao momento da sua colocação no mercado. Deste modo, desde a criação do conceito original - responsabilidade do *Game Designer* - até à implementação final - a cargo da equipa de programação - a linha de produção de um jogo de vídeo requer o contributo organizado dos esforços especializados de cada grupo profissional envolvido. Assim, para que se consiga um desempenho frutuoso numa equipa com este nível de heterogeneidade, é forçoso que sejam envidados esforços no sentido de atingir uma complementaridade eficiente e destas premissas surgirão, inevitavelmente, obstáculos que se virão somar ao conjunto de problemas que afectam habitualmente a generalidade dos projectos organizados de produção informática. Dentre estes, podem-se enumerar:

- Problemas de natureza tecnológica, que derivam de limitações, ou comportamentos inesperados e muitas vezes não documentados, da parte das tecnologias que foram seleccionadas para o desenvolvimento;
- Problemas de natureza humana e profissional, como as dificuldades a nível de comunicação dentro da equipa de desenvolvimento, que podem surgir pelo recurso a termos técnicos específicos a cada área profissional (podendo assim dificultar a transmissão eficaz de ideias, perspectivas e conhecimentos);
- Dificuldades de outros géneros, por exemplo, a nível de gestão de orçamentos, recursos materiais e humanos.

O maior risco que estas questões podem acarretar é o de propiciarem o afastamento entre as características do produto final e aquelas que compuseram o conceito inicial. Isto porque, antes do arranque da produção do título a cargo de determinada equipa, já o conceito de jogo teve de ser discutido e avaliada a sua perspectiva de rentabilidade. Caso o resultado final seja demasiado discrepante, isso pode colocar em risco a eficiência da equipa e até a própria rentabilidade do projecto, se de algum modo o resultado não corresponder às expectativas do mercado.

Neste sentido, torna-se obrigatório tomar medidas que permitam minimizar os riscos do projecto, em especial aqueles que ponham em causa a fidelidade da implementação ao conceito do jogo. Existem várias alternativas diferentes que permitem ao *Game Designer* fazer um levantamento em tempo útil destes riscos, desde as mais convencionais, às mais inovadoras. Entre as primeiras podem referir-se as reuniões regulares de *follow-up* com a equipa de implementação, no sentido

de permitir ao próprio *Game Designer* inteirar-se, em primeira-mão, dos progressos e eventuais constrangimentos encontrados. Outra possibilidade, que não obriga à comparência em tempo real, local ou remotamente, é o preenchimento assíduo de fichas de relatório por parte dos programadores. No entanto, como será evidente, estas medidas exigem um esforço organizativo adicional e a disponibilidade dos recursos e, por essas razões, não configuram a solução ideal para projectos com planificação pouco flexível e, muitas vezes, com prazos apertados.

Ainda, dado que, como já foi referido, o cargo de *Game Designer* não é ocupado necessariamente por uma pessoa da área tecnológica, menciona-se novamente a dificuldade que pode representar a comunicação dentro da equipa de produção, enquanto grupo interdisciplinar. A participação de problemas tecnológicos no sentido dos programadores para o *Game Designer* assume uma importância primordial, na medida em que a sua comunicação atempada pode permitir que sejam tomadas medidas importantes em caso de contratempos sérios, como a troca de tecnologias ou a alteração de alguns aspectos no conceito do jogo, que possibilitem salvaguardar o trabalho já realizado. Por outro lado e no sentido oposto das comunicações, do *Game Designer* para a restante equipa, pode ser necessário transmitir instruções e directivas à equipa de programação, para que estas sejam integradas no produto em desenvolvimento. Visto que um jogo é uma produção com uma grande componente artística, depreende-se a natureza por vezes subtil e abstracta dos conceitos e ideias que o *Game Designer* tem, por vezes, de passar para a equipa de programação e daí o grande interesse em que estas comunicações se processem da forma mais transparente, eficiente e livres de falhas, quanto possível.

Se fosse possível garantir, da parte do *Game Designer*, um controlo contínuo e permanente dos desenvolvimentos do *software*, durante a programação, seria possível também a detecção atempada de eventuais desvios e proceder-se logo a uma reorientação que permitisse contornar os eventuais comprometimentos do projecto. Porém, como este profissional tem outras funções e responsabilidades além da manutenção do conceito do jogo e de garantir que os desenvolvimentos convergem no seu sentido, e ainda porque, como já foi referido, também não lhe é exigido que tenha competências tecnológicas profundas, a sua inclusão eficaz no processo de implementação do *software* pode não ser produtiva.

A solução mais prática e imediata para resolver esta problemática, consiste na adopção de soluções informáticas que, por intermédio de uma interface amigável e com um recurso moderado a conceitos e termos exclusivamente tecnológicos, permitam ao *Game Designer* uma intervenção directa no processo de produção.
Estas soluções, comummente designadas "editores de jogos", permitem aos utilizadores, tipicamente o *Game Designer*, ter uma visão global da estruturação informática do projecto. No caso ideal, os editores geram um *output* que é directamente integrável no restante código do jogo, cujo desenvolvimento está a cargo da equipa de programação. Este *output* pode conter informação como imagens, animações e sons, e *meta-informação*, tais como as entidades de jogo a implementar, os cenários, a definição de eventos e as regras de jogo.

A *meta-informação* manipulada por estes editores traduzir-se-á, no baixo-nível, em orientações – ou restrições – tecnológicas para a programação do próprio jogo e podem passar por parametrizações a que o jogo deve obedecer, até à definição de classes, interfaces e respectivas propriedades. Desta forma, reduz-se a possibilidade de ocorrência de desvios e, consequentemente, promove-se a criação de um produto final mais fiel à conceptualização do *Game Designer*.

Do ponto de vista da utilização do editor, pretende-se que o trabalho realizado seja levado a cabo de forma gráfica e intuitiva, optando-se, sempre que possível, pela representação visual da informação. Por fim, o *output* será disponibilizado aos programadores, com as informações devidamente convertidas para um formato alvo – binário ou textual – e encapsuladas em recursos de fácil acesso programático.

No entanto, dada a própria natureza do mercado dos videojogos onde cada título tem características exclusivas, é também virtualmente impossível definir um conjunto de requisitos que um editor tenha de satisfazer e que garanta respostas adequadas a todas as exigências das linhas de produção de todo e qualquer jogo. Tal conjunto é, pura e simplesmente, demasiado extenso para ser exequível. A título de exemplo, pode referir-se que, para a produção de um jogo do tipo *plataformas*, seria interessante que o editor de jogos implementasse um motor físico. Por outro lado, os jogos do tipo *puzzle*, como o *Sudoku* já não utilizarão este motor. Já um jogo como o *Tetris*, apesar de poder ser classificado como um *puzzle*, poderá fazer uso desse mesmo motor, embora precise apenas de uma versão simplificada, que dê suporte a cálculos de gravidade, mecânica e colisões simples. Sintetizando, pode afirmar-se que existe demasiada variabilidade nos títulos comercializados para que os processos de produção individuais possam estar assentes somente sobre uma ferramenta de apoio ao desenvolvimento do *software*.

Numa visão diametralmente oposta, se se limitar o conjunto de requisitos ao mínimo essencial, abstraindo totalmente o editor de qualquer contexto de produção específico com vista a tornar mais abrangente as suas possibilidades de utilização, obter-se-á uma aplicação minimalista que também não dará, só por si, o apoio desejado em nenhum contexto.

Uma alternativa mais interessante seria procurar um compromisso que permita retirar o melhor dos dois quadros e atenuar as respectivas desvantagens, ou seja, construir uma aplicação que, não sendo demasiado complexa do ponto de vista da programação, se prove suficientemente maleável para se constituir como uma base sólida para futuros desenvolvimentos que lhe acrescentem valor e utilidade.

As ferramentas utilizadas na produção de videojogos costumam apresentar-se, hoje em dia, em duas variantes: colecções de pequenos aplicativos independentes, cada um com a sua própria funcionalidade e tipos de *input* e *output*; ou aplicações de maior escala, que integram um maior número de funcionalidades. No caso das soluções pequenas, estas acabam por se constituir quase como uma biblioteca de recursos que vão sendo criados, descartados e reutilizados conforme a necessidade obrigue. Ora, muitas vezes, é necessário que haja integração das várias

funcionalidades e o procedimento usual é o consumo posterior – como *input* – dos *outputs* dos programas utilizados anteriormente (ver Figura 1).

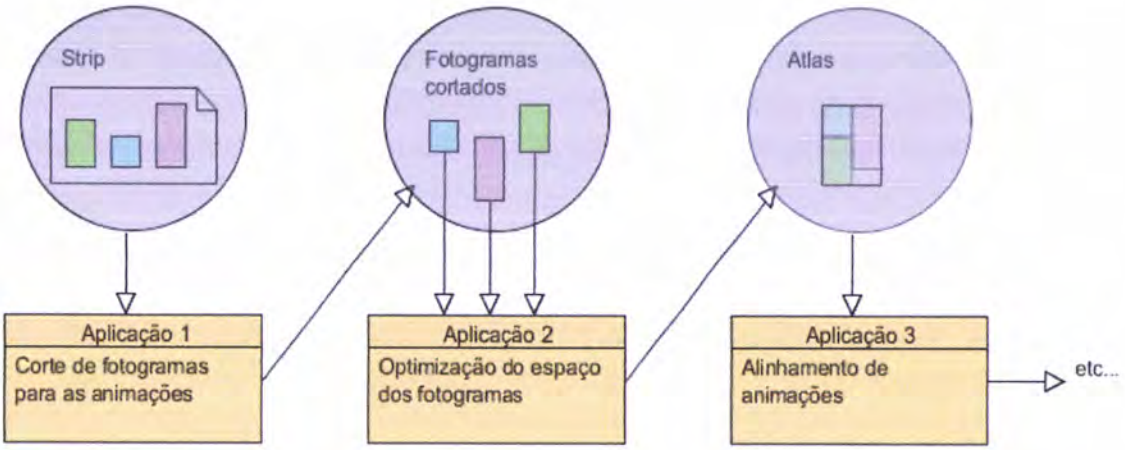


Figura 1. Sequência de trabalho com vários editores de funcionalidade específica. Os círculos a cinzento representam a informação, conforme esta entra e sai das aplicações de edição, que estão representadas como rectângulos cor-de-laranja.

Embora isto não seja um problema só por si, a utilização de várias aplicações em sequência tem a desvantagem de, quando é necessário refazer algumas acções executadas em fases anteriores, obrigar a repetir toda a sequência que se lhe seguiu, repetindo as mesmas acções que foram realizadas posteriormente com as restantes aplicações. Por outras palavras, é necessário repetir toda a linha de produção, desde a acção que teve de ser refeita. Também existem, naturalmente, editores mais avançados e que, estando geralmente enquadrados no ambiente de produção de determinadas empresas, dão respostas mais abrangentes a vários tipos de problema (ver sub-capítulo 2.2). Estes costumam ser desenvolvidos como aplicações não distribuídas, operando sobre informação que é guardada localmente em ficheiros, sendo necessário proceder a uma instalação do editor em cada máquina onde se pretenda utilizá-lo. Por outro lado, como são soluções unificadas que agregam várias funcionalidades, ao longo do tempo tendem a ser alvo de correcções e desenvolvimentos sucessivos, muitas vezes por programadores distintos, para poderem continuar a responder adequadamente a novos requisitos, conforme estes forem surgindo. A desvantagem que isto comporta é que, caso tenha sido utilizado algum tipo de arquitectura nos desenvolvimentos iniciais, a urgência que costuma caracterizar as novas alterações não permitirá este género de contemplações, e ela tenderá a perder-se (ver sub-capítulo 2.2). Por outro lado, como é necessário manter várias instalações da mesma aplicação, existe sempre o problema das versões. Ou seja, quando for necessário alterar algo na versão actual da aplicação, ou criar uma nova versão, será também obrigatório proceder-se à sua reinstalação em todas as máquinas onde esteja a ser utilizada ou actualizar todas essas instalações.

1.3. Solução Proposta

O *2D Game Editor On-line* é um sistema de edição de videojogos a duas dimensões, baseados em *sprites* e a correr em telemóveis. Porém, as funcionalidades implementadas são genéricas o suficiente para alargar este âmbito, e assim os projectos trabalhados com esta aplicação podem ser dirigidos a qualquer sistema onde corram jogos nas condições enumeradas.

A aplicação foi programada para correr sobre ambiente *web*, tal como o próprio nome indica, e pode ser disponibilizada a partir de qualquer servidor *HTTP* que tenha uma instalação de *PHP* e *MySQL*. Por outro lado, os acessos a ela podem ser realizados a partir de qualquer *browser*, conquanto estes tenham a máquina virtual de *Java* instalada e dêem suporte a *JavaScript* e *AJAX*. Estes recursos do lado do cliente são necessários para conseguir que o *2D Game Editor On-line* não fique, do ponto de vista da interactividade, posicionado em desvantagem em relação aos editores de videojogos mais completos actualmente em utilização e com os quais pretende competir. Estes, por se tratarem com frequência de aplicações de execução local, têm bastante mais facilidade em utilizar directamente os recursos da máquina onde correm.

Devido à enorme variabilidade que caracteriza o mercado do entretenimento, patente quer na oferta de jogos, quer na variedade de plataformas onde estes correm, não é possível definir um conjunto de requisitos que garantam que um editor possa dar suporte à produção de todos os jogos imagináveis. Deste modo, actualmente, o sistema funciona como uma aplicação de criação de *sprites*, ou seja, permite o carregamento de *strips* de fotogramas, a selecção e corte desses fotogramas, o alinhamento e o teste de animações, e a exportação desta informação para formatos padronizados, como o XML e ficheiros de imagem *png*. Para este processo de exportação, por seu lado, foi desenhado um algoritmo que realiza um arranjo espacial dos fotogramas a exportar numa imagem maior, o atlas. Deste modo, consegue-se reduzir os tempos de descarregamento das imagens no âmbito do trabalho com o próprio editor, bem como optimizar a alocação dos recursos das plataformas onde estes jogos irão correr.

No entanto, apenas este conjunto limitado de funcionalidades não torna o *2D Game Editor On-line* num sistema que pode competir com aqueles que são utilizados actualmente nas empresas. Então, como alternativa à implementação de todas as funcionalidades possíveis, optou-se por conformar a arquitectura do sistema a uma adaptação do padrão *Model-View-Controller*, com vista a facilitar a criação, adição e remoção de componentes funcionais que se prevejam especialmente úteis na produção de determinados jogos.

Também pela referida variabilidade da oferta neste mercado, o *2D Game Editor On-line* inclui um expediente que facilita a portabilização dos projectos, para que corram entre plataformas com esforço reduzido da parte dos utilizadores: a metáfora "plataforma". Este recurso, conceptualmente simples, permite que determinado projecto seja trabalhado até ao fim para uma plataforma específica e depois copiada essa informação para outra plataforma. Então, é possível aos utilizadores procederem aos ajustes necessários para optimizar o projecto para

essa nova plataforma. Desta forma, facilita-se o trabalho de criar uma nova versão do jogo, a partir de outra já implementada, com um mínimo de trabalho e sem ter de repetir passos anteriores.

1.4. Principais Contribuições

Com este trabalho pretende-se:

- Apresentar uma solução, baseada na *web*, para os problemas mais comuns da produção de videojogos para telemóvel, a duas dimensões e baseados em *sprites*;
- Relatar algumas das opções tecnológicas que foram tomadas para implementar a solução;
- Realçar os contributos dos padrões de arquitectura, especificamente o Model-View-Controller, como base sólida para a criação de sistemas editáveis e escaláveis;
- Propor um algoritmo de ordenação espacial de áreas rectangulares sem limitações espaciais definidas *à-priori*, bem como várias estratégias de crescimento do arranjo.

1.5. Estrutura do Documento

O presente documento está dividido nos seguintes capítulos, organizados segundo a sua temática:

- No Capítulo 1 dá-se uma visão global da realidade actual do mercado dos videojogos e faz-se o levantamento de algumas questões inerentes à actividade de produção para este mercado. Caracteriza-se também, em termos gerais, a aplicação proposta no presente documento.
- No Capítulo 2, faz-se um levantamento de algumas aplicações de edição de viedojogos nos moldes tradicionais, ou seja, pequenas aplicações orientadas à resolução de apenas um problema. Por oposição, descreve-se também um editor mais completo que estava em produção na empresa *YDreams SA*, no ano de 2007 e relata-se o estudo que foi feito sobre os padrões de arquitectura.
- No Capítulo 3 relata-se a implementação do *2D Game Editor On-line*, desde a fase de levantamento de requisitos à implementação, propriamente dita, do sistema.
- Por último, no Capítulo 4, estão as conclusões do presente trabalho, bem como algumas propostas de trabalho futuro.

Capítulo 2. Estado da Arte

Interpretando a já referida popularidade dos videojogos 2D casuais, como um indício da vitalidade deste mercado, é expectável que abundem os desenvolvimentos de aplicativos vocacionados para o suporte aos processos de produção. Pelas facilidades enunciadas no sub-capítulo 1.1, a produção deste tipo de videojogos é realizada quer por empresas e estúdios dedicados, quer por *Game Designers* amadores. No caso da produção industrial, os editores de videojogos em utilização consistem, maioritariamente, em aplicações desenvolvidas internamente, sem fins comerciais em si mesmo e que são planeados tendo em vista a resolução objectiva de problemas concretos que estejam a afectar a produção de determinado jogo, num dado momento. Por outro lado, no tocante aos editores de jogos utilizados pelos *Game Designers* amadores, que podem ou não ter conhecimentos tecnológicos ou acesso a alguém que os tenha, a oferta é mais variada mas limitada a pequenos sistemas, demasiado especializados para darem um apoio abrangente. Apesar de se ter feito aqui uma correspondência entre os o produtores e o tipo de editores utilizados, deixa-se aqui a ressalva que esta não é absoluta, não se devendo, portanto, excluir cenários onde estas tendências não se verifiquem (empresas que utilizem vários pequenos editores ou amadores que disponham de um editor de maior monta).

Segue-se a descrição de alguns exemplos mais representativos deste tipo de oferta que, à semelhança da proposta para o *2D Game Editor On-line*, são acedidas através da *Internet* ou a utilizam para algumas das suas funcionalidades.

2.1. As Ferramentas de Apoio à Produção de Videojogos

2.1.1. Image Packer

URL: *http://homepage.ntlworld.com/config/imagepacker/*

O Image Packer é uma pequena aplicação de agregação de imagens, descarregável e gratuita, que tem como entrada os vários fotogramas que compõem as animações das *sprites*. Como *output* terá a exportação desta informação, agregada numa imagem - o atlas - de dimensões seleccionáveis e cuja área útil está maximizada (o que será o mesmo que dizer que houve um esforço dos programadores no sentido de diminuir a percentagem de área não ocupada dentro da imagem exportada). Na Figura 2 pode ver-se a interface geral da aplicação, bem como um atlas composto com alguns fotogramas.

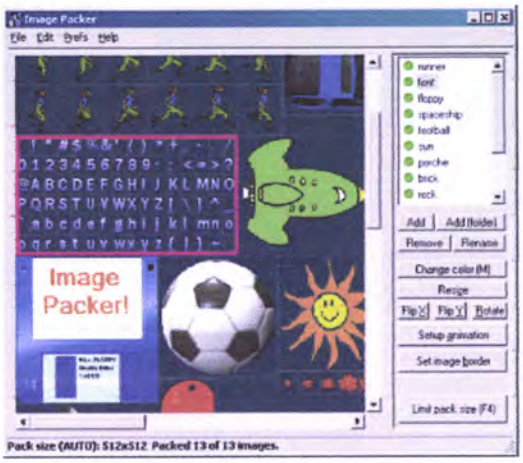


Figura 2. Interface da Aplicação Image Packer.

No entanto, apesar da preocupação com a questão do desperdício de espaço, a aplicação não inclui a possibilidade de editar os próprios fotogramas, assumindo que as imagens já tenham sido previamente cortadas e ajustadas às dimensões pretendidas. Isto pode ser um problema se a produção dos artistas for em formato de *strips*, contendo fotogramas de tamanhos variáveis, como ilustrado a seguir, na Figura 3:

Figura 3. *Strip* de fotogramas com tamanhos variáveis[1].

Sabendo que é necessário guardar informações relativas a cada fotograma depois de concluído o processo de empacotamento, como é o caso das posições dentro da imagem agregada, encontra-se aqui a primeira falha desta solução: o ficheiro com essa *meta-informação* não utiliza nenhum formato padrão, mas sim composições em variantes da linguagem *Blitz*, da empresa *Blitz Research, Ltd*. Esta característica permite que a exportação seja directamente integrável em programações *Blitz*, mas coloca problemas caso se pretenda integrar a ferramenta em processos de desenvolvimento de *software* para outras plataformas, como a *J2ME*.

Na documentação fornecida, é dada a indicação de que as funcionalidades do *Image Packer* são melhor aproveitadas quando integradas com as de outras do mesmo autor, como o *SpriteMaster Pro* ou *ImageMaster*.

[1] Este é um corte da strip apresentada no Anexo II.

2.1.2. Slick - 2D Game Library Based on LWJGL - Aplicação Pack-U-Like

URL: *http://slick.cokeandcode.com/static.php?page=about*

Esta solução apresenta-se como uma biblioteca, ou seja, uma colecção de pequenas ferramentas independentes, onde cada uma está vocacionada à resolução de um problema muito específico, comummente encontrado na produção de jogos. As aplicações estão programadas em Java e são disponibilizadas sobre o protocolo *JNLP* ou, em alternativa, podem ser descarregadas e executadas localmente no computador cliente.

Em particular, nesta biblioteca existe uma aplicação que concorre directamente com algumas das propostas de desenvolvimento para o *2D Game Editor On-line*, a aplicação *Pack-U-Like*. Trata-se de uma pequena ferramenta que, correndo sobre a plataforma *Java Web Start*, permite a entrada de imagens – os fotogramas de uma *sprite* – e constrói uma imagem maior, o atlas (ver sub-capítulo 3.3.1), com dimensões pré-definidas.

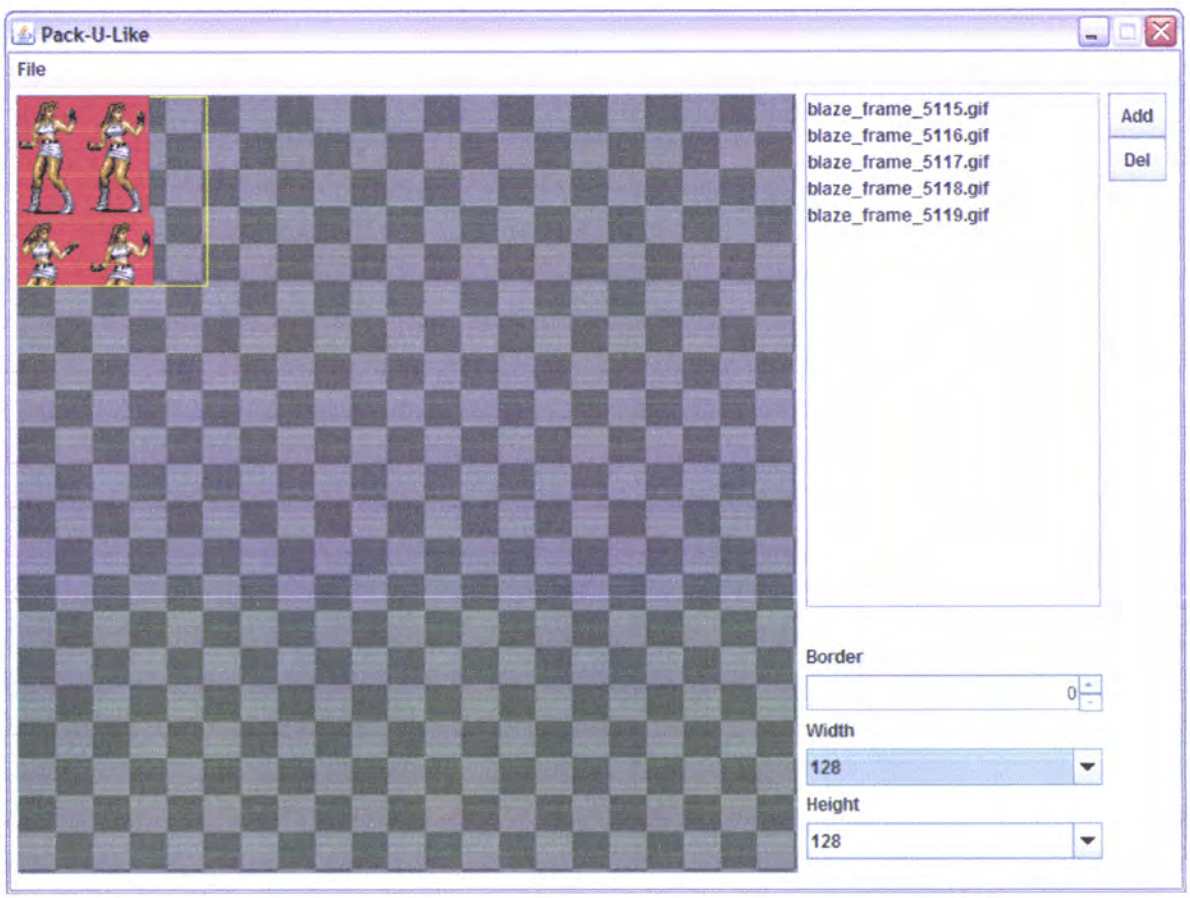


Figura 4. Aplicação *Pack-U-Like.*

Além da criação desta imagem composta, a aplicação também exporta informação sobre os fotogramas arranjados. Essa *meta-informação* consiste no identificador do fotograma, nas coordenadas do canto superior esquerdo no atlas e na sua altura e largura. Estes dados são guardados num ficheiro *XML*, com o mesmo nome do

ficheiro de imagem criado, o que permite a importação directa em qualquer ambiente programático ou em aplicativos capazes de ler *XML*.

No entanto, evidencia-se uma limitação nesta aplicação: é necessário que seja o utilizador a indicar os limites de largura e de altura do atlas. Ou seja, para que o programa consiga construir um atlas organizado e com o mínimo possível de área desperdiçada, é preciso que o utilizador encontre as medidas mais adequadas, tipicamente através de um processo de tentativa erro. A limitar mais ainda este quadro, refere-se que as dimensões seleccionáveis para o atlas pertencem a um conjunto discreto, ou seja, não é possível experimentar outras medidas que não aquelas. O referido intervalo contém os seguintes valores, em píxeis: 64, 128, 256, 512, 1024 e 2048, pelo que apenas é possível construir atlas rectangulares cujas dimensões laterais sejam combinações destes valores. Caso, o *Pack-U-Like* não consiga realizar um arranjo dos fotogramas dentro da área definida, o excedente será descartado, tal como visível na Figura 4.

Não está implementado o conceito de *sprite*, enquanto agregação de animações. Assume-se, assim, que o interrelacionamento dos fotogramas arranjados, em animações de uma *sprite,* seja uma tarefa que tenha de ser executada externamente à aplicação.

2.1.3. UnChaos

URL: *http://www.torquepowered.com/community/forums/viewthread/106183/2*

Esta é uma aplicação dedicada exclusivamente à composição de imagens a partir do arranjo de imagens mais pequenas e independentes.

Embora a aplicação possa apresentar resultados muito bons para um universo de poucas dezenas de fotogramas, consome tempos demasiadamente longos para uma aplicação local – mais de 15 minutos para 500 fotogramas, pelo que não constitui uma opção adequada para a produção de um jogo com muitas *sprites* e fotogramas. Não obstante, tem a mesma limitação que as outras, pedindo as dimensões do atlas final como parâmetro de entrada. Caso não seja possível realizar um arranjo dentro dessas dimensões, o excedente é descartado.

Como se pode ver na Figura 5, é também necessário indicar qual o tipo de ordenação inicial para os fotogramas a serem organizados entre três possibilidades: nome da imagem (alfabética), tamanho da imagem ou aleatoriamente.

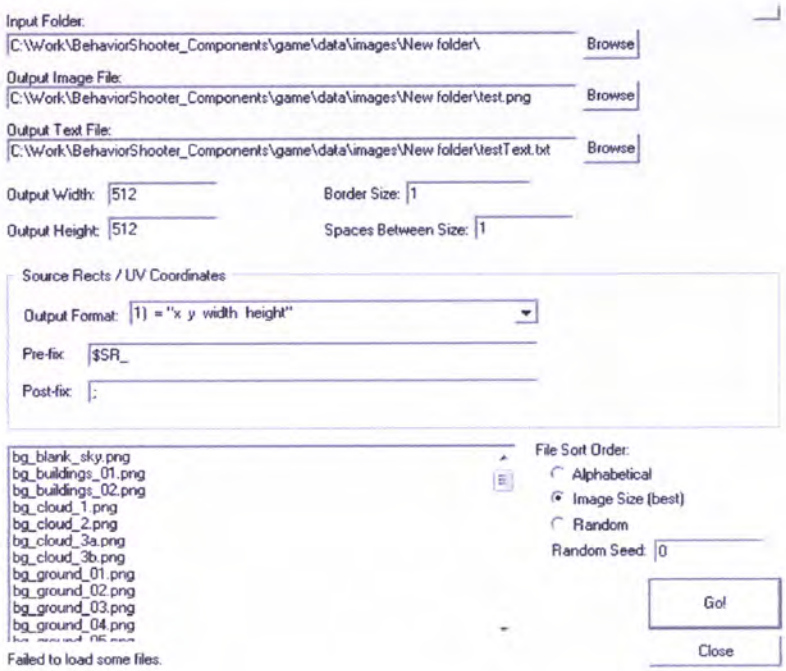


Figura 5. Interface da aplicação UnChaos

O *output* é gerado na forma de um ficheiro *txt*, (Figura 6) com as informações da área do atlas que é ocupada por cada fotograma (coordenadas do canto superior esquerdo, largura e altura do fotograma) e o ficheiro de imagem correspondente ao próprio atlas em si.

Esta aplicação também não implementa o conceito de *sprite*, tratando os fotogramas como imagens independentes.

```
// Sorting order: Image Size

$SR_BG_BLANK_SKY = "1 1 128 128";
$SR_BG_CLOUD_1 = "130 1 128 128";
$SR_BG_CLOUD_2 = "259 1 128 128";
$SR_BG_CLOUD_3A = "1 130 128 128";
$SR_BG_CLOUD_3B = "130 130 128 128";
$SR_BG_MOUNTAINS_01 = "259 130 128 128";
$SR_BG_MOUNTAINS_02 = "1 259 128 128";
$SR_BG_MOUNTAINS_03 = "130 259 128 128";
$SR_BG_MOUNTAINS_04 = "259 259 128 128";
$SR_T2D_MENUOVERLAY = "1 388 480 23";
$SR_PAUSED = "1 412 128 64";
$SR_ENEMYSHIP3A = "388 1 117 59";
$SR_BG_BUILDINGS_01 = "388 61 64 64";
$SR_BG_BUILDINGS_02 = "388 126 64 64";
$SR_PARTICLES1 = "388 191 64 64";
$SR_PARTICLES5 = "388 256 64 64";
$SR_PARTICLES6 = "388 321 64 64";
$SR_BG_GROUND_01 = "453 61 45 45";
$SR_BG_GROUND_02 = "453 107 45 45";
$SR_BG_GROUND_03 = "453 153 45 45";
$SR_BG_GROUND_04 = "453 199 45 45";
```

Figura 6. Exportação da informação, pelo UnChaos, relativa ao posicionamento dos fotogramas dentro do atlas.

2.1.4. Affinity

O *Affinity* é uma ferramenta baseada na *web* embora, ao contrário do *2D Game Editor On-line*, não seja uma verdadeira aplicação *web*, dado que o acesso não é feito através de *browsers*. Tem o propósito de se proporcionar como uma ferramenta educativa, que desenvolva nos utilizadores sem qualquer experiência de programação, aptidões de aprendizagem e de colaboração. Deste modo, no âmbito deste sistema, os jogos de vídeo surgem mais como a motivação para o trabalho com a aplicação e não como o propósito da própria aplicação. Dá ênfase ao "desenvolvimento baseado em componentes" que permite que os utilizadores desenvolvam os seus próprios componentes e os partilhem com a comunidade. Na visão do *Affinity*, um jogo é constituído por três tipos de elementos:

- **Componentes** – implementações de comportamentos e funcionalidades tipicamente encontradas nos jogos. Podem ser desenvolvidos e partilhados através de um recurso próprio para o efeito, o *Affinity Game Framework*; as pessoas que desenvolvem as *componentes* são chamadas de *"desenvolvedores"*;
- ***Assets*** – Arquivos relacionados com a criação artística de um jogo como as texturas, sons e modelos 3D; quem trabalha com os *assets* denominam-se *"artistas"*;
- **Dados** – No fundo, trata-se da *meta-informação* de um jogo, onde se indicam as parametrizações, os *assets* utilizados, tipos de entidades, posicionamentos e outros; as pessoas que lidam com os *dados* são denominados de *"projetistas"*.

O *Affinity* está muito orientado para o trabalho em colaboração em grandes comunidades, daí a necessidade de especificar de forma tão restrita os papéis que cada utilizador pode desempenhar no sistema. O sistema gera directamente os jogos concluídos, pelo que não é uma aplicação desenhada para ser integrada em ambientes de produção. A linguagem utilizada pelo *Affinity*, na geração dos jogos (componentes e dados) é o *Microsoft XNA*.

Figura 7. Esquema da arquitectura do *Affinity*.

A sua arquitectura geral (ver Figura 7) segue o modelo cliente-servidor e funciona sobre a *Internet*, embora não seja, como já foi referido neste sub-capítulo, uma verdadeira aplicação *web*. Os clientes não correm sobre *browsers*, sendo programas independentes que correm nas máquinas cliente - denominados *Affinity Game Editor* -, e o servidor limita-se a guardar os dados e a disponibilizá-los sempre que algum cliente os peça. Porém, o *Affinity* também permite que as informações trabalhadas sejam exportadas para ficheiros. Resta referir que, de acordo com o artigo explorado, esta é uma aplicação ainda em desenvolvimento (Gerosa et al.).

2.1.5. Image Packer (2)

URL: *http://www.devmaster.net/forums/showthread.php?t=5401*

Esta é uma aplicação muito simples que foi desenvolvida por um programador de videojogos, sem intenção comercial, e que realiza, apenas, o arranjo das imagens de entrada em composições agregadas. Segundo as palavras do próprio autor, no fórum que contém o *link* para o *download*, não foi dada grande atenção à questão da ordenação das imagens (visível na Figura 8). Deste modo, caso não esteja satisfeito com o arranjo actual, a única possibilidade que é dada ao utilizador para optimizar o espaço ocupado consiste na escolha activa de outro algoritmo de ordenação, dentre aqueles que são disponibilizados: "Auto", "Pack right", "Pack Down" e "Center Pack". É de relevar que esta aplicação não coloca limites ao tamanho final do atlas, realizando o corte automaticamente, conforme o arranjo que conseguiu realizar.

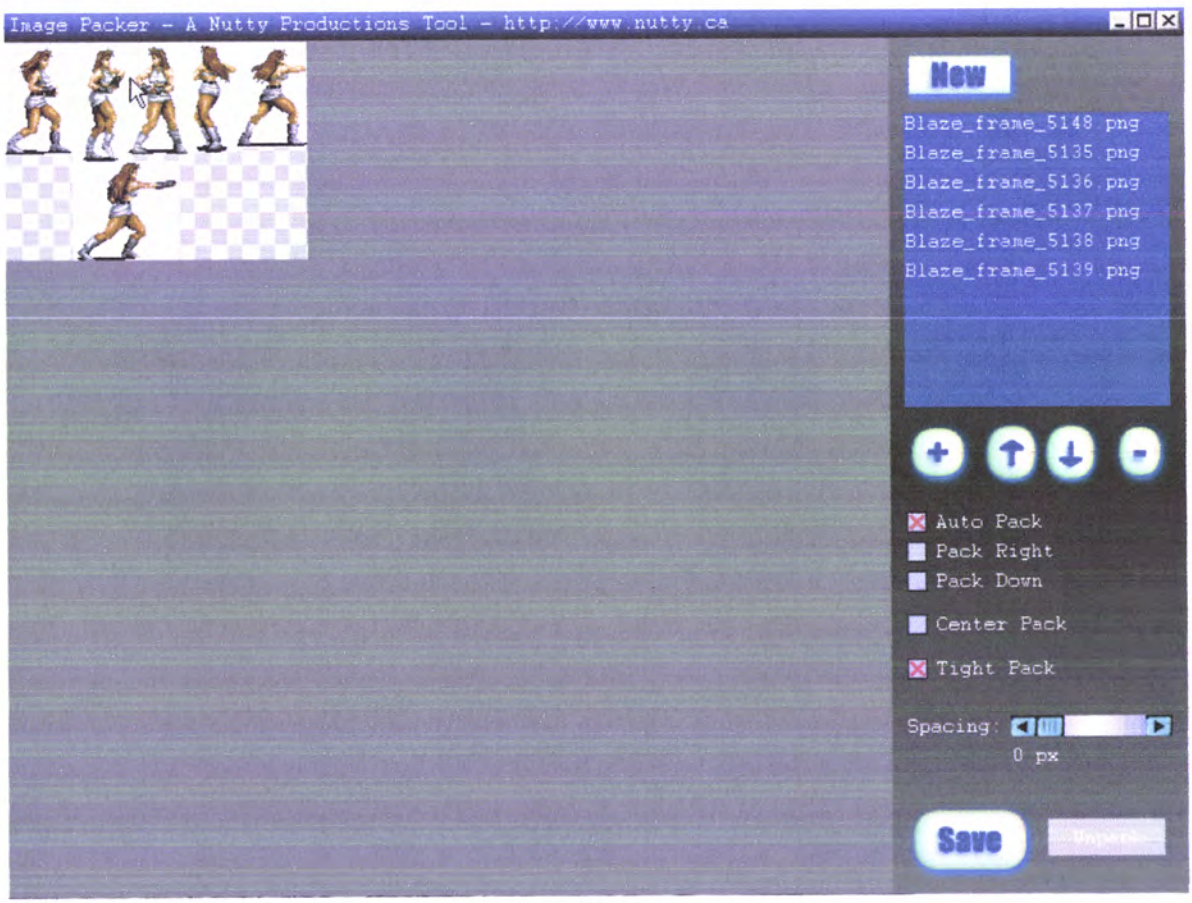


Figura 8. Aplicação Image Packer (2).

A aplicação exporta, além da imagem em formato a definir (não trabalha com *gif*), um ficheiro *xml* com apontadores para os diversos fotogramas arranjados. Novamente se verifica que não existe a implementação do conceito de *sprite*, enquanto agregação de imagens.

2.2. Um exemplo prático

A título de exemplo do que já foi referido no Capítulo 1, relativamente às aplicações utilizadas nos ambientes de produção industrial de videojogos, relata-se o caso do editor de jogos que estava integrado em ambiente de produção, na empresa *YDreams SA*, no ano de 2007. A aplicação - denominada *2D Game Editor* - constituiu um projecto desenvolvido internamente pela própria empresa e estava implementado em C#. Tratando-se de uma aplicação de execução local, portanto sem recurso a programação de servidores, a informação era manipulada directamente na máquina onde corria o editor. O trabalho com esta aplicação terminava com a geração de um *output* binário, que era utilizado por outras ferramentas que davam apoio à produção dos jogos.

