Pós-Graduação em Ciência da Computação

“Xpose: um framework para facilitar o balanceamento manual de jogos digitais”

Por

RENAN PEREIRA GOUVEIA DE LIMA

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, AGOSTO/2013

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RENAN PEREIRA GOUVEIA DE LIMA

"XPOSE: UM FRAMEWORK PARA FACILITAR O BALANCEAMENTO MANUAL DE JOGOS DIGITAIS"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: GEBER LISBOA RAMALHO
CO-ORIENTADOR: ANDRE LUIS DE MEDEIROS SANTOS

RECIFE, AGOSTO/2013

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Lima, Renan Pereira Gouveia de
Xpose: um framework para facilitar o balanceamento manual de jogos digitais / Renan Pereira Gouveia de Lima. - Recife: O Autor, 2013.
84 f.: il., fig., tab.

Orientador: Geber Lisboa Ramalho.
Dissertação (mestrado) - Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2013.

Inclui referências e anexo.

1. Engenharia de software. 2. Jogos digitais. 3. Game design. I. Ramalho, Geber Lisboa (orientador). II. Título.

005.1 CDD (23. ed.) MEI2013 – 113

Dissertação de Mestrado apresentada por Renan Pereira Gouveia de Lima à Pós Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"XPose: Framework de Exposição de Atributos para Balanceamento de Entidades em Jogos Digitais"** orientada pelo **Prof. Geber Lisboa Ramalho** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Geber Lisboa Ramalho
Centro de Informática / UFPE

_____

Prof. Andre Menezes Marques das Neves
Departamento de Design / UFPE

_____

Prof. Giordano Ribeiro Eulalio Cabral
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 23 de agosto de 2013.

_____

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Aos meus pais e a todos que me orientaram nesta pesquisa.

Obrigado!

Resumo

O Balanceamento de jogos digitais é uma atividade comum a todo jogo em suas fases de finalização. Esta atividade é bastante crítica, pois afeta diretamente a jogabilidade e consequentemente o sucesso do jogo. Em jogos digitais, balancear é o processo em que o game designer decide sobre modificações de determinados atributos de entidades de jogo afim de tornar o jogo mais desafiador, divertido e atraente. Muitas pesquisas na área de balanceamento automático existem, mas alguns pontos negativos são apontados pela indústria, e o que realmente se vê é que o método tradicional, em que game designer e programador precisam conversar para fazer o balanceamento manual, ainda é o mais utilizado. Por ser, este segundo caso, uma atividade que depende de pelo menos dois profissionais, game designer e programador, o processo demanda um tempo extra de comunicação entre as partes, mesmo que seja uma alteração simples de um único atributo numa entidade. Vários problemas podem ser identificados nessa necessidade de comunicação, como falta de tempo por parte dos profissionais, localização geográfica da equipe, limitações nas possibilidades de experimentações, entre outros. Além disso, a falta de disponibilidade imediata entre as partes quebra o ritmo do balanceamento, dificultando e atrasando o processo como um todo. O objetivo principal deste trabalho é tornar o game designer mais autônomo nas fases de balanceamento de um jogo, dependendo menos do trabalho do programador para balancear os atributos das entidades, e sem que ele precise entrar em contato direto com o código fonte. Para atingir o objetivo foi elaborado um framework, chamado de Xpose, em que o programador poderá expor os atributos das entidades. Dessa forma, com o auxilio de uma interface e um protocolo de comunicação, o game designer consegue alterar valores de atributos do jogo de forma rápida e sem interrupções. Foram feitos experimentos com game designers atuantes no mercado de jogos de Recife no estado de Pernambuco e os resultados apontaram para uma melhoria na eficiência do balanceamento usando o Xpose.

Palavras-chave: Xpose, balanceamento, balanceamento manual, balanceamento automático, framework, jogos, game designer, GDD.

Abstract

Game Balancing is a common activity to every game project at the final stages of development. This is a very critical activity, because it directly affects the gameplay and consequently the success of the game. In digital games, game balancing is the process in which the game designer tweaks the game entities' attributes, in order to make the game more challenging, fun and appealing. While some research is being done on the topic of automatic balancing, some negative points are been indicated by the industry. What happens in the industry is that the traditional method, where game designer and programmer need to talk to do manual balancing, is still used the most. The second case is an activity that depends on at least two professionals, the game designer and the programmer, and so the process requires extra time for communication between parties, for even the simplest of changes of a single attribute in an entity. Several problems can be identified in the traditional method, such as lack of time by professionals, geographical location of staff, limitations on the possibilities of experimentation, among others. Furthermore, the lack of availability between parties breaks the rhythm of balancing. The main objective of this research is to make the game designer more autonomous during the game balancing phase. To depend less on the programmer`s job while performing attribute alterations on a game's entities, he needs to be able to work without direct contact with source code. To achieve this goal we designed a framework, called Xpose, in which the programmer can expose the attributes of the entities easily. Thus, with the aid of an interface and a communication protocol, the game designer can change them without having to move directly to the source code. Experiments were done with game designers working in the game market of Recife in Pernambuco and the results showed an improvement in the efficiency of balancing a game using Xpose.

Keywords: Xpose, balancing, manual balancing, automatic balancing, framework, games, game designer, GDD.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Há algum tempo, a atividade de fazer um jogo digital era trabalho unicamente de programadores[1]. Não existia uma grande preocupação em como o processo de desenvolvimento de um jogo deveria ser difundido em uma equipe multidisciplinar e até mesmo o documento de especificação do jogo muitas vezes não existia. Estes profissionais trabalharam em um ambiente onde o código mais rápido e o de menor tamanho eram o principal objetivo, e onde eles tinham total controle de todo o código que estava sendo executado na máquina (ROCHA, 2003) . Os jogos costumavam ser fruto da imaginação e responsabilidade de poucos programadores, como foi o caso do jogo Pong criado por dois programadores, Nolan Bushnell e Ted Dabney, por volta de 1972, que deu origem a empresa Atari (ISAACSON, 2011). Esses jogos não tinham a real necessidade de documentação para o sucesso de seu desenvolvimento e como não existia uma grande equipe envolvida, problemas com comunicação ou falta de informações eram facilmente resolvidos.

Com o crescimento da indústria de jogos (já equiparada a indústria de cinema (FURTADO, 2006)) cresceu também a necessidade de aumentar a equipe de trabalho, que além dos programadores, conta atualmente com artistas visuais, designers de interface, músicos, físicos, entre outros profissionais de diversas áreas de conhecimento. 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 (BRATHWAITE; SCHREIBER, 2009). Uma das atividades principais deste profissional é a criação do documento de *game* design ou *game design document* (GDD), instrumento que serve como guia para toda a equipe de desenvolvimento.

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. Essas definições iniciais feitas pelo *game designer* no GDD tendem a sofrer mudanças durante o desenvolvimento do jogo. Algumas delas, são as mudanças dos atributos das entidades. Os atributos de uma entidade são as variáveis que caracterizam o seu

[1] Nesta pesquisa, chamarei de programadores aqueles responsáveis pela codificação dos jogos, desenvolvedores ou equipe de desenvolvimento são todos que trabalham durante o projeto do jogo, sejam eles game designers, artistas, músicos, e os próprios programadores.

comportamento dentro do jogo. Por exemplo, a aceleração de um carro de corrida ao se acionar o comando de aceleração, a potência do soco de um personagem num jogo de luta, a altura do pulo de um personagem num jogo de aventura, entre outros atributos que são parte da caracterização e comportamento das entidades. Alguns desses atributos são representados por mais de uma variável[2], como a agressividade de um certo personagem pode ser medida com a força de seu golpe mais a velocidade de seu deslocamento, nesse caso o atributo é um composto de mais de uma variável. 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.

Cabe ao game designer estipular os valores dos atributos das entidades de um jogo para torna-lo balanceado. Esta atividade deixará o jogo mais próximo dos objetivos procurados pelo GD. Um jogo não pode ser muito difícil a ponto de fazer o jogador desistir de joga-lo, nem muito fácil a ponto dele não se sentir desafiado. Conseguir encontrar esse equilíbrio, de forma progressiva e tornando o jogo cada vez mais interessante ao jogador é o grande desafio do *game designer* nas etapas de balanceamento.

