Ferramenta para Balanceamento de Jogos

Aluno	Antonio Vildes Barbosa	{avb@cin.ufpe.br}
Orientador	Geber Lisboa Ramalho	{glr@cin.ufpe.br}

RECIFE,

SETEMBRO/2013

Resumo

O balanceamento de jogos digitais é uma parte importante do processo de desenvolvimento e é comum a todos os jogos. É uma atividade muito importante pois o sucesso do jogo está diretamente relacionado a um bom balanceamento. Esta atividade consiste em garantir que o jogo é divertido e difícil nas medidas necessárias para fazer com que o jogador se interesse e mantenha este interesse pelo jogo. Este trabalho tem como objetivo realizar uma extensão de um framework para facilitar no balanceamento, e em consequência, o desenvolvimento de jogos. Este deve ser simples e prover uma abstração grande o suficiente para que o game designer, que fará o balanceamento, seja capaz de realizar esta tarefa sem a ajuda de um desenvolvedor. O framework escolhido para a extensão é chamado *Xpose* e foi desenvolvido por Renan Lima na Universidade Federal de Pernambuco. Ele fornece ao game designer um ambiente que torna possível testar e modificar as variáveis em tempo real. O objetivo da extensão descrita neste trabalho é resolver um subproblema gerado pelo uso do *framework* em jogos que possuem uma grande quantidade de variáveis a ser balanceadas. Este tipo de ferramenta se faz necessária pois a etapa de balanceamento é bastante lenta devido a esta grande quantidade de interações realizadas entre o desenvolvedor e o game designer.

Palavras-chave: *Xpose*, Extensão, Balanceamento, Balanceamento Manual, *Framework*, Jogos, *Game Designer*, GDD.

Abstract

Game balancing is a important part of the game development process and it's common to every game project. This activity is very important because the success of a game is directly related to a good game balancing. It consists in making sure that the game is fun and hard enough to make the player want and keep playing the game. This work's objective is to extend a framework that makes the game balancing more efficient, and in consequence, the game development itself. It must be simple and provide a sufficiently big abstraction to make sure that the game designer, which will do the balancing, be capable of doing it without the aid of a game programmer. The chosen framework for the extension is called Xpose and was developed by Renan Lima in the Federal University of Pernambuco. It provides an enviroment that makes possible to the game designer to change and test the game variables in real time. This extension's objective is to solve an issue created by the use of this framework in games that have a big quantity of variables to balance. This kind of framework is necessary because the game balancing activity is very slow given the great number of interactions between the game designer and the game programmer during the game development.

Keywords: Xpose, Extension, Balancing, Manual Balancing, Framework, Games, Game Designer, GDD.

Agradecimentos

À minha família, meu pai, minha mãe e meus irmãos, que com muito amor e carinho tiveram influência direta na minha formação pessoal,

À Manuela Muller de Andrade, minha namorada, que me faz ser uma pessoa feliz e me dá a coragem necessária para lutar pelos meus objetivos acima de todas as dificuldades,

À família de Manuela, que sempre me recebe com muito carinho e que, junto com a minha família, fazem meus dias mais felizes,

À todos os amigos, que tiveram influencia na minha formação, seja direta ou indireta, e que sempre estiveram a disposição para ajudar, mesmo nos momentos difíceis,

À Geber Ramalho, meu orientador, e a Renan Lima, que me orientaram durante o desenvolvimento deste trabalho,

“Imagination is more important than knowledge. For knowledge is limited to all we now know and understand, while imagination embraces the entire world, and all there ever will be to know and understand.”
— Albert Einstein

Índice

Lista de Figuras

1. Introdução

No início do mercado de jogos digitais, as equipes de desenvolvimento envolviam apenas programadores. Os jogos tinham escopos pequenos e eram frutos da imaginação deles e por isso quase não havia documentos de especificação ou sequer alguma preocupação com o processo de desenvolvimento. E de fato não seriam necessários visto que com uma equipe muito pequena, não muito mais que dois programadores, os problemas de comunicação entre eles eram resolvidos muito facilmente. “Por mais de uma década, os programadores de jogos ficaram trabalhando em um ambiente onde o código mais rápido e o de menor tamanho eram o principal objetivo, e onde o programador tinha total controle de todo o código que estava rodando na máquina” (ROCHA, 2003).

O avanço da tecnologia teve grande impacto no processo de desenvolvimento de jogos. Hoje são necessários diversos profissionais de diferentes áreas para a produção de um jogo, em contraste com os primeiros jogos produzidos apenas por programadores. Segundo (ROCHA, 2003), “mesmo para os jogos mais simples, o desenvolvimento dessas aplicações envolve a utilização de conceitos de várias áreas da Ciência da Computação, tais como Engenharia de Software, Computação Gráfica, Inteligência Artificial, Redes de Computadores e Computação Musical. Além disso, outras áreas como Educação, Psicologia, Artes e Estratégia Militar, também são envolvidas”.

Com isso, é esperado da equipe de desenvolvimento do jogo uma produção de qualidade em prazos cada vez mais curtos. Para atingir esses prazos é preciso otimizar os processos de desenvolvimento a partir do uso de métodos e ferramentas destinadas a aumentar a qualidade e reduzir ao máximo as dependências entre esses profissionais.

Segundo (BRATHWAITE & SCHREIBER, 2009), “na atual indústria de jogos, o papel preponderante no desenvolvimento é o do *game designer* (GD), que é o profissional responsável pela idealização e concepção dos jogos”. Como estes projetos envolvem grandes equipes, é parte do trabalho do *game designer* desenvolver um documento, conhecido como *Game Design Document* (GDD), que tem o objetivo de descrever da melhor maneira possível o projeto que será

desenvolvido. Durante a criação do GDD, o game designer especifica as principais entidades que farão parte do jogo, assim como quais as suas características e atributos.

Segundo (LIMA, 2013) “essas definições iniciais feitas pelo *game designer* no GDD tendem a sofrer mudanças durante o desenvolvimento do jogo”. Estas acontecem da seguinte forma: o *game designer* testa o jogo utilizando uma certa configuração das variáveis do jogo e sugere alterações nela, o programador, por sua vez, realiza essas alterações no código e permite que o *game designer* realize novos testes. Este ciclo continua até que o *game designer* encontre uma configuração que chegue próximo das características presentes no GDD. Ainda segundo o mesmo autor, “praticamente todo jogo digital, após desenvolvido ou nas fases finais de desenvolvimento, passa pelo período de ajustes dos atributos das entidades descrito acima, essa fase é chamada de balanceamento”.

O balanceamento é importante para deixar o jogo o mais próximo possível do que o desejado pelo *game designer*. Segundo (ANDRADE, 2006), “balancear consiste em criar mecanismos que desafiem adequadamente os jogadores, evitando entediá-los com tarefas triviais ou frustrá-los com tarefas intransponíveis”.

Em alguns jogos são lançados pacotes de atualização (*patch*) que agem para corrigir problemas de balanceamento e *bugs* que podem ter passados despercebidos pelas fases de testes que acontecem antes do lançamento do jogo. Um exemplo disso é o caso do jogo *League of Legends,* que após o seu lançamento continua desenvolvendo estes pacotes. Neste caso, um destes teve o objetivo de ajustar os atributos de um personagem chamado Talon. Seu dano de ataque base estava muito elevado e seu golpe especial gerava um bônus de ataque após o uso. Um dos *patchs* corrigiu este erro reduzindo levemente o valor atribuído a estes parâmetros.

“O balanceamento é uma atividade difícil e decisiva para o sucesso de um jogo, um jogo com uma boa ideia, mas com um mau balanceamento pode está fadado ao fracasso” (LIMA, 2013).

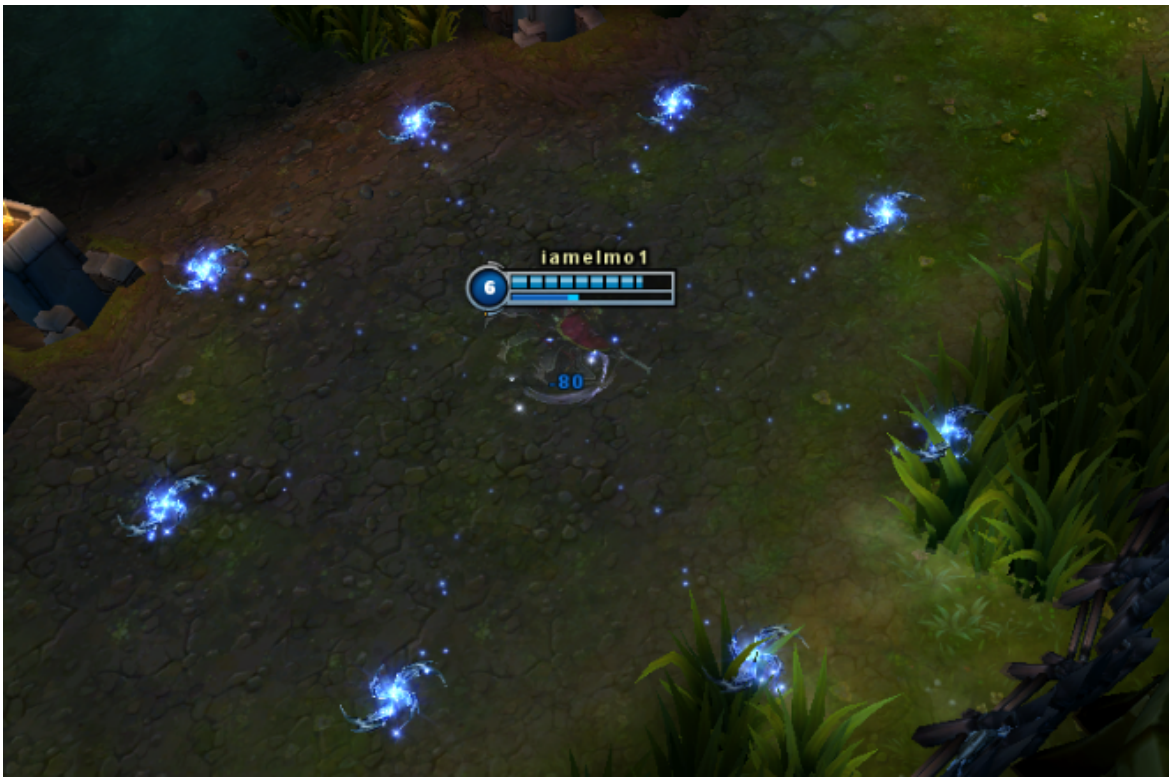

Figura 1.1 – Imagem do jogo *League of Legends* mostrando o personagem *Talon.*
Fonte: http://www.youtube.com/watch?v=drNRCr8KTPA