Além das questões técnicas, a aplicação tinha sido concebida como um meio optimizado de comunicação dentro da equipa, que permitia a transmissão de material de trabalho e informação entre o *Game Designer* e a equipa responsável pela implementação final de um jogo. Como foi referido no Capítulo 1, o cargo de *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 outro 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* visava 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*, este editor representava a informação de uma forma visual e o mais compreensível possível (ver Figura 9 e Figura 10).

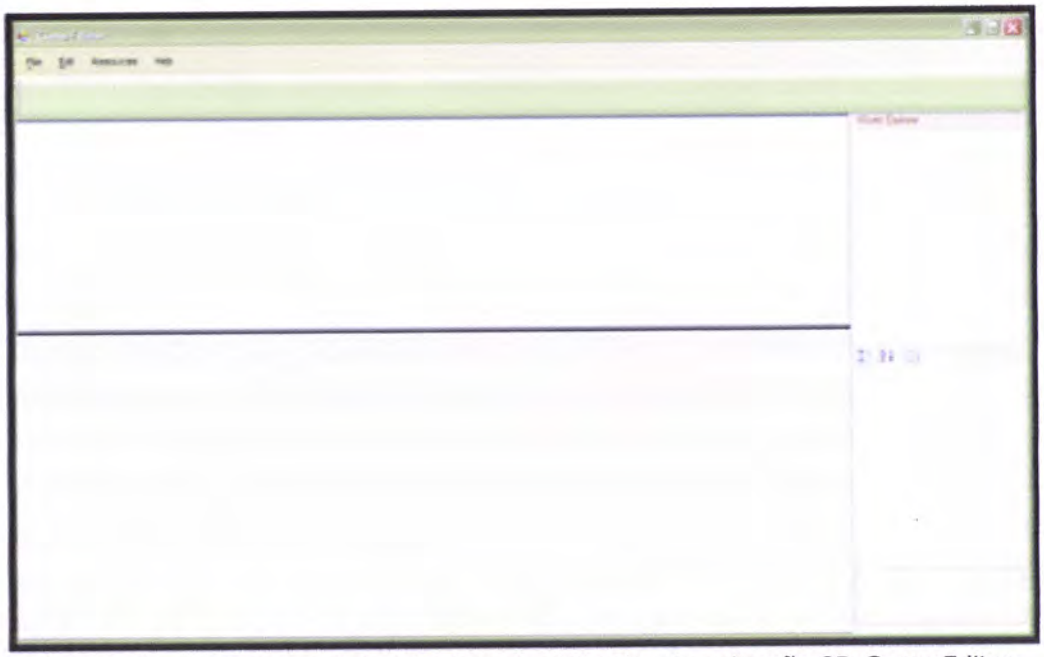


Figura 9 – Arranjo geral da interface de utilizador da aplicação 2D Game Editor.

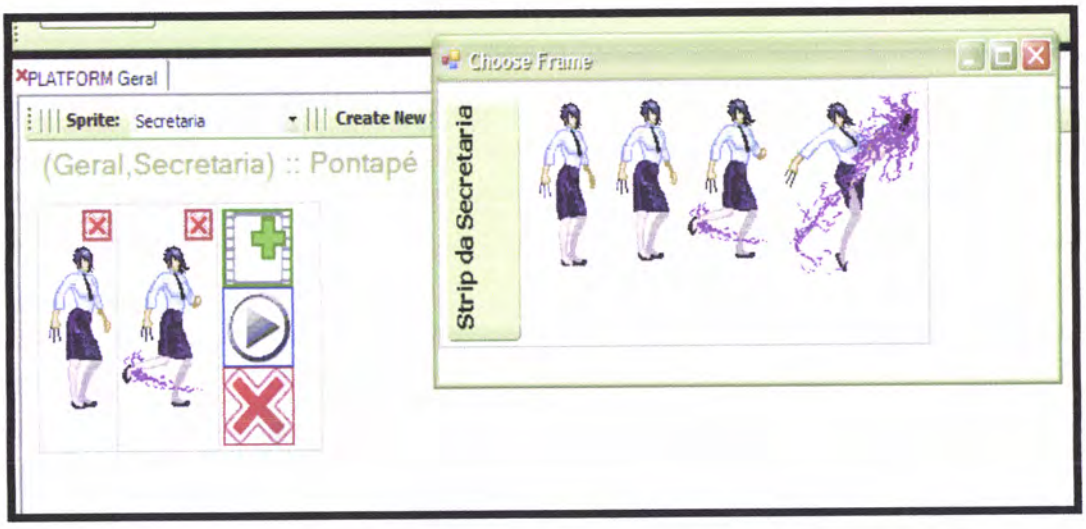


Figura 10. Representação visual dos fotogramas de uma animação.

No entanto, após a conclusão da 1ª versão do *2D Game Editor*, foram surgindo problemas graves relacionados com a arquitectura do *software* e com a sua interface com o utilizador, que se traduziram, em última análise, no seu abandono pelo *Game Designer,* e na relutância em aceitar trabalhos de alteração do código por parte dos membros da equipa de programação. Deixa-se aqui a nota, nesta altura, que 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, ou seja, não houve desde o início uma normalização da arquitectura, tendo ficado ao critério do programador a estruturação e organização 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 perda da sua estruturação interna.

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 evidente ou fácil. A par destas questões, a aplicação registava também problemas a nível do desempenho da interface com o utilizador quando o volume de informação em memória era elevado, e as mesmas dificuldades na manutenção de códigos e versões que costumam ser encontradas em aplicações de execução local. Para a solução deste problema, contribuiu a alteração da arquitectura da aplicação segundo as regras do padrão *Model-View-Controller* (Cardoso e Romão, 2008). Esta conformação veio a facilitar a resolução dos problemas a nível da interface com o utilizador, dado que reforçou a separação entre os componentes dedicados à interacção com o utilizador e os que gerem o modelo de dados e também estabeleceu regras sólidas para as futuras intervenções de manutenção e evolução do código da aplicação (ver sub-capítulo 2.3.3, secção *Model-View-Controller*).

Por outro lado, além das ilações que se poderão retirar dos problemas do editor em si, existe um conhecimento mais valioso que pode ser adquirido, se se estudarem os requisitos originais que motivaram a criação do *2D Game Editor*. Destes, a maioria são inerentes à própria actividade de criação de videojogos e comuns, aliás, a quaisquer ambientes de produção de jogos para telemóvel, independentemente das empresas que os desenvolvam:

- Permitir a manipulação de *strips* e corte de fotogramas;
- Portabilizar os jogos produzidos para várias plataformas;
- Alinhar e parametrizar as animações;
- Recorrer o mínimo possível a termos tecnológicos e representar visualmente a informação;
- Possibilitar a pré-visualização do trabalho já executado.

Atendendo à transversalidade destes requisitos, estes podem constituir-se como uma base perfeitamente válida para a fase de levantamento de requisitos de qualquer aplicação de edição que esteja apostada em posicionar-se no mercado como uma alternativa competitiva. Foi, então, este o ponto de partida para o desenvolvimento e planeamento de uma nova solução, mais generalista e aberta que o *2D Game Editor* da empresa *YDreams SA*, de forma a cativar o interesse geral da industria dos videojogos para telemóvel.

2.3. Padrões

2.3.1. Definição e Âmbito de Utilização

Um padrão, enquanto noção aplicada ao contexto geral da engenharia, foi definido como *"uma ideia que foi útil num determinado contexto prático e que vai,*

provavelmente, tornar a ser útil em outros" (Fowler, 1996). É, então, um conceito radicado num âmbito exterior ao da informática, tendo a sua origem nos trabalhos de planeamento urbano e na arquitectura de edifícios, datados dos finais dos anos 70, do arquitecto Christopher Alexander.

Conforme a citação de Fowler, a justificação da utilização de padrões fundamenta-se na premissa de que, se uma determinada solução provou ser eficiente no passado também tornará a sê-lo no futuro ou, no mínimo, poderá vir a providenciar as bases para encontrar uma solução eficiente e mais aplicável. Esta definição tão abrangente obriga a que os padrões sejam, por defeito, muito claros no que diz respeito às vantagens e desvantagens da sua utilização, como, quando, porquê e onde se devem aplicar. Desta forma, a decisão de utilizar, ou não, um padrão, pode ser devidamente considerada sopesando as vantagens, desvantagens e os objectivos que se pretendam obter.

2.3.2. Os Padrões na Engenharia de *Software*

No caso específico da engenharia de *software*, os padrões formam uma disciplina vocacionada para a resolução de problemas que emergiram da comunidade da programação orientada a objectos. A manter as directrizes aplicadas aos padrões em geral, cada tipo está orientado à resolução de conjuntos de problemas de programação específicos e recorrentes, procurando dar-lhes resposta de uma forma segura e testada em situações anteriores semelhantes. Regra geral, um padrão não é um plano acabado e pronto a traduzir para código, mas sim uma descrição, um conjunto de normas orientadas à resolução de um determinado tipo particular de situações.
Sumariando as características associadas aos padrões em engenharia de *software*, tem-se (Cunningham, 1999):

- Um padrão descreve uma solução para um problema recorrente que surge em situações específicas.
- Os padrões não são inventados "ad-hoc"; ao invés, foram gradualmente "destilados" através da experiência empírica.
- Os padrões descrevem um grupo de componentes, as interacções que se estabelecem entre estes e as responsabilidades de cada um. Por outras palavras, um padrão é uma abstracção de mais alto nível que os conceitos dos paradigmas de programação, como objectos e classes.
- Os padrões providenciam um vocabulário comum a ser usado entre *designers* de *software* e programadores, facilitando a comunicação entre a equipa de desenvolvimento. A escolha do nome de um padrão é, assim, de importância primordial.
- Os padrões ajudam a documentar a visão arquitectónica de um determinado desenho. Se esta visão for claramente compreendida, é de esperar que seja menos provável violá-la ao proceder a modificações no sistema.
- Os padrões providenciam uma estrutura conceptual para a solução de um problema de desenho e, desta forma, potenciam a construção de *software* consistente e com propriedades bem definidas.
- Os padrões são blocos de construção para o desenvolvimento gradual de sistemas mais complexos.
- Os padrões ajudam os *designers* a gerir a complexidade do *software*.

Em engenharia de *software* existe a tendência para distinguir os padrões conforme a granularidade, em (Cunningham, 1999):

- **Padrão de Arquitectura** – Padrões de *software* que oferecem soluções testadas e seguras para problemas arquitecturais recorrentes encontrados em engenharia de *software*. Um padrão de arquitectura expressa uma organização estrutural fundamental para um sistema de *software*. Este esquema consiste na implementação de subsistemas predefinidos, especificando claramente as suas responsabilidades e relações e inclui regras e linhas de orientação para as organizar. Um exemplo de um padrão de arquitectura é o padrão *Pipes and Filters* [1].
- **Padrões de Desenho** – Micro arquitecturas de mais baixo nível que os padrões de arquitectura, aplicáveis em várias situações. Providenciam um esquema para refinar os subsistemas ou componentes de *software*, ou as suas relações num contexto particular, tratando-se de uma abstracção de nível médio. A escolha de um padrão de desenho não vai afectar a estrutura fundamental do sistema de *software*, mas sim a de um subsistema. Tal como os padrões de arquitectura, os padrões de desenho tendem a ser independentes da linguagem de implementação. Um exemplo de uma aplicação de um padrão de desenho seria a aplicação do padrão *Adapter*, ou *Wrapper*, que adapta a interface de um tipo de objecto existente para a mesma interface de um outro tipo de objecto [2], ou o padrão *Iterator* [3].
- **Padrões de Idioma** – Padrões específicos de uma linguagem de programação. Descreve como implementar aspectos particulares de componentes ou das suas relações, utilizando as características e mecanismos específicos da linguagem sendo considerado, dessa forma, um padrão de baixo nível. Em *Java*, o iterador específico da linguagem definido para implementar a interface *Iterator,* pode ser considerado uma aplicação de um padrão de idioma. É uma implementação exclusiva de *Java* para o padrão de desenho *Iterator*.

---

[1] Em Unix, por exemplo, um filtro é um programa que lê um fluxo de *bytes* do *standard input* e escreve um fluxo transformado para o seu *standard output*. Estes programas podem ser ligados com o *output* de um filtro a funcionar de *input* do próximo filtro na sequência pelo mecanismo de *pipe*. Sistemas maiores podem ser, então, construídos a partir de componentes simples que, de outro modo, operariam independentemente. Outros exemplos seriam sistemas de operação por camadas, como o modelo *OSI*, sistemas de arquitectura *Blackboard* com operação em espaço comum e o padrão *Model-View-Controller* utilizado em interfaces gráficas de utilizador (*GUI*).

[2] Por exemplo, uma das implementações do tipo abstracto de dados *RankedSequence* em *Java* utiliza o padrão Adapter. A classe *VectorRankedSeq.java* adapta a classe *Vector* para corresponder à especificação da interface *RankedSequence*.

[3] Este padrão define mecanismos para avançar através de dados numa estrutura, elemento a elemento. Por exemplo, se fizermos em *Java* a invocação do método *iterator()* numa instância da classe *Vector*, este retorna um objecto que implementa a interface *Iterator*. Invocações sucessivas do método *next()* desse objecto vão retornar os sucessivos elementos do objecto *Vector* inicial.

2.3.3. Padrões com orientação à Arquitectura do Software

No caso particular do *2D Game Editor On-line*, tratando-se de uma aplicação com forte componente de interfaces de utilizador interactivas e que lida com um modelo de dados relativamente complexo, é interessante encontrar um padrão que esteja adequado a este género de sistemas. Então, de entre aqueles que aparentaram estar especialmente adequados, sobressaíram dois:

- **Presentation-Abstraction-Control** – De acordo com (Buschmann, F. et al. 1996), 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 (Coutaz, J. 1987).
- **Model-View-Controller** – Segundo (Buschmann, F. et al. 1996), procura separar a implementação de uma aplicação interactiva em três componentes, por responsabilidades: o *Model* é responsável pelos dados e pelos métodos de manipulação destes; a *View* disponibiliza a informação contida no *Model* que lhe corresponda e o *Controller*, por seu lado, reage aos *inputs* do utilizador. O par *View-Controller* forma a interface com o utilizador da aplicação. Este modelo foi sugerido no ambiente de programação do *SmallTalk* (Krasner G. et al. 1988).

Qualquer um destes padrões, por levarem em conta uma forte componente de interacção com o utilizador e proporem a cisão da arquitectura em componentes conforme as responsabilidades, tem o potencial de se adequar aos requisitos do *2D Game Editor On-line*. Segue-se, então, uma breve explicação dos seus modos de funcionamento.

Presentation-Abstraction-Control

Este padrão define uma estrutura para sistemas de *software* interactivo, na forma de uma hierarquia entre agentes cooperativos. Cada agente é, então, responsável por um aspecto específico das funcionalidades da aplicação e consiste em três componentes: apresentação, abstracção e controlo. Esta subdivisão separa os agentes responsáveis pela interacção utilizador-computador da sua funcionalidade e da sua comunicação com outros agentes (Buschmann, F. et al. 1996), conforme o esquema da Figura 11.

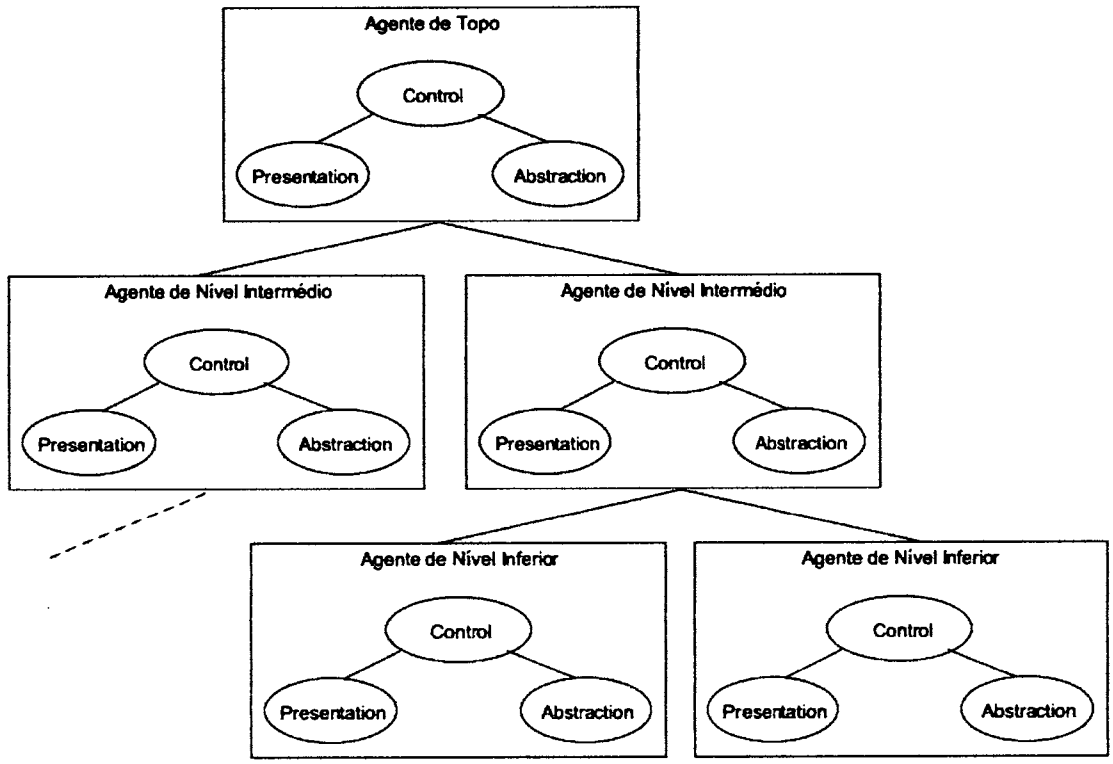


Figura 11. Esquema de agentes do padrão *Presentation-Abstraction-Control*.

Os componentes dos agentes são:

- ***Presentation*** – Mostra a informação mantida pela componente *Abstraction*.
- ***Control*** – Processa eventos externos (como *inputs* do utilizador) e actualiza o modelo de dados. Tem a responsabilidade de actualizar directamente os componentes *Presentation* e/ou *Abstraction* que pertençam ao mesmo agente e, depois, de passar a informação de actualização para o componente *Control* do agente hierarquicamente acima.
- ***Abstraction*** – Este componente mantém os dados. No entanto, esses dados podem ser apenas parte do modelo de dados geral da aplicação e não tem papel activo na notificação de alterações.

A mecânica de propagação de alterações funciona conforme a figura que se segue:

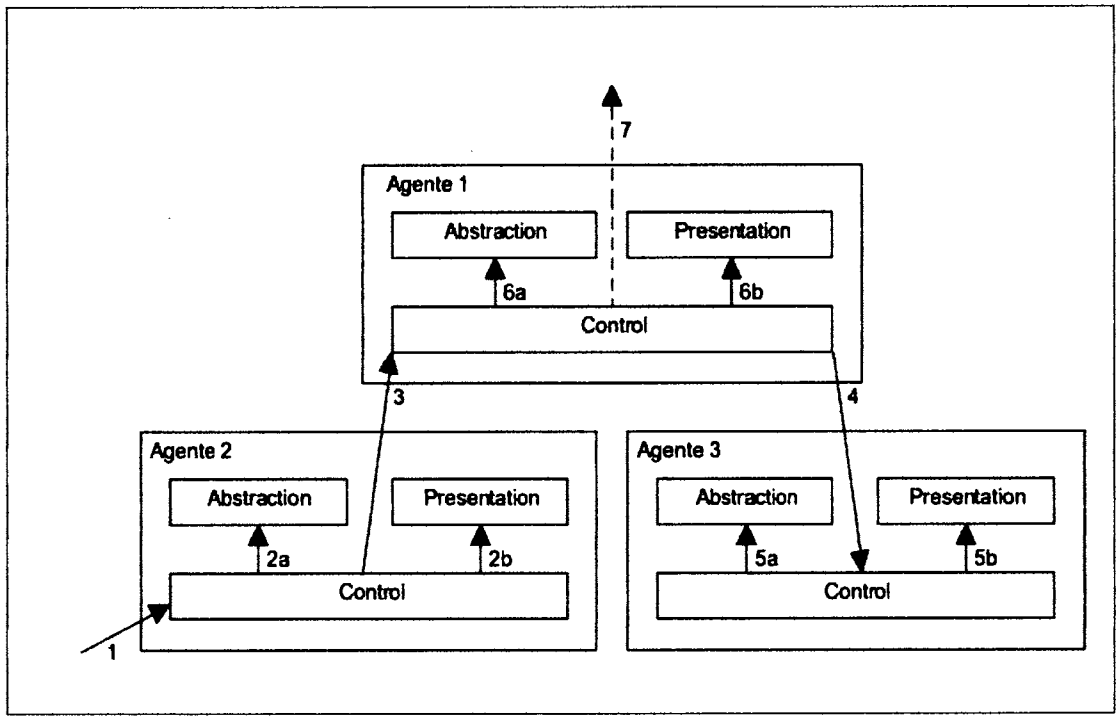


Figura 12. Funcionamento do padrão Presentation-Abstraction-Control

Em primeiro lugar, os vários agentes são instanciados pelo agente inicial (Agente 1). Quando um agente recebe *input* (1), o seu componente *Control* actualiza o seus restantes componentes, o *Abstraction* (2a) e o *Presentation* (2b), ou ambos. Depois, passa essa informação ao agente que o gerou (3). Este, reencaminha a notificação de actualização para todos os seus filhos, exceptuando aquele que o notificou a si inicialmente (4). O componente *Control* de cada filho que receba a notificação actualizará, por seu lado, os respectivos componentes (5a e 5b) e, só depois, o agente inicial se actualizará (6a e 6b). Por último, caso este agente tenha sido também gerado por outros, a notificação de actualização prosseguirá para cima, do mesmo modo que foi descrito (7).

Model-View-Controller

Este padrão é frequentemente utilizado em sistemas com elevado destaque para a interactividade humana, e propõe uma forma eficiente de separação entre a lógica da aplicação e a interface de utilizador, pelo recurso à implementação de uma camada intermédia. O objectivo é a construção de sistemas mais simples de desenhar e de manter, dado que se podem realizar intervenções ao nível de qualquer uma das componentes sem haver necessidade de alterar também as outras (isto depende, naturalmente, da extensão das alterações, podendo haver algumas mais radicais que requeiram actualizações transversais a toda a arquitectura). A aplicação correcta do *Model-View-Controller* tem como resultado a modularização do código e o consequente favorecimento do potencial de reutilização dos módulos já programados.

Como se poderá constatar nesta altura, estes objectivos correspondem na perfeição aos do *2D Game Editor On-line*, e vale a pena fazer uma avaliação mais profunda das características deste padrão de arquitectura. Em termos de implementação, a

conformação da arquitectura de um sistema aos preceitos do *Model-View-Controller* implica a sua padronização, ou seja, a sua divisão em três componentes, conforme a responsabilidade que tenham no sistema: o *Model* (*Modelo*) contém a informação e fornece métodos para interagir com esta; a *View* (*Vista*) disponibiliza a informação ao utilizador e o *Controller* (Controlador) que, por seu lado, gere o *input* do utilizador (Dix et al. 1998). Um mecanismo de propagação de alterações assegura a consistência entre a interface do utilizador e o *Model* (Buschmann, F. et al. 1996).

Apesar de existirem várias variantes do *Model-View-Controller*, a sua essência (que é também a sua maior vantagem) reside em conseguir-se a disjunção entre o *Model* e os pares *View-Controller* que lhe estejam afectados. Consegue-se assim, com relativa facilidade, proceder à substituição das *Views* sem que isso obrigue a qualquer actualização no *Model* e vice-versa. É também possível alterar o *Model* com o mínimo (se é que com alguma) alteração aos componentes de visualização.

Apesar da relativa complexidade deste padrão de desenho, convém frisar que a sua correcta implementação implica que seja totalmente transparente para o utilizador, quer a nível de componentes, quer de procedimentos de comunicação, ficando as suas vantagens apenas perceptíveis do lado da programação.

Numa perspectiva de baixo nível, uma aplicação que tenha a sua arquitectura padronizada segundo estas normas, terá a sua programação global implementada como uma agregação de unidades-padrão funcionais, cada uma composta, por seu lado, por uma tríade de componentes: um *Model*, uma *View* e um *Controller*, podendo dar-se o caso de haver mais de um par *View-Controller* ligados ao mesmo *Model* (ver Figura 13).

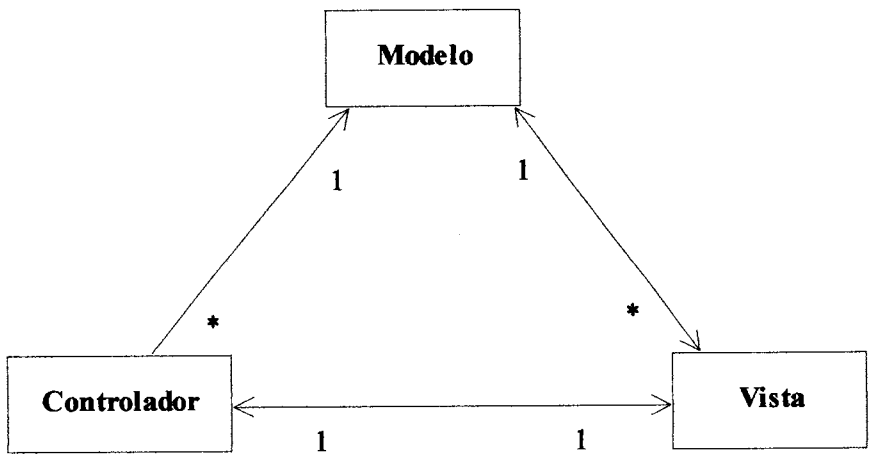


Figura 13. Tríade MVC, exemplificando a aridade (*arity*) das relações que se podem estabelecer entre os componentes.

Apresenta-se, uma ilustração (inspirada nos diagramas de classes UML) sobre o funcionamento tradicinal das comunicações entre estes componentes (Figura 14).

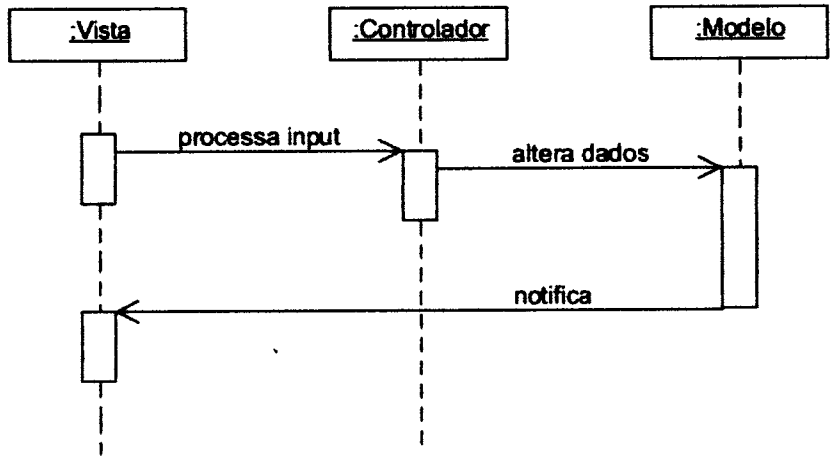


Figura 14. Mecânica de transmissão de eventos entre componentes MVC.

Como se pode observar, quando a *View* detecta *input* do utilizador invoca, no componente *Controller,* os métodos de processamento desse *input*. De acordo com o tipo e contexto em que o sistema tenha recebido esse *input*, o *Controller* invoca no *Model* os métodos de actualização dos dados. Por último, este componente notifica a *View* sobre a ocorrência de alteração nos dados (sem que tenha, no entanto, de a informar especificamente sobre o tipo de alteração) e esta tratará de se actualizar.

2.4. As metáforas de interface e a usabilidade

O principal objectivo do *2D Game Editor On-line* é que possa, uma vez implementado, vir a substituir as soluções actualmente em utilização pelas empresas de produção de videojogos ou, pelo menos, que consiga competir directamente com estas. Os costumes e a habituação dos utilizadores às soluções já enquadradas na produção, em adição à inevitável resistência à mudança nos seus hábitos de trabalho, poderão colocar-se como obstáculos à adopção de um novo sistema.

É razoável assumir de antemão que as expectativas dos utilizadores, face a uma aplicação que venha substituir outra, irão no sentido de que a usabilidade da nova seja, pelo menos, tão eficaz como a da que se propõe substituir. A usabilidade de uma aplicação define-se pela facilidade com que os utilizadores com ela interagem a fim de atingirem um dado obejctivo. Tradicionalmente, a usabilidade está associada aos seguintes cinco atributos (Nielsen, 1993):

- Learnability (Aprendizagem) – O sistema deve ser fácil de aprender, para que os utilizadores possam começar a trabalhar com ele rapidamente e com pouco tempo investido em aprendizagem;
- Efficiency (Eficiente) – Deve ter uma utilização eficiente para que, depois de devidamente familiarizados, possibilite aos utilizadores atingirem níveis elevados de produtividade;
- Memorability (Memorizável) – O sistema deve ser fácil de memorizar, para que os utilizadores ocasionais possam voltar a usar o sistema após períodos de inactividade, sem ter de o reaprender;
- Errors (Baixa ocorrência de erros) – Deve ter uma baixa taxa de ocorrência de erros e, caso estes aconteçam, o sistema deve permitir uma rápida recuperação. Em adição, têm de ser tomadas as devidas precauções para que não ocorram erros que possam comprometer a integridade da informação ou causar a sua perda;
- Satisfaction (Satisfação) – O sistema deve estar concebido de forma a permitir uma utilização agradável.

Uns dos recursos mais comuns para facilitar a utilização de um novo sistema, que se relaciona intimamente com a questão da usabilidade, são as metáforas de interface. As metáforas tratam-se, então, de modelos conceptuais que foram desenvolvidos para se assemelharem, em certos aspectos, com uma entidade física, podendo esta tratar-se de um objecto real ou de uma actividade.

As metáforas provaram ser altamente eficazes, pois estabelecem pontes cognitivas entre modelos conceptuais familiares aos utilizadores e as entidades virtuais (dados, relações entre estes ou processos de manipulação) operadas pelo sistema (Preece et al., 2002).

Assim, a inclusão cuidadosa de metáforas numa interface com o utilizador, além de facilitar a aprendizagem necessária à utilização do sistema, contribui também para a efectividade das comunicações entre os utilizadores. Isto porque permite estender os conceitos do sistema para fora do ambiente virtual criado e, assim, possibilita

que os utilizadores discutam o trabalho realizado sem necessidade de acesso ao sistema. Além destas vantagens, estes recursos, quando bem aplicados, têm ainda o potencial de evitar que os utilizadores sejam obrigados a compreender detalhes tecnológicos que, por não estarem directamente relacionados com a sua actividade profissional, podem contribuir para o mau desempenho do sistema enquanto ferramenta de trabalho.

Como exemplo de metáforas recorrentes em aplicações, temos o emprego do termo "pasta" a um directório informático. Isto permite estabelecer uma conexão mental e cognitiva entre uma pasta de arquivo tradicional, comum em ambientes de trabalho clássicos, e o conceito abstracto de directório, enquanto estrutura virtual do sistema de ficheiros. Outra metáfora muito comum é a de ambiente de trabalho, *desktop*, que concretiza algumas das funcionalidades mais comuns de um sistema operativo, relacionando-as com o ambiente de trabalho físico de um escritório.

Resumindo, o papel de uma metáfora na interface de utilizador é o de facilitar a aprendizagem, orientar o utilizador permitindo-lhe a formação e manutenção de conceitos sobre o programa e suas funcionalidades, ou seja, a criação de um modelo mental com analogias a outros mais familiares (Szabó, K. 1995).

Deste modo, pode concluir-se que um sistema que seja desenhado com vista a proporcionar uma boa experiência de utilização deve fazer uso de metáforas, sempre que as suas vantagens incrementem a sua usabilidade, e isto reveste-se de particular importância quando esta propõe substituir-se a aplicações já em utilização.

No caso particular do *2D Game Editor On-line*, foi explorada a utilização de metáforas no sentido de apurar quais as mais pertinentes e, uma vez concluído esse estudo, incluíram-se aquelas cuja implementação pudesse trazer benefícios ao desempenho da sua interface com o utilizador. No fundo, todos os objectivos destes recursos coincidem, quando bem aplicados, na perfeição com aqueles que foram definidos para a interface com o utilizador do *2D Game Editor On-line* (ver sub-capítulo 3.1). Assim, visto que se pretende criar um sistema de utilização fácil e agradável, que permita atingir bons níveis de produtividade e que tenha potencial para ter uma rápida aceitação pelos utilizadores, o contributo das metáforas para a interface com estes pode ser uma mais-valia de grande destaque (ver sub-capítulo 3.2.3).

Capítulo 3. O 2D Game Editor On-line

3.1. Levantamento de Requisitos

Pretende-se, então, desenvolver uma aplicação que tenha potencial suficiente para competir com as soluções actualmente em utilização em ambientes de produção industrial de videojogos 2D para telemóvel.

Como já foi referido no sub-capítulo 1.2, estas soluções costumam ser aplicações *desktop,* cujas funcionalidades foram desenvolvidas para dar resposta unicamente às necessidades que, na altura da sua criação, se fizeram sentir. A variabilidade que se verifica a nível da oferta no mercado dos videojogos reflecte-se, também, nas linhas de produção de diferentes títulos e isto motiva a que uma aplicação de edição que esteja perfeitamente adequada à produção de determinado jogo, tenha grandes probabilidades de não dar o apoio necessário quando se pretender utilizá-la na de outros títulos.

O ideal seria, naturalmente, criar uma aplicação que desse uma resposta optimizada a todos os problemas. Porém, um sistema que seja tão completo será, necessariamente, também muito complexo. Além dos problemas ao nível da legibilidade e da clareza que esta abordagem traria para a interface de utilizador, comprometer-se-ia também o próprio desempenho da aplicação a médio e a longo prazo, pois, para implementar funcionalidades tão abrangentes correr-se-ia o risco de tornar a programação menos legível e compreensível e, portanto, mais difícil de manter (conforme o exemplo apresentado no capítulo 2.2).

Se, por outro lado, a escolha recaísse sobre um sistema minimalista, que disponibilizasse apenas o mínimo aplicável na produção de qualquer jogo, obter-se-ia uma solução que, apesar de potencialmente integrável em várias linhas de produção, não teria a solidez nem o potencial necessários para conquistar os utilizadores que, por seu lado, teriam sempre que recorrer a soluções externas para realizarem as tarefas específicas que não estivessem contempladas no editor.

Pesadas as contrapartidas, os esforços devem apontar para a implementação de um sistema que, não sendo demasiado completo (leia-se, complexo), também não corra o risco de ser catalogado como insuficiente, dê apoio às necessidades mais prementes e generalistas da produção de videojogos e não impossibilite as respostas a requisitos mais específicos.

Uma hipótese razoável consistirá, então, em fazer o levantamento de um conjunto de requisitos que sejam comuns à produção de vários géneros de jogos (sem entrar em detalhes específicos a cada um), implementar o sistema com base nestes e incluir uma estratégia que facilite as tarefas de produção, integração e remoção de funcionalidades que se verifiquem indispensáveis, ou muito úteis, a projectos concretos.

Como também já foi referido, existe uma grande heterogeneidade nas qualificações dos grupos profissionais que integram uma equipa de produção de videojogos e,

consequentemente, não é expectável que todos tenham conhecimentos técnicos profundos. Deste modo, a aplicação deve fazer recurso moderado a termos tecnológicos, e tem de haver um esforço no sentido de possibilitar a representação visual da informação sempre que exequível e da forma mais clara e de fácil compreensão possível. Por outro lado, a dinâmica deste mercado também se reflecte na própria natureza das equipas que desenvolvem os jogos e, assim, a aplicação de edição deve ser alheia a quaisquer pressupostos acerca da composição daquelas como, por exemplo, a quantidade de profissionais que estarão envolvidos, o seu nível de envolvimento no projecto e as suas localização. É possível que seja vantajoso conseguir-se a colaboração de vários profissionais em simultâneo e também se deve considerar a possibilidade de os membros integrantes poderem estar dispersos geograficamente. Isto porque, numa área do género daquela em que estão inseridos os videojogos, onde existe uma procura contínua de talento e oportunidades estratégicas de mercado, não se pode descartar um cenário em que parte da equipa esteja num determinado país e a outra parte localizada noutro ou, até mesmo, noutro continente.

Visto que a unidade essencial de um jogo deste género são as *sprites*, convém que, ainda em fase de produção, o editor permita algum género de visualização sobre o trabalho já realizado mas ainda não concluído. Deste modo, permitir-se-á a detecção precoce de eventuais erros ou inconsistências nas animações e garante-se alguma visibilidade sobre o produto em desenvolvimento.

Nesta perspectiva, consignam-se as seguintes observações tecnológicas sobre a natureza dos jogos 2D para telemóvel e baseados em sprites:

- A visibilidade das entidades de jogo (*sprites*) é conseguida através de imagens e não de modelos renderizados em tempo de execução;
- As animações são conseguidas através da sequenciação de fotogramas (*frames*);
- A produção destas imagens é elaborada por artistas e *designers*, tipicamente agregadas em ficheiros de imagem maiores (*strips*);
- Por correrem em sistemas de recursos limitados, devem realizar uma boa gestão desses mesmos recursos, a nível de alocação de memória, de processamento, entre outros;
- Existe demasiada variação entre as especificações técnicas das plataformas móveis para se esperar que a mesma implementação corra em todas;
- Cada jogo define um caso, quer na experiência de utilização, quer nas exigências de produção.

Para resumir este sub-capítulo e partindo da listagem de requisitos para o *2D Game Editor* da empresa *YDreams SA* (ver sub-capítulo 2.2), elencam-se os principais requisitos funcionais para a aplicação:

- Desvinculada de qualquer ambiente empresarial particular;
- Vocacionada à manipulação de imagens e animações;
- Facilitadora das tarefas de manutenção;
- Ao invés de perfeitamente adaptado para dar apoio à produção de um jogo específico, ou mesmo a um género de jogo (por exemplo, plataformas ou

estratégia), o editor deve oferecer um conjunto base sólido de funcionalidades;

- Estar preparada para receber novos módulos, desenvolvidos *ad-hoc*, sempre que seja necessário fornecer funcionalidades essenciais à produção de determinado projecto;
- Facilitar o desacoplamento dos desenvolvimentos descritos no ponto anterior;
- Permitir o trabalho de várias equipas, pressupondo o mínimo sobre as condições do seu trabalho, a sua composição ou a sua distribuição geográfica;
- Disponibilizar a informação através de uma interface de utilizador aperfeiçoada, com muita interactividade e representações visuais sempre que possível, e com o mínimo de recursos a linguagem e expressões tecnológicas;
- Proporcionar um modo de pré-visualização que permita a transmissão de detalhes concretos sobre o produto em desenvolvimento, numa perspectiva próxima do resultado final;
- Providenciar funcionalidades que garantam ao utilizador algum controlo sobre os recursos que o jogo vai utilizar, quando estiver concluído e optimizar as suas alocações;
- Facilitar o esforço de portabilização dos jogos para plataformas com especificações distintas.