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. Em jogos on-line, quando o balanceamento não está de acordo no julgamento dos jogadores é comum ver o termo informal "*imbalanced*" ser usado, nas comunidades e fóruns especializados para os jogos. Por exemplo, em *Counter Strike* (ver figura 1.1), um famoso jogo do gênero conhecido como FPS, sigla em inglês para *First Person Shooter* (tiro em primeira pessoa) e pode adquirir diversas armas para conquistar seus objetivos. Para este jogo, foi lançada uma atualização que permite a retirada, do arsenal de armas possíveis, a arma chamada de AWT, um rifle em que o jogador com um tiro em qualquer parte do corpo é abatido, estando este usando ou não proteção, como por exemplo um colete a prova de balas. Como difere bastante do comportamento das outras armas do jogo, está foi considerada "imbalanced" pelos jogadores e os produtores do jogo atendendo suas solicitações lançaram uma atualização para remoção da arma caso o jogador deseje (BURGUN, 2011).

[2] Neste trabalho, chamaremos de atributo as características das entidades do ponto de vista conceitual do game designer e, de variável, a representação do atributo dentro do código fonte.

Figura 1.1 – Imagem do jogo *Counter Strike* com o jogador usando a arma AWT.
Fonte: http://www.youtube.com/watch?v=N4x4vgCJ9lU

1.1 Motivação

É comum o trabalho de balanceamento não depender apenas do *game designer*, pois este, apesar de ser a pessoa indicada para manipular os valores dos atributos das entidades, depende também de, pelo menos, um programador que execute a mudança de fato no código do jogo. Em projetos maiores, pode depender de mais programadores. O processo passa a demandar um tempo extra, pois, mesmo que seja uma alteração simples de um atributo numa entidade, o game designer precisa abrir um canal de comunicação com o programador. Vários problemas podem ser identificados nessa necessidade de comunicação, como:

- **Falta de tempo do programador** – No momento em que o game designer precisa da mudança, nem sempre o programador estará disponível para realizar os ajustes que ele deseja testar para balancear o jogo. Como é uma atividade que demanda experimentação, é difícil prever quanto tempo ela demorará, o que pode comprometer bastante a produtividade do programador e do *game designer*.
- **Interrupção no processo de balanceamento** – Essa interrupção é algo inevitável quando há dependência no trabalho. Como consequência, do momento em que o game designer pensa em fazer as alterações, ao momento que ele vai comunicar ao programador, alguma informação pode ser perdida,

pois quem vai fazer de fato a alteração não é aquele que pensou nela. Por exemplo, quando o *game designer* descreve as alterações necessárias ao programador, este pode estar ocupado com a implementação de outras funcionalidades do jogo, só podendo então atender às solicitações do GD em outra ocasião. Em muitos casos, o GD irá se ocupar com outras atividades, interrompendo então o fluxo de pensamento sobre o balanceamento.

- **Limitação na possibilidade de experimentação** – A experimentação de várias configurações diferentes, de valores de atributos para as entidades, fica limitada ao tempo que o *game designer* tem em conjunto com o programador. Se o game designer conseguisse fazer esse trabalho de forma autônoma seria muito mais fácil para ele experimentar diversas possibilidades antes de decidir a configuração ideal.
- **Localização geográfica da equipe** – Nem sempre as pessoas estão no mesmo local, ou tem a disponibilidade de trabalhar no mesmo horário, dificultando o processo de balanceamento final do jogo, podendo com isso resultar num jogo mal balanceado com baixa qualidade, ou um atraso na entrega final do projeto.
- **Rastreabilidade do balanceamento** – 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.

Na tentativa de sanar esses problemas existem alguns trabalhos, como é o caso do projeto Frogatto[3], que trata-se de um jogo de plataforma, ver Figura 1.2, que apresenta um módulo em que é possível editar a fase do jogo e os atributos das entidades presentes nessa fase. Para isso, uma linguagem de script é disponibilizada para que os atributos sejam alterados. Apesar de ser um avanço no ponto de vista do balanceamento, o fato de ter que usar uma linguagem de programação para executar as alterações insere uma dificuldade para o *game designer*, que não está acostumado com o ambiente de programação.

[3] Frogatto & Friends. Disponível em www.frogatto.com

Figura 1.2 – Tela de edição do Frogatto.

Fonte:www.frogatto.com

Outro trabalho que se destaca neste tema é o trabalho de Bret Victor (VICTOR, 2010). Ele apresenta uma forma interativa de mudança de atributos dentro do código de um programa, ver Figura 1.3, em que é possível mudar os atributos em tempo real e já ver na execução do programa os efeitos dessa mudança. Trata-se de uma boa abordagem para mostrar como as mudanças no código alteram o comportamento dos jogos em tempo real, dando um bom entendimento ao espectador do que cada atributo é responsável na elaboração do programa ou do jogo.

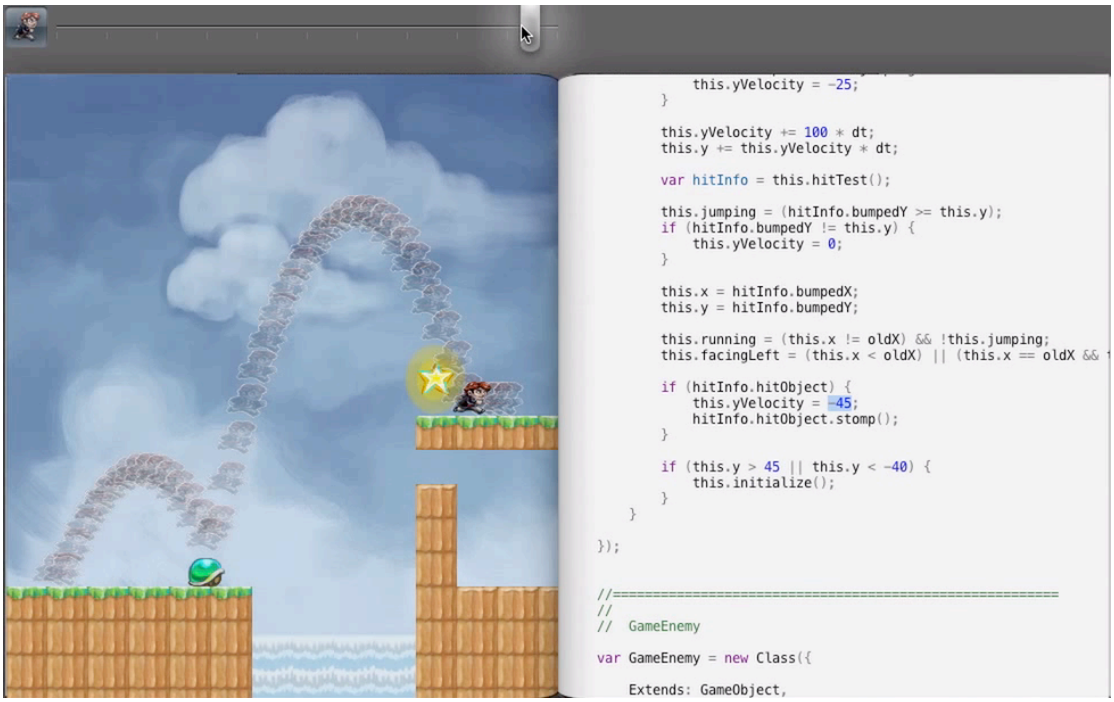


Figura 1.3 – Ferramenta de Bret Victor com as mudanças de atributos sendo refletidas em tempo real.

Fonte: http://www.youtube.com/watch?v=PlI-gPu3SPI

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. O que diferencia dos objetivos principais deste trabalho.

1.2 Hipótese de trabalho

O que se busca com essa pesquisa é avaliar a seguinte hipótese: ao tornar o *game designer* autônomo na etapa de balanceamento, consegue-se que ele seja mais eficiente no seu trabalho e com isso aumente a qualidade dos jogos. De forma prática esta hipótese sugere que se o game designer usar uma ferramenta que exponha os atributos para balanceamento, ao invés de sempre recorrer ao programador para executar mudanças de valores desses atributos, ele conseguirá se concentrar e se dedicar mais a atividade, e com isso experimentar várias configurações diferentes e ser mais eficiente em seu trabalho. A partir dessa hipótese os objetivos da pesquisa são formulados.

1.3 Objetivos

O objetivo principal deste trabalho é tornar o game designer mais autônomo nas fases de balanceamento de um jogo. Dependendo menos do trabalho do programador para balancear os atributos das entidades, sem que ele precise entrar em contato direto com o código fonte.

Para ajudar a atingir o objetivo dois artefatos foram elaborados:

- Definição de um framework que será usado pelos programadores no desenvolvimento do jogo e que traga como benefício a autonomia por parte do game designer para balancear o jogo.
- Construir uma ferramenta que seja usada pelo game designer e que use o framework apontado acima.