1.1 Motivação

É muito comum que para a realização da etapa de balanceamento o *game designer* dependa de um programador que execute a mudança de fato no código do jogo e, em projetos maiores, pode ocorrer de depender de mais de um (LIMA, 2013). Esta dependência vem do fato de serem necessárias alterações dos atributos no código para a realização das alterações propostas pelo *game designer.* E não é da competência dos game designers, pelo menos em sua maioria, entender nuances de programação.

Por ser realizada da maneira descrita na seção anterior esta etapa exige bastante tempo, tanto do programador quando do *game designer*. O motivo de ser necessário tanto tempo é justamente o fato de ser preciso uma comunicação intensa entre programadores e *game designers*. Este tempo é necessário por culpa de diversos problemas, dentre eles: distancia entre os profissionais, pois nem sempre os dois trabalham na mesma cidade ou unidade do estúdio, falta de tempo do programador, que teria de parar de realizar algumas de suas atividades para realizar

esta, o desencontro de horários, pois nem sempre os dois profissionais trabalham na mesma faixa de horário do dia.

Outro problema que afeta o balanceamento e é decorrente da forma com a qual o mesmo é realizado atualmente é o fato de que o *game designer* tem o seu tempo de experimentação limitado ao tempo que ele passa com o programador. Segundo (LIMA, 2013), “a cada momento que o game designer deseja experimentar uma nova configuração ele terá que recorrer a outra pessoa para implementá-la, com isso, algumas experimentações são deixadas de lado em razão da falta de praticidade para mudar uma característica simples do jogo”. Caso tivesse mais tempo e menos preocupações para realizar tal tarefa, o game designer seria capaz de experimentar mais e com isso testar configurações cada vez mais próximas do que foi planejado no GDD.

Dos trabalhos que tentam sanar este problema, foram levados em consideração três que apresentam uma solução próxima do ideal para o problema. A primeira delas é o projeto *Frogatto* que é um jogo de plataforma de código aberto e que possui com um módulo especial que permite a edição de *levels* e dos atributos dos elementos presentes nele. Porém, para realizar estas alterações é necessário o uso de uma linguagem de *script*. Apesar de facilitar bastante o processo de balanceamento, este fato faz com que ela esteja longe de tornar o *game designer* independente de um programador.

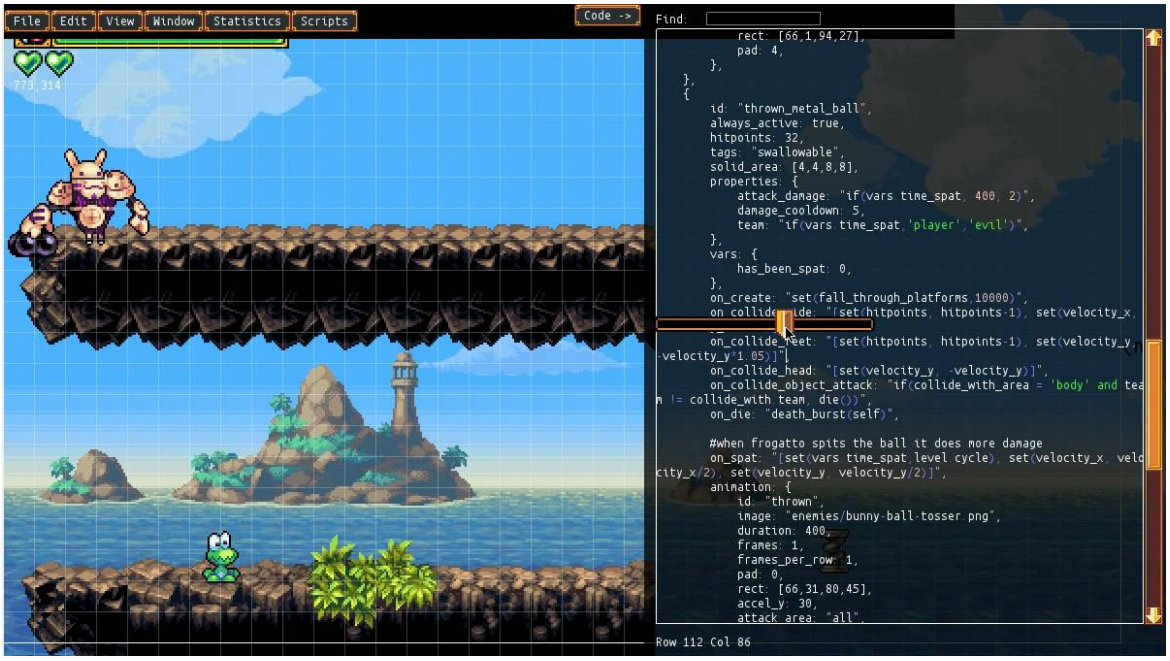

Figura 1.2 – Tela de edição do Frogatto.
Fonte:www.frogatto.com

O segundo projeto é o projeto de Bret Victor (VICTOR, 2010) que possibilita que o usuário altere os atributos, referentes aos elementos presentes no jogo, diretamente no código e a partir daí ver os efeitos dessas mudanças em tempo real como pode ser visto na figura 1.3. Esta forma interativa é uma abordagem bastante interessante para o problema, mas tanto a abordagem do *Frogatto*, quanto a de Bret Victor, entre outras abordagens encontradas para manipulação de atributos das entidades dentro de um jogo, necessitam que o manipulador tenha algum conhecimento de programação, o que não é o caso da maioria dos game designers, que são os reais responsáveis pela atividade de balanceamento (LIMA, 2013).

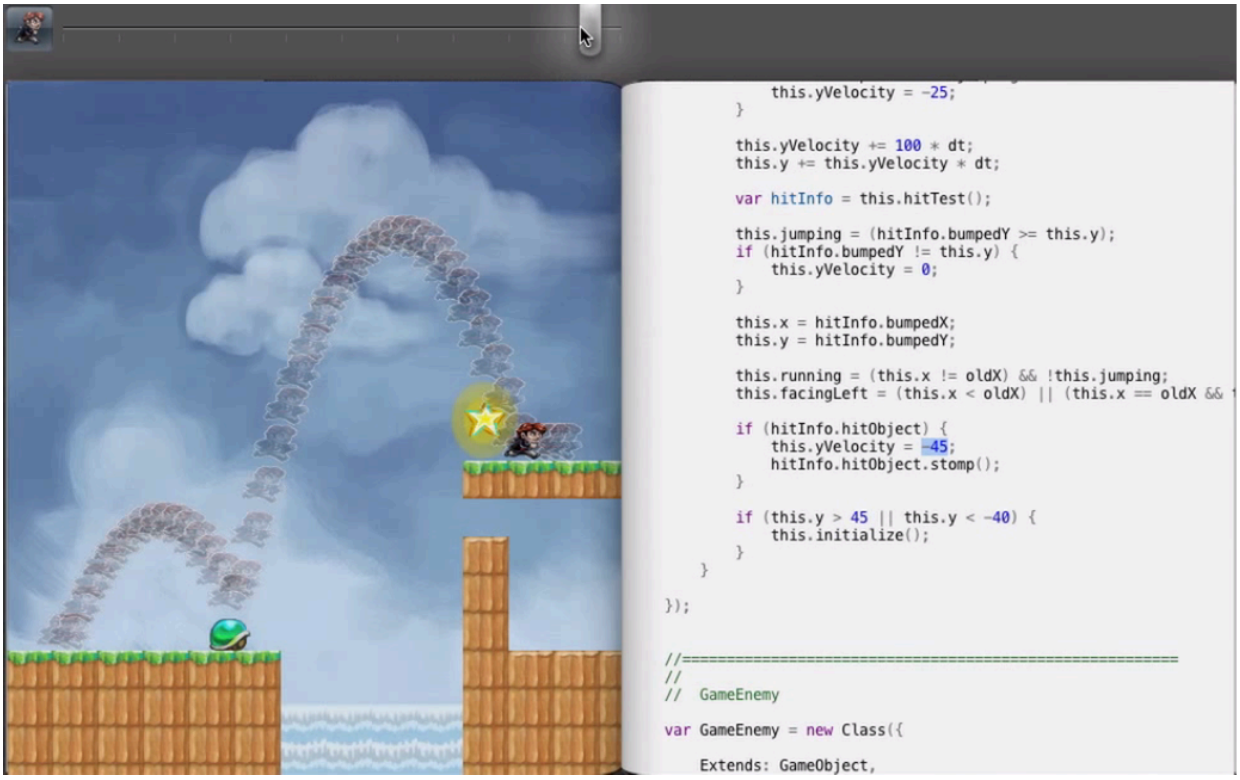

Figura 1.3 – Ferramenta de Bret Victor com as mudanças de atributos sendo refletidas em tempo real.
Fonte: http://www.youtube.com/watch?v=PII-gPu3SPI

O terceiro projeto é chamado *Xpose* e foi desenvolvido por Renan Lima na Universidade Federal de Pernambuco (UFPE). Este foi o projeto escolhido para a realização deste trabalho. Um dos critérios usados para a escolha deste foi o fato dele

propor uma abordagem que reduz bastante tanto o trabalho do programador, de modificar o jogo para usar o *framework*, quanto o do *game designer*, de realizar o balanceamento. Além disso, por natureza, o *Xpose* atende as características de fornecer ao *game designer* a possibilidade de realizar o balanceamento sem a necessidade de entrar em contato direto com o código fonte do jogo, e tampouco de auxílio de um programador. Para isto ele permite que as variáveis relativas ao balanceamento do jogo sejam visualizadas e alteradas em tempo real pelo game designer, ver figura 1.4.