3.2. A Implementação

3.2.1. O ambiente de execução

Entre as várias possibilidades para o ambiente onde se desenrolará o trabalho com o *2D Game Editor On-line*, surge uma que, pelas suas características e potenciais, se apresenta como bastante interessante: a *web*. Ao planear uma aplicação que corra sobre este contexto e faça uso adequado das facilidades de comunicação que actualmente existem, dispor-se-á também dos recursos necessários para a implementação de interfaces bastante completas e interactivas, além de que a arquitectura cliente-servidor faculta um maior controlo sobre as fontes da aplicação.

Por exemplo, facilita-se a gestão dos módulos de funcionalidades que serão desenvolvidos, acoplados e retirados sempre que necessário, dado que será necessário efectuá-la apenas uma vez, ao nível do servidor do alojamento. Ao invés, como já foi referido em capítulos anteriores, se se optar por ambientes localizados, onde a computação é processada unicamente pela máquina onde corre a própria aplicação, levantar-se-iam questões ao nível do controlo de versões: na altura em que se pretendesse proceder a actualizações ao sistema, obrigar-se-ia também a repetir essas actualizações para cada uma das suas instalações.

As aplicações baseadas na *web* também facilitam o trabalho em colaboração, permitindo que se negligenciem as questões da distância geográfica: basta que uma, ou mais, equipas tenham acesso a um *browser* para que consigam, imediata e activamente, contribuir para o desenvolvimento do projecto.

À parte as questões puramente técnicas, pode-se ainda referir a elevada aceitação que as aplicações *web* gozam, hoje em dia, por parte da sociedade. Ao se apresentar um sistema que esteja baseado sobre uma plataforma tão difundida e utilizada, que apenas requer a instalação e utilização de um *browser*, estar-se-á também a granjear algum do crédito já conquistado por inúmeras aplicações bem sucedidas, amenizando-se parte da inevitável e natural resistência que os utilizadores usualmente manifestam face à adopção de uma nova solução.

3.2.2. As tecnologias de implementação

Serão necessárias implementações tecnológicas a correr em perfeita interoperabilidade, quer do lado do servidor, quer do lado do cliente, de forma a assegurar os requisitos apontados. É, então, necessário proceder-se à selecção das tecnologias em concreto que melhor se adeqúem à implementação do sistema nos moldes apresentados nos sub-capítulos anteriores, devendo-se optar por aquelas cujas características potenciem as funcionalidades propostas e ajudem a preencher o mais possível os requisitos levantados (ver sub-capítulo 3.1).

Impõe-se, por conseguinte, uma análise cuidadosa às características dos possíveis clientes/utilizadores do *2D Game Editor On-line*. Isto porque, apesar de se pretender atingir o mercado empresarial, um dos grupos com maior receptividade para novas alternativas mais económicas são os *Game Designers* amadores. Assim, a abordagens possivelmente mais vantajosa para se conseguir *feedback* em contexto de produção, será desenhar a solução para estes, distribui-la gratuitamente, proceder aos aperfeiçoamentos que se justifiquem e só então avançar para a comercialização tendo como alvo as empresas produtoras de videojogos.

Para a presente dissertação, dado que estas conjecturas extravasam o seu âmbito, classificar-se-á o potencial de lucro como irrelevante, interessando apenas focar a sua viabilidade tecnológica. Basta, então, referir que os destinatários iniciais do *2D Game Editor On-line* serão os *Game Designers* amadores, numa perspectiva de utilização gratuita em troca de *feedback* que permita melhorá-lo.

Para o efeito, e porque não existem à partida nenhumas condições de ordem tecnológica que sejam limitativas, parece sensato optar-se, pelo menos nesta primeira fase de desenvolvimento, por soluções *open-source*. Consegue-se, assim, limitar o investimento apenas ao tempo de desenvolvimento e aos custos de disponibilização da solução (alojamento, etc), sem que lhes sejam adicionados os custos com os licenciamentos que, inevitavelmente, adviriam da adopção de tecnologias proprietárias. Esta opção sai ainda mais reforçada se se notar que, como já foi referido, este desenvolvimento inicial serve apenas como um *proof-of-concept*, com o qual se pretende realizar um ensaio em campo, a fim de asseverar se as decisões de implementação tomadas foram, ou não, as mais adequadas.

Para concluir, tomada que está a decisão sobre a filosofia tecnológica que servirá de base aos desenvolvimentos - o *open-source* - devem-se ponderar as questões concretas da implementação da solução, quer ao nível do servidor, quer do cliente.

As tecnologias do lado do servidor

Visto que o conceito a implementar é o de uma ferramenta *web* de acesso *on-line*, é absolutamente necessário o recurso a uma tecnologia de servidor que se estabeleça como um mediador entre a interface de utilizador, renderizada por aplicações *browser*, e os processos de operação sobre o modelo de dados.

Foram ponderadas várias tecnologias, porém, a última escolha centrou-se entre o *PHP* e o *Java (JSP)*, por serem referências *open-source* com muita popularidade e exemplos de concretização de linguagens orientada aos objectos e de *scripts* de servidor vocacionados para desenvolvimentos *web*, respectivamente. Existem variadíssimas razões para optar por qualquer uma destas tecnologias e dado que são ambas Turing-completas (ou com potencial para o ser, dependendo das configurações [1]), os objectivos propostos para o *2D Game Editor On-line* poderiam perfeitamente ser atingidos por qualquer uma delas. No entanto, o *PHP* apresenta uma característica com muito interesse, quer do ponto de vista da investigação, quer como desafio informático: o encapsulamento, por defeito, das funcionalidades implementadas por "detrás" de endereços *web*. Por outras palavras, o *PHP* disponibiliza as implementações e os recursos num modelo funcional bastante próximo daquele que forma a estrutura fundamental da *web*, servindo-os através da interface basilar do protocolo *HTTP*, as *URL's*. Pode dizer-se que um sistema implementado em *PHP* não pretende - a não ser que tenham sido envidados esforços programáticos nesse sentido - mascarar a *web* sob um aspecto diferente, numa tentativa comum, mas por vezes infrutífera, de torná-la mais inteligível. Esta simplicidade funcional do *PHP* torna plausível uma normalização livre das comunicações entre os componentes implementados, que será transversal a toda a aplicação. Esta padronização terá como vantagem o aumento da compreensibilidade do esquema comunicacional, e isto que permitirá focar os esforços de desenvolvimento nas tarefas de implementação das funcionalidades do *2D Game Editor On-line*. Por outro lado, também facilita as tarefas de passagem de trabalho e a eventual integração de novos elementos na equipa de desenvolvimento, dado que não força os programadores a aprender novos modelos, ou mecânicas de comunicação, antes de se poderem tornar efectivamente produtivos.

Ainda na temática da escolha das tecnologias do lado do servidor, é preciso pesar outra questão: visto que se pretende que o *2D Game Editor On-line* se assuma como uma alternativa viável às ferramentas de suporte actualmente em utilização nas linhas de produção de videojogos, é necessário suprir algumas das exigências que, tipicamente, caracterizam o trabalho com as chamadas "aplicações *desktop*". De destaque entre elas, além da interactividade da interface que será abordada no próximo sub-capítulo, está a da persistência da informação. Tendo em atenção os detalhes implicados, não é expectável que todo o esforço associado à produção de um jogo se possa concluir em apenas um dia de trabalho. Deste modo, é necessário preparar o sistema para que os projectos possam ser abandonados e retomados livremente, sem restrições nem perda de informação.

Para resolver este problema, ponderaram-se várias hipóteses. Uma delas previa que se mantivesse toda a informação de cada projecto em ficheiros com formatação própria, que ficariam guardados no computador do cliente. Esta abordagem teria a vantagem de se poupar espaço no servidor, que funcionaria apenas como ambiente de trabalho, sendo necessário carregar toda a informação

[1] O *PHP*, por defeito, vem com restrições relativas ao tempo máximo que um processamento pode levar. Por outras palavras, com esta limitação em vigor, não se pode considerar que o PHP defina realmente uma linguagem Turing-completa.

de cada vez que se desejasse trabalhar num determinado projecto. No fundo, este é o modo de funcionamento da maioria das aplicações *desktop*, onde a informação de cada projecto é guardada num ficheiro de formato próprio (dá-se como exemplo o dos ficheiros dos projectos do *Microsoft Access*, que contêm a estrutura da base de dados, os próprios dados e as definições e parametrizações dos formulários, *queries*, etc). No entanto, ao contrário deste tipo de aplicação onde os cálculos e processamentos se resolvem todos localmente (discute-se aqui uma implementação em moldes simples), numa aplicação *web* tem de se acrescentar ao tempo de execução aquele que deriva das transmissões da informação. E, visto que o *2D Game Editor On-line* trabalha sobre quantidades tendencialmente volumosas de informação (com o tamanho dos ficheiros de imagem a agravar esta condição), os tempos despendidos em comunicações correm o risco de se estender por períodos demasiado longos, directamente proporcionais à informação a transmitir. Uma possível alternativa a esta metodologia de responsabilização das tecnologias do lado do cliente, seria deixar o sistema carregar as imagens directamente para o sistema de ficheiros do servidor e, do lado do cliente, guardar-se num ficheiro a estrutura do trabalho, as parametrizações e as referências para as imagens guardadas no servidor. Porém, apesar de se subtrair o tamanho das imagens do total do volume da informação a transmitir para o servidor a cada nova sessão de trabalho[1], surgirão outras limitações quando se pretender retomar o trabalho num determinado projecto. Por exemplo, um sistema desenhado deste modo não permitirá aceder às informações guardadas no servidor a partir de outras estações de trabalho, sem que se tenha o ficheiro com as parametrizações; pior ainda, se porventura este ficheiro se perder, não será possível retomar o trabalho no projecto e, como consequência não menos grave, as imagens carregadas ficarão "perdidas" e a ocupar espaço no disco do servidor. Dá-se assim por descartada a hipótese de manter a informação, ou parte dela, nas máquinas cliente. Resta, então, ponderar a alternativa de guardar toda a informação do lado do servidor.

Colocam-se, então, as seguintes alternativas: guardar toda a informação no seu sistema de ficheiros, fazer uma utilização exclusiva de sistemas de gestão de bases de dados para esse fim ou guardar parte da informação em base de dados e a outra parte no sistema de ficheiros.

A primeira hipótese, implica que se mantenha a informação em ficheiros formatados e terá problemas de crescimento e manutenção da informação. Seria necessário, para minimizar o tempo de leitura e de acesso à informação, definir uma sintaxe própria para os ficheiros produzidos de modo a que o sistema os pudesse processar em tempo útil. Todavia, mesmo que se evitasse o esforço de criar a dita sintaxe por meio da utilização de uma formatação *standard*, como o *XML*, haveria ainda decisões a tomar a nível arquitectural: se fosse seguida uma metodologia de criar apenas um ficheiro com toda a informação de um projecto, esse ficheiro cresceria bastante devido à complexidade da informação. Se, por outro lado, fossem criados vários ficheiros mais pequenos em que cada um contivesse parte da informação do projecto, o seu número aumentaria em

[1] Neste caso, apenas ocorreria transmissão das imagens para o servidor uma vez, quando fossem carregadas. A partir daí, só haveria transmissão das imagens que se pedisse à aplicação, com a vantagem de que, geralmente, as velocidades de *download* são superiores às de *upload*.

consonância com o crescimento do projecto. No fundo, esta proposta é uma solução deselegante, dado que se trata de escolher entre um modelo de dados pesado e pouco legível ou um com demasiada entropia, respectivamente (e que acabaria por ter, como consequência, problemas de legibilidade semelhantes).
Pelas razões apontadas, é de prever que qualquer uma destas abordagens possa conduzir a problemas quando se pretender proceder a futuras manutenções.

Se, por outro lado, for utilizada exclusivamente uma base de dados para gerir toda a informação, vislumbram-se outras situações problemáticas que também desaconselham esta alternativa, estando relacionadas com a forma como a informação é representada pelo *SGBD*. O caso é que, devido ao tamanho típico dos ficheiros de imagem, para que se pudesse guardar esta informação na base de dados, ter-se-ia de optar pelo tipo *BLOB* (*Binary Large Object*), ou por uma variante deste, de acordo com as especificações do *SGBD* escolhido.

Embora isto não represente, por si só, um problema enquanto a informação for toda gerida pela mesma plataforma de base de dados, verifica-se que existem incompatibilidades entre as implementações deste formato em diferentes plataformas. Os problemas surgiriam, então, ao dar-se a eventualidade de se pretender migrar os dados para uma nova plataforma [1], que não seja compatível. Estes problemas, como quase todos na área da informática, poderão eventualmente ser resolvidos ou contornados à custa da aplicação de um esforço nesse sentido. No entanto, como os sistemas podem não ser facilmente compatibilizados, torna-se preferível apostar numa solução que não implique guardar os ficheiros de imagem dentro da base de dados.

Em face do referido, uma abordagem mista parece a mais indicada, na qual a informação seja guardada pelo *SGBD* e apenas as imagens fiquem alojadas directamente no sistema de ficheiros. Isto implica, naturalmente, que o sistema possua uma forma de referênciação das imagens, para que se mantenha a consistência entre a informação guardada nas tabelas da base de dados e os ficheiros em disco. Mas, apesar desta limitação, esta é a solução padrão encontrada em aplicações informáticas baseadas na *web*. Do modo referido, existirão referências na base de dados - tipicamente caminhos no sistema de ficheiros do servidor - que apontem para ficheiros de imagem específicos, permitindo referenciá-los e trabalhar sobre estes no contexto do *2D Game Editor On-line.*

Ao nível da tecnologia escolhida para a implementação, refere-se que a base de dados foi implementada segundo o modelo relacional, sobre o *SGBD MySql* [2]. Esta escolha foi bastante fundamentada, dado que existe um "pacote" de *software* muito difundido e totalmente gratuito, que já contém as tecnologias escolhidas prontas a utilizar, o pacote *LAMP* (*Linux*, *Apache*, *MySQL* e *PHP*). Além disto, as versões mais recentes do *MySQL* já dão suporte a sintaxes avançadas de *SQL*, a funcionalidades de gestão de bases de dados mais sólidas que implementam fielmente o modelo

[1] As possibilidades de evolução, mesmo ao nível do modelo de dados, não são de descartar, sobretudo num sistema com os requisitos de adaptabilidade do *2D Game Editor Online*.

[2] Para mais detalhes, ver sub-capítulo 3.4.

relacional, através de recursos como as *sub-queries*, as transacções e as chaves. Por outras palavras, este SGBD daria suporte a todas as exigências de funcionalidade que constam dos requisitos levantados para o *2D Game Editor On-line*. De especial relevo, as transacções revestem-se de utilidade particular, devido à relativa complexidade da informação trabalhada e ao facto de esta estar parcialmente guardada na base de dados e no sistema de ficheiros do servidor. Assim, visto que o SGBD não tem recursos para correr comandos no sistema operativo do servidor, é necessário executar alterações na base de dados e no sistema de ficheiros. A hipótese de correr várias *queries* dentro de uma transacção, e a possibilidade de a finalizar explicitamente (executar uma instrução de *COMMIT* ou *ROLLBACK*), permite que estas operações sejam sincronizadas, garantindo deste modo a consistência da informação (Converse et al. 2004).

As tecnologias do lado do cliente

A decisão de implementar o sistema como uma aplicação *web* implica que os acessos sejam realizados através de programas cliente que serão, no caso concreto, aplicações *browser*. Para que a interface de utilizador possa ser renderizada por estes, existem várias possibilidades, cada uma apresentando vantagens e desvantagens próprias.

No caso mais simples, poder-se-ia criar a interface apenas como documentos *HTML* convencionais. Desta forma estar-se-ia, no entanto, a limitar o trabalho com o *2D Game Editor On-line* ao preenchimento de formulários estáticos, o que não se coaduna com os requisitos de interactividade levantados.

Uma abordagem mais dinâmica e interactiva, mas também mais complexa e exigente do ponto de vista da programação, seria a criação de uma estrutura (*framework*) numa linguagem de *scripting* suportada pelos *browsers* – preferencialmente o *JavaScript*, por questões de compatibilidade. Do modo descrito, manipular-se-ia directamente o próprio *DOM* [1], multiplicando-se as possibilidades de interacção com o utilizador. Por outro lado, o suporte ao *JavaScript* varia consoante o *browser* e até entre versões do mesmo *browser*, o que se torna uma limitação a esta abordagem, na medida em que se podem estar a desenvolver funcionalidades e mecânicas que não funcionarão do modo esperado em todos os *browsers*.

À parte das questões relacionadas com a compatibilidade da implementação, existem outras que pesam em detrimento de uma solução baseada unicamente em *scripting* do lado do cliente: a transmissão da informação do servidor para o cliente seria concretizada de modo fragmentado, com recurso a um formato *standard* para o transporte, como o *XML* ou *JSON*[2]. Posteriormente, para que se possa renderizar adequadamente uma interface de utilizador amigável, essa informação descarregada será utilizada para preenchimento de *templates* ou de elementos *HTML*, instanciados dinamicamente pelo *JavaScript*. A hipótese que seria mais fácil de manter e desenvolver, consistiria na utilização dos referidos templates, que

[1] Document Object Model. Ver Glossário.
[2] JavaScript Object Notation. Ver Glossário.

poderiam ser implementados como ficheiros *HTML* simples (se bem que com os *placeholders* específicos do motor de *templating* escolhido) e guardados no servidor.

Esta fragmentação, no sentido em que obriga o *browser* a descarregar em primeiro lugar o documento *HTML* base, o respectivo ficheiro *JavaScript*, e só depois pedir os dados e os *templates*, obriga à realização de várias requisições ao servidor, e isso que acarretará, naturalmente, um custo temporal associado a estas comunicações.

Por outro lado, a hipótese mista, a mais tradicional em termos de *DHTML*, consiste na criação de páginas *HTML* com dinamização por *JavaScript*. Esta abordagem traz vantagens na medida em que acarreta o melhor dos dois extremos apresentados: a legibilidade e manutenção dos ficheiros *HTML* básicos, a menor quantidade de comunicações com o servidor e o dinamismo das aplicações implementadas em *JavaScript*.

Existem, no entanto, outras exigências funcionais do lado do cliente, que não podem ser satisfeitas com recurso exclusivo às tecnologias supracitadas. É necessário que o sistema possa dar, também, apoio confiável às tarefas minuciosas que fazem parte do trabalho de produção de videojogos. Refere-se, como exemplo de uma tarefa que se enquadra nestas condições de exigência de detalhe e de interactividade, o corte dos fotogramas, ou *frames*, a partir de imagens de grandes dimensões, as *strips*. Estas *strips* consistem em ficheiros de imagem, que agregam as várias texturas que as personagens de jogo assumem durante uma, ou mais, animações, os fotogramas. Deste modo, para jogos 2D baseados em *sprites*, pode-se referir que uma animação não é mais do que uma sequênciação de fotogramas, à semelhança do que sucede com os desenhos animados tradicionais e ao contrário de jogos em 3D, que exigem renderizações constantes de sólidos geométricos com recurso a motores gráficos.

A tarefa de corte dos fotogramas exige, então, uma ferramenta que permita a manipulação das imagens ao nível do píxel, para que se possa reduzir a área não ocupada, no corte realizado. Esta área desperdiçada é, como será compreensível, inevitável na medida em que nos próprios fotogramas, enquanto cortes rectangulares de uma imagem maior, existe sempre área que não está ocupada pelos píxeis da figura/*sprite* (a não ser, claro, que se trate de um fotograma onde a *sprite* assuma um formato rectangular). Como ilustração deste problema, veja-se a Figura 15.

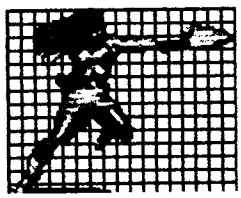


Figura 15. Área não útil de um fotograma (quadriculado verde).

O interesse nesta redução está em que, ao diminuir-se a quantidade de área não ocupada num fotograma, está-se também a reduzir o tamanho do ficheiro de imagem, conseguindo-se assim poupar na alocação da memória (*heap*) do telemóvel onde se corre o jogo. O recurso tecnológico mais simples para implementar este corte minucioso seria o *JavaScript*. Este *script*, com acesso aos procedimentos de manipulação do *DOM*, consegue realmente ter sensibilidade ao nível do píxel mas, por outro lado, além da questão da implementação variável entre diferentes *browsers*, deve-se referir que estes (*browsers*) foram desenvolvidos com o objectivo de permitir a visualização de documentos *HTML* e não para funcionarem como um ambiente de execução de aplicações.

Isto tem como consequência a existência de muitos recursos da máquina cliente que não são facilmente manipuláveis, nem mesmo com a utilização de *scriptings* locais. Será o caso, por exemplo, do nível de ampliação (zoom) com que o *browser* está a visualizar o documento. O processo para que a programação *JavaScript* possa conhecer este valor não é trivial e varia consoante o *browser* - se é que chega a ser, de todo, possível em alguns.

Devido a esta falta de controlo e segurança, verifica-se que o recurso exclusivo a implementações em *script* do lado do cliente não é inteiramente confiável, pelo que será preferível a opção por outras tecnologias que corram também do lado do cliente mas garantam um controlo programático maior.
Existem, como é natural, várias possibilidades, entre as quais se destacam as *applets* em *Java* ou o *Flash*. Dada a opção pelo *open-source* que caracteriza todas as decisões tomadas ao nível das tecnologias do *2D* Game *Editor On-line*, o *Java* será a solução mais óbvia, sendo totalmente gratuito e igualmente (ou mais) poderoso em termos das possibilidades que oferece.

Independentemente da tecnologia escolhida para o desenvolvimento inicial, foram tomadas acções no sentido de facilitar as implementações concretas destas "janelas" de interactividade. A título ilustrativo destes cuidados, pode-se referir que as funcionalidades que, de alguma forma se afastem do conceito tecnológico da "janela" (como as alterações directas ao contexto *DOM* onde a *applet* esteja inserida, ou algumas comunicações com o servidor), serão executadas por intermédio de implementações *JavaScript*. Deste modo, para se poder trocar, futuramente e com um mínimo de alterações, uma *applet* em *Java* por um recurso implementado em *Flash*, basta que neste sejam incluídos os meios para que possam invocar as mesmas funções *JavaScript*. Facilita-se, assim, a troca de componentes e aligeira-se o seu esforço de desenvolvimento, na medida em que não é necessário implementar as funcionalidades directamente, bastando invocar as funções já existentes nas páginas *web*.

Dá-se então por concluída a escolha das tecnologias a utilizar do lado do cliente. Porém, se se levar em conta que o objectivo principal do *2D Game Editor On-line* é posicionar-se como uma alternativa viável às actuais aplicações de edição de jogos, desenvolvidas internamente por cada empresa, constata-se que não estão ainda assegurados todos os requisitos que foram anteriormente levantados. Como estas ferramentas são tendencialmente implementadas como aplicações *desktop*, de execução local e sem recurso a comunicações com servidores remotos, e devido à

elaborada apresentação gráfica das suas funcionalidades, o seu potencial de interactividade com o utilizador e velocidade de resposta às suas acções é bastante elevado. Estas características impõem-se como um ponto de resistência a considerar, por parte dos utilizadores, no processo de troca dos editores habituais pelo *2D Game Editor On-line*. Este último, para poder concorrer com estes editores, terá necessariamente de dispor de uma interface de utilizador que seja, pelo menos, tão atractiva como as daquelas a que se propõe substituir. No entanto, não é este o caso das típicas aplicações on-line, baseadas em formulários e que evidenciam os tempos de espera nas comunicações com o servidor que, estão associados ao padrão *request-response* do protocolo *HTTP* subjacente, ainda que certas funcionalidades possam estar melhoradas por *applets* ou alternativas semelhantes.

Realmente, porque as aplicações *web* correm em ambientes de execução geridos por *browsers*, estão também sujeitas aos constrangimentos dos protocolos de comunicação que este utiliza, e o protocolo base da *web*, o *HTTP*, tem algumas restrições que podem limitar o desempenho da interface com o utilizador. Nomeadamente, neste protocolo a mecânica das comunicações está conforme a um modelo que não tem estados e é baseado num paradigma *request-response*. Estes princípios não permitem que sejam dados, de forma directa, os recursos e o apoio adequados à implementação de uma aplicação que se pretende altamente interactiva.

Como forma de contornar estas limitações, podem incluir-se meios que permitam ao *browser* estabelecer comunicações assíncronas com o servidor, sem que os utilizadores se apercebam dos tempos de latência que seriam por demais evidentes caso estas se apoiassem, única e exclusivamente, sobre o protocolo *HTTP*.

Existe, para o efeito, um conjunto de técnicas - não tecnologias - que permitem que, correndo em segundo plano, o *browser* consiga estabelecer comunicações deste tipo com o servidor, e que são sumariamente referenciadas como *AJAX* [1]. Este conceito não define nenhuma nova tecnologia, dado que apenas utiliza o *JavaScript* e o *XML*, e o nome está algo desajustado face às suas reais características, dado que a sua utilização não obriga à assincronia nas comunicações nem sequer à utilização do *XML* como formato para as transmissões. Apesar disto e como já foi referido, foi justamente a possibilidade de incluir comunicações assíncronas no sistema que motivou a utilização do *AJAX*. No que diz respeito ao formato da transmissão da informação, dado que as comunicações com o servidor que necessitam de correr em assincronia dizem, na sua generalidade, mais respeito à invocação de operações de transformação sobre o modelo de dados no servidor, do que propriamente à passagem de volumes de informação, o formato adoptado não foi o *XML*.
As comunicações com *AJAX* implementadas no *2D Game Editor On-line* processam-se do seguinte modo:

[1] Asynchronous JavaScript And XML. Ver Glossário.

- O cliente faz uma requisição *AJAX* directamente a uma página *PHP* (*controller MVC*) do servidor. Nesta requisição serão incluídas as informações sobre os procedimentos ou funções a correr;
- O servidor, por seu lado, transmite as informações resultantes das operações para o cliente, num formato textual simples ou, sempre que se prove mais adequado, directamente em código *HTML* ou mesmo *JavaScript* pronto a executar.

Visto que a informação passada para o servidor vai escrita directamente na requisição, detectou-se uma situação problemática quando o método escolhido foi o *GET* (que está, de qualquer modo, devidamente documentada): o *browser* indexava-a em *cache*. Isto originava comportamentos estranhos e, consequentemente, difíceis de descrever. Pode-se referir, no entanto, que a informação não era devidamente processada, quer do lado do servidor, quer do lado do cliente. Este problema foi superado trocando-se o método de requisição de *GET* para *POST*, que não implica que sejam guardadas por defeito as requisições e respectivas respostas (Berners-Lee e Connolly, 1995).

Uma das operações que convém executar em assincronia, de modo a que as alterações no modelo de dados sejam desencadeadas ao mesmo tempo que são invocadas pelo utilizador e este possa continuar a trabalhar enquanto elas decorrem, é o corte de fotogramas. Assim sendo, quando o utilizador define e confirma uma área de corte na *applet* apropriada, o *browser* envia simultaneamente uma mensagem através de *AJAX,* indicando ao servidor que deve criar um fotograma, ou seja, realizar um corte na *strip* nas mesmas condições[1] em que foi definido pelo utilizador. Uma vez recebido o resultado da operação do lado do servidor, o cliente actualiza a *applet*, numa indicação inequívoca para o utilizador, de que o fotograma foi guardado ou de que ocorreu algum problema.

A título de curiosidade, refere-se que foi detectado um comportamento implementado nos *browsers* actuais: por definição, estes apenas podem realizar duas requisições concorrentes para o mesmo domínio, baseadas em *AJAX*. Este efeito pode ser observado na *applet* quando se realizam vários pedidos de corte de fotogramas, sem esperar que os anteriores se concluam. Os cortes dos fotogramas vão sendo marcados como concluídos aos pares, conforme se vão resolvendo as requisições, pela ordem em que foram realizados os pedidos.

Em resumo, e no que tange a este sub-capítulo, as tecnologias escolhidas para implementação do lado do cliente serão o *HTML,* que define a base do documento, o *JavaScript* com recurso a *AJAX* para permitir que sejam executadas comunicações com o servidor em segundo plano e *applets* em *Java*, como forma de proporcionar um controlo mais profundo e detalhado sobre certas tarefas mais exigentes como o corte de fotogramas.

[1] As mesmas coordenadas do canto superior esquerdo e a mesma altura e largura.

3.2.3. Metáforas

Visto que o *2D Game Editor On-line* se trata de uma aplicação muito orientada ao utilizador, também faz uso de algumas metáforas, com o intuito de facilitar a aprendizagem e potenciar a compreensibilidade do sistema.

Outro detalhe de importância, a que sempre se deu atenção, foi o de criar consistência externa entre os recursos utilizados na aplicação em desenvolvimento, com os recursos de outras a aplicações populares, com elevada aceitação e difusão no mercado.

Seguem-se as descrições de algumas das opções de desenho e metáforas que foram utilizadas, no sentido de criar uma interface mais imediata e evidente.

Projectos e plataformas

A inclusão do conceito de "projecto" no sistema possibilita que o trabalho de um utilizador, num determinado jogo, fique "encapsulado" num projecto. Embora não diga respeito quotidianamente a um objecto real e tangível, permite que os desenvolvimentos de um jogo possam ser referidos, globalmente, sob o nome do seu projecto. Esta facilidade, que não é, do ponto de vista informático, mais do que informação inter-referenciada na base de dados, permite extrapolar o ambiente de trabalho gerido pelo *2D Game Editor On-line,* para dentro do contexto de produção empresarial. Assim, consegue-se que os conceitos da aplicação possam ser discutidos e apreendidos intuitivamente pelos utilizadores e a restante equipa, mesmo que nunca tenham tido contacto directo com o sistema.

Existe um detalhe importante, do ponto de vista tecnológico, que implica a inclusão de outra metáfora dentro do conceito do projecto: a plataforma. A questão que motivou esta inclusão prende-se com as capacidades de processamento e os recursos limitados de que dispõem as plataformas onde se pretende correr o jogo. Concretizando, há telemóveis no mercado que, por terem mais memória disponível, permitem também que mais imagens estejam alocadas em determinado momento sem que seja prejudicado o desempenho do programa. Assim, para estas plataformas, poder-se-ão definir *sprites* com animações bastante fluidas, obtidas pela sequenciação de muitos fotogramas.

Por outro lado, pode ser também desejável que o mesmo jogo possa correr noutros telemóveis com menos recursos. Porém, de forma a não prejudicar o desempenho nestes, uma medida tomada recorrentemente é a redução da quantidade de fotogramas que são utilizados para renderizar as mesmas animações.

Figura 16. Animação "andar" fluida, com 6 fotogramas.

Figura 17. Animação "andar" menos pesada, com 4 fotogramas.

Nas figuras acima podem-se ver duas propostas para uma animação que representa o estado "andar" de uma *sprite*. Estas animações podem ser conseguidas à custa de seis ou de quatro fotogramas. A Figura 16 define uma animação fluida, composta por seis fotogramas e isto obriga a que, para acelerar as transições entre eles, as seis imagens estejam carregadas na memória da plataforma onde se está a correr o jogo. Por outro lado, a Figura 17 mostra uma animação adequada a sistemas mais limitados, dado que obriga a uma menor alocação de memória para exibir a mesma animação.
Consegue-se assim, à custa da fluidez nas animações das *sprites*, tomar uma decisão baseada no equilíbrio desempenho-fluidez e nos resultados que isso terá na qualidade do produto final.

Dentro do *2D Game Editor On-line*, uma solução viável e simples para gerir estas variações nas implementações do jogo, consistiria na criação de um novo projecto para cada nova plataforma. Porém, esta abordagem não interessa, na medida em que com ela se iria perder a semântica da associação entre um projecto e um jogo (a relação passaria a ser entre um projecto e uma versão de um jogo). Para resolver este problema, implementou-se no *2D Game Editor On-line* uma nova metáfora, a plataforma, que consiste numa representação virtual que agrega toda a informação que foi trabalhada e optimizada especificamente para correr em determinado sistema. Desta forma, um projecto poderá estar associado a um número ilimitado de plataformas, tantas quantas sejam necessárias para garantir o bom desempenho do jogo, nas suas potencialmente várias versões, no mercado (pelo menos no tocante à questão da compatibilidade).

O sistema de ficheiros

A informação gerida pelo *2D Game Editor On-line* é, do ponto de vista estrutural, algo complexa. Dá-se um exemplo: todas os fotogramas estão relacionados com uma animação que, por seu lado, está associada a uma *sprite*. As diversas sprites relacionam-se com os desenvolvimentos para uma plataforma que, está, por último integrada num projecto, de acesso condicionado a um grupo de utilizadores, dentro das permissões concedidas ao perfil que lhes tenha sido atribuído (para mais detalhes, ver Figura 45, no capítulo 3.4). Estas relações, tipicamente de composição, entre os objectos de um projecto, podem ser representadas numa visualização em árvore, como se ilustra na figura seguinte.

Figura 18. Árvore de um projecto

Do modo representado, a estruturação dos projectos assume a forma de directórios – os nós da árvore – que contêm outros nós ou referências para os dados (Figura 18). Passa-se, portanto, a ideia ao utilizador de que, no âmbito do *2D Game Editor On-line*, os trabalhos ficam organizados num sistema de ficheiros com as relações e as hierarquias usuais nos sistemas operativos mais populares.

A ferramenta de pré-visualização

É desejável que, quando um utilizador esteja a realizar a sequenciação dos fotogramas de determinada animação, tenha a possibilidade de correr essa animação incompleta, com os fotogramas que já tenham sido alinhados até ao momento. Dessa forma, permite-se que sejam detectados, atempadamente, eventuais problemas que possam ocorrer na animação, mas que não sejam facilmente identificados olhando apenas para o alinhamento estático dos fotogramas individuais. A interface desenhada para esta ferramenta de pré-visualização foi inspirada naquelas que costumam ser utilizadas em aplicações leitoras de conteúdos multimédia, ou de vídeo, mais concretamente. Disponibiliza-se assim ao utilizador uma área de visualização, uma caixa de texto que permite a selecção da velocidade de passagem dos fotogramas, e os tradicionais botões de *play* e *pause* que lhe permitem controlar o desenrolar da animação (ver Figura 19).

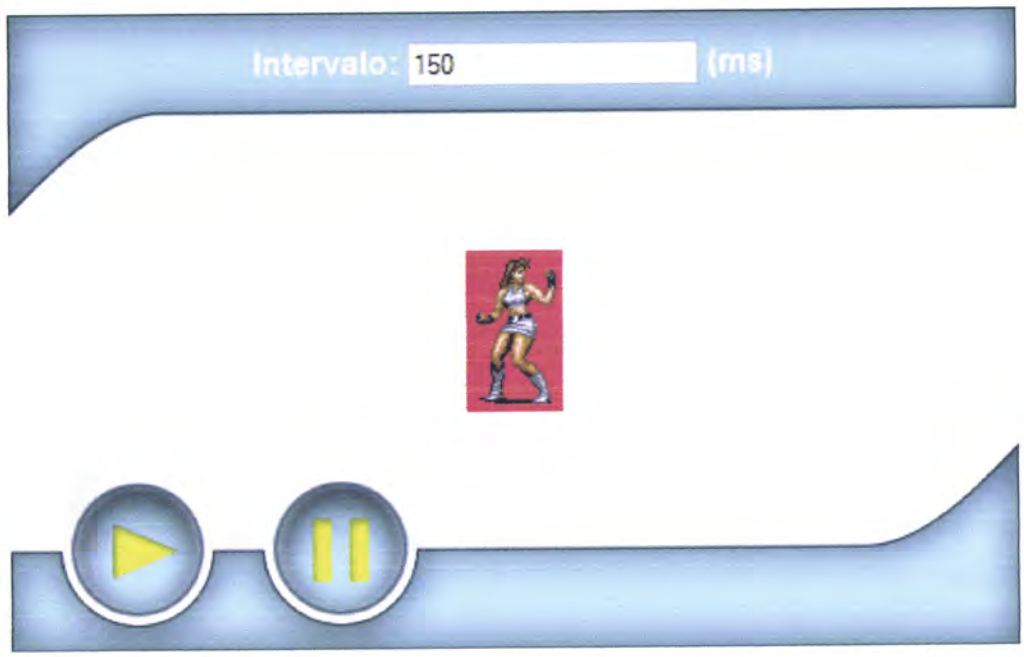


Figura 19. Ferramenta de pré-visualização de animações.

Drag and Drop

Uma das metáforas mais recorrentes, nas aplicações informáticas actuais de edição de imagens, é o *drag and drop*. Este recurso costuma ser utilizado para definir áreas onde se deseja operar alguma transformação (por exemplo, operações de corte ou remoção) ou para mover determinado elemento da interface para outro local (deslocamento de fragmentos de uma imagem). O *2D Game Editor On-line* implementa esta metáfora no processo de selecção de fotogramas, criando-se assim consistência externa entre o sistema e as referidas aplicações de manipulação de imagens.

A selecção de um fotograma a cortar é conseguida, então, por arrasto, definindo-se uma área de corte rectangular que fica delineada com tracejado vermelho, tal como se pode ver em detalhe na Figura 20.

Figura 20. Definição, por arrasto, de uma área de corte.

3.2.4. Arquitectura Interna do Sistema

Recorda-se aqui que um dos requisitos que foram levantados para o *2D Game Editor On-line*, foi o de este se constituir como um sistema desvinculado ao máximo de qualquer contexto empresarial em particular, de modo a alargar o leque de potenciais interessados. Este esforço de generalização também passa, como já foi referido em capítulos anteriores, por promover o isolamento do conjunto das funcionalidades oferecidas, daquelas que são exigidas pela produção particular de qualquer género de jogo. Se, por um lado, esta estratégia facilita a integração da aplicação dentro dos contextos laborais de produção, por outro não garante a resposta eficaz do sistema face aos requisitos individuais de cada processo concreto de produção de um jogo. Para solucionar este problema, sem perder a vantagem estratégica da desvinculação, concebeu-se a aplicação como uma plataforma estruturada, quer a nível de funcionalidades, quer de programação. Isto para facilitar a alteração, remoção e criação individual de componentes que proporcionem, já em ambiente de produção, as necessárias aproximações funcionais às necessidades reais sentidas na produção de um novo jogo. Destaca-se, então, o valor que pode advir da adopção de uma arquitectura modularizada, que mantenha legível o código produzido e sirva como guia para futuras evoluções da aplicação e de novos componentes.

Existe, no domínio da engenharia, um conceito vocacionado à resolução deste tipo de questões - os padrões - sobre os quais se realizou o estudo descrito no sub-capítulo 2.3. No sub-capítulo que se segue, justifica-se a escolha do padrão que foi aplicado ao *2D Game Editor On-line*, para resolver as questões que foram levantadas com o referido estudo.

A Escolha do Padrão

A dar sequência ao assunto do sub-capítulo 2.3, lembra-se que tanto o padrão *Presentation-Abstraction-Control* como o *Model-View-Controller* têm o potencial de responder adequadamente aos requisitos.