1.4 Abordagem

Acompanhando o objetivo principal desta pesquisa, de deixar o trabalho de balanceamento do game designer autônomo, se faz necessário a construção de uma interface de comunicação entre o game designer e os atributos do jogo. Essa interface precisa ser o mais próximo possível da realidade dos *game designers*, amigável e com resposta rápida as suas mudanças.

A construção dessa interface de comunicação é feita no decorrer do processo de desenvolvimento do jogo e segue alguns passos básicos.

- **Game designer indica os atributos** – Inicialmente o game designer deverá apontar quais os atributos que deseja manipular durante a etapa de balanceamento do jogo. Esses atributos poderão ser indicados tanto na etapa de planejamento, quanto posteriormente, caso algum outro atributo se torne interessante de ser manipulado.
- **Programador marca no código as variáveis** – Para que os atributos sejam expostos, de forma que o *game designer* consiga manipula-los, é necessário que algumas modificações sejam feitas no código. O programador irá marcar, no código fonte do jogo, quais variáveis fazem parte dos atributos que o game designer deseja manipular, mas seu trabalho será facilitado com o uso de um framework de desenvolvimento que será acoplado ao código já produzido ou em produção do jogo.
- **Interface gráfica é entregue ao *game designer*** – Após o jogo atingir uma certo nível de maturidade no desenvolvimento, o *game designer* já pode

através de uma interface gráfica, manipular os atributos que ele indicou no início do processo e experimentar a melhor configuração para o jogo, independentemente do trabalho do programador.

Um protocolo de comunicação entre as variáveis expostas no código e uma interface gráfica é estabelecido e dessa forma a ferramenta visual irá acessar os atributos expostos e apresenta-los ao game designer de forma mais amigável, assim o game designer ficará livre para testar diversas configurações diferentes, sem a necessidade de interação com o programador. Além de mexer nos atributos das entidades, o game designer poderá salvar ou carregar configurações salvas por ele para poder compara-las.

Durante a atividade de balanceamento o game designer já vai modificando o comportamento do jogo, de forma que se a equipe trabalha com sistemas de versionamento, sempre que uma nova configuração for feita pelo *game designer*, toda a equipe irá ver o resultado do balanceamento.

1.5 Organização da dissertação

Este trabalho está dividido em sete capítulos. No capítulo 2, foram definidos alguns conceitos fundamentais para o tema, conceitos sobre o *design* de jogos e o papel do *game designer* na indústria atual. No restante deste capitulo também são mostrados conceitos de balanceamento em jogos, como é feito hoje em dia nas empresas de desenvolvimento, quais os principais problemas enfrentados pelos profissionais da área e como seria uma forma próxima do ideal para que o balanceamento fosse executado de forma eficiente. No capítulo 3, o estado da arte no que se diz respeito a balanceamento é exposto. É mostrado o que tem sido estudado em relação a balanceamento automático e melhorias no balanceamento manual, e algumas críticas, que explicam porque o balanceamento automático não é unânime no mercado de jogos digitais. O capítulo 4, é uma reflexão sobre como chegar numa solução para os problemas levantados sobre a forma de balanceamento mais usada, balanceamento manual. Prós e contras de cada uma das possíveis soluções são discutidos.

No capítulo 5, é apresentada a implementação da solução escolhida neste trabalho. A arquitetura que foi pensada inicialmente e a sua evolução para se encaixar na realidade das empresas de jogos, é mostrada, assim como detalhes de como cada módulo da solução foi pensado e implementado, sua interface de comunicação,

protocolos e framework. O capítulo 6, mostra como o trabalho foi avaliado, o planejamento dos experimentos, a apresentação e a discussão dos resultados colhidos. O capítulo 7 resume as conclusões sobre o que foi estudado, as principais contribuições da pesquisa e trabalhos futuros.

2 *Game Designer* e o Balanceamento de jogos

Game Design é o processo de criação do conteúdo e das regras de um jogo (BRATHWAITE; SCHREIBER, 2009). Bons *game designs* conseguem fazer com que o jogador se sinta motivado para consumir este conteúdo através de objetivos, sejam eles explícitos e guiados pelo jogo, ou implícitos e guiados pela vontade do jogador.

A disciplina de *Game Design* é algo que vem de bem antes dos computadores, fazer o projeto (*design*) de um jogo se fez necessário desde o surgimento do termo jogo. Existem relatos do interesse humano por jogos desde muito tempo atrás, sua origem exata é desconhecida, mas estima-se que em culturas antigas, como a dos Sumérios (5000 a.c.)(FURTADO, 2006), o interesse do homem pelos desafios de jogar já era evidente. A busca por esses desafios, tanto físicos quanto intelectuais, sempre foram uma motivação para a evolução dos jogos até os tempos modernos.

2.1 Game Designer

Na indústria de jogos digitais, a pessoa responsável pela concepção do jogo, é o *game designer*. Por volta dos anos 80, existia apenas um tipo de *game designer*, que era chamado de programador. Nessa época o programador era responsável pelo *design*, programação, som e arte do jogo. Hoje em dia, jogos são maiores e dispõem de pessoas especializadas em cada uma dessas áreas. Nos últimos 5 anos, a especialização tornou-se ainda mais evidente. Em jogos que tem um grande investimento, considerados AAA (*triple A*) na indústria, existe subdivisão dentro de cada uma dessas áreas e na área de *game designer* não é diferente.

Assim como o nome *game designer* não é traduzido dentro da indústria de jogos nacional, os nomes das subdivisões também circulam em inglês dentre as pessoas do mercado. As principais subdivisões descritas por Brathwaite (2009) são: *lead designer, level designer, content designer, game writer, system designer, technical designer e UI designer*. Abaixo a descrição de cada um desses cargos.

- ***Lead designer*** - é o responsável geral pelo conceito do jogo. O líder decide como o projeto é documentado, define a mecânica principal do jogo e lidera os outros *game designers* do projeto. O *designer* líder geralmente é quem está

mais próximo dos demais lideres (líder de programação, líder de arte, líder de som) para decidir rumos importantes do jogo. Bons jogos dificilmente sairão da cabeça de uma só pessoa, logo, o *designer* líder não fará tudo sozinho, mas será o elo que organizará as diversas opiniões que aparecerão durante o desenvolvimento do jogo.

- ***Level Designer*** - é o responsável pela criação das fases (*levels*) do jogo. Tem um papel muito importante no balanceamento do jogo em si, criando uma progressão que seja motivadora para o jogador. Tipos de *level designers* ainda podem ser subdivididos dependendo do estilo de jogo.
- ***Game Writers*** - são responsáveis pela escrita do roteiro do jogo.
- ***Content Designer*** - está envolvido com a narrativa geral do jogo. Ele escreve as histórias do mundo, das personagens e objetos dentro jogo. Com o crescimento de jogos cada vez mais realistas e com diversos detalhes de mundo e histórias, esse profissional está sendo demandado mais do que nunca. Designers de conteúdo são bastante confundidos com os *Game Writers*, mas eles são pessoas diferentes dentro da indústria de jogos atual.
- ***System Designers*** - São os responsáveis por sistemas particulares dentro do jogo. Por exemplo, em um jogo de luta o *system designer* projetará como os personagens vão se movimentar em cada golpe, como a plateia da luta irá se comportar, etc.
- ***Technical Designer*** - Estes são considerados parte programadores e parte designers. São os responsáveis pela comunicação entre o designer e o time de programação. Trabalham com programação e escrevem alguns códigos e scripts durante o processo de desenvolvimento.
- ***UI Designers*** - São os responsáveis pelas interfaces do jogo. Esse profissional precisa ter boa noção de usabilidade pois irá projetar a forma como o jogador vai interagir com os menus e telas do jogo.

Além da classificação por área de atuação dentro da indústria de jogos o *game designer* é classificado por seu nível de experiência:

- ***Senior Designers*** - Um designer sênior deve ser capaz de se encaixar em qualquer das atividades listadas acima. Mas geralmente são encaixados na função de líder.
- ***Junior Designers*** - Geralmente é onde todo designer começa sua carreira,

pode ocupar qualquer atividade listada, menos a de líder.

- ***Designer Director / Creative Director*** - é o chefe geral da equipe de design. É o elo entre o time de *design* e a gestão

2.2 Balanceamento de Jogos Digitais

Mas o que vem a ser um jogo balanceado? Dentro das definições encontradas no dicionário[4], temos que balanceado significa: a ação de balancear, equilíbrio de carga, boa relação proporcional. Porém, o que é esse balanceamento no contexto de jogos digitais?