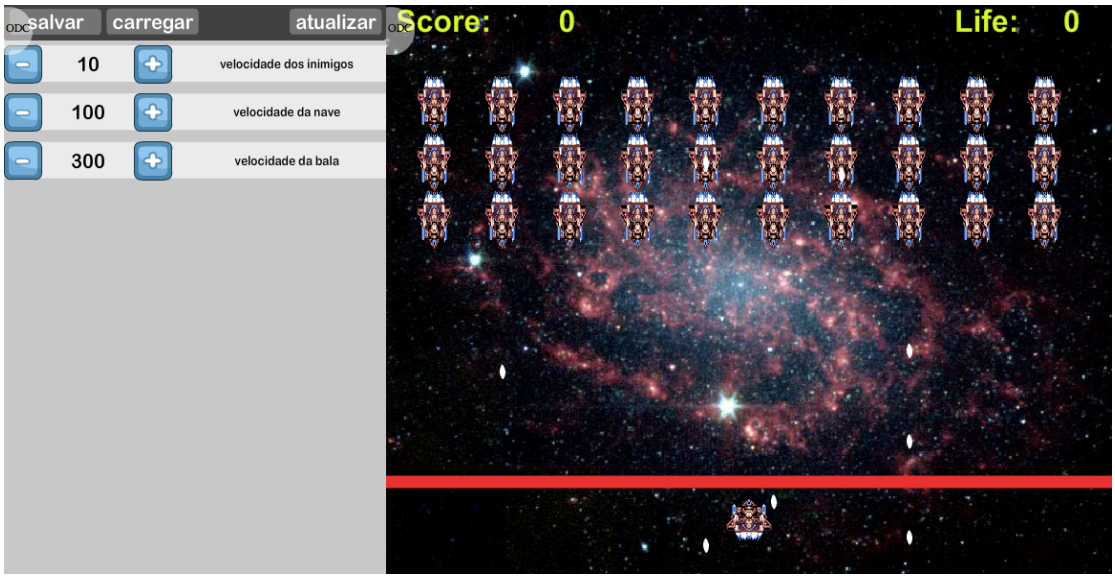

Figura 1.4 – Jogo implementado utilizando o framework Xpose e a ferramenta de interface gráfica XposeGUI (LIMA, 2013)

1.2 Objetivos

Este trabalho de graduação tem como objetivo apresentar uma extensão do *framework Xpose*, citado acima, cujo foco é tornar mais eficiente a etapa de balanceamento, e em consequência, o desenvolvimento de jogos. Para tal é importante que o *framework* dê ao *game designer* uma forma de realizar esta etapa de forma completamente independente do programador e sem exigir dele o contato com o código fonte do jogo. Escolhido o *framework*, o trabalho visa refiná-lo de modo que ele possua uma abstração cada vez maior das características de programação inerentes a esta etapa.

A extensão desenvolvida neste trabalho faz com que o *Xpose* seja capaz de permitir ao *game designer* a criação de uma variável mais abstrata do que as que ele tem expostas para ele. Isto é feito a partir do agrupamento dessas variáveis em uma nova variável que engloba todas as outras. Com isto, espera-se desta extensão que sejam expandidos os limites da capacidade de abstração da etapa de balanceamento ajudando a tornar a realização dela cada vez mais fácil para o *game designer*. Espera-se também aumentar sua facilidade para que o *game designer* possa experimentar mais e com isso realizar um trabalho melhor, considerando que, abstraindo a necessidade de um programador e de possuir noções de programação, ele poderá focar no balanceamento.

1.3 Abordagem

Partindo do framework *Xpose*, de sua interface gráfica, chamada *XposeGUI*, e do protocolo de comunicação utilizado para realizar a interação entre as duas, foi desenvolvida uma extensão que permite ao game designer escolher um grupo de variáveis expostas pelo programador e condensá-las em uma única variável mais abstrata. O objetivo disso é fazer com que o *game designer* possa trabalhar com variáveis mais abstratas e que tenham uma representação mais próxima do que seria o real comportamento dos elementos no jogo.

Usando de exemplo um inimigo cujo comportamento no jogo é se aproximar e atirar no personagem principal, ele teria variáveis do tipo: velocidade de movimento, frequência de tiros, frequência com a qual se movimenta. Todos esses atributos podem ser condensados numa única variável de nome “agressividade”. Ao alterar o valor associado a esta variável o framework fará as modificações correspondentes nas variáveis citadas acima de modo a tornar o inimigo mais ‘agressivo’.

Para isto, foram necessárias alterações nos três componentes citados acima: o protocolo de comunicação, a ferramenta de interface gráfica, *XposeGUI,* e o próprio *framework Xpose.* No protocolo de comunicação foram adicionadas mais funções, relativas as variáveis de grupo para auxiliar a comunicação entre o *Xpose* e o *XposeGUI.* No *Xpose* ficou o centro das alterações sendo estas uma nova classe para dar suporte aos grupos de variáveis e novas funções para a criação dos mesmos e comunicação com a interface gráfica. O *XposeGUI* também sofreu alterações com o

objetivo de permitir que o *game designer* crie grupos e os visualize de forma muito parecida com a qual o *framework* já fazia.

A extensão foi desenvolvida tentando evitar que um trabalho adicional tanto por parte do programador quanto do *game designer* fosse necessário para o uso da versão extendida do *framework.* Além disso, também foi foco desta extensão manter as propriedades básicas do *framework Xpose*, incluindo, tanto o funcionamento da ferramenta em tempo real quanto o fato de o *game designer* ser capaz de realizar todo o balanceamento sem entrar em contato direto com o código fonte do jogo. No fim, com a possibilidade de criação dos grupos funcionando diretamente na interface e sem gerar a necessidade de um contato do *game designer* com o programador para tal, tais objetivos foram alcançados.

2. Balanceamento de Jogos Digitais

Segundo (KOSTER, 2004), “o balanceamento de jogos (game balancing ou difficulty scaling) consiste em modificar parâmetros, cenários, ou comportamentos com o objetivo de garantir um nível adequado de desafio ao usuário, evitando os extremos de frustrá-lo porque o jogo é muito difícil ou entediá-lo porque o jogo é muito fácil”.

Counter Strike é um *First Person Shooter* (FPS) online onde dois grupos de jogadores são divididos em dois times: Terroristas e Contra-Terroristas. Neste jogo, os dois times tem a possibilidade de comprar armas e equipamentos para o seu personagem e o objetivo de um time é eliminar todos os membros do time adversário. Usando este jogo como exemplo, caso um dos dois times tivesse uma vantagem maior sobre o outro através de uma arma que só fosse disponível para uma das equipes, talvez o jogo atingisse os dois extremos que deveriam ser evitados no balanceamento. O primeiro, de frustrar o jogador por que o jogo é muito difícil aconteceria com os novatos no jogo que iniciassem o mesmo com o time mais fraco. Estes iriam perder o jogo muitas vezes e não iriam entender o motivo. Já o outro extremo, o jogador ficar entediado por que o jogo é muito fácil, seria atingido quando um jogador que já soubesse dessa vantagem que o jogo proporciona e fizesse uso da mesma para garantir a vitória sempre que jogasse.

O balanceamento tem o objetivo de corrigir este tipo de erro, que não foi percebido durante o processo de desenvolvimento do jogo, para atingir um jogo capaz de entreter a maior quantidade de jogadores possível. É comum que um jogo passe por diversas fases de balanceamento durante o seu desenvolvimento e até depois de seu lançamento ao público. Diversos jogos, como é o caso de *Counter Strike* e o já citado anteriormente *League of Legends,* mantém a prática de criar *patchs* de correção de balanceamento dos jogos levando em consideração a opinião dos jogadores. Isto é importante pois, como já citado, esta etapa do desenvolvimento de jogos é decisiva para o sucesso de um jogo, um jogo com uma boa ideia, mas com um mau balanceamento pode está fadado ao fracasso (LIMA, 2013).