No entanto, fazendo uma análise mais concreta e próxima da realidade do sistema *2D Game Editor On-line*, salienta-se que o padrão *Presentation-Abstraction-Control* possui um tecido arquitectónico mais rígido que o do *Model-View-Controller*, dado que obriga à estruturação geral da arquitectura em agentes, ou tríades, cujos elementos componentes possuem responsabilidades bem definidas. Neste padrão, toda a comunicação entre os vários agentes é realizada, única e exclusivamente, pelo componente *Controller* e propagada de forma hierárquica (ver sub-capítulo 2.3.3, secção *Presentation-Abstraction-Control*). Concretizando, um dos inconvenientes que se obteria com a adopção do padrão *Presentation-Abstraction-Control* manifestar-se-ia quando, por exemplo, se tentasse criar uma nova *widget* para representar doutra forma, que não uma visualização em árvore, a estrutura dos projectos. Seria necessário criar, então, uma nova camada ou um novo agente que estivesse integrado numa já existente, com os respectivos três componentes *Presentation*, *Abstraction* e *Control*. Em adição, seria também necessário proceder à implementação dos mecanismos de transmissão de eventos e garantir a consistência dos mecanismos de propagação entre o novo agente e os já

existentes. Ao invés, com o *Model-View-Controller*, esta alteração poderia ser implementada apenas com a criação de um novo par *View-Controller*, que fizesse referência ao *Model* já implementado para a *widget* inicial (ver Figura 21).

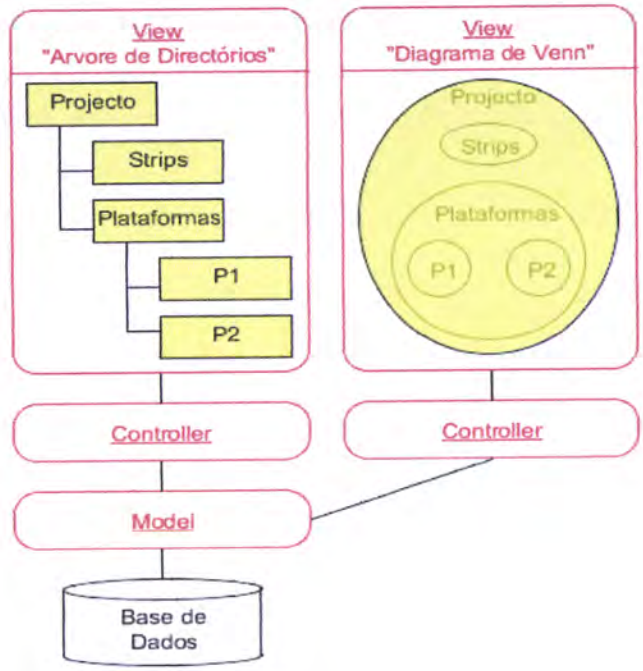


Figura 21. Nova widget para visualização da informação.

Por outro lado, dado que o *2D Game Editor On-line* ainda é um sistema em desenvolvimento, a questão da escolha da arquitectura não pode ser levada ao extremo de se colocar como um obstáculo à implementação, apenas porque se prevê que determinados problemas possam surgir no futuro. Como foi explicado no sub-capítulo 2.3, ao optar-se por determinado padrão, deve ter-se em mente que "cada tipo está orientado à resolução de conjuntos de problemas de programação específicos e recorrentes, procurando dar-lhes resposta de uma forma segura e testada em situações anteriores semelhantes". Assim, o referido "conjunto de problemas", ou requisitos, que se pretende satisfazer[1] diz simplesmente respeito à organização do código, à sua legibilidade e à incorporação, na aplicação, de uma metodologia de desenvolvimento que oriente, de forma segura, a criação de novos componentes e permita uma visão globalizada do sistema e das suas mecânicas internas.

Como não é boa prática ignorar a realidade do negócio que se pretende alcançar, frisa-se aqui o principal problema das intervenções de alteração aos editores de videojogos: a urgência das novas funcionalidades. Deste modo, se pela urgência desses desenvolvimentos não for possível utilizar o padrão, a arquitectura vigente no restante sistema não se deve colocar como um obstáculo maior à integração dos novos componentes, ainda que as suas arquitecturas possam não lhe estar conformes.

[1] Ver sub-capítulo 3.2.4.

Em resumo, pretende-se que arquitectura escolhida sirva apenas como uma referência formal que garanta que, caso seja seguida durante o desenvolvimento de um novo componente, este terá facilidade em integrar-se no sistema mas que, caso não seja, o sistema também não o rejeitará por essa razão.

Deste modo, para que se facilite o seguimento da arquitectura do *2D Game Editor On-line*, quando se desenvolvem novos componentes, a sua acessibilidade torna-se um factor deveras importante e, neste sentido, apesar da semelhança que os caracteriza, o *Model-View-Controller* é um padrão mais simples, compreensível e maleável.

Como nota final, refere-se que não existe um padrão cuja escolha se possa afirmar como absolutamente correcta ou absolutamente errada. Todos têm vantagens e desvantagens próprias e um problema, ou problemas, ao qual se direccionam.

Dessa forma, a escolha de um padrão deve partir de um conjunto de requisitos bem definido e como no caso do 2D Game Editor On-line a principal questão é a da legibilidade e compreensibilidade do código, o padrão seleccionado foi o menos restritivo, o *Model-View-Controller*.

A implementação do padrão

Apesar da adequação do padrão *Model-View-Controller* aos requisitos do *2D Game Editor On-line*, este é uma aplicação *web* – assente, portanto, sobre comunicações protocoladas pelo *HTTP*. Ora, este padrão foi desenhado, originalmente, para ser aplicado à arquitectura de sistemas com programação baseada sobre o paradigma da orientação aos objectos e assim, a aplicação do padrão de arquitectura *Model-View-Controller* difere daquela que seria realizada caso se tratasse de uma aplicação *desktop*, podendo-se referir de forma generalizada:

- A *View* é concretizada em páginas *HTML* enviadas para o *browser*;
- A implementação dos *Controllers* tem de tomar medidas, caso se justifique, que contornem a convenção sem estados (*stateless*) do protocolo – deixando de ser, eles próprios, puramente reactivos;
- Devido à separação física das máquinas onde correm a *View* e o *Model* e por facilidades de programação, deve ser o *Controller* a decidir quando a *View* deve ser actualizada.

Segue-se, então um esquema do funcionamento das comunicações na implementação adaptada do *Model-View-Controller* que foi realizada para o *2D Game Editor On-line*:

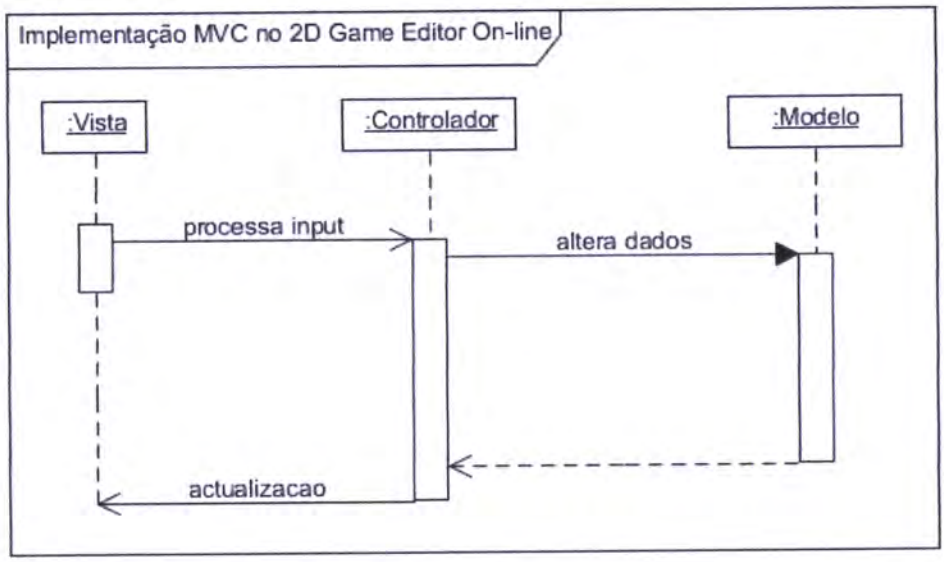


Figura 22. Comunicações entre componentes MVC, na versão do editor.

Assim, quando a *View* detecta *input* do utilizador, informa o *Controller*, que trata de o processar invocando, se for caso disso, os métodos de actualização dos dados no *Model*. Nesta implementação concreta, o *Controller* aguarda os resultados do *Model* para que possa, em sincronia, notificar a *View* para que esta se actualize.

Como se pode ver, não é este o funcionamento estabelecido pelo padrão *Model-View-Controller*, mas sim uma variante deste, adaptada para se adequar ao ambiente *web* e suficientemente sólida para poder aplicar-se transversalmente a toda a arquitectura do sistema.
Segue-se uma breve descrição das implementações realizadas dos três componentes.

O Componente View

Este componente é, conforme as definições do *Model-View-Controller*, o responsável pela renderização do *Model*, apresentando-o ao utilizador sob uma forma inteligível e interactiva, podendo existir várias *Views* distintas sobre o mesmo *Model*. No caso particular do *2D Game Editor On-line*, a implementação deste componente foi realizada através de páginas *HTML* e *JavaScript,* que são renderizadas pelo servidor *PHP* de acordo com o estado da aplicação e o modelo de dados (geridos pelo componente *Model*) sendo, depois, enviadas para o cliente. Este componente pode ser implementado em pequenos módulos independentes, como é o caso da *View* que renderiza o modelo de dados sob a forma de uma árvore de directórios, ou de páginas mais abrangentes que agregam outras vistas, como a página principal do sistema (ver Figura 23).

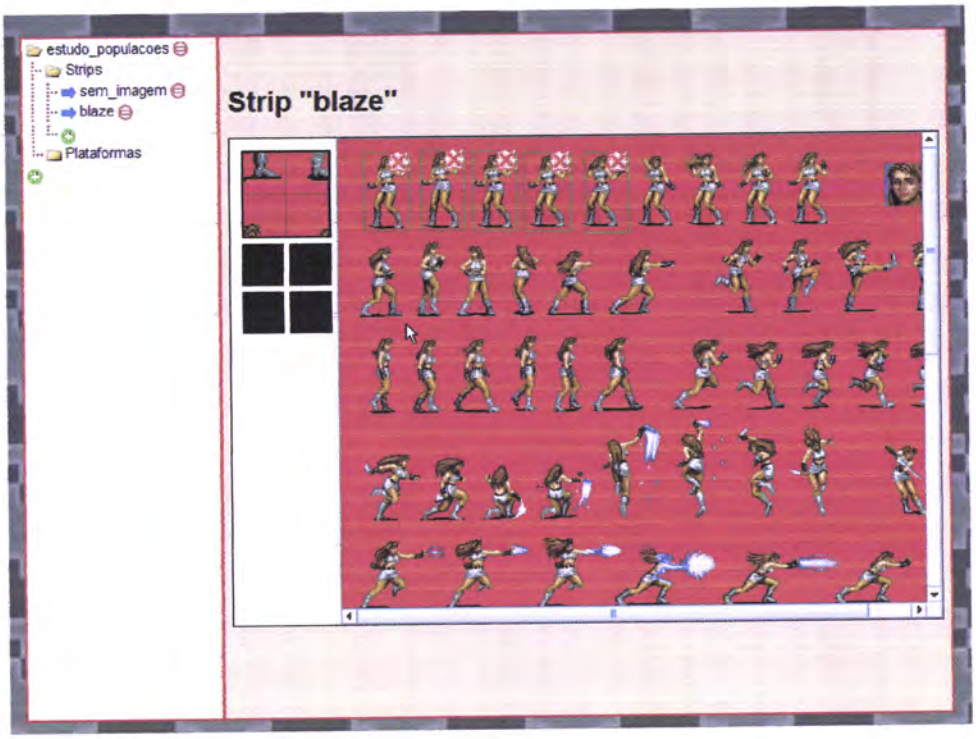


Figura 23. Vista geral do *2D Game Editor On-line*.

As funcionalidades implementadas em *JavaScript* contribuem para a dinamização da interface com o utilizador, na medida em que permitem que esta reaja a alguns tipos de *input* sem que haja necessidade de iniciar um ciclo *requisição-resposta* tradicional. A razão é clara: por questões de desempenho, todo o processamento que não necessite de acesso ao modelo de dados, vai correr do lado do cliente. Assim, distribui-se a responsabilidade do processamento pelo cliente e pelo servidor e alivia-se a carga no servidor, ficando este dedicado exclusivamente à gestão do modelo de dados e ao processamento dos *inputs* que lhe introduzam alterações.

Traduzindo para os termos da arquitectura adoptada, os processamentos puramente reactivos dão-se do lado do *browser*, estando implementados na componente *View*. Do mesmo modo, os processamentos que decorrerão no servidor são aqueles que estão no âmbito das competências do Model e do *Controller*. Esta "agilização" do esquema rígido de comunicações preconizado pelo *Model-View-Controller*, que passa a ser seguido unicamente em resposta às acções de alteração do *Model*, é bastante comum para se conseguirem respostas ao utilizador mais fluídas e imediatas, sempre que as suas acções não requeiram comunicações com o servidor.

O Componente Model

Este componente trata da implementação da lógica dos dados e dos métodos de operação sobre estes. Concretizando, trata-se de implementações em *PHP*, que disponibilizam várias funcionalidades de operação sobre a base de dados. O esforço de programação foi aliviado devido à elevada integração entre as tecnologias *PHP* e *MySQL*, e à comunidade muito activa que está dedicada a estas tecnologias. Além das comunicações com a base de dados, o *Model* é também o responsável pelas

interacções directas com o sistema de ficheiros do servidor, onde são guardados os ficheiros de imagem trabalhados.

O *Model* em si é, genericamente, uma camada lógica que assenta sobre o suporte de dados, conferindo-lhes valor semântico no âmbito do projecto, e disponibiliza alguns métodos que os manipulam, mantendo sempre a sua integridade e consistência. Devido às boas práticas de programação seguidas, os componentes *Model* encapsulam a camada de acesso aos dados, ficando os referidos métodos implementados como funções. Deste modo, não é possível fazer requisições directas por *URL* às páginas que implementam o *Model*. A invocação destas funções é feita unicamente pelo componente *Controller* que lhe esteja associado e este sim, está programado para responder a requisições por *URL's*, segundo as normas básicas do protocolo *HTTP*.

O Componente Controller

Este componente recebe o *input* da *View* e processa a resposta invocando, sempre que seja necessário proceder a alterações nos dados, as funções do *Model*. À semelhança deste último, o *Controller* está também implementado em *PHP*, embora as suas funcionalidades sejam acedidas por requisição directa às páginas de implementação. Dado que estas chamadas podem implicar alterações ao modelo de dados mantido pelo servidor, são geralmente executadas pelo método *POST*, prevenindo-se assim que sejam colocadas em *cache* pelo *browser* cliente onde corre a componente *View* que faz as invocações, conforme referido no sub-capítulo 3.2.2, secção *As tecnologias do lado do cliente*.

Como já foi referido, as funcionalidades de manipulação do componente *Model* são disponibilizadas por funções, invocadas pelo *Controller*. Quando estas invocações retornam, o *Controller* irá notificar a *View* para que esta se actualize ou às suas partes desta que tenham sido afectadas e precisem de actualização.

Considerações sobre a arquitectura implementada

O *2D Game Editor On-line* ficou com a arquitectura da sua programação segundo uma arquitectura adaptada do padrão *Model-View-Controller*. Assim, cada componente funcional do editor – os módulos – terá a sua própria implementação de uma tríade de componentes *MVC*. Na Figura 24, estão representados três dos principais módulos funcionais, evidenciando as relações dos componentes de cada tríade, bem como uma ligação entre dois desses componentes distintos.

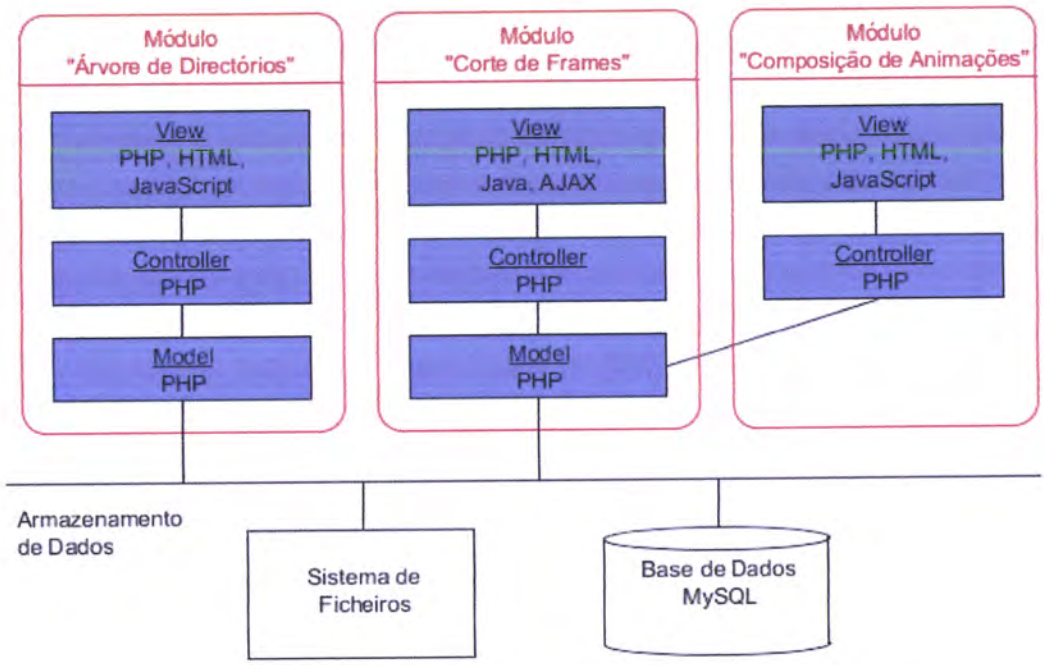


Figura 24. Representação das relações entre três dos módulos funcionais mais importantes do editor, e as componentes MVC que os implementam.

Quando se pretender desenvolver uma nova funcionalidade para o editor, será sempre necessário criar um, pelo menos, novo par *View-Controller*. Se as funcionalidades de modificação dos dados puderem ser implementadas pelos métodos já implementados em algum componente *Model* e isso fizer sentido, o novo *Controller* poderá ser directamente ligado a este (como é o caso do módulo "Composição de Animações" na Figura 24). Caso contrário, será necessário criar todos os componentes do novo módulo.

3.2.5. Algumas funcionalidades

Apresentam-se, neste sub-capítulo, duas das funcionalidades mais representativas do ponto de vista da interactividade, dentre aquelas que estão implementadas no *2D Game Editor On-line*.

Corte de fotogramas

Como já foi referido no sub-capítulo 3.2.2, secção *As tecnologias do lado do cliente*, esta funcionalidade está implementada unicamente em Java. As comunicações assíncronas com o servidor processam-se em paralelo e estão a cargo do *browser*. Na Figura 25, pode-se visualizar a interface geral da ferramenta (*applet*) de corte de fotogramas:

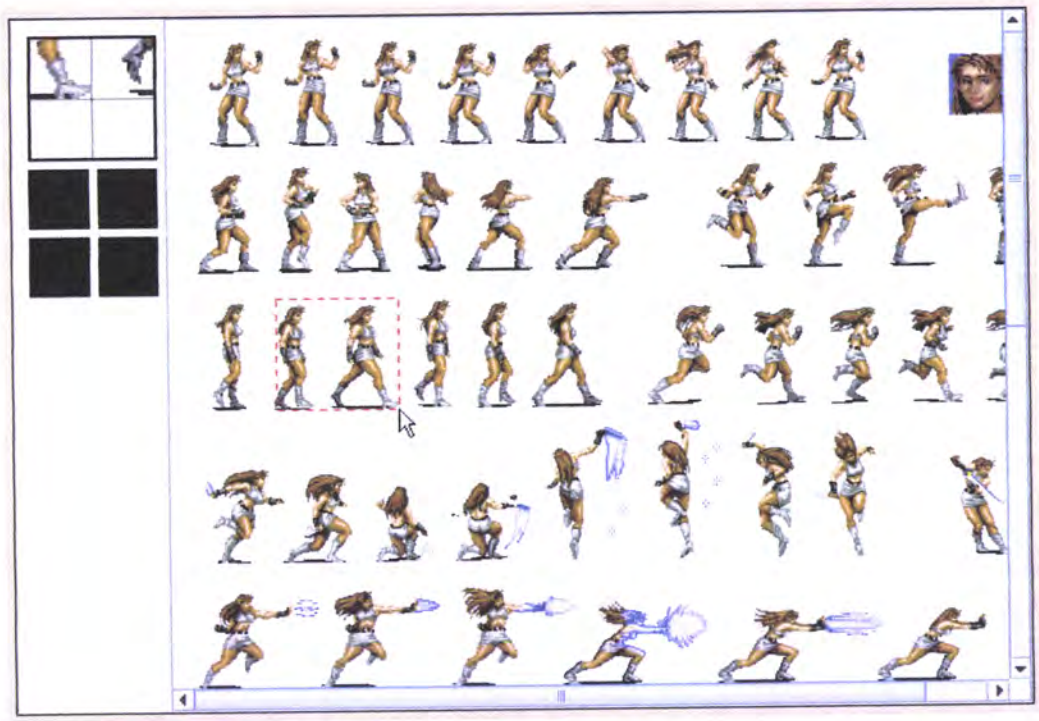


Figura 25. Interface para o corte de fotogramas.

A selecção de um fotograma a cortar é conseguida por arrasto, definindo uma área de corte rectangular que fica delineada com tracejado vermelho, tal como se pode ver em detalhe na Figura 26.

Figura 26. Selecção de uma área de corte.

Depois de concluir a definição da área, o sistema mostrará controlos que a permitirão cancelar, confirmar ou ajustar, conforme necessário (Figura 27).

Figura 27. Alteração da selecção.

Uma vez confirmado o corte, o sistema irá aguardar a resposta do servidor à acção que foi invocada, informando o utilizador desta espera através do realce, a azul, do fotograma (Figura 28).

Figura 28. Corte de fotograma em espera da resposta do servidor.

Depois da resposta, dada quando o servidor confirma o corte, o fotograma passará a ser representado com limites a verde, tal como também pode ser observado na Figura 28. Resta dizer que, para que o utilizador possa realizar um trabalho detalhado, foram incluídas janelas de detalhe (*viewports*), que ampliam a área em redor do cursor, quando este está a ser arrastado para definir a área de corte (Figura 29).

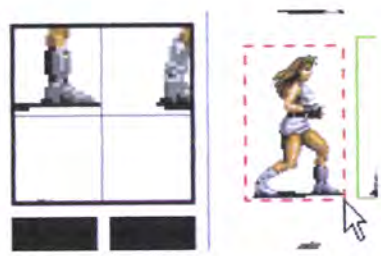


Figura 29. Janela (*viewport*) de detalhe.

Criação de animações

Ao clicar-se sobre o nó de uma animação, na árvore dos projectos (ver sub-capítulo 3.2.3, secção *O sistema de ficheiros*), o *2D Game Editor On-line* irá abrir a ferramenta de edição de animações. Esta ferramenta mostrará a sequência de fotogramas que foi definida para a animação, como se pode ver na Figura 30, e permite manipular todos os seus detalhes.
Esta funcionalidade é processada, unicamente, do lado do cliente (*browser*) e, deste modo, a sua implementação foi conseguida apenas com recurso a *JavaScript* e *HTML*.

Figura 30. Alinhamento dos fotogramas da animação "murro1"

As interacções são conseguidas através dos botões:

- **Nova Frame** – Adiciona um novo fotograma, no fim da sequência;
- **Prever Animação** – Abre a ferramenta de pré-visualização de animações (ver sub-capítulo 3.2.3, secção *A ferramenta de pré-visualização*);
- **Guardar** – Guarda a sequência de fotogramas definida até à data;
- **Cancelar** – Cancela todo o trabalho executado nesta animação, desde a última acção "Guardar".

Para apagar um fotograma, basta clicar no botão com um "X", tal como se pode ver na Figura 30. Para substituir um fotograma, é necessário clicar sobre o fotograma visado, aparecendo a janela com a interface de selecção de fotogramas que contém, prontos a seleccionar, todos os fotogramas cortados e guardados no sistema (ver Figura 31).

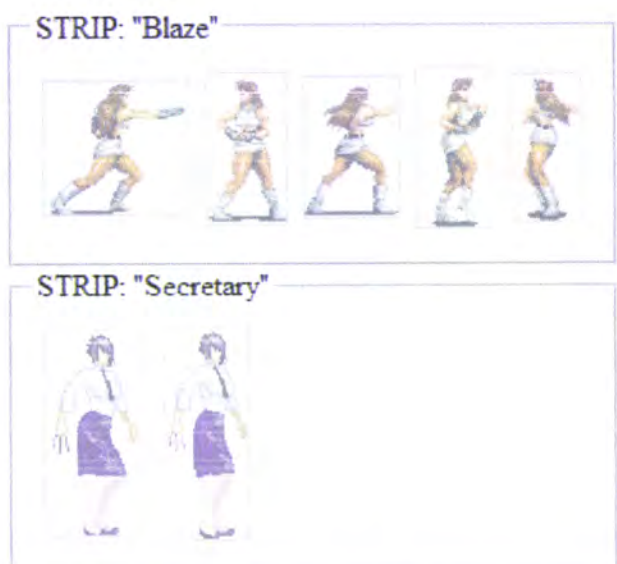


Figura 31. Janela de selecção de fotogramas cortados.

3.3. Algoritmo de composição de atlas de fotogramas

3.3.1. O atlas

Um dos maiores problemas dos motores gráficos actuais (com aplicações que podem ultrapassar os videojogos) consiste na renderização simultânea de milhares de objectos diferentes. O custo individual de cada renderização está potenciado no caso particular dos jogos, onde cada entidade visível que precisa de ser mostrada pode ter vários estados, cada um com a sua forma de renderização própria, e pode transitar livremente para novos estados, exigindo o carregamento de novo conjunto de imagens (Ivanov, 2006).

Para mitigar as elevadas exigências de processamento destas operações, é usual o emprego de atlas de texturas. Estes consistem em imagens relativamente grandes, compostas pelo arranjo de várias texturas e fotogramas, dos quais cada entidade de jogo apenas utiliza uma parte.

O quadro ideal seria a produção dos artistas sair neste formato ordenado mas, dado que pode ser necessário, posteriormente, proceder a algum arranjo na disposição das texturas e fotogramas, isso faria com que fosse obrigatório refazer o trabalho. Isto implica que se tornasse a envolver o artista nos processos de edição ou sempre que se pretendesse trocar fotogramas por outros, de dimensões distintas.

Da perspectiva do motor do jogo, para que este consiga referenciar especificamente um fotograma particular, é necessário que seja realizado um mapeamento prévio que associe um identificador à localização dentro do atlas (coordenadas do canto superior esquerdo, altura e largura). Este mapa de regiões será, então, carregado pelo motor de jogo e utilizado para associar as entidades virtuais do jogo à sua representação visual, que está embebida no atlas.

Dado que os atlas não são mais do que composições de imagens mais pequenas e independentes, será interessante reduzir ao máximo as suas dimensões, optimizando-se o preenchimento do espaço. São, actualmente, conhecidos alguns algoritmos que propõem soluções para este problema classificado como NP-difícil (*NP-hard, non-deterministic polynomial-time hard*), como é o caso do *bin packing* (Coffman, E. et al. 97), existindo várias aplicações disponíveis na *web* para resolver este problema, com desempenhos muito variáveis. Todas as aplicações gratuitas que foram encontradas partem do pressuposto que lhes será dado como parâmetro de entrada os limites máximos, vertical e horizontal, para o atlas final. Pelas observações dos resultados produzidos, os algoritmos utilizados tentam preencher o espaço segundo a lógica dos algoritmos de *bin packing*, dividindo a área em contentores. Consegue-se assim dividir o trabalho de optimização no espaço total, preenchendo primeiro ao máximo um contentor e só depois criando um novo.

Podem aqui fazer-se, desde já, algumas observações a esta abordagem: a imposição de limites ao atlas obriga a que, mesmo no caso óptimo em que não ocorra nenhum desperdício de espaço, a soma das áreas de todos os fotogramas não possa ultrapassar a área total do atlas; ao escolher uma forma para o

rectângulo do atlas, estamos a assumir que essas sejam as dimensões que melhor enquadram a composição dos fotogramas, à semelhança de um *puzzle* que, composto por formas irregulares, uma vez terminado, terá o formato de um rectângulo; por último, para que se obtenha um resultado satisfatório, é necessário, por vezes, seguir uma metodologia de tentativa e erro com a escolha dos limites, obrigando a uma repetição do trabalho por parte do utilizador.

3.3.2. Uma solução em bruto

Após alguma pesquisa na *internet*, encontrou-se um algoritmo que, com algum aperfeiçoamento, teve o potencial de satisfazer os requisitos enunciados no sub-capítulo anterior (Scott, J.). Esse algoritmo, propõe a divisão de uma área original em áreas mais pequenas conforme vai sendo realizada a inserção de fotogramas. No enunciado original, funciona do seguinte modo:

- É necessário criar uma área inicial com tamanho fixo. Esta área corresponde à do atlas, dentro do qual se vai realizar a composição (Figura 32);

Figura 32. Área inicial com dimensões fixas.

- Insere-se um fotograma no canto superior esquerdo do atlas, e divide-se o espaço do atlas, estendendo um dos limites do fotograma inserido (Figura 33);

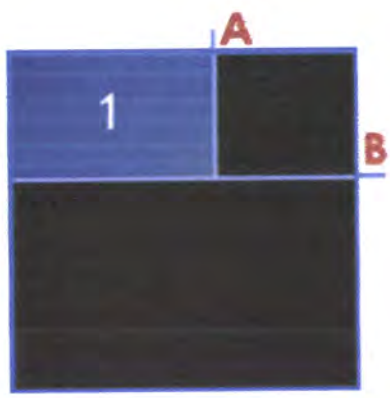


Figura 33. Atlas com fotograma (azul) inserido.

- Ao inserir o próximo fotograma, tenta-se averiguar se cabe acima de B. Se couber, então verifica-se a área à esquerda de A. Como esta área já está preenchida, tenta-se à direita de A. Se não couber, como é o caso do

exemplo, tenta-se a inserção abaixo da linha B, dividindo novamente o espaço, da mesma forma que se fez na primeira inserção. O tipo de divisão, horizontal ou vertical, será decidida de acordo com a divisão que deixar maior extensão de linha por utilizar (Figura 34);

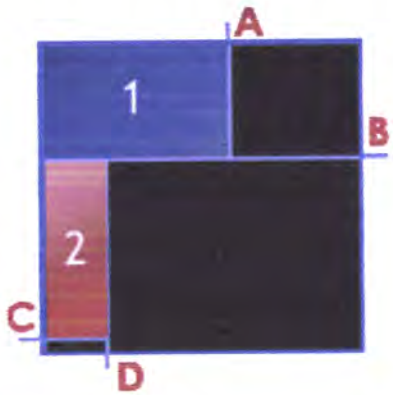


Figura 34. Atlas com o segundo fotograma (encarnado) inserido.

- Repete-se o procedimento, tentando sempre a inserção acima e abaixo das linhas já marcadas, até encontrar uma área onde o fotograma caiba.

O algoritmo não opera directamente sobre as imagens, mas sim sobre a informação estruturada segundo o esquema de uma árvore binária, que é percorrida recursivamente em busca de uma área passível de inserção, e onde se representam as rectas e áreas já preenchidas como nós e folhas da árvore.

3.3.3. Crítica ao algoritmo

O algoritmo apresentado manifesta um problema quando os fotogramas inseridos inicialmente são de dimensões mais pequenas que aqueles que lhes sucedam: ao estender os limites dos fotogramas inseridos até aos limites externos do atlas estão-se a criar "contentores" – à semelhança dos algoritmos de *bin packing* – que inutilizam faixas inteiras do atlas para fotogramas que lá não caibam.

Por outro lado, este algoritmo também requer a definição prévia das dimensões do atlas, o que é, justamente, o maior ponto fraco manifestado pelas aplicações de composição de atlas encontradas, dado que obriga o utilizador a aproximações por tentativa-erro.

A estruturação dos dados na forma de uma árvore binária modela bastante bem o problema que se procura resolver, bem como a ideia de, ao inserir um novo fotograma no atlas, se realizar a divisão do espaço livre restante.

3.3.4. Requisitos e decisões para o novo algoritmo

Pelas razões apresentadas nos sub-capítulos anteriores, é preciso implementar um algoritmo de composição de atlas que seja rápido, dê resultados satisfatórios e não obrigue, de antemão, o utilizador a fazer escolhas que poderão não ser as melhores.

Para atenuar o problema das consequências de inserções iniciais de fotogramas de pequenas dimensões que definem contentores demasiado pequenos para futuras inserções, procede-se a uma ordenação prévia, decrescente, do *array* de fotogramas, comparando as diferenças das larguras dos fotogramas com a diferença entre as suas alturas. Aquele fotograma que tiver maior diferença entre as larguras ou as alturas, será considerado maior que o outro e virá ordenado em primeiro lugar. Deste modo consegue-se, muito convenientemente, que os contentores criados a partir da inserção dos fotogramas iniciais o sejam do maior para o mais pequeno, tornando-se assim possível dividir os contentores maiores criando, dentro do seu espaço, novos contentores de menores dimensões. De outro modo, a seguir as regras do algoritmo original, seria necessário criar novos contentores sempre que o novo fotograma não coubesse nos limites daqueles já criados, o que levaria à criação de áreas desocupadas dentro da composição.

Foi abolido o conceito restringente de dimensão do atlas estabelecido a-priori, substituindo-se pelo conceito de infinito (concretizado na forma de valores muito grandes). No entanto, este novo conceito não nos dá nenhuma indicação sobre o espaço que pretendemos ou podemos ocupar e, assim, para que o crescimento do atlas se dê de forma regrada e sem perder de vista o objectivo da optimização do seu espaço útil, foi necessário definir uma estratégia que orientasse o crescimento da composição (ver sub-capítulo 3.3.6). Dado o desconhecimento sobre a população de fotogramas que o algoritmo de composição terá de arranjar, também não se fizeram quaisquer pressupostos a esse nível, não se tendo imposto quaisquer limitações em termos do número de fotogramas permitido, tamanhos ou áreas destes.

Visto que o atlas é uma imagem que se pretende o mais condensada possível, ou seja, com a maior percentagem de área ocupada por fotogramas, é de evitar ao máximo aumentar o seu tamanho. Para o efeito, definiu-se o conceito de "fronteira", enquanto limite da composição actual, correspondendo à área rectangular definida entre o canto superior esquerdo do atlas (a origem) e o ponto de abcissa e ordenada iguais às maiores abcissas e ordenadas encontradas, entre todos os píxeis ocupados por fotogramas. Embora possa parecer confuso, este conceito corresponde ao menor rectângulo possível que enquadre todos os fotogramas arranjados até ao momento, conforme se pode ver a tracejado amarelo na Figura 35.

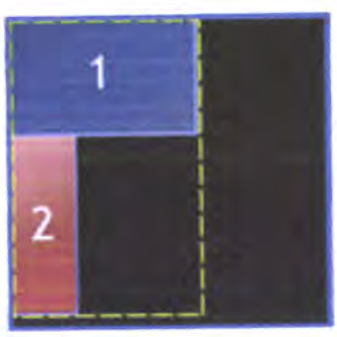


Figura 35. Exemplo do conceito de fronteira (tracejado amarelo).

Então, durante o processo de inserção de um novo fotograma, procurar-se-á primeiramente por uma área livre que esteja contida dentro fronteira actual e, caso isto não seja possível e o fotograma ultrapasse este limite depois da inserção, a fronteira será ampliada para o conter. No entanto, para se conseguir minimizar a área do atlas, é necessário que esse crescimento iterativo não se efectue de forma desregrada.

Por outras palavras, é necessário encontrar uma estratégia de posicionamento dos fotogramas no arranjo actual, de forma a reduzir o recurso aos redimensionamentos da fronteira. É, então, sobre a definição desta estratégia que versam os próximos sub-capítulos.

3.3.5. Ferramentas de avaliação das estratégias

O desenho de um algoritmo que dê uma resposta satisfatória a um problema desta complexidade, sem partir de assunções sobre o número total de fotogramas a inserir, as suas medidas ou as dimensões máximas para o arranjo final, deve considerar uma aproximação heurística. Por outras palavras, a aposta tem de ser no sentido de obter uma solução aceitável em tempo útil e não na procura da solução óptima, isto é, encontrar uma solução optimizada em detrimento da solução óptima.

A estratégia heurística pode ser concretizada de inúmeras formas, no entanto, optou-se por um método *greedy* em que se assume, a cada nova iteração do problema, que a solução descoberta na iteração anterior tenha sido a melhor. Ou seja, procura-se afinar os resultados obtidos na iteração anterior, numa perspectiva construtiva e sem refazer passos anteriores.

Visto que o algoritmo procura inserir o novo fotograma, numa primeira fase e se possível, dentro dos limites da fronteira actual seguindo uma filosofia *best-fit* [1], só é necessário recorrer à estratégia heurística quando for necessário alargar as dimensões do atlas para albergar o novo fotograma (que não coube, repete-se, nas áreas vazias dentro da fronteira). As possibilidades para estas estratégias de crescimento são inúmeras e, para testar o desempenho das várias que foram desenvolvidas, optou-se por analisar os resultados que cada uma obteve quando aplicada a 14 populações distintas de fotogramas, criadas apenas para o efeito (*benchmarking*).

Visto que o problema do algoritmo se resume ao arranjo bidimensional, com melhor aproveitamento possível do espaço, de fotogramas rectangulares dentro de um atlas também rectangular, foi necessário encontrar uma medida que permitisse a classificação e comparação intuitivas entre rectângulos (leia-se, fotogramas). Esta questão seria resolvida de forma simples adoptando-se uma medida escalar, derivada das dimensões dos fotogramas. Uma possibilidade seria a sua área, embora não seja possível assumir nada acerca dos lados do fotograma por mera observação deste valor. Outra possibilidade, seria o comprimento da diagonal do fotograma, traçada desde o seu canto superior esquerdo até ao canto inferior

[1] O fotograma será inserido dentro da área vazia onde for deixada a menor quantidade desperdiçada de espaço após a inserção.

direito. No entanto, também não se pode assumir nada acerca das dimensões laterais a partir desta grandeza. Por outro lado, se for calculada a razão R, entre os comprimentos do lado mais pequeno e do maior, segundo a fórmula $R = \frac{lado\,menor}{lado\,maior}$, teremos que $R \in]0,1]$, onde $R = 1$ corresponde a um quadrado perfeito (lado menor é igual ao lado maior) e $R \to 0$ conforme vai aumentando a diferença entre o lado maior e o menor. Com este valor já se podem inferir relações entre os comprimentos dos lados dos fotogramas e permite retirar algumas ilações acerca das populações de teste.

Assim, na tabela que se segue, estão sumariados alguns detalhes das 14 populações utilizadas, remetendo-se uma caracterização estatística mais detalhada para o Anexo I.