Para jogos digitais, a definição é um pouco mais específico, segundo Sirilin, um jogo estará balanceado se há um número razoavelmente grande de opções disponíveis para o jogador durante o jogo, especialmente durante os níveis mais altos para os jogadores com maior experiência (Sirlin, 2008 tradução nossa). Já na definição de Janssen balanceamento de jogos é um conceito no design de jogos onde os pontos fortes de um personagem ou uma estratégia particular, são compensados por uma desvantagem proporcional em outra área para evitar a dominação de um personagem ou uma abordagem de jogo (Janssen, 2011 tradução nossa). Com essas definições, há uma certa tendência a pensar que o balanceamento de um jogo seria uma necessidade de tornar justo o mundo em que o jogo está inserido, porém ainda não é bem este o ponto.

Um jogo estar balanceado não significa necessariamente que ele seja justo. Segundo Newheiser (NEWHEISER, 2009) o balanceamento de um jogo é na verdade o gerenciamento de cenários de injustiça. Um jogador iniciante enfrenta várias injustiças durante sua etapa de aprendizagem e conhecimento do jogo, essa fase se torna desafiadora a ponto do jogador crescer e adquirir habilidades para vencer as injustiças. É nesse momento que o *game designer* se depara com um dos grandes desafios do balanceamento, pois nas fases iniciais do jogo se este tem um baixo nível de desafio pode se tornar chato e fazer com que o jogador não queira continuar jogando e ao mesmo tempo se o desafio for grande demais, a ponto de frustrar o jogador, ele também não irá continuar na sua evolução. Logo, deixar o jogo completamente balanceado, a ponto de deixa-lo justo para todas as entidades não é o

[4] O dicionário consultado foi o Michaelis, disponível em: http://michaelis.uol.com.br

objetivo da atividade de balanceamento. Além disso, há várias discussões (NEWHEISER, 2009; SCHREIBER, 2009) que afirmam que nenhum jogo pode ser considerado completamente balanceado, até porque cada jogador é diferente do outro. Nem mesmo jogos simétricos de tabuleiro, como é o caso do xadrez em que os dois jogadores iniciam com a mesma quantidade de peças e podem fazer os mesmo movimentos. No xadrez, o jogador com as peças brancas tem uma vantagem por ser o primeiro a iniciar o jogo, e isso já pode ser considerado uma ligeira injustiça entre os jogadores. Porém como dito acima, a justiça não é a base do balanceamento, e é interessante que cenários de injustiça sejam incorporados nos jogos para tornar estes mais interessantes.

Existe uma parte da atividade de balanceamento que é a configuração inicial do jogo. É comum em jogos digitais, dos mais diversos gêneros, o jogador inicialmente ter o direito de escolher o seu time inicial, ou seu carro inicial, ou até mesmo a raça de seu personagem inicial, entre outros tipos de escolha existentes. Nesses jogos, a escolha inicial reserva uma quantidade de características que acabam sendo cruciais para o balanceamento do jogo. É importante que essa escolha inicial não seja determinante e exclusiva para o sucesso do jogador, quando isso ocorre acontece o que é chamado de **estratégia dominante** (NEWHEISER, 2009). A estratégia dominante é algo que deve ser evitado no momento do balanceamento das configurações iniciais de um jogo. Ao ter um personagem que sempre vence, ou um único caminho possível para atingir o sucesso, ou até mesmo um time que é muito superior aos outros times, que podem ser escolhidos, é esperado que todos os jogadores tenham a tendência de escolher essa estratégia para obter o sucesso, diminuindo bastante as possibilidades de variações dentro do jogo. Durante a atividade de balanceamento, o *game designer* precisa estar atento às possíveis estratégias dominantes que podem surgir e evita-las ao máximo.

Nessa discussão de balanceamento com base na justiça, injustiça e estratégias dominantes, uma definição que traz a tona todas essas características é que o balanceamento de um jogo depende basicamente do contexto em que ele está inserido. No artigo de Schreiber (SCHREIBER, 2009), pode ser visto quatro principais contextos em que os jogos podem ser classificados, como: jogos de um jogador apenas, jogos de vários jogadores, jogos com múltiplos caminhos e jogos de sistemas similares. Esses contextos são cruciais para que o *game designer* entenda como deve ser feito o balanceamento do jogo. Por exemplo, em um jogo para dois jogadores, a

preocupação do GD é tornar o mais próximo possível do justo, para que os dois jogadores possam duelar sem que um tenha privilégios na frente do outro, mesmo sendo impossível estar completamente justo como foi descrito acima. Já em um jogo para um jogador apenas, o conceito de injustiça é algo que não necessariamente é encarado como ruim e acaba sendo um desafio a mais para a evolução do jogador.

Entendendo o que é o balanceamento é importante então entender como ele vem sendo feito hoje na indústria de jogos e quais são as críticas e sugestões para melhorias nesse processo.

2.3 Balanceamento manual de jogos

Ao pensar num jogo, o *game designer* não é capaz de definir o valor de todos os atributos das entidades que existirão para deixá-lo balanceado. Alguns valores desses atributos precisam ser experimentados para que o *game designer* atinja o objetivo que ele imagina para o jogo. Alguns exemplos de atributos que provavelmente são ajustados durante os testes do jogo são: qual a velocidade que um certo inimigo deve se mover, para que o jogador se sinta desafiado e que ao mesmo consiga destruí-lo, ou qual a duração ideal para o combustível de um carro de corrida, ou qual o tamanho do inimigo e quantos golpes esse precisa sofrer para ser vencido, entre outros atributos que só são realmente definidos no momento em que o *game designer* joga e executa o balanceamento.

O balanceamento é então realizado quando o jogo está *"jogável"*, ou seja, quando o núcleo do jogo está implementado, o *gameplay*. Nesse momento o *game designer* inicia o processo de balanceamento. Nesse processo, é comum que o programador seja muito solicitado para fazer os ajustes necessários. É a partir daí que a interação entre o programador e o *game designer* passa a ser constante. Para que os ajustes dos atributos, apontados pelo GD, sejam realizados, ele depende bastante da disponibilidade do programador.

As técnicas para balanceamento de um jogo dependem muito do próprio jogo e da experiência que o *game designer* tem. Num artigo bastante prático Schreiber (SCHREIBER, 2009) mostra três técnicas de balancear um jogo: usando matemática, usando o instinto do *game designer*, ou testando com jogadores.

Usar a matemática para balancear jogos é uma técnica que pode ser a mais segura, se o jogo permitir esse tipo de balanceamento, mas pode ser perigosa, se os cálculos não fizerem muito sentido. Nesse caso o *game designer* precisará criar uma função de

custo e benefício para cada uma dos atributos que deseja balancear num jogo e aplicar essa fórmula para todos os casos. Por exemplo, num jogo de luta para vários jogadores, para cada personagem pode haver um poder diferente para cada golpe, força do chute, força do soco, força do poder especial. Balancear a força de cada golpe entre os diversos personagens pode ser uma função que dê pesos diferentes em cada golpe, mas que no final, o custo benefício de escolher um ou outro personagem acabe sendo o mesmo.

Usar o instinto é o que muitos *game designers* acabam fazendo, mas exige uma certa experiência com aquele tipo de jogo e aquele tipo de público. Pode ser uma técnica muito rápida e eficaz em alguns casos, dependendo da experiência do GD, mas também corre o risco de ter um jogo mal balanceado por falta de experimentação com o publico alvo. Nessa técnica, é comum que o GD trabalhe na base da tentativa e erro, pois, algumas variáveis que ele imagina ficarem boas provavelmente não estarão, quando aplicadas ao jogo em conjunto com outras. As idas e vindas dessa técnica acabam por atrasar o processo, pois, do momento que o GD percebe a necessidade de mudança, até o momento que pede ao programador para executa-la, muita informação pode ser perdida.

A técnica de testar com a audiência aparenta ser a mais segura, pois o cliente está sendo colocado como espectador, porém é um tanto custosa. Conseguir um quórum significativo de pessoas que representem o publico alvo de um jogo é algo difícil de realizar. O que muitos produtores de jogos vêm fazendo é lançar versões beta do jogo para determinados usuários (fãs dos jogos) e a partir daí colher os feedbacks deles.