2.1 Balanceamento Manual

Alguns atributos dos elementos de um jogo não podem ser balanceados sem que o *game designer* jogue ou veja alguém jogando o jogo. “Estes só podem ser ajustados quando o jogo está “jogável”, ou seja, quando o núcleo do jogo está implementado, o *gameplay”* (LIMA, 2013). Usando de exemplo *Counter Strike* e a situação detalhada na seção anterior, só seria possível perceber que uma arma específica causaria uma vantagem maior para uma das equipes jogando por bastante tempo em cada uma das equipes e testando todas as armas cuidadosamente. Alguns valores desses atributos (atributos dos elementos de um jogo) precisam ser experimentados para que o game designer atinja o objetivo que ele imagina para o jogo (LIMA, 2013).

Ao iniciar o processo de balanceamento o *game designer* se junta com um programador para realizar o ciclo de ajustes e testes desta etapa. Este é o método mais comum de balanceamento e é conhecido balanceamento manual. É baseado em perfis de usuários pré-definidos e considera que haja uma evolução não linear nas fases (conjunto de tarefas sequenciais que devem ser realizadas) de um jogo. A habilidade do jogador evolui constantemente a medida que ele consegue avançar nos desafios propostos em cada fase. Estes desafios tem o seu nível de dificuldade aumentado com o passar do jogo. Entre as limitações do balanceamento manual está o fato de que este tipo de balanceamento não é capaz de abranger todos os tipos de jogadores.

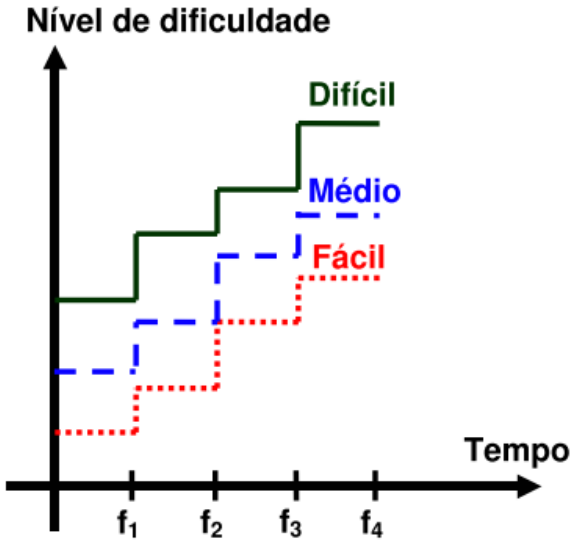

Figura 2.1 - Evolução dos níveis de dificuldade por meio de fases (ANDRADE, 2006).

O objetivo das fases é formar barreiras para garantir que o jogador que tenha se esforçado para transpor tais barreiras possua um nível mínimo de habilidade. Segundo (ANDRADE, 2006), “a ideia é que os jogadores sejam desafiados a evoluir para superar essas barreiras, evitando a situação simplista em que seja possível chegar ao final do jogo sem realizar algum esforço”.

É interessante citar o balanceamento dinâmico (ou balanceamento automático) pois ele é uma outra abordagem para o problema enfrentado neste trabalho. Segundo (ANDRADE, 2006), “o balanceamento dinâmico consiste em prover mecanismos que, periodicamente, identifiquem o nível de habilidade do jogador e, automaticamente, atualizem o nível de dificuldade do jogo para mantê-lo próximo à capacidade do usuário”. Nesta abordagem o jogo avalia o nível de dificuldade baseado na dificuldade que o jogador teve para transpor as fases do jogo. Ela é interessante pois diminui o trabalho do *game designer* e é capaz de gerar níveis de dificuldade personalizadas a partir da capacidade de cada jogador. Porém, justamente o fato de os níveis de dificuldade serem personalizados, que são a parte mais importante do balanceamento automático, são o motivo de muitos *game designers* não o utilizarem. Segundo (LIMA, 2013), “existe aquele tipo de jogador que gosta de escolher em que nível quer jogar, seja este mais fácil do que seu nível de experiência ou mais difícil. [...] Existem muitos jogadores que não gostam quando sentem que o jogo está trapaceando a favor deles quando estão ficando ruins”.

2.2 Principais Dificuldades do Balanceamento Manual

O balanceamento manual exige que o trabalho seja feito pelo *game designer* em conjunto com um programador. A dependência de um programador para realizar o balanceamento vem do fato de que, para que as alterações sejam efetuadas no código do jogo, é necessário que um programador as realize pessoalmente, pois não é da competência de um *game designer* entender nuances de programação. Em consequência desta dependência gerada entre esses dois profissionais surgem diversas dificuldades.

LIMA (LIMA, 2013) lista algumas destas dificuldades: falta de tempo do programador, interrupção no processo de balanceamento, limitação na possibilidade de experimentação, localização geográfica da equipe, rastreabilidade do balanceamento. A falta de tempo do programador, para atender as solicitações do *game designer,* ocorre pois ele não tem como prever a duração de um período de experimentação por parte do *game designer* e não pode ficar aguardando uma resposta sem comprometer sua produtividade.

A interrupção no processo do balanceamento se dá a partir de quando o programador vai realizar as alterações solicitadas pelo *game designer.* Segundo (LIMA, 2013), “nesse tempo alguma informação pode ser perdida, pois em muitos casos, o *game designer* irá se ocupar com outras atividades, interrompendo então o fluxo de pensamento sobre o balanceamento”. A limitação na possibilidade de experimentação é causada pelas interrupções geradas pelo contato necessário entre programador e *game designer.* Com isso, ele não consegue experimentar todas as configurações que ele acharia conveniente, caso pudesse ver as alterações sendo feitas em tempo real, o seu tempo de experimentação fica bastante reduzido por isso.

Problemas de localização geográfica costumam ocorrer em grandes estúdios onde programadores trabalham em unidades diferentes dos *game designers*, e por as comunicações entre eles são bastante ineficientes. Por fim, segundo (LIMA, 2013), “o controle das mudanças de atributos de uma entidade no decorrer do balanceamento é algo que se perde com o tempo. [...] É interessante rastrear as mudanças feitas para poder comparar qual a melhor configuração de balanceamento de um jogo”.

Uma solução que busque resolver estes problemas deve ser capaz de prover ao *game designer* uma maneira de realizar o balanceamento do jogo tendo dois focos principais. O primeiro, fazer de modo que o *game designer* não precise ter conhecimento algum de programação, e o segundo, fazer sem gerar nenhum trabalho extra para nenhuma das partes envolvidas (programador e *game designer*). É importante também que esta solução funcione em tempo real, ou seja, que o *game designer* seja capaz de ver as alterações realizadas por ele. Isto fará com que ele tenha uma capacidade maior para experimentar novas configurações.

3. Estado da arte

Nas seções a seguir serão apresentadas as duas abordagens que compõem o estado da arte no contexto de balanceamento de jogos digitais.

3.1 Balanceamento Automático

A abordagem de balanceamento dinâmico, também conhecida como balanceamento automático ou *Dynamic Difficult Adjustment* (DDA), consiste em utilizar mecanismos que automaticamente identifiquem as características específicas e o comportamento do jogador, e com base em sua experiência adapte os desafios ao seu nível de habilidade (FARIAS, 2011). Com o objetivo de fazer com que o jogo seja equilibrado em dificuldade e que se adapte ao nível de habilidade do jogador para tornar assim o jogo mais justo para todos os jogadores.

Como já citado na seção 2.1, o balanceamento dinâmico consiste em prover mecanismos que, periodicamente, identifiquem o nível de habilidade do jogador e, automaticamente, atualizem o nível de dificuldade do jogo para mantê-lo próximo à capacidade do usuário (ANDRADE, 2006). Isto é feito através de diversas técnicas que envolvem o uso de Inteligência Artificial. São exemplos dessas técnicas: manipulação de parâmetros, aprendizagem por reforço, scripts dinâmicos e algoritmos genéticos (FARIAS, 2011).

Apesar da grande quantidade de pesquisas e técnicas na área, o balanceamento automático não é a abordagem mais utilizada na indústria de jogos. Existem diversos motivos que explicam isto, incluindo, como dito na seção 2.1, o fato de os níveis de dificuldade serem personalizados ao invés de escolhidos pelo jogador. O uso de DDA também faz com que jogadores diferentes tenham sensações diferentes enquanto estão jogando, já que o jogo se adapta a habilidade do jogador e na maioria das vezes isto não é desejado pelo *game designer*.

Isso destrói a tentativa do *level design* de ter momentos fáceis e difíceis durante o jogo para balancear a fluidez do *gameplay*, é comparável a ouvir uma música que esta sempre na mesma altura, ou um filme que não tem momentos de clímax (LIMA, 2013). Embora estejam sendo realizadas diversas pesquisas na área e já tenham se

desenvolvido grandes melhorias nos últimos anos, o balanceamento automático ainda está longe de substituir o balanceamento manual.

3.2 Balanceamento Manual

O balanceamento manual dos jogos é a forma mais difundida na indústria, mesmo com a evolução das técnicas de balanceamento automático e, ao mesmo tempo, por ser muito presente na indústria, é pequeno o número de publicações sobre melhorias desse processo, por conta do alto nível de sigilo empregado na indústria de jogos mundial (LIMA, 2013).

Por conta do sigilo em torno dos modelos de balanceamento usados na indústria há uma pequena dificuldade de entender e definir padrão a seguir. Cada empresa possui as suas técnicas e ferramentas específicas para realizar o balanceamento. Segundo (LIMA, 2013), “algumas empresas dão indícios de como isso pode estar sendo feito em documentos de *post mortens* publicados em revistas especializadas, mas ainda assim a maioria dos detalhes são omitidos”.