População	Número de fotogramas	Media de R	Desvio-padrão de R
1	51	0,49864415	0,280696112
2	51	0,69942994	0,183974937
3	51	0,87199327	0,076557271
4	101	0,52371177	0,285162321
5	101	0,87236314	0,084043067
6	251	0,54222844	0,270403891
7	251	0,88174082	0,084188645
8	501	0,52160373	0,262789425
9	501	0,87389818	0,083867754
10	1001	0,53503785	0,270969825
11	1001	0,87420186	0,087090548
12	251	1	0
13	501	0,55054722	0,129648926
14	501	0,69917029	0,144558287

Tabela 1. Caracterização da média da razão R das 14 populações de fotogramas utilizadas para teste.

Destaca-se o caso particular da população 12 que é, como se poderá deduzir pela observação dos valores da média e do desvio-padrão de R, constituída apenas por quadrados. Esta é também a população com maior homogeneidade e a população 4 é a que apresenta maior diversidade entre os fotogramas. Note-se ainda a população 13, que é constituída apenas por rectângulos de altura fixa e largura variável. Note-se também que, no caso das populações 12 e 13, é possível compor um atlas perfeito, com 100% de aproveitamento de área, bastando alinhar todos os fotogramas lado a lado.

No sub-capítulo seguinte descrevem-se algumas das estratégias que foram testadas com as referidas 14 populações de fotogramas.

3.3.6. Aproximações heurísticas

Encontram-se, nos próximos sub-capítulos, descrições sucintas das estratégias consideradas para resolver o problema. Em todas elas, antes de se ponderar inserir o novo fotograma numa área que cruze os limites da fronteira e obrigar ao redimensionamento do atlas, procura-se por uma área vazia que esteja contida nesta e tenha tamanho adequado. Se for encontrada mais do que uma área nestas condições, a escolha é feita por um princípio *best-fit*, dando-se primazia à área que tiver as dimensões mais próximas das do fotograma a inserir. O melhor caso possível para a inserção de um fotograma, é encontrar uma área com as mesmas dimensões aproveitando-se assim o espaço ao máximo. Caso isto não seja possível, a área escolhida será aquela que, estando totalmente contida na fronteira actual, causará o menor desperdício possível. Esta abordagem tem, naturalmente, limitações, no entanto mostrou resultados melhores nos testes realizados que os procedimentos *first-fit* [1] ou *worst-fit* [2].

Incluíram-se algumas imagens ilustrativas, com a seguinte representação: a fronteira actual do atlas está desenhada numa linha contínua azul; os fotogramas estão representados em tons de cinza; o fotograma que se pretende inserir está a amarelo; a fronteira após a inserção está marcada a tracejado azul e realça-se a vermelho a medida que condicionou a colocação do novo fotograma naquela posição. Note-se ainda que o estado de partida do atlas não foi obtido mediante a aplicação do algoritmo com a estratégia em estudo. Deve ser considerado apenas como um estado inicial e sem significado, igual para todas as estratégias, meramente com propósitos ilustrativos e a partir do qual será necessário aplicar a estratégia para decidir onde colocar o fotograma.

Atlas quadrado ($R_{atlas} \approx 1$)

Tenta-se, em cada iteração, manter no atlas a razão R (ver sub-capítulo anterior) o mais próxima possível de 1. Por outras palavras, a escolha do ponto de inserção dependerá de qual deles transforma o atlas, após o redimensionamento, numa figura o mais quadrática possível (Figura 36).

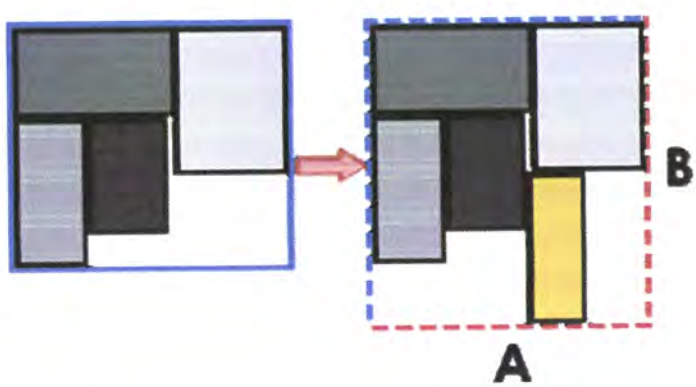


Figura 36. Inserção de um fotograma (amarelo), procurando manter a razão R do atlas próxima de 1.

[1] A área seleccionada será a primeira onde o fotograma couber.
[2] A área seleccionada será aquela que deixar mais espaço vazio após a inserção do fotograma.

No caso ilustrado, o ponto de inserção escolhido foi aquele cuja razão R, calculada por $\frac{A}{B}$, era a mais próxima de 1.

Apesar de parecer uma aproximação algo desajustada da realidade das populações de fotogramas a tratar, provou ser a segunda melhor estratégia para as populações em estudo, com uma média de preenchimento da área dos atlas, na ordem dos 90,18%.

Naturalmente, um dos casos particulares em que esta abordagem apresenta resultados óptimos, com aproveitamento de área de 100%, é aquele em que os fotogramas são quadrados iguais, e em quantidade igual ao da raiz da área do atlas. O atlas ficaria conforme a representação abaixo:

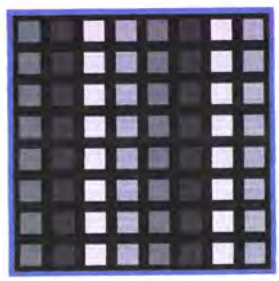


Figura 37. Atlas quadrado, composto por fotogramas quadrados e iguais.

Melhor desempenho (comparado):
população 1 - 84,89% de ocupação do atlas;
população 8 - 95,02% de ocupação do atlas.
Pior desempenho (comparado): nunca.
Média de preenchimento: 90,18%.

Atlas na proporção de um rectângulo de ouro ($R_{atlas} \approx 0{,}618$)

Considerando os resultados razoáveis obtidos com o atlas próximo de um quadrado, e que os próprios fotogramas tendem, eles próprios, a ser rectangulares, faz sentido tentar-se uma abordagem em que o atlas mantenha proporções de rectangulares. No entanto, as possibilidades para as dimensões num atlas desta natureza são inúmeras.

Foi, então, necessário realizar um estudo sobre a forma rectangular, donde sobressaiu um tipo particular de rectângulo: o "rectângulo de ouro". Dá-se esta designação a um paralelogramo cujas proporções dos lados correspondam à razão $\frac{1}{\varphi} \approx 0{,}618$, onde $\varphi = \frac{1+\sqrt{5}}{2} \approx 1{,}618(...)$. Este número irracional φ, o número de ouro, tem presença recorrente em muitas proporções verificadas em elementos naturais, arquitectónicos, matemáticos e, como foi recentemente descoberto, até ao nível da física quântica. Desde a distribuição das folhas em redor do caule de algumas plantas como forma de maximização da área de exposição ao Sol, ao valor para o qual tende a razão entre dois números sucessivos da sucessão de Fibonacci quando o número de iterações tende para infinito, o número de ouro surge como constante, regendo as proporções dos elementos constituintes com o todo.

Com aplicações tão transversais, esta é uma proporção que vale a pena tentar, atestando o sucesso do atlas quadrado e visto que é necessária uma forma rectangular para testar.

Assim, tenta-se a cada iteração do algoritmo, que a razão das dimensões do atlas esteja o mais próximo possível de 0,618 (Figura 38).

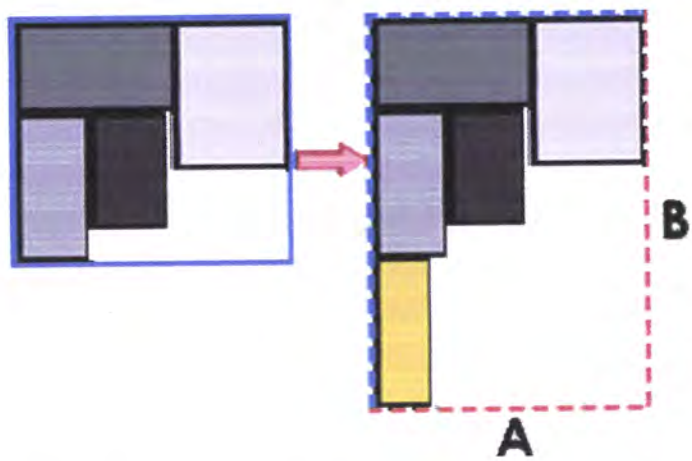


Figura 38. Inserção de um fotograma (amarelo), procurando manter a razão R do atlas próxima de 0,618.

Na ilustração, o ponto escolhido para inserir o novo fotograma foi aquele cuja razão R do atlas, calculada como $\frac{A}{B}$, esteja mais próxima de 0,618.

Melhor desempenho (comparado):
população 9 - 90,85% de ocupação do atlas.
Pior desempenho (comparado):
população 2 - 73,36% de ocupação do atlas;
população 12 - 96,54% de ocupação do atlas.
Média de preenchimento: 89%.

Distância à origem

Nesta estratégia, o ponto de inserção escolhido será aquele que tiver a menor distância até à origem (entenda-se aqui a origem como o canto superior esquerdo do atlas). É de notar que, caso não houvesse a pré-condição de se escolher primeiro uma área dentro da fronteira actual e só depois optar por redimensionar o atlas, a área ocupada registaria um crescimento tendencialmente circular (sempre a menor distância até à origem, conforme a Figura 39).

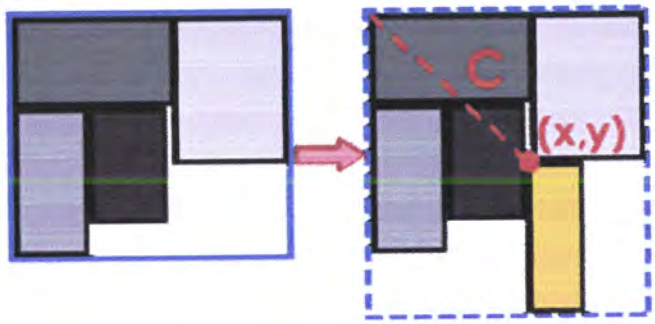


Figura 39. Inserção de um fotograma (amarelo), no ponto livre que tem a menor distância C até à origem.

Na imagem acima, o ponto escolhido foi aquele que teve o menor comprimento C, onde $C = \sqrt{x^2 + y^2}$, é correspondente à distância euclidiana desde a origem até ao canto superior esquerdo do fotograma inserido.

Melhor desempenho (comparado):
população 4 - 89,06% de ocupação do atlas.
Pior desempenho (comparado): nunca.
Média de preenchimento: 88,08%.

Maior crescimento unidimensional

O ponto escolhido para inserção será aquele que causar, após a colocação do novo fotograma, o maior crescimento horizontal ou vertical do atlas. No caso de crescer nos dois sentidos, vale aquele que tiver registado a maior diferença. Ao optar-se por um crescimento grande numa só iteração está-se, em probabilidade, a abrir espaço no atlas para que os próximos fotogramas possam ser inseridos sem necessitar de mais redimensionamentos (Figura 40).

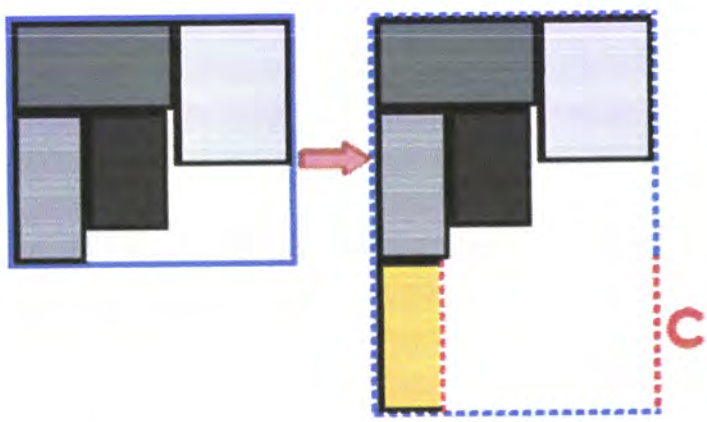


Figura 40. Inserção de um fotograma (amarelo), na área que causou a maior alteração nas dimensões (entre lado e largura) do atlas.

Na ilustração, o comprimento C, correspondente à altura do novo fotograma, é o maior crescimento vertical ou horizontal que o atlas pode registar no processo de redimensionamento decorrente da inserção.

Esta foi a estratégia com piores resultados, tendo-o demonstrado em 6 populações. Faz-se aqui a devida ressalva no sentido em que o estudo efectuado incidiu sobre apenas 14 populações pelo que não pode ser, por esta razão, considerado absolutamente conclusivo.

Melhor desempenho (comparado):
população 12 - 100% de ocupação;
população 13 - 100% de ocupação.
Pior desempenho (comparado):
população 1 - 53,04% de área ocupada;
população 5 - 80,36% de área ocupada;
população 8 - 93,32% de área ocupada;
população 9 - 86,96% de área ocupada;
população 10 - 94,72% de área ocupada;
população 14 - 41,14% de área ocupada.
Média de preenchimento: 84,01%.

Menor crescimento unidimensional

Esta estratégia é a oposta da anterior, dado que o ponto de inserção escolhido será aquele que gerar o menor redimensionamento horizontal ou vertical. Caso ocorra crescimento nos dois sentidos, vale aquele que tiver registado a maior diferença (isto pode parecer contraditório, mas visto que pretendemos encontrar o ponto de inserção que implica o menor redimensionamento, precisamos de comparar os diferentes pontos pelo maior crescimento registado). Naturalmente, ao optar por crescimentos mais pequenos em cada iteração (Figura 41), será necessário recorrer mais vezes ao redimensionamento do atlas.

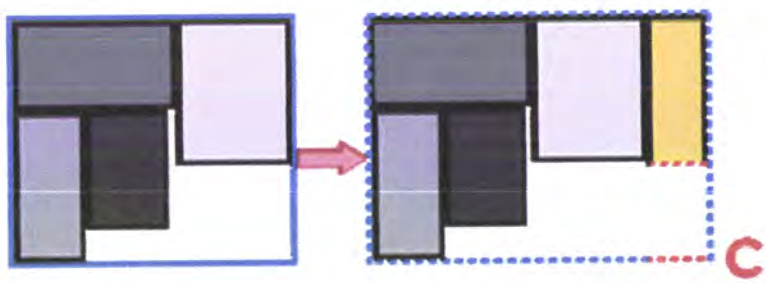


Figura 41. Inserção de um fotograma (amarelo), na área que causou a menor alteração nas dimensões (entre lado e largura) do atlas.

No exemplo acima, o comprimento C, igual à largura do fotograma inserido, corresponde ao menor crescimento, horizontal ou vertical, que o atlas regista depois da sua inserção.

Melhor desempenho (comparado):
população 6 - 94,05% de ocupação;
população 11 - 91,85% de ocupação;
população 12 - 100% de ocupação.
Pior desempenho (comparado):
população 7 - 75,64% de área ocupada;
população 13 - 91,11% de área ocupada.
Média de preenchimento: 88,14%.

Maior aumento da área

Esta estratégia opta pelo ponto de inserção cujo redimensionamento consequente provoque maior aumento na área do atlas. Procura-se, assim, minimizar a necessidade de redimensionamentos do atlas, apostando em garantir, em cada recurso ao crescimento, que seja disponibilizada área suficiente para permitir as inserções dos próximos fotogramas, conforme a Figura 42.

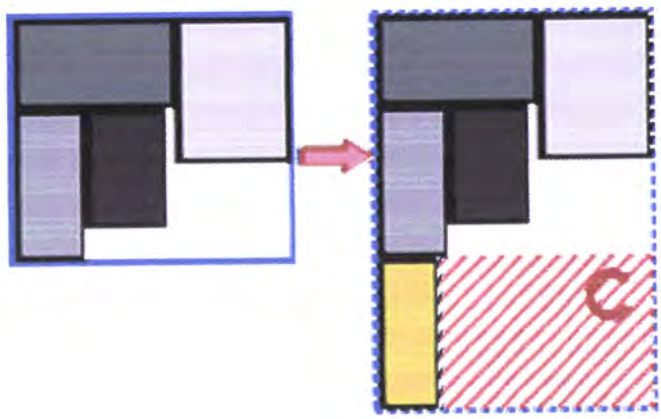


Figura 42. Inserção de um fotograma (amarelo), na área que causou o maior aumento na área do atlas.

Na imagem acima, a área C, corresponde à maior área que é possível obter no redimensionamento que se segue à inserção do novo fotograma.

Melhor desempenho (comparado):
população 10 - 96,58% de ocupação.
Pior desempenho (comparado):
população 3 - 77,36% de área ocupada;
população 4 - 81,25% de área ocupada.
Média de preenchimento: 87,53%.

Menor aumento da área

Nesta estratégia, ao contrário da anterior, procura-se inserir o novo fotograma no ponto de inserção que implique a menor variação na área do atlas. Assim, procura-se alcançar em cada iteração, um atlas com tamanho razoavelmente ajustado ao conteúdo e um mínimo de área desperdiçada.

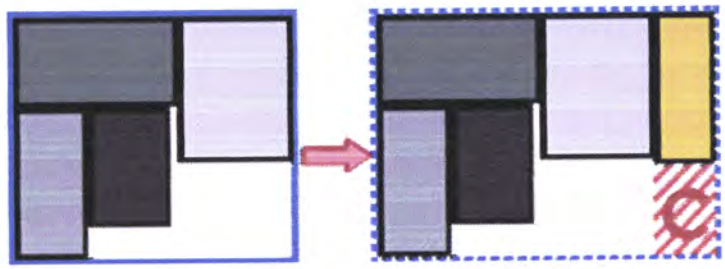


Figura 43. Inserção de um fotograma (amarelo), na área que causou o menor aumento na área do atlas.

Na Figura 43, o fotograma será inserido na posição que causar o menor aumento da área C do atlas.

Melhor desempenho (comparado):
população 3 - 86,07% de ocupação;
população 12 - 100% de ocupação;
população 13 - 100% de ocupação.
Pior desempenho (comparado):
população 11 - 83,18% de área ocupada.
Média de preenchimento: 88,1%.

Atlas com razão próxima da razão média dos fotogramas $(R_{atlas} \approx \overline{R}_{fotogramas})$

Nesta última estratégia é necessário efectuar, à-priori, o cálculo das razões de todos os fotogramas que serão utilizados para compor o atlas. Depois, utilizar-se-á esta constante como objectivo para o qual deve tender R_{atlas}, na altura em que este tenha de ser redimensionado. Ou seja, ao redimensionar-se o atlas, o ponto de inserção para o novo fotograma será aquele que mantiver R_{atlas} o mais próximo possível de $\overline{R}_{fotogramas}$, a razão média dos fotogramas.

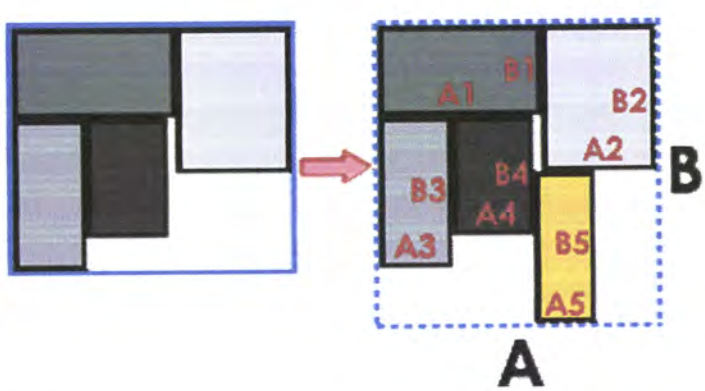


Figura 44. Inserção de um fotograma (amarelo), no ponto que irá manter R_{atlas} o mais próximo possível de $\overline{R}_{fotogramas}$.

No caso ilustrado na Figura 44, escolheu-se aquele ponto de inserção pois é aquele que coloca $R_{atlas} = \frac{A}{B}$, o mais próximo possível de $\overline{R}_{fotogramas} = \frac{R_1 + R_2 + R_3 + R_4 + R_5}{5}$, onde $R_1 = \frac{B1}{A1}$, $R_2 = \frac{A2}{B2}$, ..., $R_n = \frac{Lado\ Menor_{fotograma\ n}}{Lado\ Maior_{fotograma\ n}}$.

Esta foi a estratégia com melhor desempenho, apresentando os melhores resultados em 5 dos casos testados.

Melhor desempenho (comparado):
população 1 - 84,89% de ocupação;
população 2 - 86,96% de ocupação;
população 5 - 88,87% de ocupação;
população 7 - 89,43% de ocupação;
população 14 - 91,18% de ocupação.
Pior desempenho (comparado):
população 7 - 89,43% de ocupação.
Média de preenchimento: 90,43%.

3.3.7. Análise dos resultados

De acordo com os resultados obtidos, refere-se que a estratégia do algoritmo que teve o melhor desempenho foi a da razão média.
Segue-se a tabela com a síntese destas informações:

Aplicação	Algoritmo	Nº de vezes melhor	Nº de vezes pior	Média das percentagens de ocupação
2DGEO	Maior diferença linear	2	6	84,01%
2DGEO	Menor diferença linear	3	2	88,14%
2DGEO	Maior area	1	2	87,53%
2DGEO	Menor area	2	1	88,1%
2DGEO	Quadrado	2	0	90,18%
2DGEO	Rectângulo de ouro	1	2	89%
2DGEO	Relação Média	5	1	90,43%
2DGEO	Distancia Origem	1	0	88,08%
Slick		3	7	82,02%

Tabela 2. Resultados da aplicação das estratégias às 14 populações de teste.

Neste quadro dá-se também a ideia do desempenho do algoritmo da aplicação *Packer,* da biblioteca *2D Game Library* ver sub-capítulo 2.1.2. No entanto, a inclusão deste algoritmo nestas comparações não é totalmente conveniente, na medida em que esta última aplicação assume que lhe sejam, como já foi referido, indicadas *a-priori* as dimensões pretendidas para o atlas final e o algoritmo do *2D Game Editor On-line* não apresenta esta restrição. Deste modo, para que os resultados pudessem ser alvo de uma comparação justa, havia duas hipóteses: ou se ia indicando ao *Packer*, iterativamente, as dimensões até achar uma que fosse adequada, o que obrigaria a uma metodologia teste-erro num domínio de hipóteses potencialmente infinito, ou se dava a dimensão máxima a este algoritmo, deixando-o optimizar a distribuição dentro desta área. Naturalmente que, dada a dificuldade prática da primeira abordagem, o método escolhido foi o segundo. Ademais, a possibilidade de libertar o utilizador da responsabilidade de procurar qual a área do atlas mais adequada foi, precisamente, um dos principais factores que motivou o desenho de um novo algoritmo.

Como se pode verificar no quadro de resultados acima exarado, a melhor estratégia é aquela que procura manter a relação entre os lados do atlas o mais próximo possível da média das relações dos lados dos fotogramas. Isto implica um consumo de tempo acrescido, na medida em que é necessário percorrer o conjunto dos fotogramas um a um, para se fazer o levantamento dessa média. No entanto, como neste caso particular é mais importante optimizarem-se os resultados do que os tempos de execução (enquanto estes se mantenham aceitáveis), esta é uma troca vantajosa.

Para mais detalhes sobre o desempenho dos algoritmos nas várias populações, consultar o Anexo I.

3.3.8. Considerações sobre a implementação

Na implementação do algoritmo de composição dos atlas fez-se uso exclusivo do *PHP*, em particular do suporte que esta linguagem oferece ao paradigma da programação orientada a objectos.

Em termos práticos, a árvore que foi desenvolvida, e sobre a qual assenta a mecânica da informação, é binária e composta por instâncias da classe Nó, cada uma com referências para dois outros nós - os filhos. Sempre que é necessário proceder à inserção de um novo fotograma no atlas, esta estruturação nos dados permite que essa árvore possa ser percorrida nó a nó, recursivamente, em busca do melhor local/nó para essa operação.

Para a filosofia *proof-of-concept* que regeu o desenvolvimento deste algoritmo, e dado que ainda não está perfeitamente definido o contexto ou as especificações das máquinas onde correrá, não foram envidados esforços no sentido de o optimizar em termos temporais ou espaciais (memória ocupada), inferindo-se o grau de eficácia

relativa pela comparação dos resultados[1]. No entanto, caso se prove necessário, existem algumas propostas já pensadas e avançadas no Capítulo 4.

3.3.9. Outras perspectivas

Este algoritmo poderá ser adaptado para se enquadrar em qualquer contexto onde fosse útil produzir o arranjo de um conjunto de áreas pequenas numa composição maior, cuja área final, sendo desconhecida à partida, se pretende o menor possível.

Um exemplo seria, na área das publicações web, a utilização do algoritmo para compor um arranjo das imagens utilizadas num site ou em parte deste. Desse modo, essas imagens passariam a estar todas agrupadas num atlas que teria, contudo, um tamanho menor do que a soma dos tamanhos de todas as imagens individuais. Assim, quando um browser fizesse a requisição de uma página ao servidor, seria necessário apenas descarregar o atlas, já que este conteria o arranjo de todas as imagens necessárias para a renderização dessa página. A selecção, para visualização de cada imagem individual, poderia ser conseguida por técnicas de *CSS* e/ou *JavaScript*.

[1] Esta comparação entre os resultados das estratégias foi realizada comparando, para as mesmas populações, as percentagens de ocupação dos arranjos conseguidos por cada uma.

3.4. A Base de Dados

Como já foi anteriormente referido, a persistência da informação é assegurada pela utilização do sistema de gestão de bases de dados *MySQL*. Foi, então, desenhada uma base de dados assente sobre o modelo relacional, que permite consultas e manipulações de forma rápida e eficaz. Foi, assim, criado o seguinte modelo:

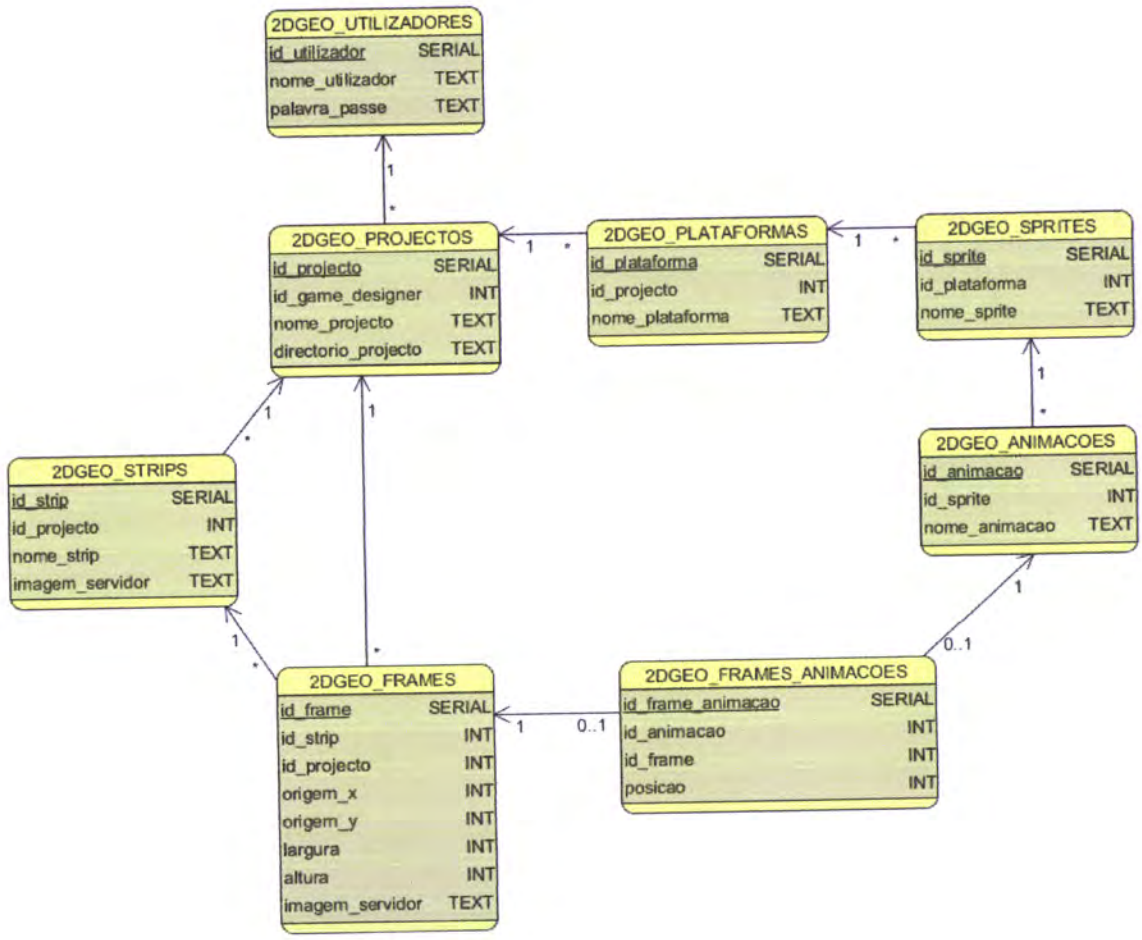


Figura 45. Esquema da base de dados.

O esquema apresentado na Figura 45 foi inspirado no diagrama de classes do *UML*, e serve apenas como recurso ilustrativo do esquema da base de dados e das relações que se estabelecem entre as tabelas.

3.4.1. Listagem de tabelas

Segue-se uma breve listagem das tabelas utilizadas, e dos campos que nelas figuram.

2dgeo_utilizadores

Esta tabela contém os registos dos utilizadores no sistema.

- ***id_utilizador***: o identificador do utilizador; chave primária da tabela.
- ***nome_utilizador***: o nome utilizado no formulário de *login*.
- ***palavra_passe***: a palavra-passe do utilizador, transformada por *md5*.

2dgeo_projectos
Contém o registo dos projectos guardados no sistema.

- ***id_projecto***: o identificador do projecto; chave primária desta tabela.
- ***id_game_designer***: o identificador do utilizador que criou o projecto; actualmente, apenas o *Game Designer* pode aceder ao projecto [1]; chave estrangeira da tabela *2dgeo_utilizadores*.
- ***nome_projecto***: o nome dado ao projecto.
- ***directorio_projecto***: o directório que, no servidor, será utilizado para guardar as imagens trabalhadas.

2dgeo_plataformas
Registos das plataformas associadas a determinado projecto.

- ***id_plataforma***: o identificador da plataforma; chave primária desta tabela.
- ***id_projecto***: o identificador do projecto a que pertence esta plataforma; chave estrangeira da tabela *2dgeo_projectos*.
- ***nome_plataforma***: o nome dado à plataforma.

2dgeo_sprites
Os registos das sprites criadas.

- ***id_sprite***: o identificador da *sprite*; chave primária desta tabela.
- ***id_plataforma***: o identificador da plataforma; chave estrangeira da tabela *2dgeo_plataformas*.
- ***nome_sprite***: o nome da *sprite*.

2dgeo_animacoes
Os registos das animações das *sprites*.

- ***id_animacao***: o identificador da animação; chave primária desta tabela.
- ***id_sprite***: o identificador da *sprite* a que esta animação pertence; chave estrangeira da tabela *2dgeo_sprites*.
- ***nome_animacao***: o nome da animação.

[1] Ver sub-capítulo 0.

2dgeo_strips
Os registos das *strips* carregadas no sistema.

- ***id_strip***: o identificador da *strip*; chave primária da tabela.
- ***id_projecto***: o identificador do projecto a que pertence esta *strip*; chave estrangeira da tabela *2dgeo_projectos*.
- ***nome_strip***: o nome da *strip*.
- ***imagem_servidor***: o nome do ficheiro carregado no servidor, a partir do qual serão cortados os fotogramas.

2dgeo_frames
Esta tabela regista os fotogramas cortados.

- ***id_frame***: o identificador do fotograma; chave primária desta tabela.
- ***id_strip***: o identificador da *strip* de onde o fotograma foi cortado; chave estrangeira da tabela *2dgeo_strips*.
- ***id_projecto***: o identificador do projecto a que pertence este fotograma; chave estrangeira da tabela *2dgeo_projectos*.
- ***origem_x***: a abcissa do canto superior esquerdo deste fotograma, em relação à origem definida no canto superior esquerdo da *strip*.
- ***origem_y***: a ordenada do canto superior esquerdo deste fotograma, em relação à origem definida no canto superior esquerdo da *strip*.
- ***largura***: a largura do rectângulo cortado.
- ***altura***: a altura do rectângulo cortado.
- ***imagem_servidor***: o nome do ficheiro de imagem correspondente ao corte deste fotograma.

2dgeo_frames_animacoes
Esta é uma tabela de relações, ligando uma frame cortada a uma animação.

- ***id_frame_animacao***: identificador desta ligação; chave primária desta tabela.
- ***id_frame***: o identificador do fotograma associado à animação.
- ***id_animacao***: o identificador da animação associada ao fotograma.
- ***posicao***: a ordem onde se encontra este fotograma, dentro da sequência de fotogramas da animação.

3.4.2. Considerações finais sobre o suporte de dados

Foram feitos alguns esforços no sentido de aprimorar o desenho da base de dados. Assim, esta está em conformidade com a 3ª forma normal e, consequentemente, também com a 1ª e 2ª. Deste modo, previnem-se algumas das vulnerabilidades que podem advir de futuras manipulações dos dados (actualizações, inserções e remoções), bem como se torna a organização dos dados mais legível para os utilizadores ou programadores que lhe acedam directamente.

Foram também implementados alguns mecanismos que reforçam a consistência e a integridade dos dados ao nível do próprio *SGBD,* como as restrições de chave estrangeira, realizando a sua declaração explícita e a utilização de restrições de integridade como *CASCADES* em acções de remoção de entradas.

No entanto, a base de dados não está desenhada para poder ser utilizada com outras interfaces que não o *2D Game Editor On-line*, havendo algumas acções de manipulação do modelo de dados que são efectuadas a nível programático, pelo componente *Model* correspondente, que encapsula a informação a ser alterada. A título de exemplo, pode-se referir o processo de eliminação de *strips*: é necessário que, após toda a informação relativa à *strip* em questão ter sido apagada, que o sistema também proceda à eliminação do próprio ficheiro de imagem correspondente. Garante-se que, desse modo, não ficam ficheiros de imagem "perdidos" no servidor (sem referencias na base de dados) e a consumir espaço em disco.

Fez-se, como seria de esperar num sistema desta natureza, bastante uso das capacidades transaccionais implementadas no *SGBD MySQL*. Usando o mesmo exemplo, quando se apaga uma *strip*, o sistema começa por dar início a uma transacção. Depois são removidas as referências directas àquela *strip* (e algumas indirectas, como os fotogramas cortados a partir dela) e, uma vez concluído o processo de actualização da base de dados, tenta-se remover o ficheiro de imagem do disco. Dependendo do resultado desta última operação e o sistema consiga ou não apagar o ficheiro, far-se-á o *COMMIT* ou o *ROLLBACK* da transacção, respectivamente.

3.5. Sessões de teste

A implementação do *2D Game Editor On-line* foi alvo de duas sessões de teste: a primeira correu numa fase inicial do desenvolvimento, quando ainda só estavam implementadas as funcionalidades de corte de fotogramas e a representação dos projectos numa estrutura do tipo árvore; e segunda sessão deu-se quando a implementação estava já concluída e nos termos em que se encontra descrita no presente trabalho.

Foram sempre os mesmos utilizadores que realizaram os testes embora, na primeira sessão, tenham estado envolvidas cinco pessoas e na segunda apenas quatro. Todas elas são licenciadas em áreas afins da informática ou pelo menos com frequência universitária nela, e a trabalhar com tecnologias de informação há dois anos no mínimo, e catorze no máximo. Uma delas era, inclusivamente, *Game*

Designer amador, tendo já criado dois jogos para telemóvel, embora não os tenha publicado.

Note-se que, como se pretende que o *2D Game Editor On-line*, seja um sistema fácil de usar e que tenha presentes todos os aspectos que definem um programa com bastante usabilidade (ver sub-capítulo 2.4), as particularidades deste grupo de utilizadores assumiram particular relevância para os testes. Fazendo a devida excepção para o utilizador que era *Game Designer* amador, quase todos os restantes envolvidos poderiam ser considerados inexperientes neste tipo de sistemas. Deste modo, as opiniões que exprimiram e a observação directa das suas dificuldades foram muito importantes para aperfeiçoar a compreensibilidade imediata do sistema.

Depois das sessões de testes, enquanto os utilizadores trocavam opiniões e impressões relativas à experiência de utilização, notou-se que havia bastante coerência nos diálogos e também alguma facilidade na partilha dos conhecimentos sobre os procedimentos do sistema que o *Game Designer* amador, tendo já alguma familiaridade com este tipo de aplicações, dominou rapidamente. Nestas discussões, a maioria das referências feitas ao sistema foram baseadas, ou relacionadas, com as metáforas implementadas. Ou seja, estas auxiliavam à discussão objectiva do ambiente virtual criado pelo *2D Game Editor On-line*, mesmo quando os utilizadores não tinham acesso directo a ele. Isto pode, claramente, ser considerado como uma confirmação do potencial que estes recursos têm para incluir um sistema, e respectivas funcionalidades, no contexto humano envolvente.

Como forma de contextualização foi explicado/relembrado aos utilizadores, antes das sessões, o objectivo do projecto, algumas das suas funcionalidades e foram também fornecidos ficheiros de imagem com fotogramas (as *strips*), para que pudessem ser carregadas no sistema.

Além disto, de forma a poder-se testar a usabilidade da aplicação com alguma objectividade, definiu-se uma lista com as tarefas mais representativas das potencialidades do sistema (Nielsen, 1993) e que são, no fundo, também as mais habituais durante a fase de implementação de um novo jogo 2D, baseado em sprites.

Segue-se, então, uma breve listagem das tarefas que foram propostas aos utilizadores na primeira sessão de testes, bem como algumas das sugestões mais pertinentes que foram registadas na altura.

Criação de um novo projecto

Passos necessários:

- Fazer *Login*;
- Seleccionar o ícone '+', na árvore, ao nível dos projectos;
- Preencher o formulário de criação de novo projecto.

Resultados do teste:

- Foi levantada uma crítica, por 2 dos 5 utilizadores, ao facto de a árvore de projectos ser a única forma de navegação e acesso às

funcionalidades dentro da aplicação. Porém, depois de assimilado, este procedimento tornou-se rápido e não houve mais questões.

Carregamento de *strips*

Passos necessários:

- Expandir o nó do projecto criado;
- Seleccionar o ícone '+', na árvore, ao nível das *strips*;
- Preencher o formulário de carregamento de nova *strip*, inclusivamente com a indicação do ficheiro de imagem a carregar.

Resultados do teste:

- 2 dos 5 utilizadores manifestaram problemas em encontrar as áreas interactivas, na árvore, para a expansão do nó do projecto. A solução foi encontrada por tentativa-erro, ou seja, os utilizadores *clicaram* até conseguirem atingir o objectivo. Uma vez assimilado o funcionamento da árvore, o problema não tornou a surgir.
- A parte correspondente ao preenchimento do formulário de carregamento das *strips* processou-se sem problemas.