2.3.1 Problemas enfrentados no balanceamento manual

Independentemente da técnica escolhida para o balanceamento do jogo, acaba sempre sendo um trabalho que envolve o *game designer* e o programador. Essa interação entre programador e GD nem sempre é possível de ser feita em determinados momentos, pois cada profissional pode estar envolvido em outras atividades. Isso toma parte do tempo do programador que poderia estar sendo usado para o desenvolvimento de novas funcionalidades já planejadas, atrasando a entrega do jogo. E por outro lado, limita a liberdade do *game designer* que acaba não fazendo todos os ajustes que desejaria para as variáveis, pois sente que atrapalha o desenvolvimento das funcionalidades novas. A comunicação entre GD e equipe de desenvolvimento é um dos maiores problemas citados em entrevistas feitas com

profissionais da área (MACHADO et al., 2012) que afirmam o quão dependentes o *game designer* e o programador acabam ficando um do outro, a ponto de atrasar o desenvolvimento do jogo.

A descontinuidade no trabalho é algo bem difícil de ser medido, mas é evidente que atrapalha no desenvolvimento e ajustes de qualidade do jogo. Do momento em que o *game designer* descobre uma necessidade de mudança, ao momento que ele vai comunicar isso ao programador, para que a mudança ocorra de fato, há uma descontinuidade no trabalho, exatamente por conta da dependência existente entre o *game designer* e o time de desenvolvimento. Isso limita também a experimentação, pois 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.

A rastreabilidade das mudanças também acaba sendo comprometida da forma como o balanceamento é executado usualmente. Em alguns momentos o *game designer* não tem certeza de qual a melhor configuração de atributos as entidades devem ter, sendo muito comum que experimentações novas fiquem piores do que outras feitas anteriormente, nesse caso, se o game designer não tiver uma forma de organizar todas as mudanças que está fazendo acaba perdendo um trabalho realizado. Existem várias criticas (COOK, 2011; LANG, 2009; MACHADO, 2009; MACHADO et al., 2012) que corroboram com a afirmação da dificuldade que os *game designers* têm em rastrear as mudanças que são feitas durante o desenvolvimento do jogo. Mesmo quando tem o GDD para auxiliá-los a guardar o histórico das mudanças, muitos não costumam atualizar o GDD durante o desenvolvimento e atributos descritos nas etapas de conceituação do jogo acabam ficando desatualizados.

2.4 Recomendações para um balanceamento eficiente

Diante de todos os problemas citados, ficam claros alguns requisitos ou recomendações essenciais para uma ferramenta que ajude a deixar o balanceamento de um jogo mais fácil e eficiente. Esses requisitos são:

- Permitir autonomia ao *game designer* em relação ao programador, para que aquele tenha a liberdade de executar o balanceamento dos atributos das

entidades do jogo. Com isso, tanto *game designer* quanto programador ganham, pois deixam de depender um do outro para executar suas atividades.

- Enxergar a consequência da mudança dos atributos em tempo real. É interessante para o *game designer* ver o que está mudando no momento em que a mudança ocorre. Isso dá um dinamismo maior ao processo de balanceamento.
- Conseguir ter controle no tempo do jogo a ponto de poder voltar, adiantar e pausar livremente o que aconteceu, para melhor ajuste das variáveis. Junto com o requisito anterior de enxergar as mudanças em tempo real, se *o game designer* puder ter um controle ainda maior da ação das mudanças em diferentes momentos do jogo, tendo um controle do tempo, vai facilitar e acelerar o seu trabalho.
- Salvar as configurações que foram criadas e compará-las com configurações anteriores. A rastreabilidade dessa forma ficaria garantida para futuros arrependimentos ou comparações.

3 O Estado da Arte

O problema do balanceamento é algo bastante discutido na academia (ANDRADE, 2006; ANDRADE et al., 2004; MACHADO et al., 2012). A busca por melhorias no processo de balanceamento é discutida por diversos estudos na área de tecnologia da informação. O estado da arte no contexto de balanceamento de jogos pode ser dividido em balanceamento automático e balanceamento manual.

3.1 Balanceamento Automático

Segundo Andrade (ANDRADE, 2006), "O balanceamento dinâmico (ou automático) 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". O balanceamento automático de jogos digitais é feito usando técnicas de inteligência artificial. Alguma dessas técnicas são: sistemas de classificadores genéricos, arvores de decisão, scripts dinâmicos, aprendizagem por reforço e aprendizagem supervisionada. (ANDRADE et al., 2004).

Embora técnicas de aprendizagem e balanceamento automático (conhecido no meio acadêmico como *dynamic difficulty adjustmen*t - DDA), sejam bastante populares na academia, ainda é incomum na indústria de jogos.

3.1.1 Dificuldades de aplicação do balanceamento automático na indústria de jogos

A tentativa de usar o balanceamento automático na indústria é criticada tanto pelas pessoas que produzem os jogos quanto pelos próprios jogadores. Como o jogo passa a ser automaticamente adaptável, pessoas diferentes vão acabar tendo sensações diferentes ao jogar, o que muitas vezes não é desejado pelo *game designer.*

Imagine, por exemplo, um jogo que será usado numa competição e que usa DDA para balancear o *gameplay*. Não há como não duvidar do resultado final dessa competição, uma vez que o balanceamento automático deixará o jogo muito mais

difícil para o jogador mais experiente que para o iniciante, que disputará a mesma competição num nível mais fácil.

Outra crítica ao DDA é que, com o balanceamento automático, é comum não existir a escolha inicial por parte do jogador de qual nível ele deseja jogar, pois, já que o jogo aprende com o jogador, aquele tentará encaixar este no nível ideal à sua experiência. Apesar de parecer mais justo, isso não é o desejo de todos os jogadores. 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. É aconselhável que esse desejo do jogador seja sempre possível de acontecer. Dessa forma, em jogos que usam DDA, é recomendável que o jogador possa desligar essa funcionalidade.

Uma outra crítica pode ser encontrada no artigo de Adams (ADAMS, 2012) para a revista gamassutra[5] ao DDA, ele, que é *game designer*, assume que DDA é bom para alguns jogos, porém apresenta defeitos em outros. Ele afirma que alguns jogadores detestam DDA, e que se for utilizar essa funcionalidade em um jogo é altamente recomendado que o jogador consiga desliga-la. Existem muitos jogadores que não gostam quando sentem que o jogo está trapaceando a favor deles quando estão ficando ruins. Para eles, jogar e tentar vencer um jogo que é difícil, é a grande diversão, nem que isso dure 500 rodadas para atingir o sucesso. Outro problema é que jogadores podem aprender como o DDA funciona, sendo piores para serem melhores, e dessa vez é o jogador que trapaceia para conseguir êxito no jogo.

O DDA pode também criar absurdos num jogo. Por exemplo, em jogos de corrida se o jogador bate o carro e perde muito tempo, os seus oponentes ficam mais lentos para que aquele consiga alcança-los. O mesmo ocorre no inverso, se o jogador vai muito rápido, seus oponentes estarão sempre perto para deixar o jogo mais desafiador. Esse é um efeito existente na maioria dos jogos de corrida conhecido como *rubber-band*. É aceitável em alguns jogos, em que não necessariamente o jogador está preocupado com a realidade dos fatos, porém, em jogos de simulação, fica muito evidente e acaba frustrando o jogador que procurava por realismo.

Se for bem implementado o DDA tende a deixar todos os desafios no mesmo nível. 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

[5] Gamassutra é um portal destinado a discussões e artigos relacionados a jogos dos mais diversos gêneros e tecnologias. Mais informações acessar www.gamassutra.com

esta sempre na mesma altura, ou um filme que não tem momentos de clímax. O clímax é algo buscado no balanceamento do jogo e muitas vezes a utilização de DDA pode cortar esses momentos.

O DDA é algo que vem melhorando muito com o passar dos anos e das pesquisas, porém, na realidade, ainda não conseguiu substituir o balanceamento manual feito pelo *game designer*. Além de ser algo bastante difícil de ser bem implementado, muitas vezes não é o que o jogador deseja. Então, vale muito a pena pensar bem antes de usar em jogos comerciais, pois, consome um longo tempo de desenvolvimento e nem sempre atinge o objetivo esperado. Muitas vezes o velho modo de balancear pode ser o mais indicado.

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 (ANDRADE, 2006; ANDRADE et al., 2004; APONTE; LEVIEUX; NATKIN, 2009) 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.