Embora haja todo esse sigilo em torno dos métodos utilizados para realizar o balanceamento manual alguns trabalhos que se destacam e “ditam tendências na indústria e no estado da arte do balanceamento manual dos jogos” (LIMA, 2013). Dos citados no trabalho de (LIMA, 2013) os mais relevantes são as pesquisas de Bret Victor (VICTOR, 2010) e *Frogatto & Friends*.

As pequisas de Bret Victor (VICTOR, 2010), tem como objetivo ensinar programação de uma forma diferente, através de elementos visuais e intuitivos. A partir delas foi criada uma ferramenta que mostra aos programadores os efeitos de suas alterações no código em tempo real. Já o *Frogatto & Friends* é um jogo que possui um editor de *levels* com um funcionamento bastante parecido com tais pesquisas. O editor permite que características dos elementos presentes no *level* tenham seus atributos alterados sem que seja necessário realizar a alteração diretamente no código. Isto é muito próximo do que seria ideal para que os *game designers* conseguissem realizar o balanceamento sem depender da presença de um programador.

Estas pesquisas trazem abordagens bastante interessantes para o problema do balanceamento manual de jogos digitais, no entanto, nenhuma delas tem seu foco em resolver tal problema. Tanto a forma com a qual Bret Victor (Victor, 2010) usou para expor as variáveis do código para que o programador pudesse alterar enquanto estivesse rodando a aplicação, quanto a forma com a qual o editor de *levels* do *Frogatto & Friends* faz a comunicação entre a interface gráfica e o código serviram de inspiração para Renan Lima (LIMA, 2013). Ele utilizou as características destes trabalhos para desenvolver um *framework* focado em tornar mais eficiente o balanceamento.

Para conseguir isto foi criado o *Xpose* (LIMA, 2013) que possuiu como objetivos na sua criação ser um *framework* que não exigisse muito trabalho do programador para utilizá-lo e tornar possível que a etapa de balanceamento seja realizada minimizando ao máximo o contato do *game designer* com o programador. O *Xpose* (LIMA, 2013) é um framework que foi desenvolvido com o objetivo de facilitar o trabalho de programadores e game designers na etapa de balanceamento de um jogo.

Neste caso, fazer com que o *game designer* se torne independente do programador e possa realizar o balanceamento dos atributos das entidades de um jogo sem precisar entrar em contato com ele. Para isto o *framework* permite ao programador que ele exponha os atributos das entidades presentes no jogo. E, através de uma interface amigável e um protocolo de comunicação, o *game designer* consegue alterar estes atributos sem a necessidade de entrar em contato com o programador ou editar diretamente o código fonte.

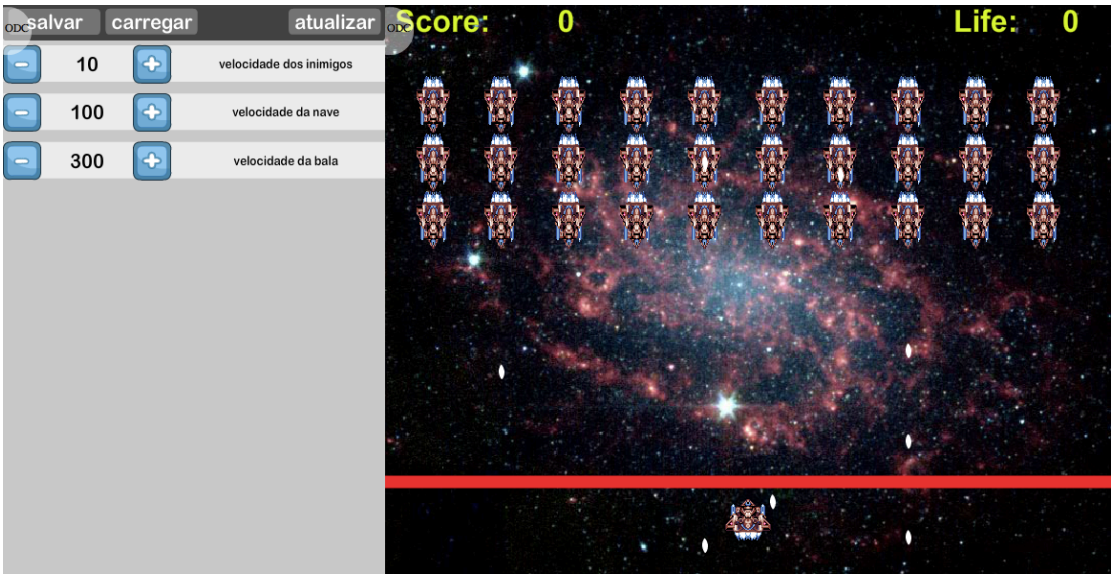

Figura 3.1 – Jogo implementado utilizando o framework Xpose e a ferramenta de interface gráfica XposeGUI (LIMA, 2013)

O uso do protocolo de comunicação, escrito na linguagem de programação *Javascript*, permite que tudo isto aconteça em tempo real. Outra vantagem do uso deste protocolo é o fato de que ele permite que o projeto do *Xpose* seja facilmente replicado em outras linguagens de programação. Respeitando o protocolo de comunicação, isto garante que é possível utilizar o *framework* no desenvolvimento de qualquer jogo, independente da linguagem utilizada no mesmo.

Porém, o que acontece quando o jogo que se deseja balancear possui uma quantidade muito grande de variáveis?

3.2.1 Subproblema

Se um jogo possui uma quantidade muito grande de variáveis o uso do *Xpose* fica muito prejudicado. Usando de exemplo um jogo de estratégia em tempo real, (em inglês, *Real-time Strategy* ou RTS), que possui uma quantidade muito grande de elementos e para cada um deles uma quantidade grande de variáveis relacionadas. No capítulo 5, é detalhado um experimento que foi realizado para validar a extensão criada neste projeto e no anexo 1 é possível ver, no teste aplicado para a simulação de balanceamento de um RTS, exemplos da quantidade e de que tipo de variáveis é necessário utilizar em um projeto como o citado acima.

Com uma grande quantidade de variáveis para controlar individualmente, é bastante provável que o *game designer* perca o controle do balanceamento, dada a complexidade do jogo em si. Como uma forma de tornar o *Xpose* um *framework* mais completo este trabalho propõe uma solução para este problema.

4. Solução

A proposta deste trabalho é apresentar uma extensão do *framework Xpose* que foi criada com o objetivo de tentar solucionar o problema apresentado na seção 3.2.1. Para isto a extensão torna possível que o *game designer* condense um grupo de variáveis em uma variável mais abstrata. O objetivo é dar ao *game designer* a possibilidade de fazer agrupamentos de modo que as variáveis geradas representem comportamentos mais complexos de tais elementos no jogo.

Tomando por exemplo um jogo cujo objetivo é tomar conta de um animal, uma simulação como *Nintendogs*, jogo da *Nintendo*, ou um *Tamagotchi* jogo que fez bastante sucesso nos anos 90 (ver figura 4.1). O animal pode ter vários atributos relacionados a ele: quantidade de refeições por dia, quantidade de passeios por dia, quantas vezes foi ao banheiro, quantas vezes se encontrou com um outro animal. Todos estes atributos podem ser condensados em um grupo chamado “Felicidade”, e para considerar um animal “feliz” ou não, basta que o *game designer* controle essa variável e automaticamente o *framework* irá reduzir e/ou aumentar o valor das variáveis que a compõem para que este comportamento seja refletido no jogo.

Figura 4.1 – Jogo *Nintendogs* da *Nintendo* à esquerda, e à direita o *Tamagotchi*, jogo que fez bastante sucesso nos anos 90

O objetivo disto é fazer com que o *game designer* consiga trabalhar com variáveis mais abstratas e com isso obter uma melhor eficiência nas suas experimentações. Com uma quantidade muito grande de variáveis para controlar a complexidade do balanceamento cresce bastante, no entanto, transformar estas variáveis em uma quantidade menor de grupos de variáveis faz com que o ambiente que o *game designer* terá de trabalhar se torne mais controlado.

4.1 *Xpose* Extendido

A solução foi implementada com o objetivo de manter as antigas características do *Xpose* de ser um *framework* que não exige que o programador tenha que mudar drasticamente o código do jogo para usar, e de fazer com que o *game designer* seja capaz de realizar o balanceamento sem entrar em contato com o código fonte e tendo o mínimo de contato com o programador. Para isto foram necessárias algumas alterações na interface do *Xpose,* a *XposeGUI*, para realizar a exibição destes grupos na tela e permitir a criação dos mesmos. Na figura 3.1 é possível ver uma imagem de como a interface gráfica funcionava antes das alterações realizadas e na figura 4.2 pode-se ver como ficou a *XposeGUI* após a extensão.