Sugestões:

- Foi sugerido que a árvore fosse mais interactiva, eventualmente até com alguma redundância, como a invocação dos eventos a partir de áreas alternativas (por exemplo, o texto com a designação dos projectos).

Acesso às *strips* carregadas

Passos necessários:

- Expandir o nó do projecto criado;
- Expandir o nó das *strips* do projecto;
- *Clicar* sobre a *strip* que se pretende manipular.

Resultados do teste:

- Não foram encontrados constrangimentos, possivelmente devido à familiarização dos utilizadores com a mecânica da árvore.

O corte de fotogramas

Passos necessários:

- Arrastar a área de corte sobre a *strip* aberta para edição, enquadrando o fotograma que se pretenda cortar;
- *Clicar* no botão de confirmação de selecção.

Resultados do teste:

- Como a *applet* de corte de fotogramas tem o funcionamento inspirado naquele que costuma ser encontrado nas aplicações de desenho, fazendo uso da metáfora *drag-and-drop* para definir a área de corte por arrasto, o procedimento de selecção foi facilmente aprendido por tentativa-erro, por todos os elementos que testaram a aplicação, sem precisarem de apoio.
- Todos os utilizadores manifestaram problemas quando procediam ao ajuste do enquadramento dos fotogramas, em especial nos de menores dimensões. A aplicação não oferecia soluções alternativas que permitissem contornar esta questão.

- Três dos utilizadores levantaram ainda questões quanto ao feedback que a aplicação dava, sobre o sucesso da operação de corte dos fotogramas no servidor, classificando-o de insuficiente. A aplicação não oferecia soluções alternativas que permitissem contornar esta questão.

Sugestões:

- Uma proposta consensual do grupo foi no sentido de implementar uma ferramenta de ampliação, que permitisse a definição das áreas de corte ao detalhe do *píxel*.

Depois da sessão de testes descrita, outra se lhe seguiu, já depois da conclusão das implementações propostas para esta primeira versão do *2D Game Editor On-line*. Houveram esforços no sentido de corrigir os problemas apontados na primeira sessão de testes e, para avaliar os efeitos dessas medidas, tornou-se a pedir aos utilizadores que repetissem as mesmas tarefas, além de testarem as novas funcionalidades implementadas. Segue-se, então, a descrição desta segunda fase de testes.

Criação de um novo projecto

Passos necessários:

- Iguais aos da primeira fase de testes.

Resultados do teste:

- Não foram manifestadas dificuldades.

Carregamento de *strips*

Passos necessários:

- Iguais aos da primeira fase de testes.

Alterações desde a primeira fase de testes:

- As áreas da árvore de ficheiros que reagem aos *inputs* foram complementadas com outros pontos que invocam os mesmos eventos.

Resultados do teste:

- Os problemas encontrados na primeira fase de testes não se tornaram a verificar.

Acesso às *strips* carregadas

Passos necessários:

- Iguais aos da primeira fase de testes.

Resultados do teste:

- Não foram manifestadas dificuldades.

O corte de fotogramas

Passos necessários:

- Iguais aos da primeira sessão.

Alterações desde a primeira fase de testes:

- Foram incluídas algumas janelas (*viewports*) que permitem ver partes ampliadas da *strip*, enquanto se está a arrastar a área de corte. Estas áreas estão centradas nos quatro cantos da área em definição.
- O resultado da operação de corte no servidor é agora mostrado aos utilizadores por um código de cores. Enquanto está em arrasto, a área está delineada com tracejado vermelho; depois de ter sido concluído o arrasto, os limites dos fotogramas estão a amarelo; quando se confirma a área de corte e até que o servidor devolva a resposta, ficam sombreados a azul; quando o servidor confirma a resposta, ficam a delimitados a verde.

Resultados do teste:

- As áreas de ampliação não se provaram suficientes para o nível de detalhe exigido por esta tarefa, pois, dependendo do formato da imagem do fotograma, muitas vezes não ampliam a secção apropriada.
- Não se verificaram mais problemas.

Sugestões:

- Criar uma ferramenta de ampliações que, nos moldes tradicionais, permita ampliar toda a *strip*.

Criação de uma plataforma [1]:

Passos necessários:

- Expandir o nó "plataformas" de um projecto;
- Preencher o formulário de criação de nova plataforma.

Resultados do teste:

- Não foram manifestadas dificuldades.

Alinhar uma animação

Passos necessários:

- Seleccionar uma animação;
- Criar fotogramas para a animação;
- Eliminar fotogramas da animação;
- Pré-visualizar uma animação.

Resultados do teste:

- Não foram manifestadas dificuldades.

Exportação da informação trabalhada

Passos necessários:

- Seleccionar uma plataforma;
- *Clicar* no ícone de exportação.

Resultados do teste:

- Não foram manifestadas dificuldades.

A concluir esta última sessão de testes, sobressaíram outras sugestões importantes, como os métodos de controlo para o trabalho concorrencial (ver Capítulo 4). Os problemas que tinham sido apontados na primeira fase de testes foram resolvidos, e as alterações correctivas bem recebidas pelos utilizadores envolvidos na segunda. Foi manifestado interesse particular pelo algoritmo de composição espacial e pelas possibilidades de aplicação fora do âmbito da indústria dos videojogos como, por exemplo, a produção de *sites web*.

[1] A criação de sprites e animações são semelhantes à das plataformas, e os resultados dos testes foram iguais, pelo que não serão descritos.

Capítulo 4. Conclusões e Trabalho Futuro

4.1. Conclusão

O *2D Game Editor On-line* é uma aplicação que tem o objectivo de se tornar numa ferramenta confiável, onde as empresas possam encontrar o apoio necessário para a realização de algumas das muitas actividades que estão envolvidas na produção dos jogos a duas dimensões, baseados em *sprites* e a correr em telemóveis. Porém, para atingir este objectivo maior, é necessário que o sistema demonstre corresponder aos elevados padrões de qualidade que, normalmente, são exigidos para as aplicações de suporte à produção industrial. Para efeitos de conseguir uma primeira aproximação a essa meta, pretende-se despertar o interesse da comunidade de *Game Designers* amadores, muito activa dentro e fora de Portugal. Espera-se então que, a troco das funcionalidades oferecidas, se venha a obter *feedback* e sugestões que conduzam à valorização do sistema.

Esta decisão sobre o público-alvo inicial teve um peso considerável na decisão sobre as tecnologias a utilizar na implementação, na medida em que, devido à natureza não profissional das actividades que desenvolvem na área, a disposição destes *Game Designers* para investimentos será, também, menor. Então, como forma de reduzir os custos de desenvolvimento e também porque não existem impedimentos ou desvantagens a nível tecnológico, o *2D Game Editor On-line* foi implementado, única e exclusivamente, com recurso a tecnologias *open-source* (*PHP*, *MySQL*, *HTML*, *Java*, *JavaScript*).

A arquitectura de alto nível da aplicação está baseada num ambiente de execução particular que tem, recentemente, despertado o interesse generalizado do público: a *web*. Esta tem, progressivamente, vindo a afastar-se do conceito de meio estático para a publicação de documentos *HTML* e a convergir para se estabelecer como uma base, cada vez menos limitada, para o desenvolvimento de aplicações e oferta de soluções. Os melhoramentos consideráveis a nível das velocidades de comunicação, da qualidade das aplicações *browser*, da segurança acrescida face à perda de dados, da integração de recursos externos[1] e das facilidades de acesso têm contribuído, cada vez mais, para que os serviços informáticos sejam procurados neste ambiente. Veja-se, por exemplo, o surgimento do conceito de *cloud computing*, que propõe a criação de "um modelo que permita o acesso por rede, conveniente e a pedido, a um conjunto partilhado de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser acedidos e abandonados rapidamente e com um mínimo de interacções com os fornecedores" (Mell e Grance, 2009).

A seguir o exemplo, o *2D Game Editor On-line* é uma aplicação cuja arquitectura se enquadra perfeitamente nessa definição, numa lógica de *SaaS*, ou *Software as a Service*, na medida em que permite a utilização do software disponibilizado de uma forma simples, requerendo unicamente os recursos habituais para a navegação *web*

[1] Leitor de pdf, a JVM, o leitor de flash, etc.

tradicional (embora com *cookies*, *JavaScript* e a máquina virtual do *Java*, instalados e a correr).

Este sistema tem implementadas as funcionalidades elementares para o desenvolvimento de jogos baseados em *sprites* para dispositivos móveis, como o corte de fotogramas a partir de ficheiros de imagem (*strips*); o arranjo e o mapeamento dos fotogramas para uma única imagem (atlas); a sequenciação dos fotogramas em animações; a exportação destas informações para um ficheiro em formato binário, passível de importação por aplicações externas e o suporte à criação de versões consoante a plataforma onde se pretenda correr o jogo.

Do ponto de vista tecnológico refere-se que, por requisitos vários a nível de interactividade e detalhe, nem todas as funcionalidades propostas para o editor poderiam ser implementadas através da programação nas linguagens básicas das aplicações *web*, como o *DHTML*. Desse modo, foi necessário introduzir recursos externos na aplicação que permitissem oferecer ao utilizador a mesma interactividade que costuma ser encontrada nos editores baseados em arquitecturas *desktop*: as *applets* Java. Estes recursos estão contidos, na medida em que não comunicam directamente com o servidor, mas apenas com o *browser* onde correm, e limitam-se a realizar uma tarefa previamente definida (como o corte de fotogramas). Este isolamento tem o propósito de facilitar, caso interesse por alguma razão, a troca desses recursos, por outros, possivelmente implementados em linguagens diferentes.

Pela multiplicidade das tarefas que têm de ser cumpridas para produzir um jogo, e tendo em mente que estas variam consoante o tipo e as características do título que está a ser desenvolvido, não é possível construir um editor que dê um suporte universal. Deste modo, em detrimento de tentar implementar todas as funcionalidades necessárias à produção de todos os videojogos, optou-se por conformar a arquitectura do *2D Game Editor On-line* a uma adaptação das normas preconizadas pelo padrão de arquitectura *Model-View-Controller*. Permite-se assim dar aos utilizadores a possibilidade de desenvolverem, eles próprios, os componentes funcionais que realizem as acções que pretendam e, uma vez concluídos estes desenvolvimentos, facilitar a sua integração dentro do sistema. A simplicidade própria deste padrão abre a arquitectura do sistema a novos componentes, mesmo quando a programação destes não está arquitectada segundo as suas normas. Isto deve-se principalmente a uma característica essencial do *Model-View-Controller*: a separação entre o código que implementa a interface de utilizador (par de componentes *View-Controller*), da camada de dados (componente *Model*). Isto permite ligar novos pares *View-Controller*, ou qualquer outro tipo de módulo de interface, independentemente da sua arquitectura, a um *Model* já existente.

A interface com o utilizador do *2D Game Editor On-line* foi projectada para um perfil de utilizador sem conhecimentos técnicos profundos, havendo recurso à representação visual da informação sempre que possível. Isto porque o cargo de *Game Designer* não é, obrigatoriamente, ocupado por uma pessoa com formação, ou sequer com competências, nas áreas tecnológicas.

Foram realizadas duas sessões de teste, sempre com bons resultados e críticas que conduziram evoluções positivas na aplicação, em especial na sua interface com o utilizador. Não foram manifestadas grandes dificuldades em compreender o sistema ou as funcionalidades implementadas. Foi demonstrado bastante interesse no algoritmo que realiza o arranjo dos fotogramas num atlas, em especial pelo seu potencial de utilização em contextos exteriores ao do *2D Game Editor On-line* como, por exemplo, os *sites web*.

Entretanto, o conceito do *2D Game Editor On-line* foi já motivo para estabelecimento de contactos com um colega que está a frequentar o mestrado na Universidade Nova de Lisboa e que pretendia informações mais detalhadas sobre este projecto de mestrado. Esse colega está a desenvolver um motor de videojogos que integra editores (novamente a ideia de construir aplicações pequenas e independentes, cada uma vocacionada à resolução de um determinado tipo de problema) e procurava uma solução do género desta que pudesse ser integrada na sua própria solução. Mostrou-se particularmente interessado na natureza distribuída do editor (que é, nas suas palavras, uma novidade neste tipo de programas).

4.2. Trabalho Futuro

O *2D Game Editor On-line* é uma aplicação que ainda pode ser bastante aperfeiçoada, quer ao nível das funcionalidades base que disponibiliza por defeito, quer ao nível da interface com o utilizador. É de esperar que surjam alterações e evoluções a fazer quando for colocado, para testes, à disposição do público de *Game Designers* amadores. Porém, existem alguns melhoramentos, a vários níveis, que já foram identificados e que irão orientar os trabalhos a realizar no futuro próximo.
Segue-se a sua listagem nos sub-capítulos seguintes.

Implementação das funcionalidades desfazer/refazer
Visto que o trabalho com a aplicação *2D Game Editor On-line* se processa, essencialmente, de forma gráfica, é muito provável que ocorram erros derivados de *inputs* indevidos da parte do utilizador. A implementação de uma funcionalidade desfazer/refazer, assente sobre uma mecânica de pilha, e com acesso rápido por atalhos, seria uma mais-valia para a correcção de eventuais erros. No entanto, deve-se ficar ciente que esta aplicação está distribuída, pelo que a implementação deste mecanismo terá de ser realizada quer a nível dos recursos do cliente, quer do servidor (para anular tanto as acções que ainda não foram guardadas, como aquelas que já o tenham sido).

Melhoramento do corte de fotogramas
Uma das funcionalidades mais procuradas por *Game Designers* amadores é o corte dos fotogramas. No *2D Game Editor On-line*, apesar de se terem tomado cuidados no sentido de tornar este processo fácil e minucioso, é necessário ainda seleccionar a área rectangular de corte através de um sistema *drag-and-drop*. Este processo é

razoavelmente eficiente, embora possa ser melhorado com a introdução de uma funcionalidade simples, a selecção por *flooding*, como as ferramentas *magic wand*, que realizam a selecção por cor, e que costumam fazer parte da interface das aplicações de edição de imagem. Assim, se o utilizador clicar sobre qualquer pixel preenchido da imagem, o algoritmo propagar-se-á, recursivamente, para todos os pixeis em seu redor, até atingir um que tenha a cor definida como limite (a cor de fundo da *strip*). Seguem-se abaixo as linhas gerais de uma possível proposta de implementação:

Seja Q uma pilha vazia;
Seja cor_final a cor de fundo da *strip*, geralmente transparente;
Se a cor do píxel clicado == cor_final, retorna;
$x1, y1 = +\infty$;
$x2, y2 = -\infty$;
Adiciona píxel clicado a Q;
Para cada píxel P de Q:
    Se a cor de P != cor_final
        $x1 = min(x1, P \rightarrow x)$;
        $y1 = min(y1, P \rightarrow y)$;
        $x2 = max(x2, P \rightarrow x)$;
        $y2 = max(y2, P \rightarrow y)$;
        $x1 = x1 - 1$, até que a cor à esquerda de $pixel(x1, P \rightarrow y)$ seja igual à cor_final;
        $x2 = x2 + 1$, até que a cor à direita de $pixel(x2, P \rightarrow y)$ seja igual à cor_final;
        Para cada píxel $P2$ entre $pixel(x1, P \rightarrow y)$ e $pixel(x2, P \rightarrow y)$:
            Se a cor do píxel acima de $P2$!= cor_final, adiciona esse píxel a Q;
            Se a cor do píxel abaixo de $P2$!= cor_final, adiciona esse píxel a Q;
Continua ciclo até terminarem os píxeis de Q;
Retorna;

Este algoritmo foi inspirado no algoritmo tradicional de coloração, o *flood fill*, conforme foi encontrado num artigo da *Wikipedia* (Wikipedia, 2009). Depois de processar cada píxel, apenas toma atenção nos píxeis que se lhes estejam situados directamente acima, abaixo, à esquerda ou à sua direita, "espalhando-se" nestes quatro sentidos. Quando encontra píxeis com a cor do fundo da *strip*, retorna. No fim, pode-se cortar o rectângulo definido entre o canto superior esquerdo, $pixel(x1, y1)$, e o canto inferior direito $pixel(x2, y2)$, que corresponderá à melhor

área de corte, portanto com menor desperdício de espaço, para determinado fotograma.

Para aqueles fotogramas que tenham mais do que uma área, separadas por espaço vazio, comum em, por exemplo, *sprites* que representem explosões, será depois necessário incluir um modo de concatenar várias selecções (o mesmo sistema que se encontra nas referidas aplicações de edição de imagem, que permitem seleccionar áreas separadas através de *cliques* em cada uma, enquanto se pressiona a tecla *shift*). Visto que um fotograma é uma imagem normal rectangular, é necessário que se guardem as coordenadas mais afastadas, de todos os píxeis que pertençam à (ou às) selecções.

Incorporação de rotação no algoritmo

O algoritmo desenvolvido para a composição dos atlas a partir de fotogramas também pode ser alvo de optimizações. Na primeira implementação efectuada, não se equacionou a hipótese de rodar os fotogramas antes (ou durante) a sua inserção na árvore do arranjo. Ao considerar-se esta possibilidade, é mais provável que o algoritmo consiga encontrar um ponto de inserção dentro da fronteira actual do atlas e com um mínimo de espaço desperdiçado, evitando assim o recurso ao redimensionamento. Esta funcionalidade trará, sem dúvida, mais complexidade ao algoritmo, virtualmente duplicando o número de avaliações inserções da proposta inicial (duas para cada fotograma, uma na posição vertical e a outra na horizontal).

Mecanismo de equilíbrio na árvore de representação do espaço

Actualmente, os nós da árvore binária que representa o arranjo no espaço do atlas e na qual se baseia o algoritmo de composição do atlas, estão ordenados pela forma como o espaço é dividido. Exemplificando, uma área que seja dividida em três quando se dá a inserção de um novo fotograma (a área ocupada e as duas áreas resultantes da divisão horizontal ou vertical do espaço livre) terá como representação na árvore um nó cujos filhos correspondem às duas áreas livres criadas.

Ora, sem preocupações de ordenação, esta árvore pode, com a sucessão das operações de inserção, tornar-se bastante desequilibrada em termos das alturas das sub-árvores inferiores (radicadas a partir de qualquer nó-filho) e isso contribuirá, naturalmente, para prejudicar o desempenho das operações que requeiram travessias nessa estrutura.
No entanto, cada nó dessa árvore pode ter como valor, por exemplo, a área do espaço livre que representa. Fazendo uso desta grandeza escalar, é possível a implementação de uma mecânica que permita a manutenção do equilíbrio da árvore, reduzindo assim o tempo consumido pelo algoritmo em travessias. Como exemplo de um possível benefício que poderá advir desta estratégia para a operação de inserção de um novo fotograma, consistirá em que qualquer nó referenciado como o filho esquerdo de outro nó, terá uma área inferior e, se o fotograma a inserir não couber nesta área, certamente também não caberá em nenhuma das áreas representadas nos nós da sub-árvore à esquerda desse nó, e a pesquisa não prosseguirá nesse sentido.

Exportação para código do jogo

Nesta altura, o *2D Game Editor On-line* está preparado para exportar a informação trabalhada (ver Capítulo 3) para um ficheiro binário, após um processo de serialização. Para que este formato possa ser utilizado nas fases seguintes da produção de um videojogo, é necessário que se proceda à *des-serialização* dessa informação e isso poderá conduzir a problemas de compatibilidade entre arquitecturas (por exemplo, as questões de implementações em *little-endian* ou *big-endian* para a ordenação dos *bytes* de inteiros).

Uma possibilidade interessante para o formato exportado, será permitir que o utilizador possa decidir qual pretende. À parte as opções mais comuns e intuitivas, como o *XML*, poder-se-á equacionar a hipótese da criação de classes em *Java*[1], ou até mesmo um rudimento de *framework*, que tenha já implementados os métodos para aceder à uma determinada *sprite*, correr determinada animação, deslocá-la, etc.

Assim, a restante programação do jogo final seria no sentido de criar uma "matriz" funcional que permitisse a integração e utilização destas exportações, mas não tivesse de se preocupar com os detalhes da implementação das sprites, carregamento e gestão da memória alocada para os fotogramas ou outros detalhes que seriam tratados, de forma automática, pelo código exportado.

Perfis de utilização

Durante a fase de desenvolvimento, o *2D Game Editor On-line* apenas contemplou um utilizador por projecto, o *Game Designer*. Mas esta não é a realidade das empresas, onde as equipas que trabalham no desenvolvimento dos videojogos podem ser bastante alargadas e heterogéneas. Assim, será eventualmente preciso criar um sistema de perfis de utilização e associá-lo aos utilizadores registados, atribuindo adequadamente as permissões e restrições. Por exemplo, os utilizadores do perfil *Game Designer* poderão realizar todas as tarefas dentro de um projecto, enquanto que um do perfil *Artista* apenas poderá carregar e cortar os fotogramas.

Controlo de acessos concorrenciais

Uma das sugestões que vieram dos testes executados foi, precisamente, uma forma de gerir os acessos concorrenciais à informação. A proposta foi baseada em experiência prévia que a pessoa tinha com o sistema *Microsoft SourceSafe* e incluía restringir os acessos a determinados recursos quando se verifica um acesso. Deste modo, e até que o recurso seja libertado – através de uma rotina que verifique os acessos ou até que o utilizador aceda activamente a outro recurso – os acessos ficam limitados. Como esta não é uma solução verdadeiramente concorrente, foram sugeridas outras hipóteses, no sentido de permitir a utilizadores distintos trabalhar sobre a mesma informação. Uma proposta consistiria em implementar no servidor um sistema de *pooling* nas conexões *HTTP* estabelecidas com os clientes, que não permitisse o seu fecho. Desse modo, assim que um utilizador alterasse a informação do servidor, este faria a propagação dessa alteração para todos os

[1] Houve, inclusivamente, uma sugestão da parte de um *Game Designer* profissional no sentido de produzir código para o *SDK* do *iPhone*.

clientes que lhe estivessem conectados. Ainda outra hipótese, consistiria na implementação de um mecanismo, nos clientes, que realizasse a execução de verificações assíncronas e periódicas no servidor (por *AJAX*, eventualmente). Então, sempre que um cliente que estivesse a aceder a determinado recurso questionasse o servidor e a resposta deste fosse no sentido de comunicar uma alteração feita por outro utilizador, o cliente local actualizar-se-ia.

Outras funcionalidades

O *2D Game Editor On-line* constituiu-se, até à data, como um *proof-of-concept* para o qual foram escolhidas as funcionalidades mais exigentes e representativas da globalidade do trabalho com um editor deste género. No entanto, para aumentar as possibilidades de sucesso da aplicação, esta terá de dar suporte a outras tarefas, como a composição de cenários de jogo em matrizes posicionais ou a possível integração de um motor físico simples, que detecte colisões e implemente um sistema de forças parametrizável como, por exemplo, a gravidade que será utilizada em jogos do género do *tetris* ou plataformas.

Outro recurso, muito importante para os jogos 2D baseados em *sprites,* é o conceito do ponto *pivot*. Este ponto corresponde à distância relativa entre um ponto fixo no ecrã do jogo e o canto superior esquerdo de cada fotograma. No decorrer da animação, ao fazer-se a passagem dos fotogramas, o ponto *pivot* de cada um vai estar sempre localizado sobre o mesmo ponto fixo no ecrã dando a impressão ao utilizador que, apesar da diferença no tamanho de cada fotograma, a *sprite* se mantém sempre fixa no ecrã, assente sobre aquele ponto e é a partir dele que se desenrolam as animações.

Referências Bibliográficas

Alves, L. 2008. *Estado da Arte dos games no Brasil: trilhando caminhos*. Actas - ZON Digital Games 2008. Nelson Zagalo e Rui Prada. Centro de Estudos de Comunicação e Sociedade, Instituto de Ciências Sociais, Universidade do Minho, Braga, pp. 9-18.

Arnaud, J. 2009. *INFOS - Na Europa, mercado português de videojogos foi o que mais cresceu*. Site web acedido a 12 de Setembro de 2009, em: http://www.gamerstek.com/noticias-7897.php

Berners-Lee, T. e Connolly, D. 1995. *Hypertext Markup Language - 2.0*. Site web acedido a 1 de Dezembro de 2009, em: http://www.w3.org/MarkUp/html-spec/html-spec_toc.html

Buschmann, F. et al. 1996. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons Lda, West Sussex, Inglaterra.

Cardoso, B e Romão, T. 2008. *Um Caso Prático de Reabilitação de um Editor de Jogos Móveis Utilizando o Paradigma Model-View-Controller*. Interacção 2008 - Actas da 3ª Conferência Nacional em Interacção Pessoa-Máquina. J. C. Campos, Daniel Gonçalves, T. Romão e L. Rato (editores). Grupo Português de Computação Gráfica, Porto, Portugal, pp. 109-114.

Coffman, E. et al. 1997. *Approximation Algorithms for NP-Hard Problems*. D. Hochbaum, PWS Publishing, Boston, EUA.

Converse, T. Park, J. e Morgan, C. 2004. *PHP5 and MySql Bible*. Wiley Publishing, Inc. Indiana, USA.

Celeritas. *Should I Build a Windows or Web Application*. Site web acedido a 8 de Julho de 2008, em: http://dotnet.celeritas.com/Windows-or-Web-Application.html

Coutaz, J. 1987. *PAC, an object oriented model for dialog design*. Human - Computer Interaction - Interact'87. H. J. Bullinger and B. Shackel, Elsevier Science Publishers, North Holland, Amsterdam, pp. 431-436.

Cunningham, H. 1999. *Object Oriented Design and Programming*. Site web acedido a 20 de Dezembro de 2009, em: http://www.cs.olemiss.edu/~hcc/csci581oo/notes/patterns.html

Dix, A. et al. 1998. *Human-Computer Interaction*. Prentice Hall Europe, London.

Dusina, M. 2006. *Bin Packing*. Site web acedido a 3 de Janeiro de 2010, em: http://www.developerfusion.com/article/5540/bin-packing

Entertainment Software Association. 2009. *2009 Sales Demographic and Usage Data - Essential Facts About Video Game Industry*. Site web acedido a 30 de Janeiro de 2010, em: http://www.theesa.com/facts/pdfs/ESA_EF_2009.pdf

Fowler, M. 1996. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, Boston, EUA.

Gerosa, L. e Cury, D. *Um Framework para a Construção Cooperativa de Jogos.* XIX Simpósio Brasileiro de Informática na Educação. Universidade Federal do Espírito Santo. Vitória. Brasil. Site web acedido a 20 de Janeiro de 2010, em: http://200.169.53.89/download/CD%20congressos/2008/SBIE/workshops/workshop%203/Um%20Framework%20para%20a%20Constru%C3%A7%C3%A3o%20Cooperativa%20de%20Jogos.pdf

Instituto Nacional de Estatística. 2009. *Sociedade da Informação e do Conhecimento - Inquérito à Utilização de Tecnologias da Informação e da Comunicação pelas Famílias (indivíduos dos 10 aos 15 anos).* Site web acedido a 26 de Novembro de 2009, em: http://www.ine.pt/ngt_server/attachfileu.jsp?look_parentBoui=59744386&att_display=n&att_download=y

Ivanov, I. 2006. *Pratical Texture Atlases.* Site web acedido em 12 de Dezembro de 2009, em: http://www.gamasutra.com/view/feature/2530/practical_texture_atlases.php

Krasner, G. E. e Pope, T. 1988. *A cookbook for using the model-view-Controller user interface paradigm in SmallTalk-80.* Journal of Object-Oriented Programming. v. 1, n. 3, SIGS Publications, USA, pp. 26-49.

Mell, P. Grance, T. 2009. *The NIST Definition of Cloud Computing.* Site web acedido a 20 de Janeiro de 2010, no Web Site do National Institute of Standards and Technology, Information Technology Laboratory, em: http://csrc.nist.gov/groups/SNS/cloud-computing

Nielsen, J. 1993. *Usability Engineering*, Academic Press Inc

Preece, J. et al. 2002. *Interaction Design Beyond Humam-Computer Interaction.* John Wiley and Sons Inc., New York, EUA.

Schofield, M. 2008. *PHP4MVC: An MVC architecture in PHP 4.* Site web acedido a 10 de Dezembro de 2009, em: http://mark.schofield.free.fr/articles

Scott, J. *Packing Lightmaps.* Site web acedido a 3 de Janeiro de 2010, em: http://www.blackpawn.com/texts/lightmaps/default.html

Szabó, K. 1995. *Metaphors and the User Interface.* Site web acedido a 3 de Janeiro de 2010, em: http://www.katalinszabo.com/metaphor.htm

Vandevenne, L. 2004. *Lode's Computer Graphics Tutorial - Flood Fill.* Site web acedido a 12 de Janeiro de 2010, em: http://www.student.kuleuven.be/~m0216922/CG/floodfill.html

Wikipedia. 2009. *Flood Fill.* Site web acedido a 20 de Janeiro de 2010, em: http://en.wikipedia.org/wiki/Flood_fill

Glossário

AJAX – *Asynchronous Javascript and XML*. É um conjunto de técnicas utilizadas em browsers, com o objectivo de permitir comunicações com o servidor de forma assíncrona, em segundo plano, que é geralmente utilizado como recurso para criar aplicações web interactivas.

Atlas – Literalmente, uma colecção de mapas. No contexto do presente trabalho, este termo é utilizado para designar uma imagem construída a partir do arranjo de vários fotogramas, e que deve ser mapeada para que cada fotograma possa ser utilizado independentemente.

Benchmarking – Método de avaliação (geralmente utilizada para comparar desempenhos), através da utilização de um indicador único. Os testes são executados colocando-se os objectos em estudo no mesmo cenário e registando-se os desempenhos individuais de cada um. Esta métrica pode, depois, ser utilizada para classificar comparativamente os objectos.

Cloud Computing – Trata-se de um modelo que permita o acesso por rede, conveniente e a pedido, a um conjunto partilhado de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser acedidos e abandonados rapidamente e com um mínimo de interacções com os fornecedores" (Mell e Grance, 2009)

DHTML – *Dynamic HyperText Markup Language*. União de *HTML, JavaScript,* e uma linguagem de estilos, como as *CSS*, aliada ao *DOM*, para permitir alterações dinâmicas às páginas sem necessidade de comunicações com o servidor.

DOM – *Document Object Model*. Convenção independente de plataforma ou linguagens, que representa e interage com elementos HTML, XHTML, e XML, representando-os como objectos. Esses objectos podem, então, ser acedidos e manipulados directamente, independentemente da linguagem utilizada.

Fotograma – Um fotograma (ou *frame*) corresponde, em linguagem cinematográfica, a uma imagem desenhada numa *strip*.

Framework – Uma abstracção de *software*, onde uma base comum de código que realiza funcionalidades genéricas, pode ser selectiva e parcialmente substituída por novas implementações de funcionalidades específicas. Existem algumas diferenças marcantes com uma *API* nos moldes tradicionais como, por exemplo, é a própria framework que gere o fluxo de controlo da aplicação.

Game Designer – Na indústria dos jogos de vídeo, trata-se do cargo atribuído à pessoa encarregue de desenvolver o arranjo geral do jogo, incluindo desenho da área de jogo, escrita de texto e a entrada e manuseamento de valores que balançam e afinam a experiência de jogo.

Heap (memória de telemóvel) – Uma área de memória utilizada para alocações dinâmicas, em tempo de execução, de um programa.

JSON – *JavaScript Object Notation*. Formato de troca de informação, com sintaxe derivada do *JavaScript*. Utilizada para representar estruturas de dados simples e *arrays* associativos (aos quais se dá o nome de objectos, no contexto *JSON*), é muito usada na transmissão de dados dos servidores para os *browsers* cliente.

LAMP – *Linux*, *Apache*, *MySQL* e *PHP* (*Perl* ou *Python*). Pacote que agrega as tecnologias de *software* citadas. Muito popular devido à elevada integração das as tecnologias que o compõem, baixo custo (*open source*) e ao facto de, se utilizadas em conjunto, configurarem um servidor *web* bastante completo.

Model-View-Controler (padrão de arquitectura) – Este padrão de arquitectura procura separar a implementação de uma aplicação interactiva em três componentes, por responsabilidades. O *Model* (*Modelo*) é responsável pelos dados e pelos métodos de manipulação destes. O *View* (Vista) disponibiliza a informação contida no respectivo modelo. O *Controller* (Controlador) reage, para um determinado par *View-Model*, ao *input* do utilizador. O par *View-Controller*, ou o conjunto de todos os pares, formam a interface do utilizador. (Buschmann, F. et al. 1996)

Padrão de Arquitectura – Modela a arquitectura geral de um sistema ou aplicação.

Padrões de Desenho – Micro arquitecturas de mais baixo nível que os padrões de arquitectura, aplicáveis em várias situações e vocacionados para a modelação da implementação de subsistemas com funções especificas.

Padrões de Idioma – Padrões específicos de uma linguagem de programação que descreve como implementar certos componentes ou partes destes através dos mecanismos específicos da linguagem de implementação.

Portabilização – Em inglês, *porting*. O processo de adaptação de *software* para que um programa executável possa correr num ambiente de computação diferente daquele para que foi, originalmente, desenhado. Numa palavra, o processo de portabilizar uma determinada aplicação.

SGBD – Sistema de Gestão de Bases de Dados. *Software* dedicado à gestão de bases de dados que, geralmente, disponibiliza formas de comunicação para poder ser integrado, como componente funcional, em sistemas mais complexos.

Sprite – Uma *sprite* é uma imagem ou animação em duas dimensões. Foram originalmente inventadas como uma forma de compor várias imagens juntas em jogos de vídeo bidimensionais, utilizando *software* especializado. À medida que o desempenho dos computadores melhorou, esta optimização tornou-se desnecessária e o termo evoluiu para se referir especificamente às imagens bidimensionais que estão integradas numa cena.

Strip – Uma *strip*, literalmente "fita", é uma sequência de desenhos, produzidos pelos artistas encarregues da arte do jogo, representando várias poses de uma personagem ou objecto.

Templates – Documento modelo, sem conteúdos, apenas com as formatações e indicações ao nível da meta-informação (ou seja, informações sobre o tipo de dados que se espera). Muito utilizado em sistemas *web*, em especial para a produção de relatórios e implementação de interfaces reutilizáveis.

Widget – Elemento reutilizável de interfaces de utilizador. Trata-se de um ponto de interactividade, mostrando determinada parte do modelo de dados sob uma representação própria e permite, de forma directa, a sua manipulação.

Anexos

Anexo I

Caracterização estatística das populações e dos testes realizados.

Este anexo descreve, estatisticamente, as populações de fotogramas que foram utilizadas para comparar as estratégias de crescimento do atlas, de acordo com a implementação do seu algoritmo de composição. Naturalmente, não se tratam de verdadeiros fotogramas enquanto ficheiros de imagem, mas sim de rectângulos abstractos, criados aleatoriamente, para simular o desempenho do algoritmo de composição de atlas em populações de características variáveis.

Note-se que, como medida de comparação, se utiliza recorrentemente a razão R, calculada a partir das medidas laterais dos indivíduos, da seguinte forma: $R = \frac{lado\,menor}{lado\,maior}$, onde $R \in]0,1]$. Particularmente, quando $R = 1$, significa que o paralelogramo em causa é um quadrado (lado menor = lado maior) e $R \to 0$, conforme aumenta o valor absoluto da diferença entre os seus lados.

Nas descrições apresentadas, os indivíduos de cada população foram agrupados em intervalos de classe, definidos segundo a razão R descrita acima, de modo a possibilitar a elaboração de um histograma que dê uma visão geral da população.

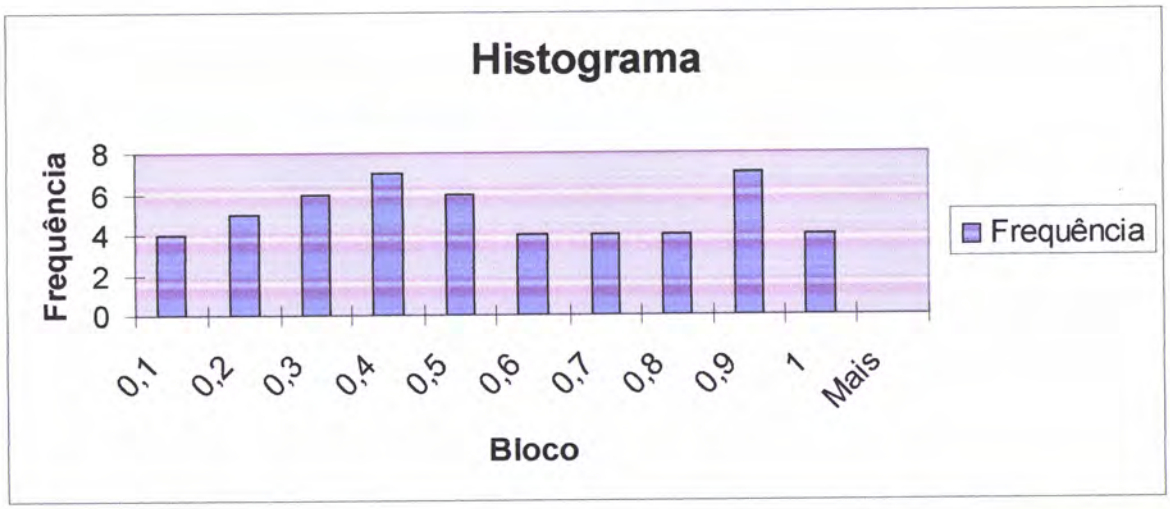
Incluem-se também tabelas com a descrição dos resultados da aplicação do algoritmo de composição do atlas a cada população particular, bem como das suas variações conforme as estratégias de crescimento testadas. Por fim, a ilustrar estas informações, figuram também as miniaturas dos atlas que foram criados.

População 1

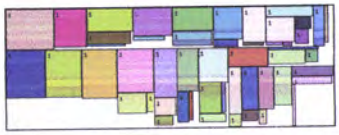
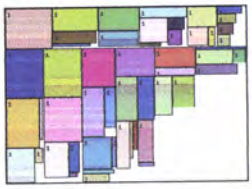
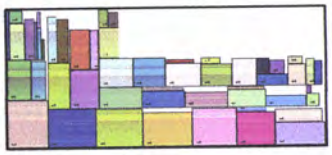
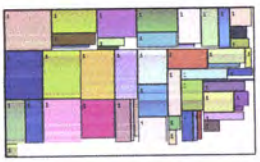
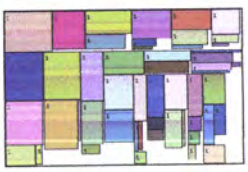
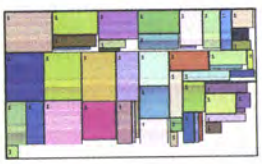
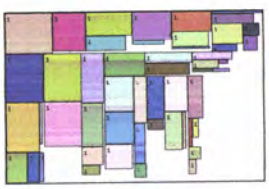
Composta por 51 fotogramas; não existem dois elementos com o mesmo valor de R, ou seja, dois rectângulos de dimensões iguais (inexistência de moda); elevada variação nas dimensões (coeficiente de variação > 0,5).