É evidente que cada empresa dispõe de técnicas e ferramentas apuradas para chegar ao nível atual de balanceamento dos jogos que vem sendo lançados no mercado. Algumas empresas dão indícios de como isso pode estar sendo feito em documentos de *post mortens* (SHEFFIELD, 2009) publicados em revistas especializadas, mas ainda assim a maioria dos detalhes são omitidos.

Porém, alguns trabalhos surgem diante de todo esse sigilo e despontam como soluções interessantes e ditam tendências na indústria e no estado da arte do balanceamento manual dos jogos, são eles: as pesquisas de Bret Victor (VICTOR, 2010), Frogatto & Friends, e o projeto aberto *Live Scratchpad* da *Mozilla* (VIVIER, 2012).

Um dos requisitos apontados na seção 2.4 (enxergar a consequência da mudança dos atributos em tempo real) é algo que vem sendo bastante buscado em alguns trabalhos desenvolvidos e foi uma inspiração inicial para o desenvolvimento desse trabalho, que busca dar ao *game designer* maior agilidade no processo de balanceamento.

Um pesquisador que se destaca nas técnicas de mostrar mudanças em programas em tempo real é Bret Victor (VICTOR, 2010). A solução apresentada por Victor, usando, como exemplo, o jogo Braid[6], mostra como é possível aplicar, de uma forma bastante eficaz, a mudança de variáveis dentro de um código e mostrar, em tempo real, essas mudanças.

Apesar do principal objetivo dele não ter muita ligação com o balanceamento das variáveis, mas sim, com a criação de um sistema para ensinar pessoas a programar de uma forma diferente da usual (ver Figura 3.1) as técnicas usadas para expor ao programador como cada variável de código influencia o comportamento das entidades é bem semelhante ao que é desejável mostrar ao *game designer* no momento que ele deseja balancear um jogo.

A mudança de uma variável no código significa a mudança de um atributo para uma entidade, que é exatamente o que o *game designer* deseja experimentar durante a etapa de balanceamento de um jogo. Como trata-se de uma ferramenta para a aprendizagem de programação, a solução de Victor acaba sendo orientada ao código fonte, o que é bem diferente do universo em que o *game designer* está acostumado a trabalhar, logo, esta solução é uma inspiração para encontrarmos a solução ideal para o problema do balanceamento. Porém, é preciso ter um certo cuidado em relação a como esses atributos serão expostos ao GD, pois ele não deve manipular diretamente o código fonte.

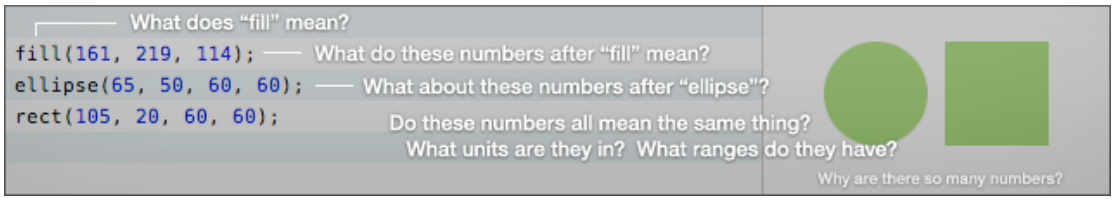


Figura 3.1 – Exemplo do trabalho de Bret Victor com o objetivo de educar os programadores.
Fonte: http://worrydream.com/LearnableProgramming/

Victor é apontado como inspiração em vários trabalhos que vem sendo construídos. Um deles é o projeto *Live Scratchpad* da *Mozila* (VIVIER, 2012), um *plugin* para o navegador *firefox* que permite que os desenvolvedores vejam em tempo

[6] http://www.youtube.com/watch?v=PlI-gPu3SPI

real as modificações que estão fazendo no código fonte, mais uma vez, é algo muito direcionado a programadores, mas é evidente a busca por visualização das mudanças em tempo real também nessa solução.

Na mesma linha por verificação de mudanças de variáveis e atributos em tempo real e no balanceamento de jogos está jogo Frogatto & Friends, um jogo multi-plataforma que apresenta um editor de fases onde o usuário pode alterar as características da fase e ver as alterações em tempo real. Com versões para iOS, Android, Mac OS, Windows e Linux, o Frogatto & Friends é um jogo que apresenta como diferencial uma engine de criação de levels onde o usuário pode criar diversos levels em 2D e edita-los em tempo real. A semelhança com o esta pesquisa se dá por conta de que os elementos do level podem ser editados em tempo real sem muito esforço de programação. No Frogatto & Friends, é usada uma linguagem própria para fazer as mudanças de variáveis chamada de FSON. Mesmo não sendo preciso mexer diretamente no código fonte do jogo, apenas sendo necessário conhecimento nos scripts da linguagem FSON, ainda não é um ambiente amigável para ser usado pelo *game designer*.

Outro problema identificado é que o Frogatto & Friends consegue fazer jogos bem característicos do estilo plataforma, com limitações de paletas de cores e formatos de objetos. É uma ferramenta muito interessante para jogos desse estilo e protótipos de jogos maiores, mas não apresenta a facilidade de fazer outros tipos de jogos. Também não fica claro como é feita a exposição das variáveis dentro da engine usada.

4 Encontrando a Solução

Antes de chegar ao nível de implementação das ferramentas para atingir os objetivos listados na seção 1.3, vale a pena uma reflexão sobre qual a filosofia por trás desta necessidade de implementação de algo que facilite principalmente o trabalho do *game designer*. Relembrando o objetivo principal (tornar o *game designer* mais autônomo na fase de balanceamento de um jogo, menos dependente do trabalho do programador para balancear os atributos das entidades, sem que ele precise entrar em contato direto com o código fonte) é importante destrinchar com mais detalhes o que este objetivo realmente busca.

Para deixar o *game designer* autônomo do programador e para que ele possa alterar os valores dos atributos das entidades do jogo, se faz necessário a criação de mecanismos que façam com que estes atributos sejam expostos e possam ser manipulados intuitivamente, sem requerer contato direto com o código fonte. Isto é necessário porque não é da competência dos *game designers* entender nuances de programação.

Além de expor as variáveis para o *game designer*, também como objetivo do trabalho, procura-se facilitar a forma como o programador deve trabalhar, para que esta exposição seja feita sem grandes mudanças na forma com que ele já está acostumado a implementar o jogo, pois, não faria sentido apenas facilitar um lado do desenvolvimento, dando ferramentas ao *game designer*, e dificultar o outro lado, desta forma o ganho não seria completo.

Diante deste cenário, algumas soluções foram discutidas no período de pesquisa até ser encontrada a solução atual. Idealmente, o que estava sendo procurado era uma forma de enxergar, no código fonte de qualquer jogo, quais os pontos que precisariam ser mudados para que fosse possível expor os atributos necessários ao balanceamento do jogo pelo *game designer*. Esta forma de enxergar as variáveis passou por algumas ideias como:

- utilização de arquivos de configuração,
- utilização de uma linguagem de domínio específico (DSL – *Domain Specific Language*),

- através de engenharia reversa e exploração de código compilado,
- criando uma extensão na *engine* de desenvolvimento do jogo,
- criando um *framework* que pudesse ser acoplado ao código do jogo, criando um protocolo de comunicação entre o jogo e alguma interface de controle para o GD.

Algumas soluções pareceram mais promissoras que outras ao se analisar os objetivos e ao se conversar com os *game designers*, como veremos a seguir.

4.1 Utilizando arquivo de configuração de variáveis

Utilizar um arquivo de configuração, onde as variáveis ficariam todas escritas e o *game designer* poderia manipula-las antes de executar o jogo, é uma forma que muitas empresas acabam adotando na indústria de jogos de hoje (PIRANHA, 2012; ROMERO, 2012). É uma forma prática e que acaba resolvendo boa parte dos problemas de comunicação entre o game designer e o programador, pois estando com o arquivo de configuração em mãos, de forma que as mudanças feitas no arquivo se reflitam no jogo, o game designer pode mudar as variáveis sem precisar recorrer ao programador.

As formas com que essa abordagem pode ser feita variam muito com a tecnologia e a linguagem de programação utilizada no desenvolvimento do jogo. Esses arquivos geralmente apresentam algum nível de sintaxe que precisa ser respeitado, porém, é bem mais simples do que a manipulação direta no código fonte, ficando perfeitamente viável e possível de ser manipulado pelo *game designer* sem maiores dificuldades.

Apesar de ser uma forma bastante promissora e simples para execução do balanceamento do jogo por parte do *game designer*, essa solução não atende completamente aos requisitos para um balanceamento eficiente, que foram listados na seção 2.4. Os dois principais requisitos que não foram atendidos são: a mudança em tempo real das variáveis e a interface amigável ao *game designer*.