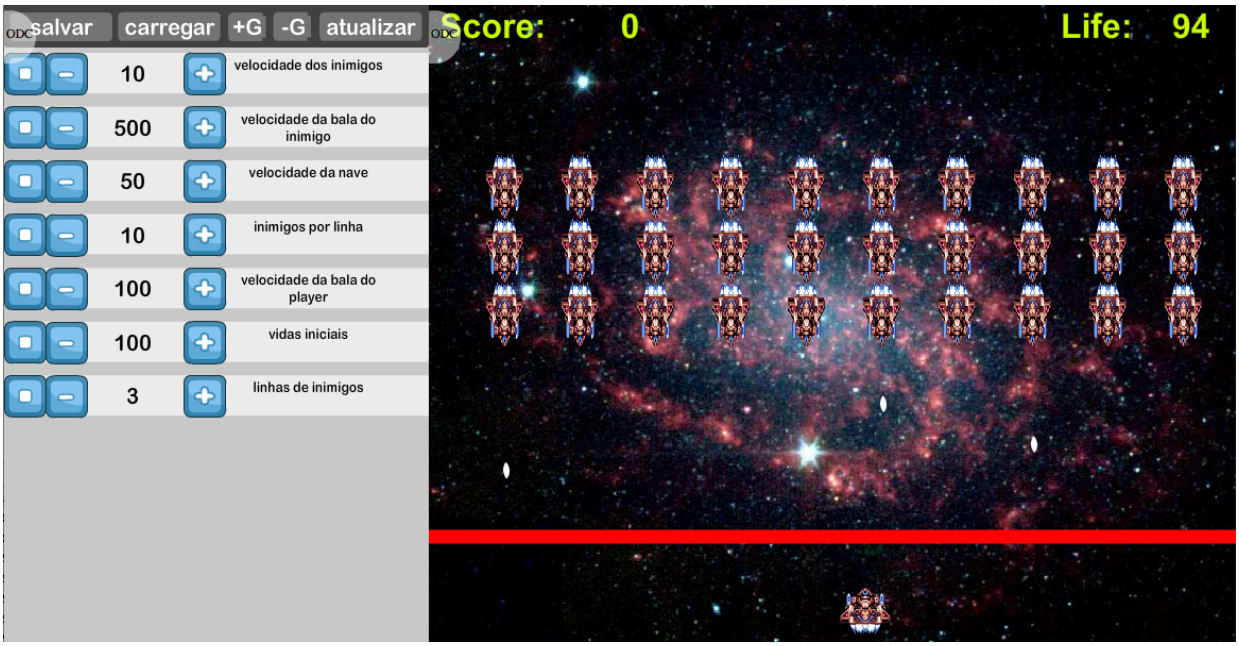

Figura 4.2 – *XposeGUI* após a extensão realizada

Como pode ser notado nas imagens comparando as imagens foram feitas poucas alterações na interface gráfica *XposeGUI*. O objetivo foi fazer com que as alterações realizadas não tivessem influência no uso do *framework* e que um *game designer* que já conhecesse a ferramenta pudesse realizar o seu trabalho ignorando completamente as novas funcionalidades criadas nesta extensão.

As alterações da tela vista na figura 4.2 foram apenas os botões à esquerda de todas as variáveis para permitir que o usuário selecione as variáveis que farão parte do grupo, e os botões de adicionar e remover um grupo que ficam ao lado dos botões de salvar, carregar e atualizar. Além destes botões, foi criada também uma nova tela para a criação dos grupos.

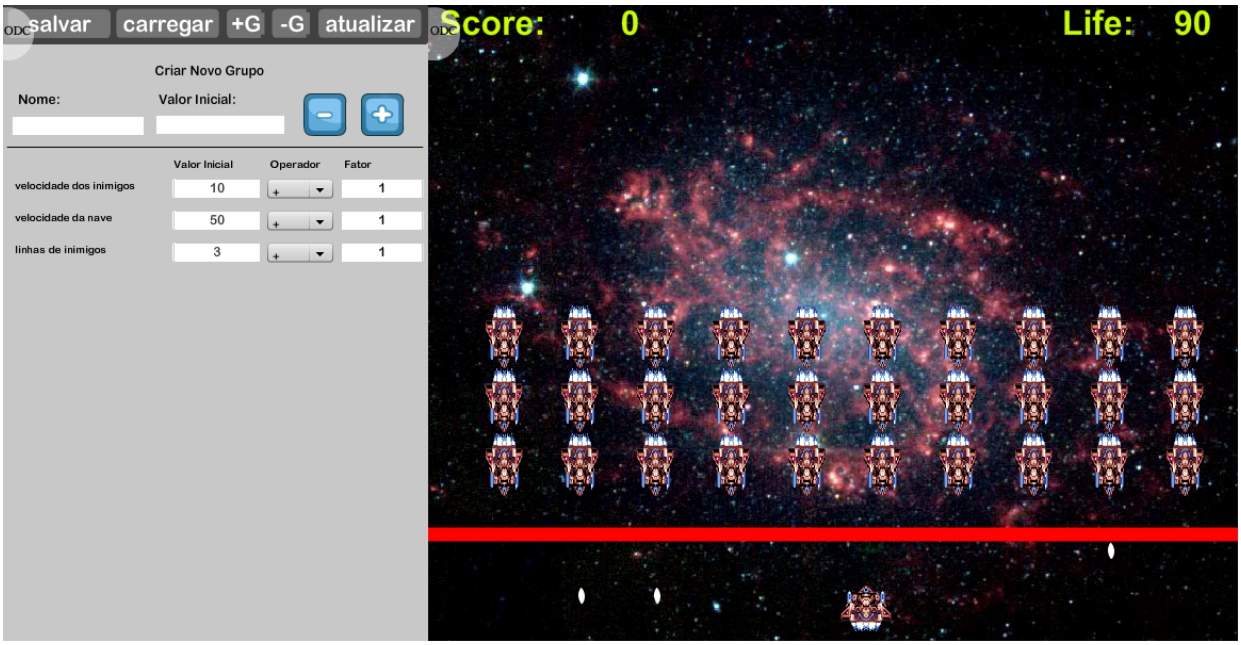

Figura 4.3 – Tela de criação de grupos da nova *XposeGUI*

Na figura 4.3 é possível ver como ficou a tela de criação de novos grupos. A tela possui alguns campos a serem preenchidos e dois botões, o botão “-” para cancelar a operação e o botão “+” para confirmar. Para realizar a operação é necessário definir um nome fantasia (LIMA, 2013) e um valor inicial para o grupo assim como valores iniciais, um operador e um fator para cada uma das variáveis selecionadas (ver figura 4.4).

Figura 4.4 – Seleção do operador na tela de criação de grupos da nova *XposeGUI*

Na versão extendida do *Xpose* existe um novo tipo de variável, a variável de grupo que representa a criação de um grupo. Toda vez que o valor de um grupo é alterado os valores das variáveis que pertencem àquele grupo também são. Por isso é preciso definir um valor inicial para todas elas, para que o novo valor seja calculado

em cima deste valor inicial, um operador e um fator. E como é feito o cálculo do valor das variáveis?

De acordo com a mudança feita na variável de grupo pelo *game* designer o que o *Xpose* faz é: dada a diferença entre o valor atual de um grupo e seu valor inicial, caso esta seja positiva, o valor de cada variável é definido como sendo a aplicação da operação selecionada no seu valor inicial e na multiplicação do fator selecionado pela diferença obtida. Caso a diferença seja negativa, no caso da adição e subtração as operações são invertidas automaticamente ao realizar a multiplicação do fator pela diferença. No caso da multiplicação e da divisão o valor usado no calculo é o módulo da diferença e o fato de ser negativo ou positivo apenas serve como direcionamento da operação a ser utilizada.

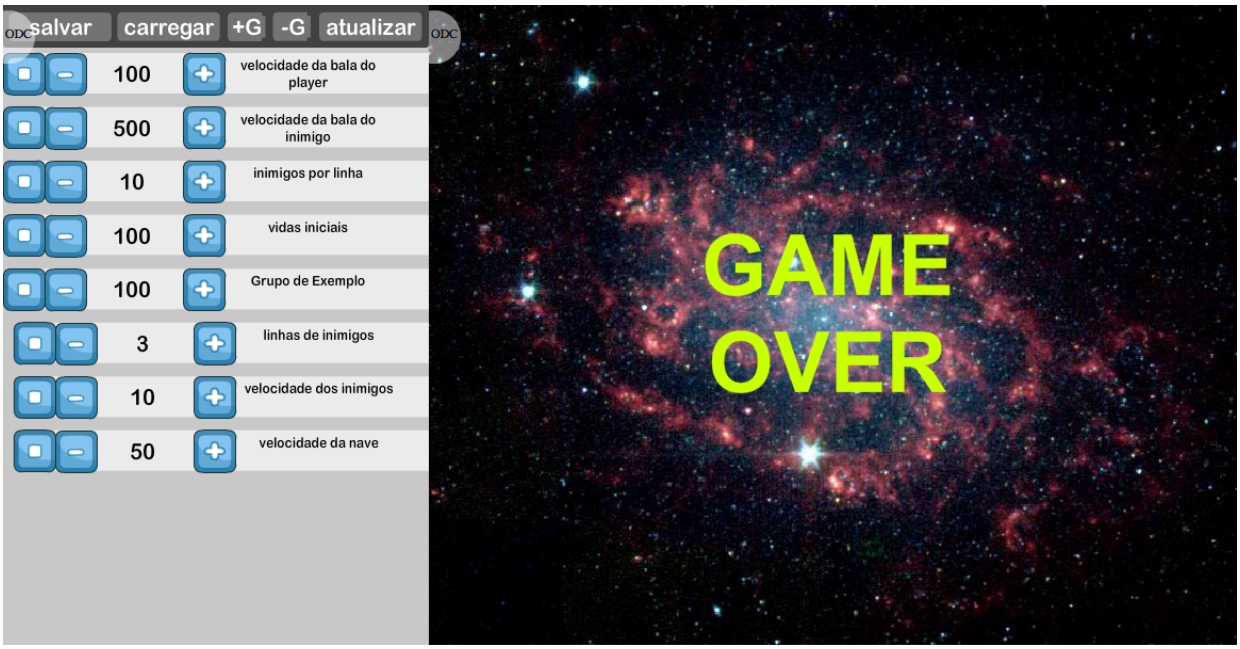

Figura 4.5 – Tela da nova *XposeGUI* exibindo um grupo de exemplo

Na figura 4.5 é possível ver como ficou a exibição de um grupo criado na versão extendida do *framework*. Com a variável de grupo sendo exibida como uma variável comum e as variáveis que fazem parte do grupo ainda aparecendo na tela, para que o *game designer* possa alterá-las individualmente caso seja necessário, porém agora com uma identação em relação outras variáveis.