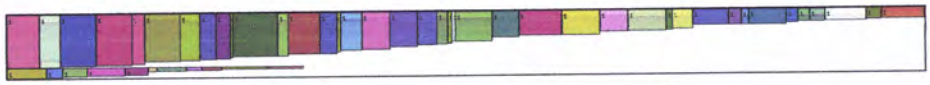
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	4	7,84
]0.1, 0.2]	5	9,8
]0.2, 0.3]	6	11,76
]0.3, 0.4]	7	13,73
]0.4, 0.5]	6	11,76
]0.5, 0.6]	4	7,84
]0.6, 0.7]	4	7,84
]0.7, 0.8]	4	7,84
]0.8, 0.9]	7	13,73
]0.9, 1]	4	7,84

Estatísticas de R	
Média	0,498644148
Erro-padrão	0,039305318
Mediana	0,488636364
Moda	---
Desvio-padrão	0,280696112
Variância da amostra	0,078790307
Curtose	-1,253873459
Assimetria	0,118073722
Intervalo	0,95
Mínimo	0,05
Máximo	1
Coeficiente de variação	0,562918693
Soma das Areas	123514
Contagem	51



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	365	638	232870	53,04
2DGEO	Menor diferença linear	657	240	157680	78,33
2DGEO	Maior area	482	347	167254	73,85
2DGEO	Menor area	278	638	177364	69,64
2DGEO	Quadrado	500	291	145500	84,89
2DGEO	Rectângulo de ouro	476	308	146608	84,25
2DGEO	Relação Média	500	291	145500	84,89
2DGEO	Distância à Origem	517	339	175263	70,47
Slick		1968	119	234192	52,74

Estratégia	Área (px)	Arranjo da população (dimensões a 25%, aprox.)
Maior diferença linear	232870	
Menor diferença linear	157680	
Maior área	167254	
Menor área	177364	
Quadrado	145500	
Rectângulo de ouro	146608	
Relação Média	145500	
Distância à Origem	175263	

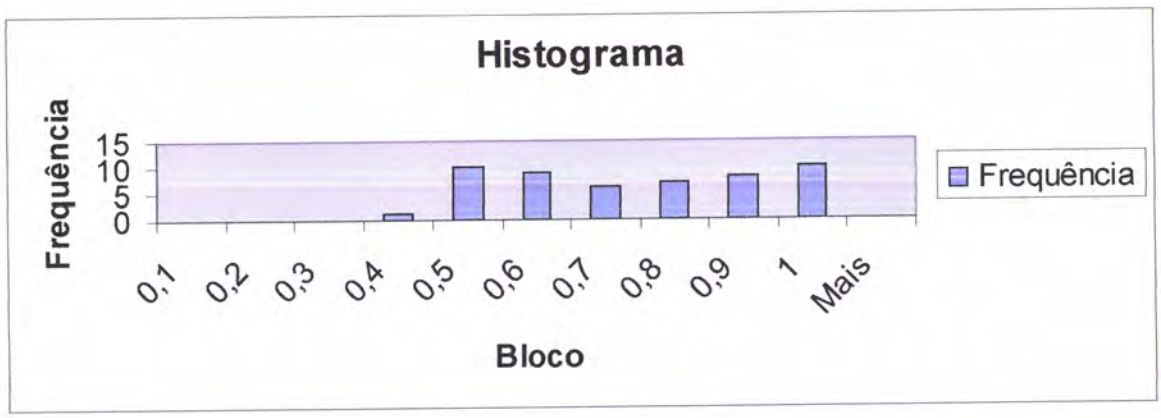
Estratégia	Área (px)	Arranjo da população (dimensões a 25%, aprox.)
Slick	234192	

População 2

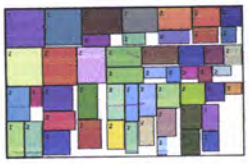
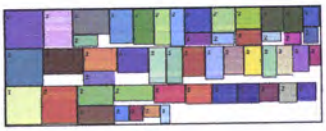
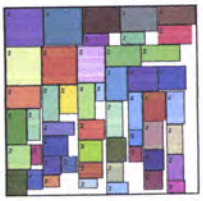
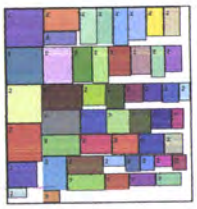
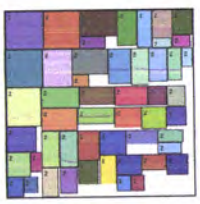
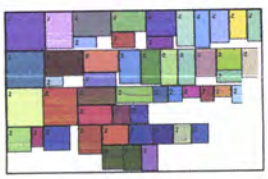
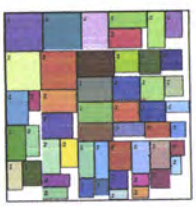
Composta por 51 fotogramas; existe, pelo menos, um quadrado na população (máximo=1); maior parte dos fotogramas é um rectângulo cujo lado maior é superior ao dobro do menor (média>0.6).

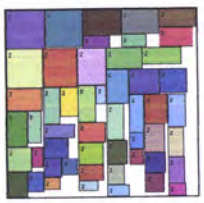
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	0	0
]0.1, 0.2]	0	0
]0.2, 0.3]	0	0
]0.3, 0.4]	1	1,96
]0.4, 0.5]	10	19,61
]0.5, 0.6]	9	17,65
]0.6, 0.7]	6	11,76
]0.7, 0.8]	7	13,73
]0.8, 0.9]	8	15,69
]0.9, 1]	10	19,61

Estatísticas de R	
Média	0,699429936
Erro-padrão	0,025761644
Mediana	0,68115942
Moda	0,5
Desvio-padrão	0,183974937
Variância da amostra	0,033846778
Curtose	-1,311995266
Assimetria	0,114177738
Intervalo	0,605263158
Mínimo	0,394736842
Máximo	1
Coeficiente de variação	0,263035549
Soma das Áreas	120328
Contagem	51



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	474	297	140778	85,47
2DGEO	Menor diferença linear	631	230	145130	82,91
2DGEO	Maior area	383	373	142859	84,23
2DGEO	Menor area	370	388	143560	83,82
2DGEO	Quadrado	377	371	139867	86,03
2DGEO	Rectângulo de ouro	511	321	164031	73,36
2DGEO	Relação Média	369	375	138375	86,96
2DGEO	Distância à Origem	383	373	142859	84,23
Slick		2035	110	223850	53,75

Estratégia	Área (px)	Arranjo da população (dimensões a 25%, aprox.)
Maior diferença linear	140778	
Menor diferença linear	145130	
Maior área	142859	
Menor área	143560	
Quadrado	139867	
Rectângulo de ouro	164031	
Relação Média	138375	

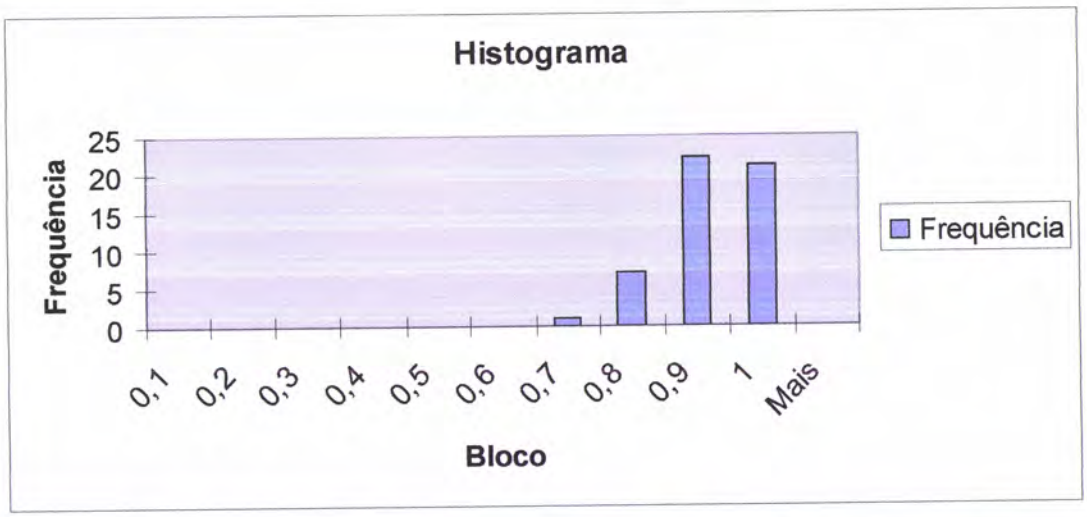
Estratégia	Área (px)	Arranjo da população (dimensões a 25%, aprox)
Distância à Origem	142859	
Slick	223850	

População 3

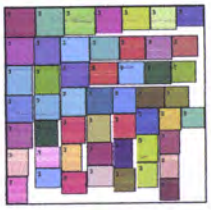
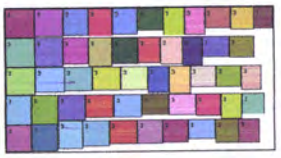
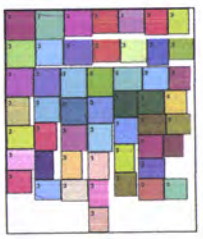
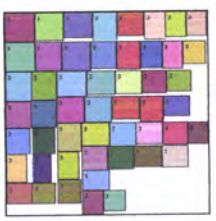
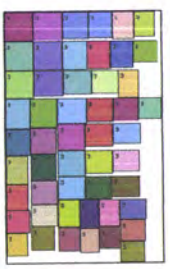
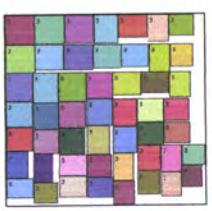
Composta por 51 fotogramas; maior parte dos fotogramas são rectângulos cujo lado maior é superior a 4/5 do menor (média>0,8); rectângulos de dimensões pouco variáveis (coeficiente de variação<0,09).

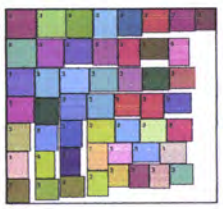
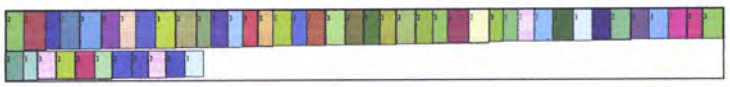
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	0	0
]0.1, 0.2]	0	0
]0.2, 0.3]	0	0
]0.3, 0.4]	0	0
]0.4, 0.5]	0	0
]0.5, 0.6]	0	0
]0.6, 0.7]	1	1,96
]0.7, 0.8]	7	13,73
]0.8, 0.9]	22	43,14
]0.9, 1]	21	41,18

Estatísticas de R	
Média	0,871993273
Erro-padrão	0,010720162
Mediana	0,87755102
Moda	0,962962963
Desvio-padrão	0,076557271
Variância da amostra	0,005861016
Curtose	-0,552827578
Assimetria	-0,417792297
Intervalo	0,293678161
Mínimo	0,689655172
Máximo	0,983333333
Coeficiente de variação	0,087795713
Soma das Áreas	129200
Contagem	51



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	396	393	155628	83,02
2DGEO	Menor diferença linear	537	286	153582	84,12
2DGEO	Maior area	377	443	167011	77,36
2DGEO	Menor area	170	883	150110	86,07
2DGEO	Quadrado	405	404	163620	78,96
2DGEO	Rectângulo de ouro	313	501	156813	82,39
2DGEO	Relação Média	395	381	150495	85,85
2DGEO	Distância à Origem	419	387	162153	79,68
Slick		2014	105	211470	61,1

Estratégia	Área (px)	Arranjo da população (dimensões a 25%, aprox)
Maior diferença linear	155628	
Menor diferença linear	153582	
Maior área	167011	
Menor área	150110	
Quadrado	163620	
Rectângulo de ouro	156813	
Relação Média	150495	

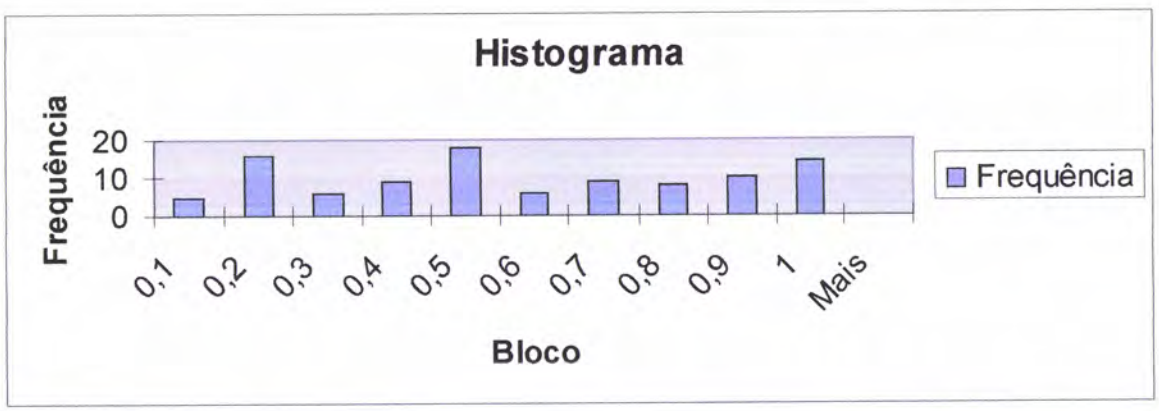
Estratégia	Área (px)	Arranjo da população (dimensões a 25%, aprox.)
Distância à Origem	162153	
Slick	211470	

População 4

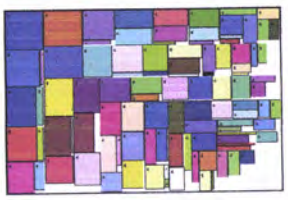
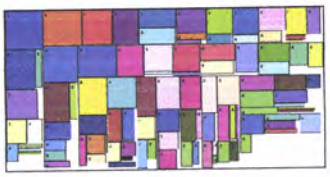
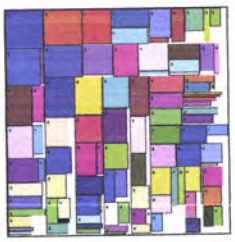
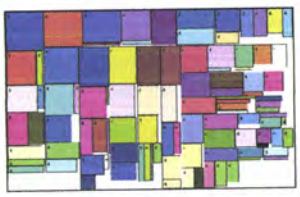
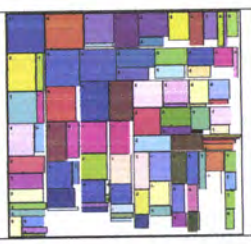
Composta por 101 fotogramas; existe, pelo menos, um quadrado na população (máximo=1); maior parte dos fotogramas é um rectângulo cujo lado maior está proximo do menor (média aprox. 0,5); elevada variação nas dimensões (coeficiente de variação > 0,5).

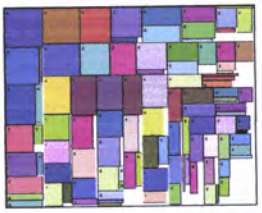
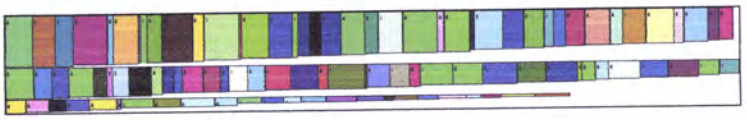
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	5	4,95
]0.1, 0.2]	16	15,84
]0.2, 0.3]	6	5,94
]0.3, 0.4]	9	8,91
]0.4, 0.5]	18	17,82
]0.5, 0.6]	6	5,94
]0.6, 0.7]	9	8,91
]0.7, 0.8]	8	7,92
]0.8, 0.9]	10	9,9
]0.9, 1]	14	13,86

Estatísticas de R	
Média	0,523711772
Erro-padrão	0,028374711
Mediana	0,481481481
Moda	0,862068966
Desvio-padrão	0,285162321
Variância da amostra	0,081317549
Curtose	-1,233926624
Assimetria	0,080252103
Intervalo	0,946808511
Mínimo	0,053191489
Máximo	1
Coficiente de variação	0,544502408
Soma das Áreas	275529
Contagem	101



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	685	456	312360	88,21
2DGEO	Menor diferença linear	532	614	326648	84,35
2DGEO	Maior area	776	437	339112	81,25
2DGEO	Menor area	796	402	319992	86,1
2DGEO	Quadrado	558	564	314712	87,55
2DGEO	Rectângulo de ouro	719	446	320674	85,92
2DGEO	Relação Média	587	553	324611	84,88
2DGEO	Distância à Origem	625	495	309375	89,06
Slick		2017	187	377179	73,05

Estratégia	Área (px)	Arranjo da população (dimensões a 20%, aprox)
Maior diferença linear	312360	
Menor diferença linear	326648	
Maior área	339112	
Menor área	319992	
Quadrado	314712	
Rectângulo de ouro	320674	
Relação Média	324611	

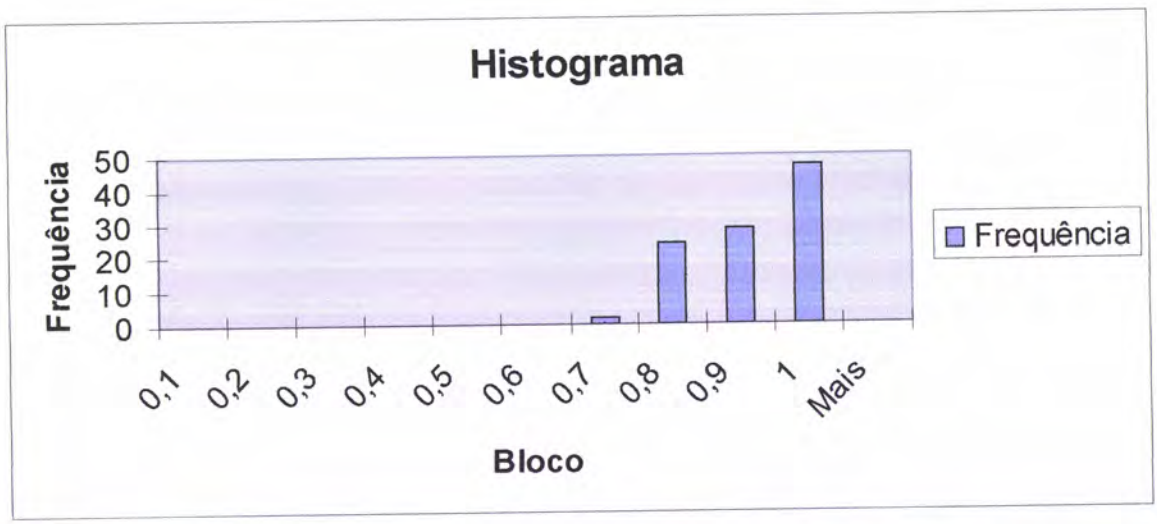
Estratégia	Área (px)	Arranjo da população (dimensões a 20%, aprox)
Distância à Origem	309375	
Slick	377179	

População 5

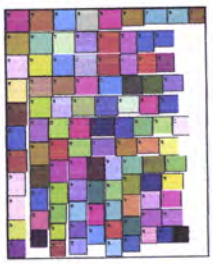
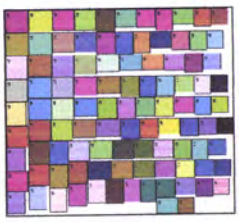
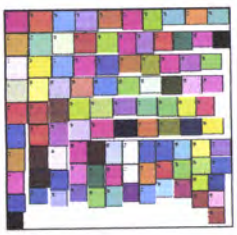
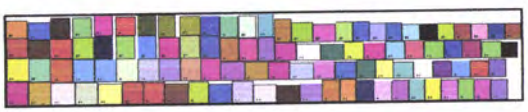
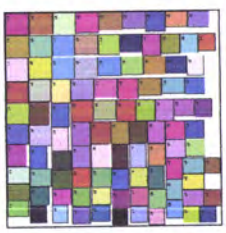
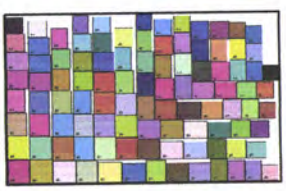
Composta por 101 fotogramas; existe, pelo menos, um quadrado na população (máximo=1); rectângulos de dimensões pouco variáveis (coeficiente de variação<0,1).

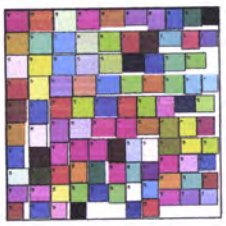
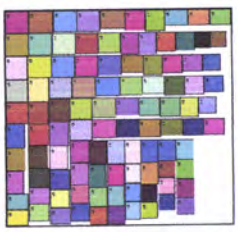
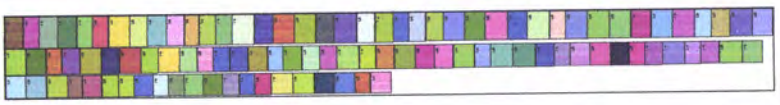
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	0	0
]0.1, 0.2]	0	0
]0.2, 0.3]	0	0
]0.3, 0.4]	0	0
]0.4, 0.5]	0	0
]0.5, 0.6]	0	0
]0.6, 0.7]	2	1,98
]0.7, 0.8]	24	23,76
]0.8, 0.9]	28	27,72
]0.9, 1]	47	46,53

Estatísticas de R	
Média	0,872363137
Erro-padrão	0,008362598
Mediana	0,893617021
Moda	0,716666667
Desvio-padrão	0,084043067
Variância da amostra	0,007063237
Curtose	-0,981569445
Assimetria	-0,459680213
Intervalo	0,310344828
Mínimo	0,689655172
Máximo	1
Coeficiente de variação	0,096339545
Soma das Áreas	250101
Contagem	101



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	502	620	311240	80,36
2DGEO	Menor diferença linear	570	512	291840	85,7
2DGEO	Maior area	567	543	307881	81,23
2DGEO	Menor area	223	1288	287224	87,08
2DGEO	Quadrado	536	535	286760	87,22
2DGEO	Rectângulo de ouro	420	684	287280	87,06
2DGEO	Relação Média	534	527	281418	88,87
2DGEO	Distância à Origem	566	537	303942	82,29
Slick		2035	155	315425	79,29

Estratégia	Área (px)	Arranjo da população (dimensões a 20%, aprox)
Maior diferença linear	311240	
Menor diferença linear	291840	
Maior área	307881	
Menor área	287224	
Quadrado	286760	
Rectângulo de ouro	287280	

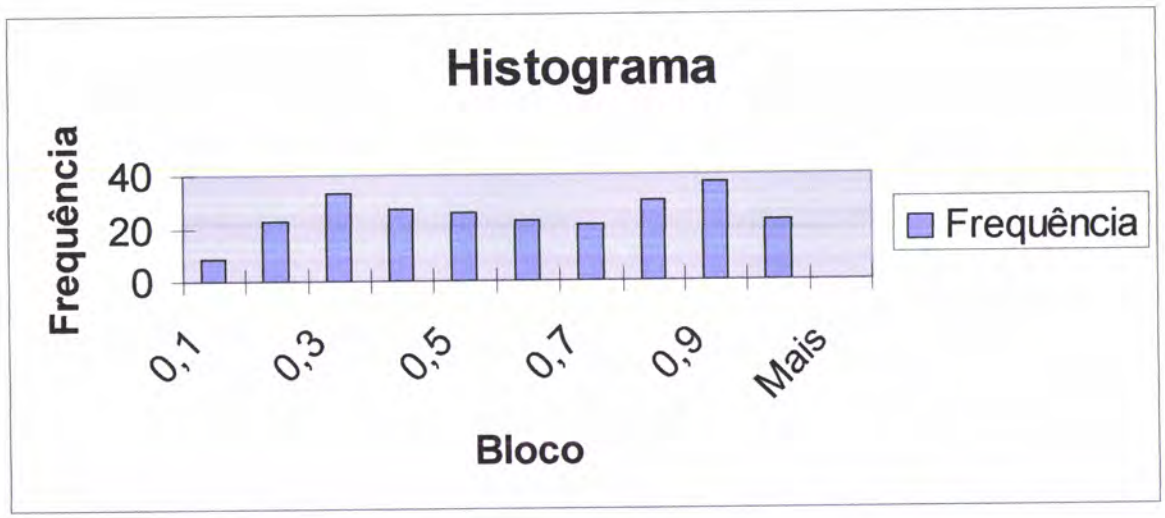
Estratégia	Área (px)	Arranjo da população (dimensões a 20%, aprox)
Relação Média	281418	
Distância à Origem	303942	
Slick	315425	

População 6

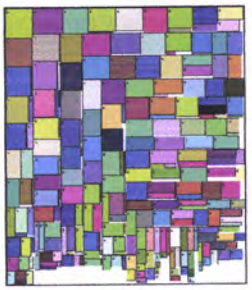
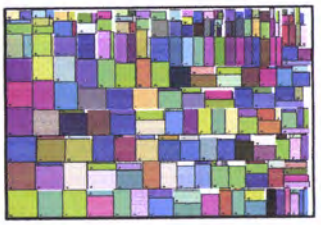
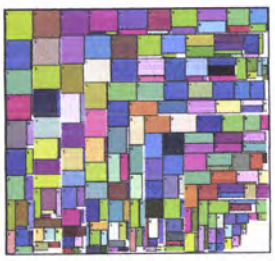
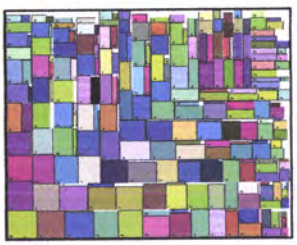
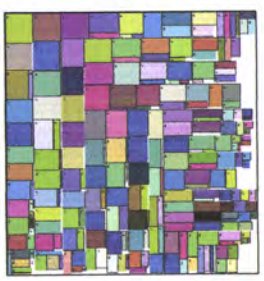
Composta por 251 fotogramas; existe, pelo menos, um quadrado na população (máximo=1); elevada variação nas dimensões (coeficiente de variação prox. 0,5).

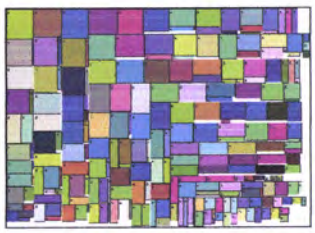
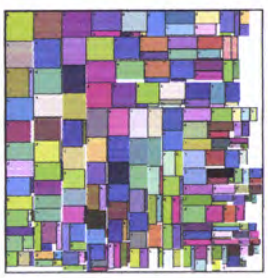
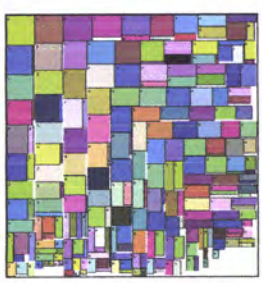
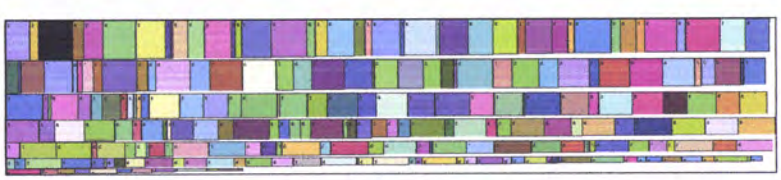
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	8	3,19
]0.1, 0.2]	23	9,16
]0.2, 0.3]	33	13,15
]0.3, 0.4]	27	10,76
]0.4, 0.5]	26	10,36
]0.5, 0.6]	23	9,16
]0.6, 0.7]	21	8,37
]0.7, 0.8]	30	11,95
]0.8, 0.9]	37	14,74
]0.9, 1]	23	9,16

Estatísticas de R	
Média	0,542228437
Erro-padrão	0,017067742
Mediana	0,526315789
Moda	0,5
Desvio-padrão	0,270403891
Variância da amostra	0,073118264
Curtose	-1,296036305
Assimetria	-0,019527903
Intervalo	0,948453608
Mínimo	0,051546392
Máximo	1
Coeficiente de variação	0,498689984
Soma das Áreas	672261
Contagem	251



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	803	920	738760	91
2DGEO	Menor diferença linear	696	1027	714792	94,05
2DGEO	Maior area	883	812	716996	93,76
2DGEO	Menor area	756	956	722736	93,02
2DGEO	Quadrado	840	871	731640	91,88
2DGEO	Rectângulo de ouro	1016	727	738632	91,01
2DGEO	Relação Média	856	870	744720	90,27
2DGEO	Distância à Origem	837	871	729027	92,21
Slick		2048	387	792576	84,82

Estratégia	Área (px)	Arranjo da população (dimensões a 15%, aprox.)
Maior diferença linear	738760	
Menor diferença linear	714792	
Maior área	716996	
Menor área	722736	
Quadrado	731640	

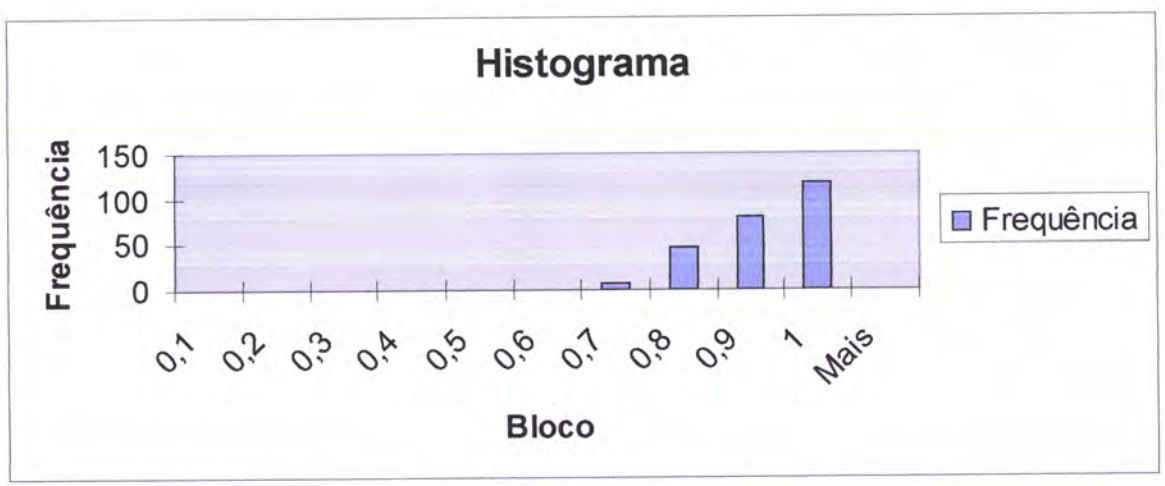
Estratégia	Área (px)	Arranjo da população (dimensões a 15%, aprox.)
Rectângulo de ouro	738632	
Relação Média	744720	
Distância à Origem	729027	
Slick	792576	

População 7

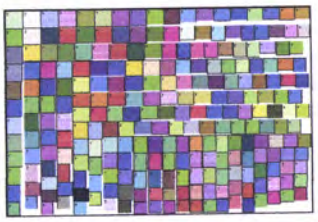
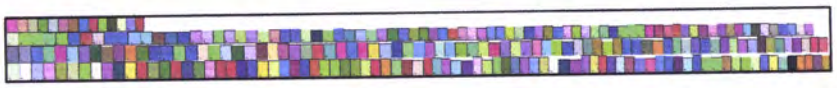
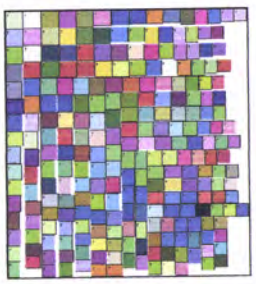
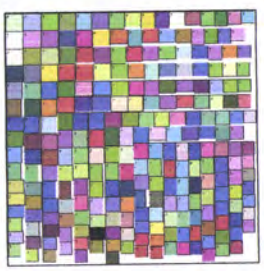
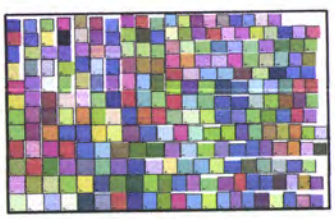
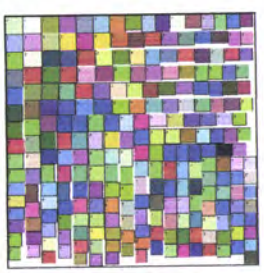
Composta por 251 fotogramas; existe, pelo menos, um quadrado na população (máximo=1); rectângulos de dimensões pouco variáveis (coeficiente de variação<0,1).

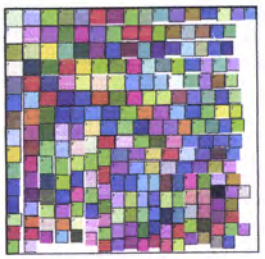
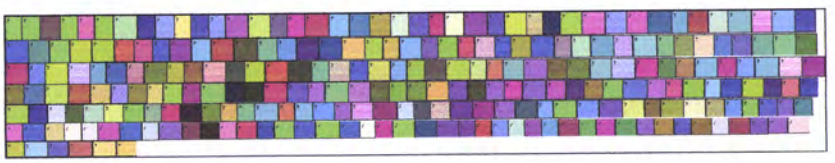
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	0	0
]0.1, 0.2]	0	0
]0.2, 0.3]	0	0
]0.3, 0.4]	0	0
]0.4, 0.5]	0	0
]0.5, 0.6]	0	0
]0.6, 0.7]	7	2,79
]0.7, 0.8]	46	18,33
]0.8, 0.9]	80	31,87
]0.9, 1]	118	47,01

Estatísticas de R	
Média	0,88174082
Erro-padrão	0,00531394
Mediana	0,896551724
Moda	1
Desvio-padrão	0,084188645
Variância da amostra	0,007087728
Curtose	-0,638083848
Assimetria	-0,527985008
Intervalo	0,333333333
Mínimo	0,666666667
Máximo	1
Coeficiente de variação	0,095480036
Soma das Áreas	622048
Contagem	251



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	1020	683	696660	89,29
2DGEO	Menor diferença linear	207	3973	822411	75,64
2DGEO	Maior area	811	888	720168	86,38
2DGEO	Menor area	60	12497	749820	82,96
2DGEO	Quadrado	838	838	702244	88,58
2DGEO	Rectângulo de ouro	653	1069	698057	89,11
2DGEO	Relação Média	833	835	695555	89,43
2DGEO	Distância à Origem	864	831	717984	86,64
Slick		2047	351	718497	86,58

Estratégia	Área (px)	Arranjo da população (dimensões a 15%, aprox.)
Maior diferença linear	696660	
Menor diferença linear	822411	
Maior area	720168	
Menor area	749820	Demasiada distorção. Atlas com 60×12497 píxeis (L×A)
Quadrado	702244	
Rectângulo de ouro	698057	
Relação Média	695555	

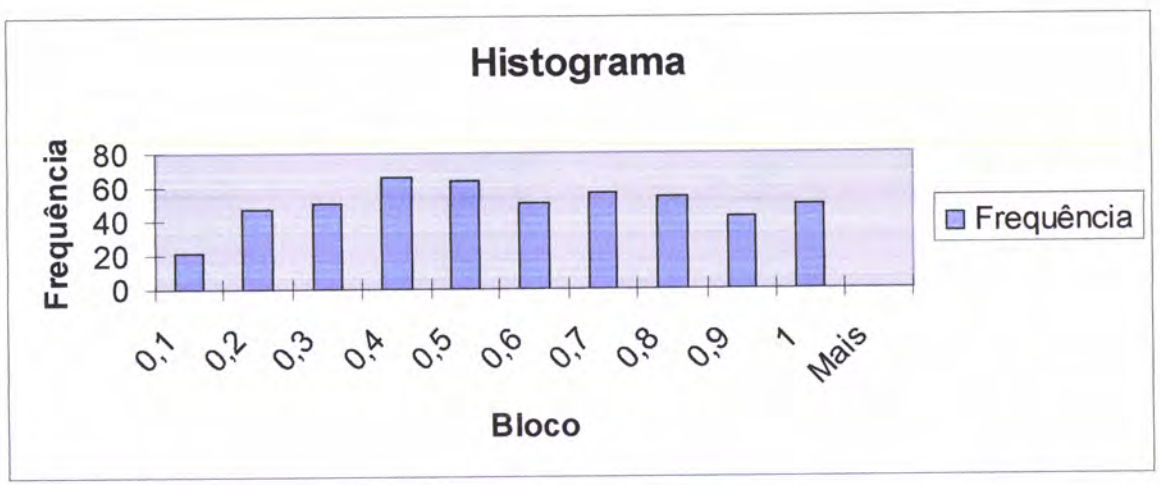
Estratégia	Área (px)	Arranjo da população (dimensões a 15%, aprox.)
Distância à Origem	717984	
Slick	718497	

População 8

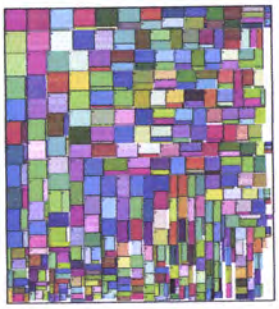
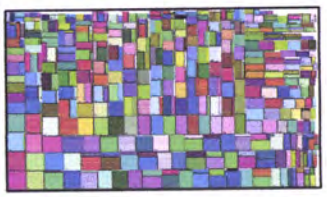
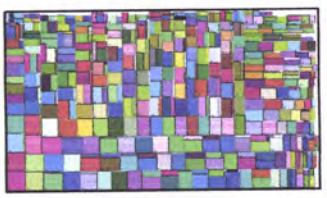
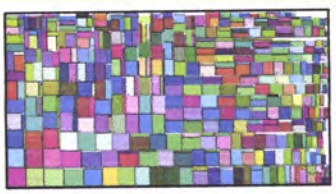
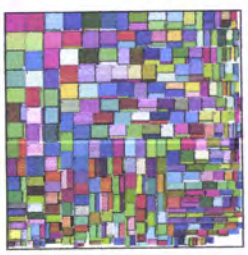
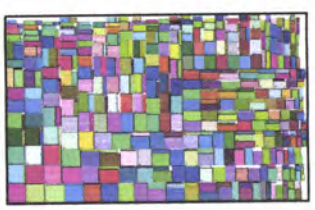
Composta por 501 fotogramas; existe, pelo menos, um quadrado na população (máximo=1); elevada variação nas dimensões (coeficiente de variação > 0,5).