Com a utilização de arquivos de configuração, o *game designer* precisaria, sempre que mudasse alguma variável dentro do arquivo, executar novamente o jogo para ver a mudança. Isso se dá por conta da forma como a implementação da leitura dos arquivos de configuração é feita. Os arquivos são lidos no início do jogo, logo, qualquer mudança feita quando ele estiver sendo executado não é refletida nas suas entidades. É possível fazer com que esse arquivo fique sendo lido sempre que alguma

mudança seja feita, porém, essa forma acabaria prejudicando a performance do jogo, o que traria mais prejuízos do que benefícios.

Apesar de os arquivos de configuração apresentarem uma sintaxe simples, que o *game designer* pode facilmente dominar, esta pode não ser a melhor maneira de apresentar atributos para que ele os manipule. Muitas vezes, os nomes das variáveis são estranhos e acabam desmotivando o GD a ir mais a fundo no balanceamento. Essa característica foge do objetivo de ter uma interface amigável e próxima da realidade dos *game designers*.

Mesmo com esses pontos negativos levantados acima, usar arquivos de configuração é uma forma prática e simples de persistir os valores dos atributos das entidades que são balanceados pelo *game designer*. Por esse motivo, foi utilizado neste trabalho esse método para salvar e carregar balanceamentos feitos pelo *game designer*. Com isso, a cada configuração que o GD ache interessante, ele pode salvar num arquivo de configuração que pode ser editado manualmente, ou carregado de volta para a interface de manipulação dos atributos para posteriores ajustes.

4.2 Utilizando uma Linguagem de Domínio Específico (DSL)

Uma DSL é uma linguagem pequena, geralmente declarativa, que oferece poder de expressão orientado a um determinado domínio de problema (FURTADO, 2012). Muitas vezes, DSLs são traduzidas para linguagens de propósito geral e ficam escondidas dentro da implementação, tornando ainda mais transparente o seu uso. É uma linguagem bastante limitada em relação a complexidade de alguns problemas, porém muito poderosa para resolver problemas específicos (FURTADO; SANTOS, 2010).

Alguns benefícios conhecidos do uso de DSL segundo Furtado (FURTADO, 2012) são:

- DSL permite que a solução seja expressa na linguagem do problema. Especialistas em resolverem esses problemas podem usar a DSL naturalmente, pois ela estará retratando o problema em uma linguagem com que ele já está bem familiarizado.
- Programas feitos usando DSL são concisos e podem ser facilmente reusados.
- Uma DSL bem feita aumenta a produtividade, facilita a manutenção e a portabilidade.

- DSLs permitem a validação e a otimização no nível do domínio.
- DSLs facilitam os testes.

DSLs se diferenciam de arquivos de configurações por seu poder de computação, que mesmo sendo mais limitado que o de linguagens de proposito geral, como C++, Java e Actionscript 3, apresentam, além de uma sintaxe, uma semântica bem definida. Outra característica interessante do uso de DSLs é que elas tanto podem ser textuais, quanto visuais (FURTADO, 2005), podendo então ser uma boa saída para a utilização de interfaces amigáveis a serem usadas pelo *game designer*.

Por outro lado, ainda segundo Furtado, algumas desvantagens em usar DSLs podem ser apontadas, como:

- O alto custo para o projeto de uma DSL as vezes não compensa sua elaboração
- O custo na educação dos usuários
- A dificuldade de achar escopo bem definido para o uso da DSL
- Quando usar DSL e linguagem de propósito geral?
- Potencial perda de desempenho das aplicações que usem alguns tipo de DSL, como DSLs que são interpretadas em tempo real.

No projeto da interface gráfica utilizada neste trabalho para comunicação com o *game designer,* foram utilizados vários conceitos de DSLs visuais, onde existe um conjunto de entidades que possuem uma semântica bem definida para cada um dos atributos que podem ser manipulados pelo GD. Na seção 4.3, onde mais detalhes da implementação são apresentados, esses conceitos ficam mais claros.

4.3 Utilizando um *Framework* e protocolo de comunicação

Segundo Govoni (GOVONI, 1999), *framework* é uma coleção abstrata de classes, interfaces e padrões dedicados a resolver uma classe de problemas através de uma estrutura extensível e flexível. Um *framework* assume um papel ativo dentro da implementação de uma solução. Difere de uma biblioteca de classes exatamente nesse ponto, pois uma biblioteca tem funções passivas à chamada da aplicação, nesse caso a aplicação é a responsável por fazer as ligações e chamadas às funções da biblioteca. Já no caso do *framework*, além de possuir funções passivas que podem ser chamadas pela aplicação, ele também pode chamar as funções da aplicação, com isso a implementação feita pelo programador resume-se a implementar o comportamento

das funções que serão chamadas pelo *framework* quando for necessário. A Figura 4.1 exemplifica com um esquema esta diferenciação.

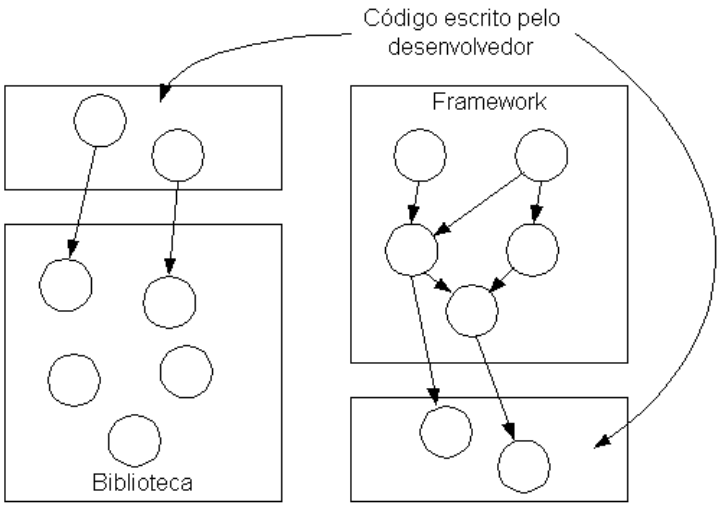


Figura 4.1 - Diferença entre Framework e Biblioteca de classes (SAUVÉ, 2005)

A característica ativa do *framework* pode ser observada quando há a necessidade de mudança de uma variável por parte do GD. O *framework* capta essa necessidade e passa a mudança para a aplicação, no caso, para as entidades implementadas pelo jogo. Mais detalhes de como essa mudança é feita podem ser vistos na seção 5 que trata da implementação do *framework* em questão.

O *framework* se torna uma opção para uma estrutura ativa na mudança das variáveis e ajuda o programador a expor essas variáveis de forma fácil, sem precisar mudar muito a forma com que ele está acostumado a programar os jogos.

Até aqui, foram apresentadas duas estruturas que acabam não se comunicando bem, o jogo, junto com o framework e a DSL, representada pela interface de mudança das variáveis. Essas estruturas podem ser implementadas utilizando as mais diversas linguagens de programação, como será detalhado na seção 5, e para que a comunicação entre essas estruturas seja estabelecida, é necessária a criação de um protocolo de comunicação entre as partes (ver Figura 5.2). As mensagens que são trocadas entre a interface gráfica e o jogo fazem com que essas duas estruturas sejam implementadas de forma completamente desconectadas, facilitando com isso o reuso desta interface pelos mais diversos jogos que usem o *framework* para expor suas variáveis. A seguir, será apresentado mais detalhes o protocolo estabelecido e como é feita a comunicação entre as estruturas.

5 Implementação do Xpose

Baseando-se no objetivo principal (tornar o game designer mais autônomo nas fases de balanceamento de um jogo), foi elaborado um *framework* que facilita a exposição de variáveis por parte do programador, sem que ele precise mudar muito a forma como está acostumado a trabalhar. Junto a este *framework,* também foi desenvolvida uma interface amigável ao *game designer* para que ele consiga executar as modificações nos atributos das entidades e, consequentemente, nas variáveis expostas pelo programador.

O *framework* de desenvolvimento, chamado de Xpose, apresenta uma forma de facilitar a implementação, quando existe a necessidade de expor variáveis do código de um jogo. O Xpose foi implementado usando a linguagem de programação orientada a objetos, Actionscript 3, porém os conceitos aqui definidos são independentes de linguagem de programação. Como contribuição principal deste trabalho, está então a proposição de um *guideline* de desenvolvimento: um conjunto de boas práticas e arquitetura, que podem ser utilizadas para a criação de um *framework* que expõe variáveis do código sem muito esforço por parte do programador e que sejam apresentadas de forma amigável ao *game designer*.