4.2 Implementação

Para que a extensão funcionasse foi necessária a implementação de novas classes e funções tanto no módulo da interface gráfica, para lidar com os novos tipos de variáveis, quanto no próprio *Xpose,* para que agora dê suporte a criação de grupos, e no protocolo de comunicação.

No *Xpose* está implementada toda a parte de criação, remoção e calculo de valores dos grupos e de suas variáveis relacionadas. Para isso foi preciso criar uma nova classe que foi chamada de "*Group*" para auxiliar a classe "*ParameterCollection*" no tratamento dos grupos. O funcionamento da nova classe é parecido com o da classe "*ExposedObjects*" já que possui o mesmo propósito, de facilitar a manipulação dos valores dos grupos e suas variáveis por parte do *framework* sem criar nenhum esforço para o programador que utilizará a ferramenta. Na classe "*ParametersCollection*" foram criadas as funções que fazem o tratamento do grupo quando criado, processo iniciado pela função "createGroup", a remoção de um grupo já existente, através da função "removeGroup". Também foi criada uma função chamada "*updateParameters*" dentro da classe "*ParametersCollection*", que realiza a alteração das variáveis dentro e fora dos grupos no código do jogo.

No *XposeGUI* além das mudanças realizadas para adicionar os grupos e suas variáveis na tela, e a tela de criação de grupos, foi adicionada uma função chamada "*calculateNewValue*" na classe "*ParametersScreen*" que calcula o novo valor das variáveis antes de passar os valores para o *Xpose.*

O protocolo de comunicação foi alterado para tornar possível a comunicação destas novas funções do *Xpose* e da interface gráfica *XposeGUI*. O protocolo pode ser visto por completo no anexo 1 deste documento.

5. Validação

Após a implementação da extensão do *Xpose* foi feita uma avaliação para validar a sua utilidade. O teste ideal seria aplicar o *framework* em um jogo que já estivesse em desenvolvimento e se enquadrasse no problema citado na seção 3.2.1, que seria um jogo que possuísse uma quantidade de variáveis muito superior do que os jogos já testados por Renan Lima (LIMA, 2013). O objetivo disto seria levar o foco do balanceamento para o uso da extensão e poder assim avaliar a sua eficiência.

Porém, como o tempo disponível para a realização deste trabalho não permitiu que tal situação fosse testada, foi realizada uma simulação de balanceamento com um *game designer* experiente e inserido na indústria de jogos. Fabio Florêncio *game designer* do C.E.S.A.R. (Centro de Estudos e Sistemas Avançados do Recife) , realizou a simulação, que pode ser encontrada no anexo 2 deste documento, e fez alguns comentários sobre a ferramenta.

O teste consistiu em simular o balanceamento de um jogo do gênero RTS (*Real-time Strategy*) já conhecido. O jogo escolhido foi *Age of Empires* (AoE) pois, por ser um jogo muito conhecido, não criou a dificuldade para o *game designer* de balancear um jogo desconhecido. As variáveis escolhidas para o teste foram pré-selecionadas de modo que o balanceamento não necessitasse que o *game designer* levasse muito mais de uma hora para realizar a tarefa.

Os comentários de Fabio Florêncio a respeito da ferramenta e do teste podem ser encontrados a seguir:

> "Realização do teste:
> Parte do teste foi usado com a ferramenta propriamente dita e parte com sua pseudo utilização. Pseudo pois foi pensado no seu uso com um jogo não em andamento: as mudanças não eram visíveis em tempo real ou depois de uma compilação. Embora conceitualmente estivesse tudo correto, não se viu se o comportamento desejado estava de fato sendo executado.
>
> Sobre a ideia e utilidade:
> Conforme um game cresce em complexidade, suas muitas variáveis passam a obter maior influência sobre as demais, gerando com isso comportamentos emergentes. Perder as variáveis de foco é perigoso para um bom balanceamento do jogo por se sobreporem umas sobre as outras. Agrupar variáveis, facilita o trabalho de quem está realizando o game balancing pois garante que determinados comportamentos sejam evocados apenas quando

são necessários, desde que esse agrupamento seja feito corretamente. Com isso o balancing gera outra emergência: a da relação entre esses mesmos agrupamentos. Porém, com menos entidades disponíveis, o ambiente desse sistema complexo se torna mais controlado.

Sobre a usabilidade:
Acredito que, uma vez entendido quais variáveis se quer agrupar e sua relação com as demais, a ferramenta vem como um facilitador do processo de level design. Quanto mais simples a apresentação e visibilidade de cada elemento variante, mais efetivo o trabalho será, e nesse critério a ferramenta atinge seu objetivo.”

6. Conclusões

Este trabalho foi desenvolvido a partir do documento e do *framework* criados por Renan Lima (LIMA, 2013) e seus objetivos foram atingidos com sucesso. Primeiro, o objetivo de manter as características do projeto original de manter a ferramenta como uma ferramenta de fácil implantação por parte do programador responsável por preparar o *framework* para que o *game designer* pudesse utilizá-la. Segundo, o objetivo de ser uma ferramenta que tornasse o trabalho do *game designer* independente do trabalho dos programadores e reduzisse ao máximo a interação necessária entre estes dois profissionais.

Além disso, a extensão mantem as características principais do *Xpose* de permitir que o *game designer* veja as alterações realizadas em tempo real sem que seja necessário alterar diretamente o código fonte ou que o *game designer* tenha algum conhecimento de programação. E como foi visto no documento o uso da extensão é completamente opcional, sendo possível utilizar o *framework* normalmente caso o uso desta não seja necessário.

6.1 Trabalhos Futuros

Durante o desenvolvimento e o teste de validação da extensão foram identificadas algumas possíveis modificações a serem implementadas porém o tempo de desenvolvimento não permitiu que estas chegassem a ser concretizadas e testadas. A seguir serão listadas as possíveis melhorias em ordem decrescente de importância.

A primeira delas é uma renovação na interface gráfica do *framework,* a *XposeGUI*, de forma completa, desde a organização e posicionamento dos itens até o design dos botões. Esta deve ser feita visando uma melhor usabilidade da ferramenta por parte do *game designer*. Isto é importante para tentar evitar que a ferramenta possa de alguma forma atrapalhar o trabalho do *game designer* e facilitar o aprendizado por parte dos novos usuários.

A segunda diz respeito aos itens de grupo exibidos na tela. Usando como base a figura 6.1, é possível ver como os itens relativos a um grupo são mostrados na

interface. Do modo como funciona atualmente, cada vez que uma variável de grupo é criada, uma nova variável é inserida na tela. Apesar de a interface realizar uma identação para as variáveis que pertencem ao grupo, a tela continua muito poluída, o que poderá reduzir a eficiência da ferramenta nos jogos cuja a quantidade de variáveis é muito grande. Como solução é sugerida a adição da funcionalidade de expandir/esconder as variáveis que pertencem a um grupo. Com isto, os itens só seriam vistos pelo *game designer* quando fosse necessário.

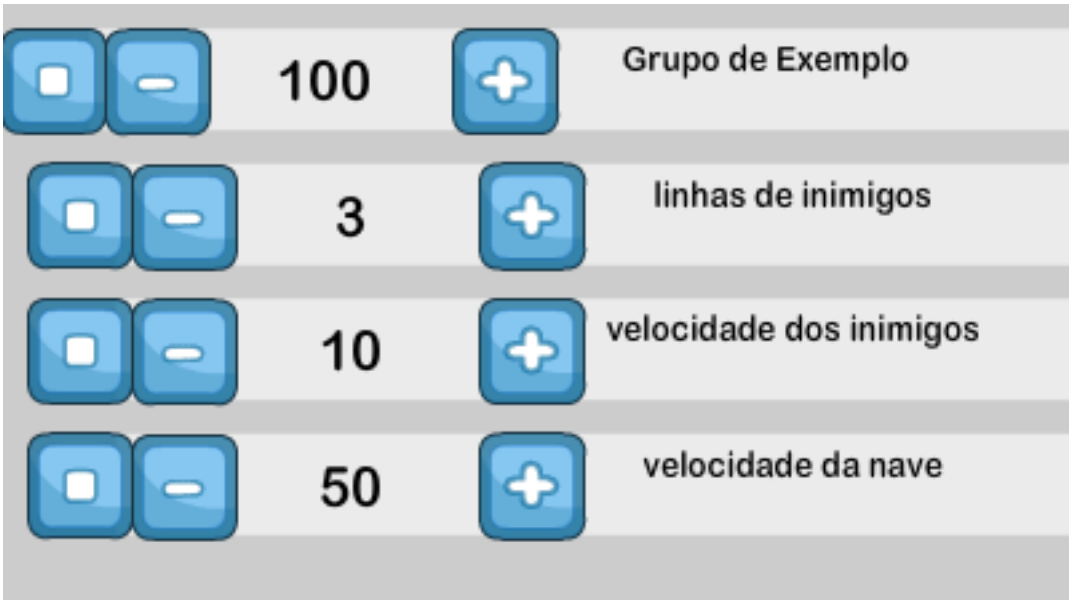

Figura 6.1 – Tela da nova *XposeGUI* mostrando a exibição de um grupo

A terceira modificação proposta é a possibilidade de se criar um grupo de um grupo de variáveis. Dado que já existem alguns grupos criados, tornar possível que seja criado um grupo que condense todos os outros em um único. Antes de realizar a modificação é importante realizar um estudo para verificar a validade e necessidade desta.