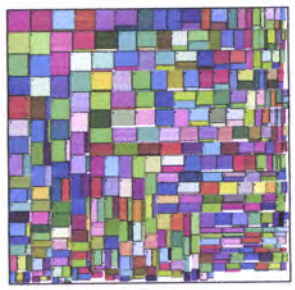
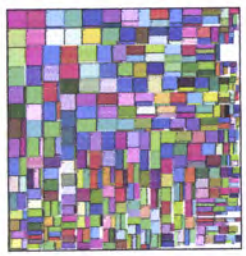
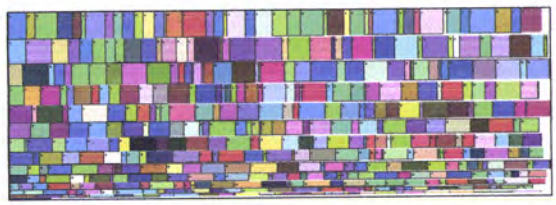
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	21	4,19
]0.1, 0.2]	47	9,38
]0.2, 0.3]	51	10,18
]0.3, 0.4]	66	13,17
]0.4, 0.5]	63	12,57
]0.5, 0.6]	51	10,18
]0.6, 0.7]	56	11,18
]0.7, 0.8]	54	10,78
]0.8, 0.9]	42	8,38
]0.9, 1]	50	9,98

Estatísticas de R	
Média	0,521603729
Erro-padrão	0,011740566
Mediana	0,510638298
Moda	1
Desvio-padrão	0,262789425
Variância da amostra	0,069058282
Curtose	-1,084601656
Assimetria	0,081089147
Intervalo	0,947916667
Mínimo	0,052083333
Máximo	1
Coeficiente de variação	0,503810481
Soma das Áreas	1310753
Contagem	501



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	1130	1243	1404590	93,32
2DGEO	Menor diferença linear	887	1564	1387268	94,48
2DGEO	Maior area	1054	1317	1388118	94,43
2DGEO	Menor area	864	1610	1391040	94,23
2DGEO	Quadrado	1173	1176	1379448	95,02
2DGEO	Rectângulo de ouro	928	1504	1395712	93,91
2DGEO	Relação Média	1199	1166	1398034	93,76
2DGEO	Distância à Origem	1159	1201	1391959	94,17
Slick		2033	700	1423100	92,11

Estratégia	Área (px)	Arranjo da população (dimensões a 10%, aprox.)
Maior diferença linear	1404590	
Menor diferença linear	1387268	
Maior area	1388118	
Menor area	1391040	
Quadrado	1379448	
Rectângulo de ouro	1395712	

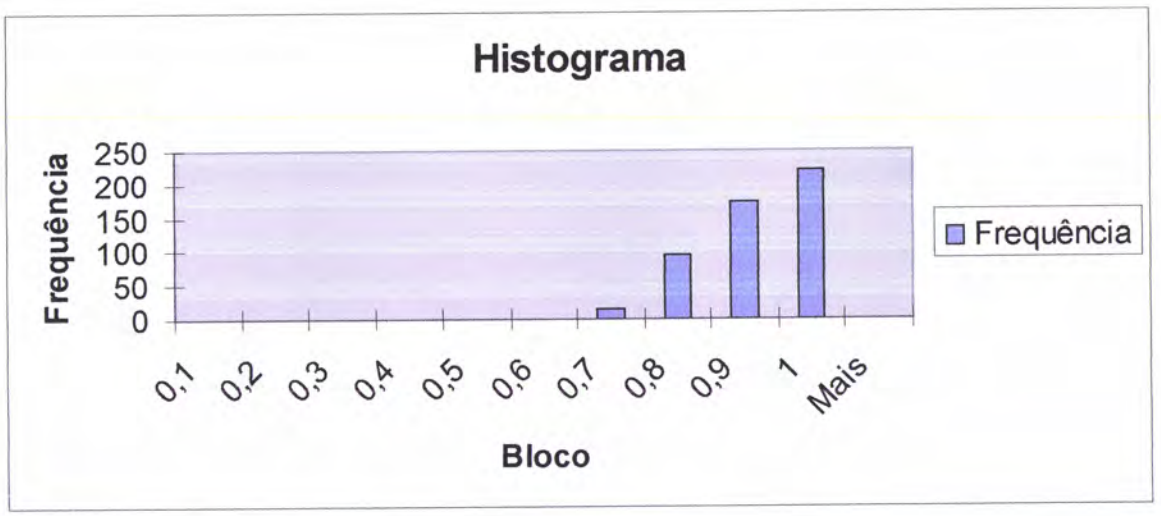
Estratégia	Área (px)	Arranjo da população (dimensões a 10%, aprox.)
Relação Média	1398034	
Distância à Origem	1391959	
Slick	1423100	

População 9

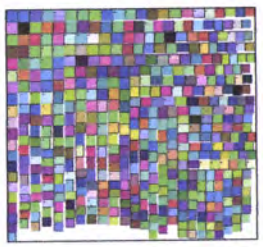
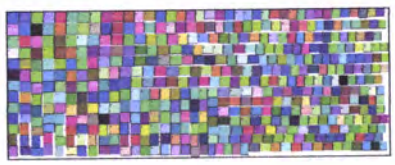
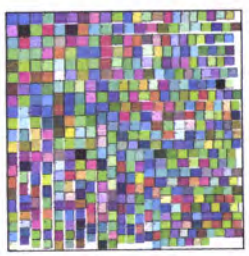
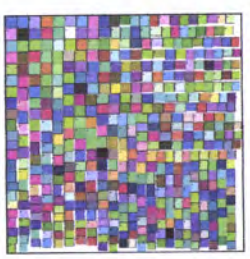
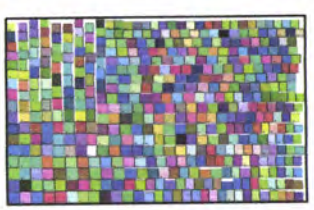
Composta por 501 fotogramas; existe, pelo menos, um quadrado na população (máximo=1); rectângulos de dimensões pouco variáveis (coeficiente de variação<0,1).

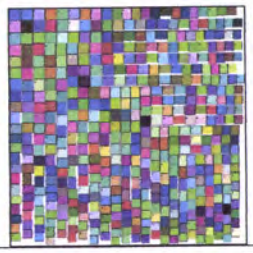
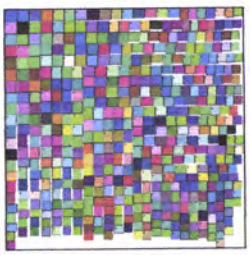
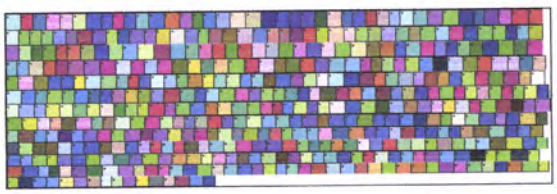
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	0	0
]0.1, 0.2]	0	0
]0.2, 0.3]	0	0
]0.3, 0.4]	0	0
]0.4, 0.5]	0	0
]0.5, 0.6]	0	0
]0.6, 0.7]	15	2,99
]0.7, 0.8]	94	18,76
]0.8, 0.9]	173	34,53
]0.9, 1]	219	43,71

Estatísticas de R	
Média	0,873898175
Erro-padrão	0,003746935
Mediana	0,886792453
Moda	1
Desvio-padrão	0,083867754
Variância da amostra	0,0070338
Curtose	-0,766875176
Assimetria	-0,425280339
Intervalo	0,333333333
Mínimo	0,666666667
Máximo	1
Coeficiente de variação	0,09596971
Soma das Áreas	1249180
Contagem	501



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	1248	1151	1436448	86,96
2DGEO	Menor diferença linear	1908	732	1396656	89,44
2DGEO	Maior area	1177	1186	1395922	89,49
2DGEO	Menor area	464	3051	1415664	88,24
2DGEO	Quadrado	1182	1182	1397124	89,41
2DGEO	Rectângulo de ouro	921	1493	1375053	90,85
2DGEO	Relação Média	1183	1184	1400672	89,18
2DGEO	Distância à Origem	1196	1197	1431612	87,26
Slick		2039	653	1331467	93,82

Estratégia	Área (px)	Arranjo da população (dimensões a 10%, aprox.)
Maior diferença linear	1436448	
Menor diferença linear	1396656	
Maior area	1395922	
Menor area	1415664	
Quadrado	1397124	
Rectângulo de ouro	1375053	

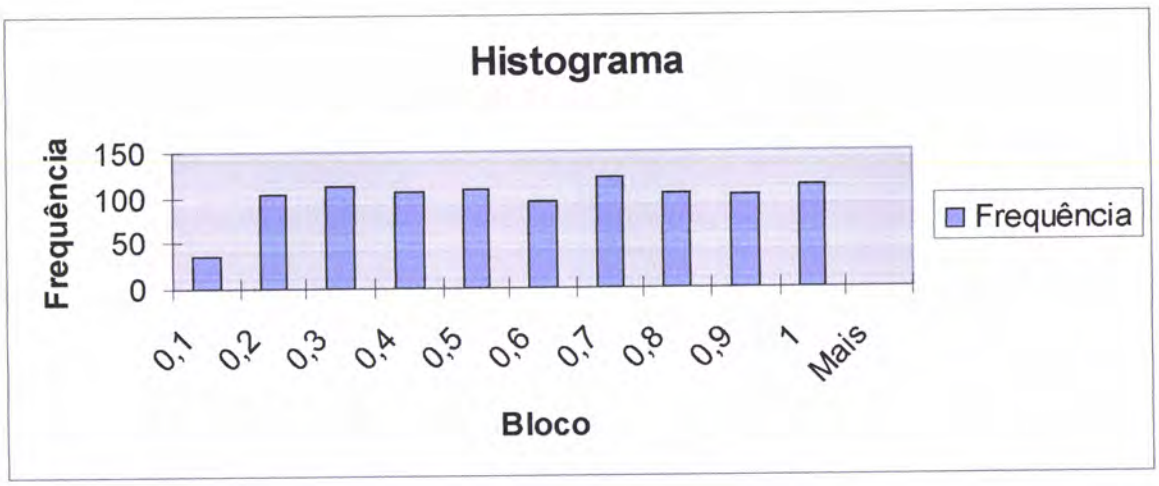
Estratégia	Área (px)	Arranjo da população (dimensões a 10%, aprox.)
Relação Média	1400672	
Distância à Origem	1431612	
Slick	1331467	

População 10

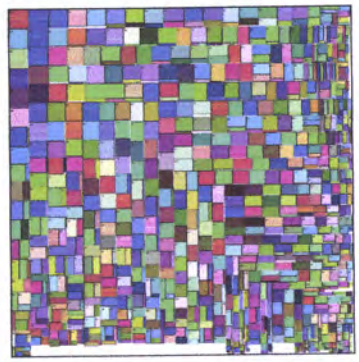
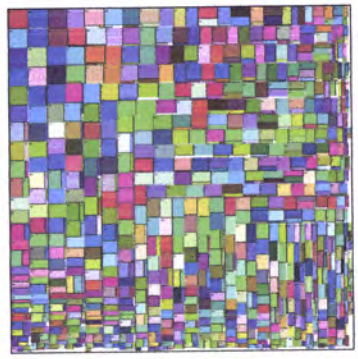
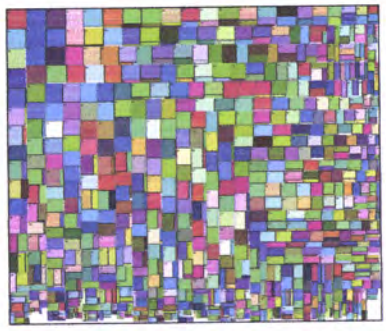
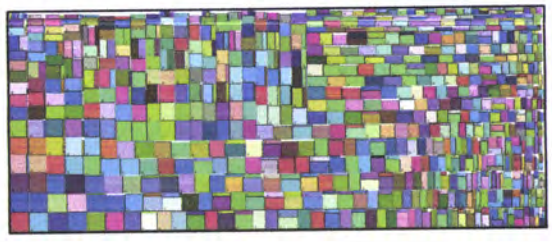
Composta por 1001 fotogramas; existe, pelo menos, um quadrado na população (máximo=1); elevada variação nas dimensões (coeficiente de variação > 0,5).

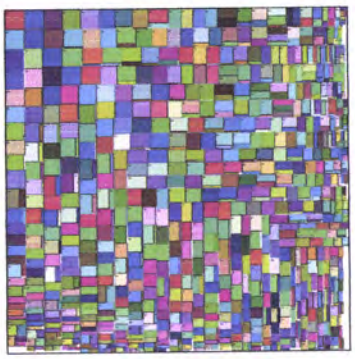
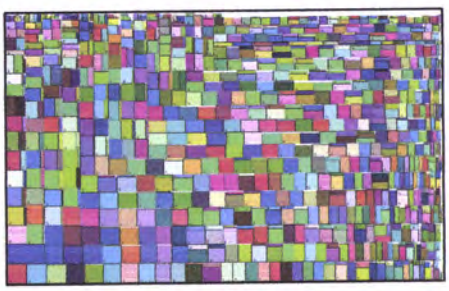
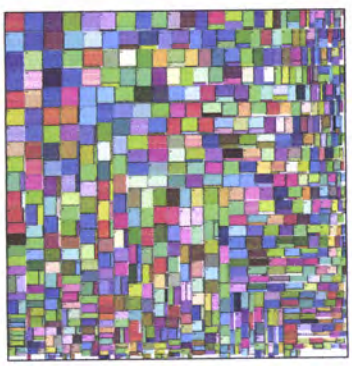
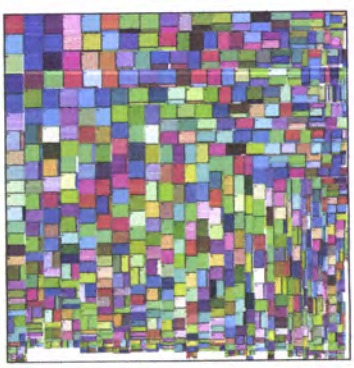
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	36	3,6
]0.1, 0.2]	104	10,39
]0.2, 0.3]	112	11,19
]0.3, 0.4]	106	10,59
]0.4, 0.5]	108	10,79
]0.5, 0.6]	95	9,49
]0.6, 0.7]	122	12,19
]0.7, 0.8]	104	10,39
]0.8, 0.9]	102	10,19
]0.9, 1]	112	11,19

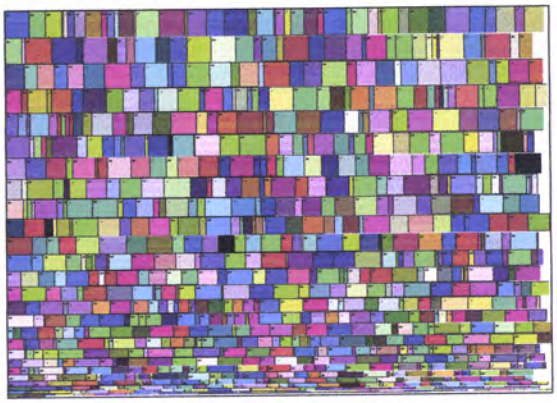
Estatísticas de R	
Média	0,53503785
Erro-padrão	0,008564537
Mediana	0,533333333
Moda	0,5
Desvio-padrão	0,270969825
Variância da amostra	0,073424646
Curtose	-1,224747287
Assimetria	-0,000904295
Intervalo	0,949494949
Mínimo	0,050505051
Máximo	1
Coeficiente de variação	0,506449824
Soma das Áreas	2815037
Contagem	1001



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	1714	1734	2972076	94,72
2DGEO	Menor diferença linear	1712	1718	2941216	95,71
2DGEO	Maior area	1860	1567	2914620	96,58
2DGEO	Menor area	1096	2683	2940568	95,73
2DGEO	Quadrado	1712	1712	2930944	96,05
2DGEO	Rectângulo de ouro	1351	2185	2951935	95,36
2DGEO	Relação Média	1695	1732	2935740	95,89
2DGEO	Distância à Origem	1709	1735	2965115	94,94
Slick		2044	1446	2955624	95,24

Estratégia	Área (px)	Arranjo da população (dimensões a 10%, aprox.)
Maior diferença linear	2972076	
Menor diferença linear	2941216	
Maior area	2914620	
Menor area	2940568	

Estratégia	Área (px)	Arranjo da população (dimensões a 10%, aprox.)
Quadrado	2930944	
Rectângulo de ouro	2951935	
Relação Média	2935740	
Distância à Origem	2965115	

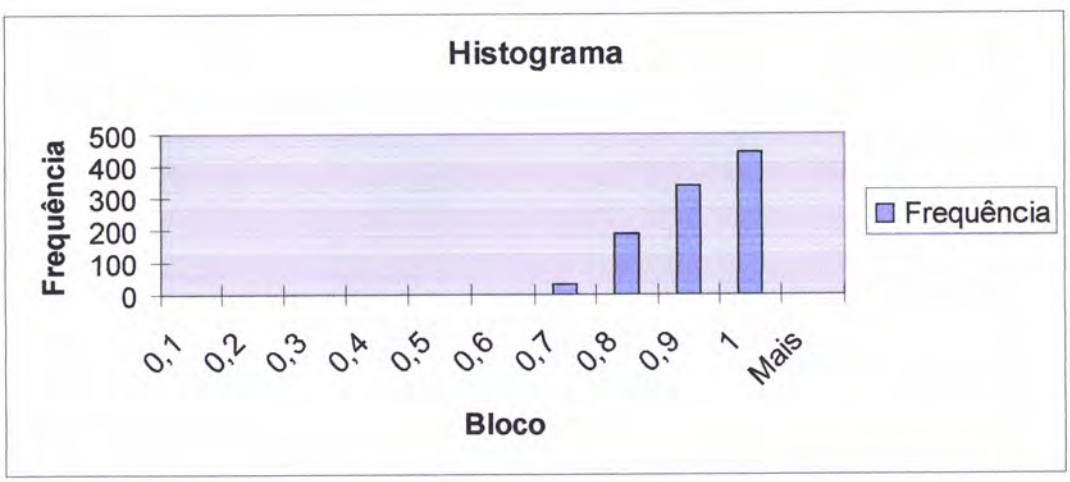
Estratégia	Área (px)	Arranjo da população (dimensões a 10%, aprox.)
Slick	2955624	

População 11

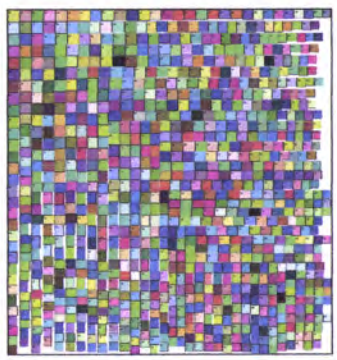
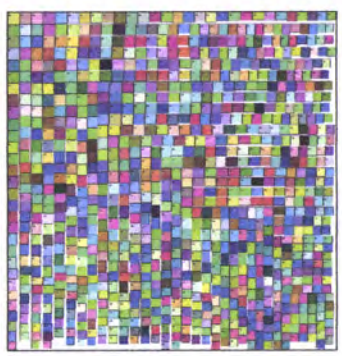
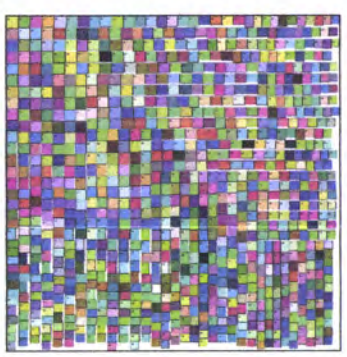
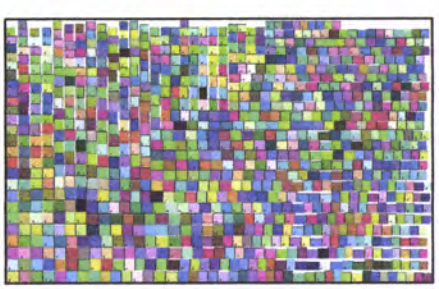
Composta por 1001 fotogramas; existe, pelo menos, um quadrado na população (máximo=1); rectângulos de dimensões pouco variáveis (coeficiente de variação<0,1).

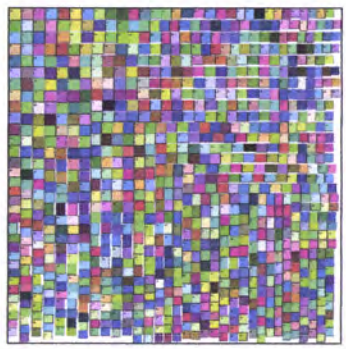
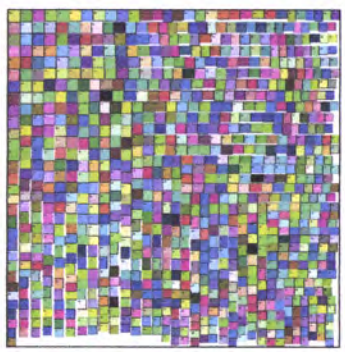
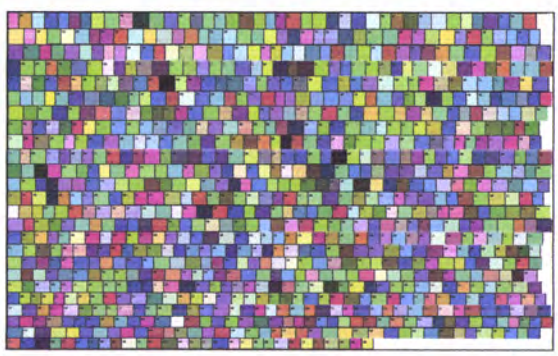
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	0	0
]0.1, 0.2]	0	0
]0.2, 0.3]	0	0
]0.3, 0.4]	0	0
]0.4, 0.5]	0	0
]0.5, 0.6]	0	0
]0.6, 0.7]	33	3,3
]0.7, 0.8]	190	18,98
]0.8, 0.9]	335	33,47
]0.9, 1]	443	44,26

Estatísticas de R	
Média	0,874201855
Erro-padrão	0,002752669
Mediana	0,886792453
Moda	1
Desvio-padrão	0,087090548
Variância da amostra	0,007584764
Curtose	-0,819383381
Assimetria	-0,420079661
Intervalo	0,333333333
Mínimo	0,666666667
Máximo	1
Coeficiente de variação	0,099622927
Soma das Áreas	2504637
Contagem	1001



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	1624	1720	2793280	89,67
2DGEO	Menor diferença linear	216	12625	2727000	91,85
2DGEO	Maior area	1649	1688	2783512	89,98
2DGEO	Menor area	60	50185	3011100	83,18
2DGEO	Quadrado	1672	1672	2795584	89,59
2DGEO	Rectângulo de ouro	1312	2124	2786688	89,88
2DGEO	Relação Média	1666	1671	2783886	89,97
2DGEO	Distância à Origem	1660	1679	2787140	89,86
Slick		2047	1259	2577173	97,19

Estratégia	Área (px)	Arranjo da população (dimensões a 10%, aprox.)
Maior diferença linear	2793280	
Menor diferença linear	2727000	Demasiada distorção. Atlas com 216×12625 píxeis (L×A)
Maior area	2783512	
Menor area	3011100	Demasiada distorção. Atlas com 60×50185 píxeis (L×A)
Quadrado	2795584	
Rectângulo de ouro	2786688	

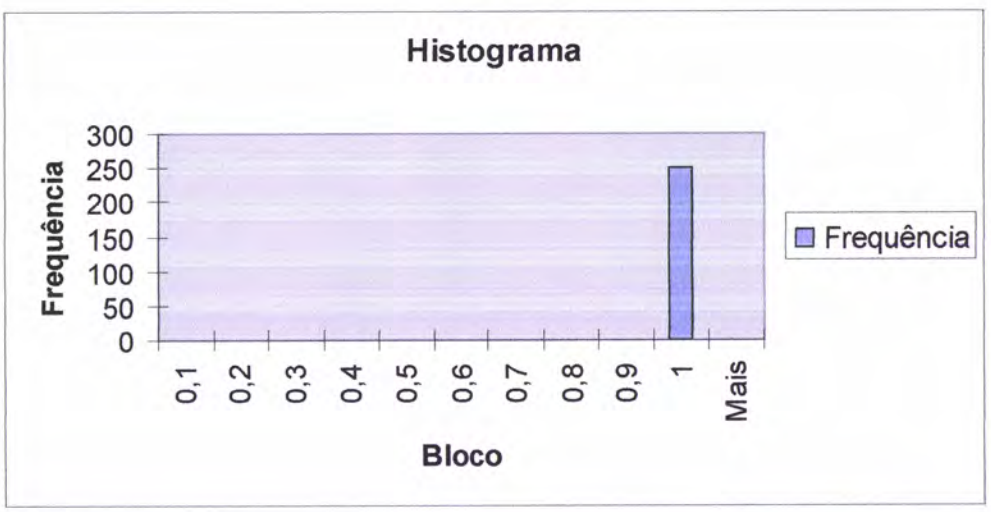
Estratégia	Área (px)	Arranjo da população (dimensões a 10%, aprox.)
Relação Média	2783886	
Distância à Origem	2787140	
Slick	2577173	

População 12

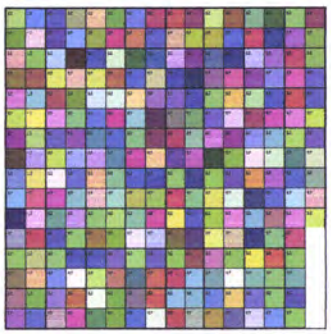
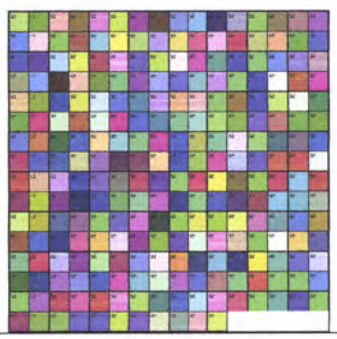
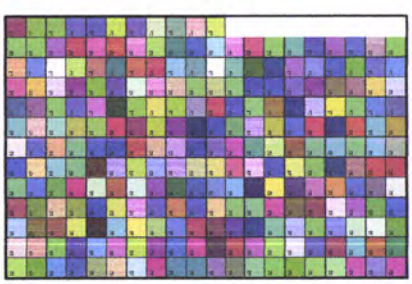
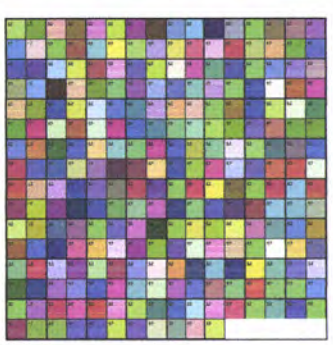
Composta por 251 fotogramas; constituída inteiramente por quadrados.

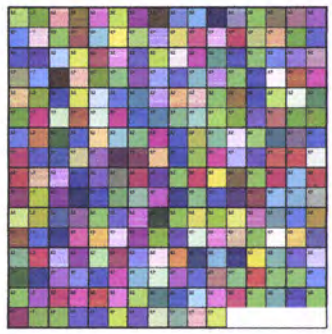
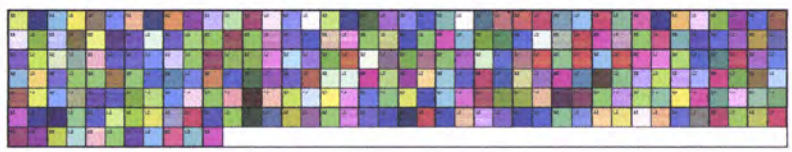
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	0	0
]0.1, 0.2]	0	0
]0.2, 0.3]	0	0
]0.3, 0.4]	0	0
]0.4, 0.5]	0	0
]0.5, 0.6]	0	0
]0.6, 0.7]	0	0
]0.7, 0.8]	0	0
]0.8, 0.9]	0	0
]0.9, 1]	251	100

Estatísticas de R	
Média	1
Erro-padrão	0
Mediana	1
Moda	1
Desvio-padrão	0
Variância da amostra	0
Curtose	---
Assimetria	---
Intervalo	0
Mínimo	1
Máximo	1
Coeficiente de variação	0
Soma das Áreas	627500
Contagem	251



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	50	12550	627500	100
2DGEO	Menor diferença linear	50	12550	627500	100
2DGEO	Maior área	800	800	640000	98,05
2DGEO	Menor área	50	12550	627500	100
2DGEO	Quadrado	800	800	640000	98,05
2DGEO	Rectângulo de ouro	650	1000	650000	96,54
2DGEO	Relação Média	800	800	640000	98,05
2DGEO	Distância à Origem	800	800	640000	98,05
Slick		2000	350	700000	89,64

Estratégia	Área (px)	Arranjo da população (dimensões a 20%, aprox.)
Maior diferença linear	627500	Demasiada distorção. Atlas com 50×12550 píxeis (L×A)
Menor diferença linear	627500	Demasiada distorção. Atlas com 50×12550 píxeis (L×A)
Maior area	640000	
Menor area	627500	Demasiada distorção. Atlas com 50×12550 píxeis (L×A)
Quadrado	640000	
Rectângulo de ouro	650000	
Relação Média	640000	

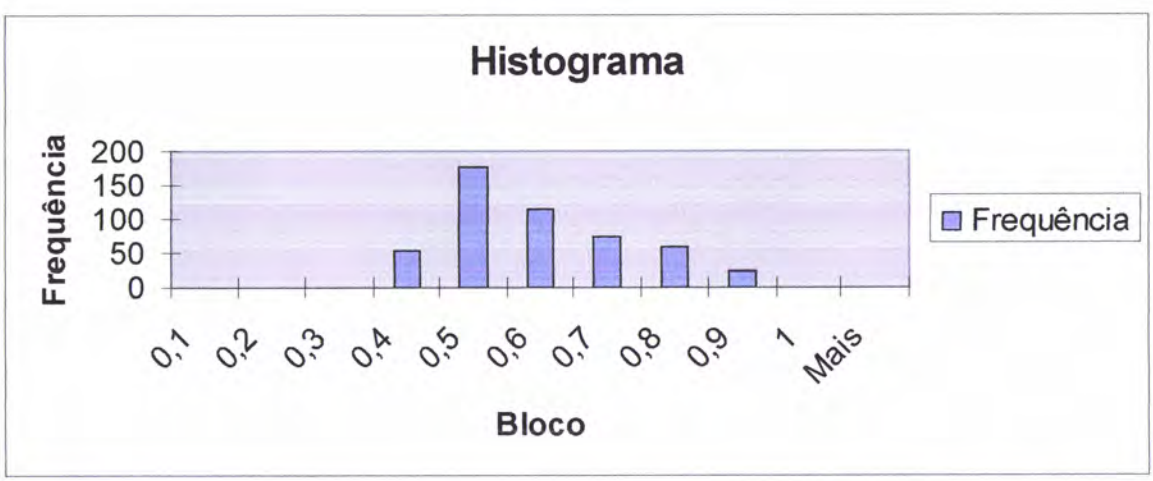
Estratégia	Área (px)	Arranjo da população (dimensões a 20%, aprox.)
Distância à Origem	640000	
Slick	700000	

População 13

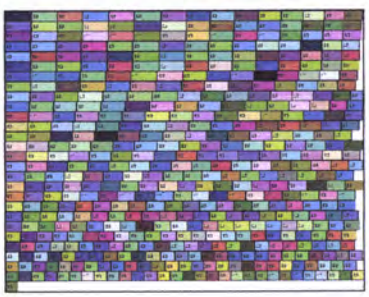
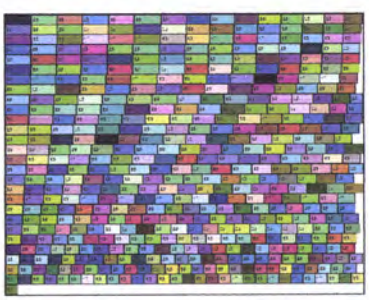
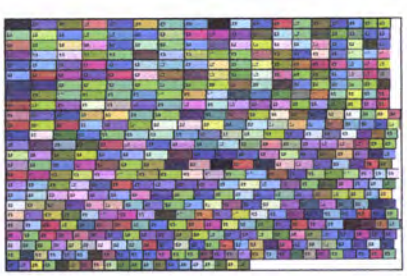
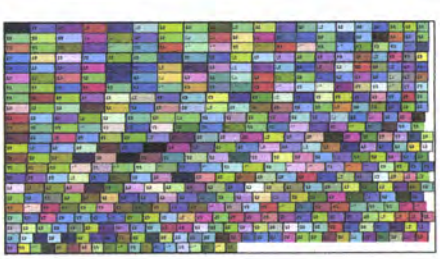
Composta por 501 fotogramas; não existe nenhum quadrado na população (máximo!=1).

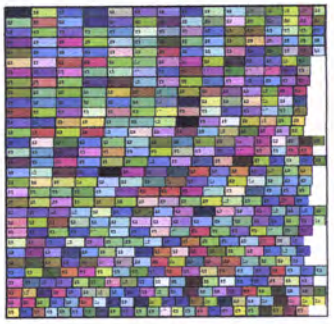
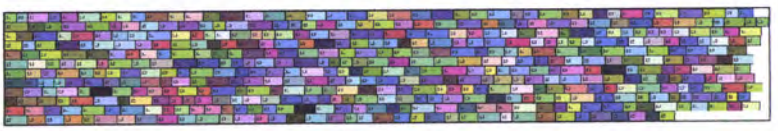
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	0	0
]0.1, 0.2]	0	0
]0.2, 0.3]	0	0
]0.3, 0.4]	52	10,38
]0.4, 0.5]	175	34,93
]0.5, 0.6]	116	23,15
]0.6, 0.7]	73	14,57
]0.7, 0.8]	60	11,98
]0.8, 0.9]	25	4,99
]0.9, 1]	0	0

Estatísticas de R	
Média	0,550547222
Erro-padrão	0,005792287
Mediana	0,520833333
Moda	0,390625
Desvio-padrão	0,129648926
Variância da amostra	0,016808844
Curtose	-0,809651918
Assimetria	0,595273141
Intervalo	0,448717949
Mínimo	0,384615385
Máximo	0,833333333
Coeficiente de variação	0,235491019
Soma das Áreas	599000
Contagem	501



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	23960	25	599000	100
2DGEO	Menor diferença linear	91	7225	657475	91,11
2DGEO	Maior area	896	700	627200	95,5
2DGEO	Menor area	23960	25	599000	100
2DGEO	Quadrado	781	775	605275	98,96
2DGEO	Rectângulo de ouro	990	625	618750	96,81
2DGEO	Relação Média	1076	575	618700	96,82
2DGEO	Distância à Origem	800	775	620000	96,61
Slick		2048	300	614400	97,49

Estratégia	Área (px)	Arranjo da população (dimensões a 20%, aprox.)
Maior diferença linear	599000	Demasiada distorção. Atlas com 23960×25 píxeis (L×A)
Menor diferença linear	657475	Demasiada distorção. Atlas com 91×7225 píxeis (L×A)
Maior area	627200	
Menor area	599000	Demasiada distorção. Atlas com 23960×25 píxeis (L×A)
Quadrado	605275	
Rectângulo de ouro	618750	
Relação Média	618700	

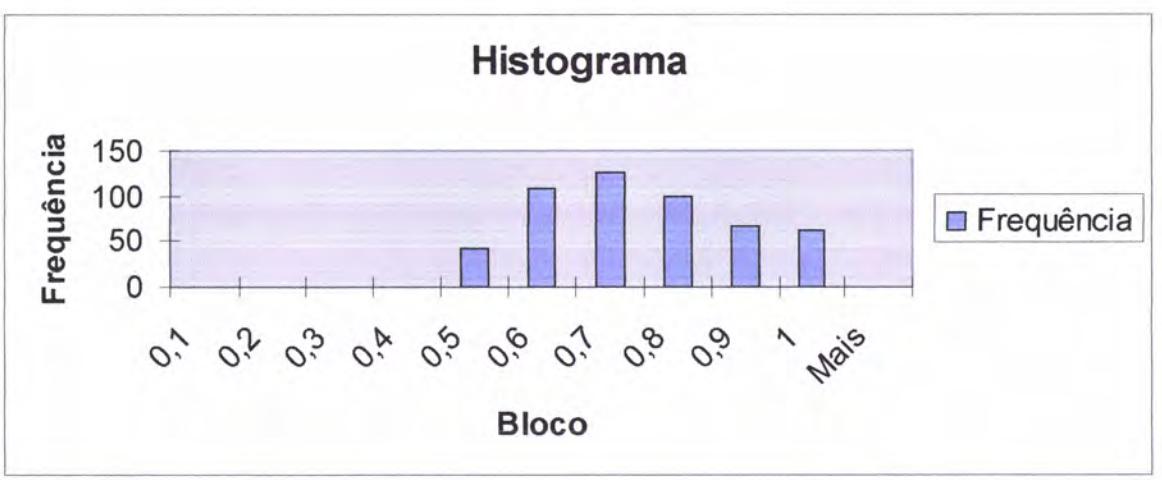
Estratégia	Área (px)	Arranjo da população (dimensões a 20%, aprox.)
Distância à Origem	620000	
Slick	614400	

População 14

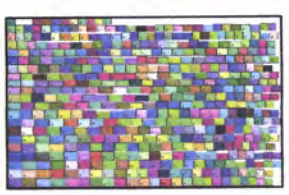
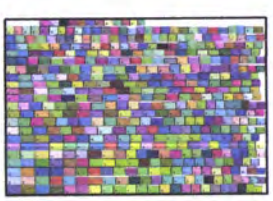
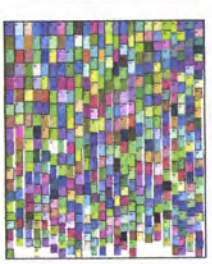
Composta por 501 fotogramas; existe, pelo menos, um fotograma quadrado na população (máximo=1).

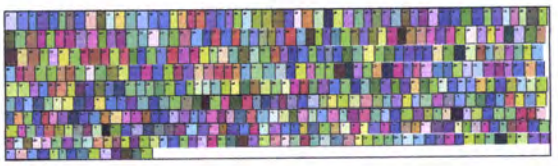
Intervalos de R	*Frequência*	*Frequência Relativa*
]0, 0.1]	0	0
]0.1, 0.2]	0	0
]0.2, 0.3]	0	0
]0.3, 0.4]	0	0
]0.4, 0.5]	41	8,18
]0.5, 0.6]	108	21,56
]0.6, 0.7]	125	24,95
]0.7, 0.8]	99	19,76
]0.8, 0.9]	66	13,17
]0.9, 1]	62	12,38

Estatísticas de R	
Média	0,699170292
Erro-padrão	0,006458388
Mediana	0,68
Moda	0,666666667
Desvio-padrão	0,144558287
Variância da amostra	0,020897098
Curtose	-0,847766752
Assimetria	0,28208939
Intervalo	0,571428571
Mínimo	0,428571429
Máximo	1
Coeficiente de variação	0,206756907
Soma das Áreas	1034397
Contagem	501



Aplicação	Estratégia	Largura (px)	Altura (px)	Área Total (px)	% Ocupação
2DGEO	Maior diferença linear	135	18626	2514510	41,14
2DGEO	Menor diferença linear	3676	326	1198376	86,32
2DGEO	Maior area	1091	1137	1240467	83,39
2DGEO	Menor area	45	27568	1240560	83,38
2DGEO	Quadrado	1074	1067	1145958	90,26
2DGEO	Rectângulo de ouro	840	1375	1155000	89,56
2DGEO	Relação Média	872	1301	1134472	91,18
2DGEO	Distância à Origem	1000	1180	1180000	87,66
Slick		2042	554	1131268	91,44

Estratégia	Área (px)	Arranjo da população (dimensões a 10%, aprox.)
Maior diferença linear	2514510	Demasiada distorção. Atlas com 18626×135 píxeis (L×A)
Menor diferença linear	1198376	
Maior area	1240467	
Menor area	1240560	Demasiada distorção. Atlas com 45×27568 píxeis (L×A)
Quadrado	1145958	
Rectângulo de ouro	1155000	
Relação Média	1134472	
Distância à Origem	1180000	

Estratégia	Área (px)	Arranjo da população (dimensões a 10%, aprox.)
Slick	1131268	

Anexo II

Exemplo de uma Strip.