Bibliografia

ANDRADE, G. D. De. (2006). *Balanceamento Dinâmico de Jogos : Uma Abordagem Baseada em Aprendizagem por Reforço*. *Universidade Federal de Pernambuco*. Universidade Federal de Pernambuco.

BRATHWAITE, B., & SCHREIBER, I. (2009). *Challenges for Game Designers*.

FARIAS, D. L. De. (2011). *Estudo sobre balanceamento dinâmico de jogos baseado em sistemas classificadores*. Universidade Federal de Pernambuco.

KOSTER, R. (2004) *Theory of Fun for Game Design*, Paraglyph Press, Phoenix.

LIMA, R. P. G. de. (2013). *Xpose : um Framework para facilitar o balanceamento manual de jogos digitais*. *Universidade Federal de Pernambuco*. Universidade Federal de Pernambuco.

ROCHA, E. J. T. S. (2003). *Forge 16V: Um Framework para Desenvolvimento de Jogos Isométricos*. Universidade Federal de Pernambuco.

VICTOR, B. *No Title*. Disponível em: <http://worrydream.com/#!/InventingOnPrinciple>. Acesso em: 20 setembro. 2013.

Anexos

Anexo 1

```
//xpose protocol

function getExposedParameters()
{
	return document.getElementById('game').getExposedParameters();
}
function prepareParametersToSave()
{
	return document.getElementById('game').prepareParametersToSave();
}
function loadParameters(data)
{
	document.getElementById('game').loadParameters(data);
}
function changeParameter(fantasyName, value)
{
	document.getElementById('game').changeParameter(fantasyName, value);
}
function getParameterValue(fantasyName)
{
	return document.getElementById('game').getParameterValue(fantasyName);
}
function getParameterType(fantasyName)
{
	return document.getElementById('game').getParameterType(fantasyName);
}
function getGroupVars(fantasyName)
{
	return document.getElementById('game').getGroupVars(fantasyName);
}
function createGroup(fantasyName, defaultValue, objects)
{
	document.getElementById('game').createGroup(fantasyName, defaultValue,
objects);
}
function removeGroup(fantasyName)
{
	document.getElementById('game').removeGroup(fantasyName);
}
function updateParamValues(fantasyName, fator, operador)
{
	document.getElementById('game').updateParamValues(fantasyName, fator,
operador);
}
```

```
function getParameterFator(fantasyName)
{
	return document.getElementById('game').getParameterFator(fantasyName);
}
function getParameterOperador(fantasyName)
{
	return document.getElementById('game').getParameterOperador(fantasyName);
}
function getParamDefaultValue(fantasyName)
{
	return document.getElementById('game').getParamDefaultValue(fantasyName);
}
function setParamDefaultValue(fantasyName, value)
{
	document.getElementById('game').setParamDefaultValue(fantasyName, value);
}
```

Anexo 2

Experimento de Balanceamento de Variáveis Agrupadas:

Simular o balanceamento do jogo Age of Empires utilizando as variáveis abaixo. Ao realizar o balanceamento, avaliar o uso de uma ferramenta que possibilita o agrupamento de variáveis e redigir um comentário a respeito da ideia e sua utilidade e sobre a usabilidade da ferramenta.

Variáveis para o balanceamento do jogo Age of Empires:

Configuração de inicio:

Inicio: Quantidade de Madeira
Inicio: Quantidade de Carne
Inicio: Quantidade de Ouro
Inicio: Quantidade de Pedra
Inicio: População Total
Inicio: Quantidade de Aldeões

UNIDADES:

Aldeão:

Velocidade de movimento do aldeão
Velocidade de coleta de ouro do aldeão
Velocidade de coleta de madeira do aldeão
Velocidade de coleta de carne do aldeão
Velocidade de coleta de pedra do aldeão
Resistencia do Aldeão
Dano causado pelo Aldeão
Quantidade Máxima de Aldeões por Civilização

Arqueiro:

Velocidade de construcao de um arqueiro
Velocidade de movimento de um arqueiro
Frequencia de ataque de um arqueiro
Dano causado pelo arqueiro
Resistencia de um arqueiro

Soldado:

Velocidade de construcao de um soldado
Velocidade de movimento de um soldado
Frequencia de ataque de um soldado
Dano causado pelo soldado
Resistencia de um soldado

CONSTRUÇÃO:

Base:

Velocidade de construcao de um aldeao (Base)
Velocidade de construcao da Base
Custo da Base
Resistencia da Base
Dano causado pela Base

Prédio dos soldados:

Velocidade de construcao do predio de soldados
Custo do predio de soldados
Resistencia do predio de soldados

Prédio dos arqueiros:

Velocidade de construcao do predio de arqueiros
Custo do predio de arqueiros
Resistencia do predio de arqueiros

Casa:

Velocidade de construção de uma casa
Custo de uma casa
Resistencia de uma casa

Mina:

Velocidade de construcao de uma Mina de ouro/pedra
Resistencia de uma Mina de ouro/pedra
Custo de uma mina de ouro/pedra
Velocidade de construcao de uma Mina de madeira
Resistencia de uma mina de madeira
Custo de uma mina de madeira

RESULTADOS:

Auto balanceamento do soldado:
A medida que mais soldados são produzidos, maior é sua eficácia no combate… tornam-se assim, unidades mais custosas e de difícil aquisição

Velocidade de movimento de um soldado -1
Frequencia de ataque de um soldado -1
Dano causado pelo soldado +2
Resistencia de um soldado +2

Auto balanceamento do arqueiro:
A medida que mais arqueiros são produzidos, menor é sua eficácia no combate… por isso devem atacar em grande número

Velocidade de movimento de um arqueiro +1
Frequencia de ataque de um arqueiro +1
Dano causado pelo arqueiro -2
Resistencia de um arqueiro -2

Combate arqueiro x soldado batalha épica:
Batalha entre dosi exércitos rivais. Os soldados são o front de batalha e por isso, as unidades mais caras do exército. Os arqueiros podem ser produzidos com mais facilidade, custo baixo, velocidade de construção alta porém são mais frágeis.

Soldado: +2
Arqueiro: -1
Velocidade de construcao de um arqueiro +2
Velocidade de construcao de um soldado -1
Custo de arqueiro -1
Custo de soldado +2

Edificação em vacas magras:
Em dado momento é interessante que os recursos passam a ser limitados e a população produz pouco por desânimo.

Quantidade Máxima de Aldeões por Civilização -2
Inicio: Quantidade de Carne -1
Inicio: Quantidade de Ouro -1
Custo de uma casa -2
Inicio: Quantidade de Madeira +3
Inicio: Quantidade de Pedra +2
Velocidade de coleta de ouro do aldeão -1
Velocidade de coleta de madeira do aldeão -1
Velocidade de coleta de carne do aldeão +2

Velocidade de coleta de pedra do aldeão -1

Agricultura e pecuária:
População de coletores natos e esfomeados. A população inicia-se baixa. Os custos de construção são geralmente altos.

Velocidade de coleta de ouro do aldeão +4
Velocidade de coleta de madeira do aldeão +3
Velocidade de coleta de carne do aldeão -1
Velocidade de coleta de pedra do aldeão -2
Quantidade Máxima de Aldeões por Civilização +3
Velocidade de construção de uma casa +1
Custo de uma casa +4
Inicio: Quantidade de Madeira -2
Inicio: Quantidade de Carne -3
Inicio: Quantidade de Ouro +4
Inicio: Quantidade de Pedra -1
Inicio: Quantidade de Aldeões -2

Comentários de Fabio Florencio, *game designer* que realizou o teste:

"Realização do teste:
Parte do teste foi usado com a ferramenta propriamente dita e parte com sua pseudo utilização. Pseudo pois foi pensado no seu uso com um jogo não em andamento: as mudanças não eram visíveis em tempo real ou depois de uma compilação. Embora conceitualmente estivesse tudo correto, não se viu se o comportamento desejado estava de fato sendo executado.

Sobre a ideia e utilidade:
Conforme um game cresce em complexidade, suas muitas variáveis passam a obter maior influência sobre as demais, gerando com isso comportamentos emergentes. Perder as variáveis de foco é perigoso para um bom balanceamento do jogo por se sobreporem umas sobre as outras. Agrupar variáveis, facilita o trabalho de quem está realizando o game balancing pois garante que determinados comportamentos sejam evocados apenas quando são necessários, desde que esse agrupamento seja feito corretamente. Com isso o balancing gera outra emergência: a da relação entre esses mesmos agrupamentos. Porém, com menos entidades disponíveis, o ambiente desse sistema complexo se torna mais controlado.

Sobre a usabilidade:
Acredito que, uma vez entendido quais variáveis se quer agrupar e sua relação com as demais, a ferramenta vem como um facilitador do processo de level design. Quanto mais simples a apresentação e visibilidade de cada elemento variante, mais efetivo o trabalho será, e nesse critério a ferramenta atinge seu objetivo."