5.1 Solução inicial

Como citado na seção anterior, até chegar à implementação atual, algumas outras soluções foram testadas e sofreram melhorias após a execução dos testes. É interessante observar as dificuldades encontradas em soluções prévias para melhor entendimento de como a solução atual chegou ao ponto em que se encontra.

Inicialmente, foi elaborada uma solução em que o *framework* pudesse ser implementado junto a *engine* de desenvolvimento do jogo, como pode ser observado na arquitetura inicial, Figura 5.1, em que o framework, a *engine* e o jogo fazem parte de um mesmo sistema.

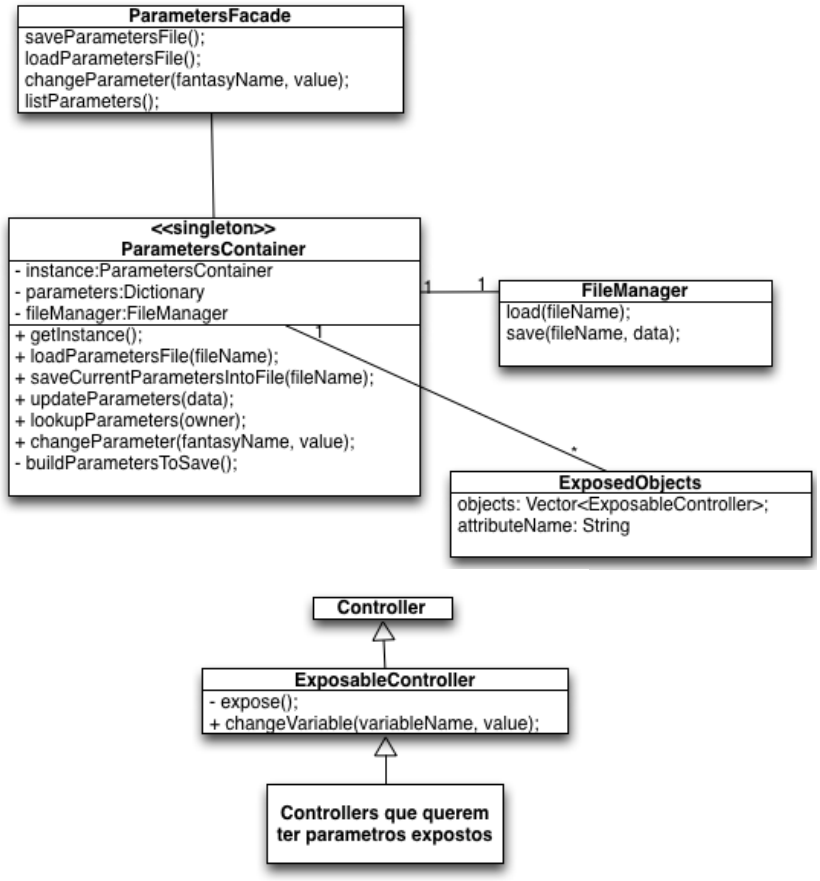


Figura 5.1 – Arquitetura definida inicialmente

Algumas classes importantes de serem destacadas nessa arquitetura são:

- ***ParametersFacade*** – Representa a classe fachada da implementação. A partir dela, a interface gráfica acessa os métodos para exposição e manipulação das variáveis. Nessa implementação, apenas algumas poucas operações eram possíveis, como: salvar e carregar uma configuração de variáveis, listar os atributos expostos e mudar o valor de um atributo.
- ***ExposableController*** – Classe herdada pelos controladores (entidades) do jogo que desejam expor suas variáveis. Nessa implementação, o programador precisa fazer com que suas classes de controle das entidades (ex. *Player*, Inimigos, etc.) herdem desta classe e insiram uma anotação nas variáveis que desejam expor, para que o *framework* processe a classe e exponha essas variáveis. O fato de usar herança nessa arquitetura foi um problema que acarretou algumas mudanças e o abandono desta solução, como veremos mais a frente.

- ***ParametersContainer*** – *Singleton*(GAMMA et al., 1994) que guarda os parâmetros expostos dos *ExposableControllers*. Apresenta duas funções que merecem destaque na implementação: o *lookupParameters*, que é chamado pelo *ExposableController* para cadastrar as variáveis que deseja expor e o *changeParameter* que muda o valor de uma variável. É a classe que faz o principal trabalho dentro do *framework*, gerenciar os atributos das entidades. Essa classe foi utilizada na arquitetura atual do framework como pode ser visto nas seção 5.2.

Um problema encontrado na arquitetura elaborada, foi o alto nível de acoplamento entre as classes da *engine* e do *framework*. Esse alto acoplamento pode ser visualizado na estrutura de herança mostrada na Figura 5.1, onde a classe *Controller*, que faz parte da *engine* usada para desenvolver o jogo, é herdada pela classe *ExposableController*, que é uma classe do *framework* que implementa os métodos responsáveis por expor as variáveis (*expose*) e mudar uma variável (*changeVariable*). As classes do jogo que representam as entidades e detêm as variáveis que representam seus atributos, normalmente, iriam herdar da classe *Controller*. Com a inserção do *framework* elas passariam então a herdar da classe *ExposableController*. Esta solução apresenta vários prós, como: deixar o código mais elegante, encapsular o *framework* dentro da *engine*, fazendo com que qualquer jogo que utilize esta *engine* consiga facilmente usar o *framework*, entre outras vantagens que a engenharia de software proporciona com esse tipo de arquitetura.

A fragilidade deste modelo ficou clara durante os testes realizados em empresas reais. Os jogos que já estavam em fase de implementação ou de balanceamento, precisariam sofrer muitas mudanças no código para implantar o *framework*, o que complicava a implantação da solução. Outro problema que também foi identificado, foi que nem sempre as *engines* possuem código aberto e a facilidade de inserir um nível a mais na hierarquia de classes não existia, impedindo o uso do *framework* em algumas situações.

Com isso, foi pensada uma outra abordagem para a solução, em que o jogo, o framework e a interface, que o *game designer* iria usar para executar o balanceamento fossem estruturas independentes.

5.2 Solução atual

A solução atual é composta por três estruturas, são elas: o jogo, o *framework* (Xpose) e a interface gráfica (Xpose GUI). Estas estruturas se comunicam através de um protocolo (Figura 5.2).

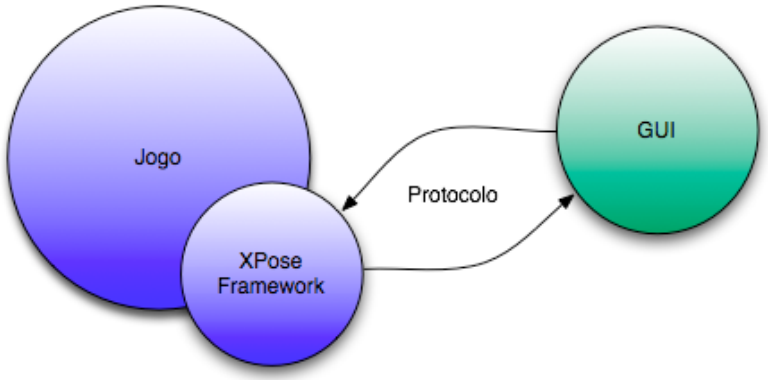


Figura 5.2 – Ligação entre jogo, Xpose Framework e Interface gráfica

A ligação entre o *framework* e o jogo é feita de forma a permitir um fácil desacoplamento entre o código do jogo e o código específico para o *framework*. Dessa forma, quando o jogo já se encontrar balanceado, o programador pode retirar as referências ao Xpose e à ferramenta de interface gráfica de balanceamento.

Como dito anteriormente, o *framework* foi desenvolvido usando a linguagem de programação Actioscript 3. A escolha de Actionscript 3 para desenvolver o *framework* foi feita pelo motivo de haver uma grande quantidade de jogos sendo desenvolvidos nas empresas candidatas a experimentação do *framework* usando essa linguagem. Porém, por usar princípios básicos de orientação a objetos, o *framework* pode ser implementado em qualquer linguagem orientada a objetos, seguindo-se as especificações mostradas neste trabalho.

A utilização de um protocolo de comunicação entre o *framework* e uma interface gráfica, desacopla os projetos de interface de balanceamento do projeto do jogo, possibilitando com isso que uma mesma interface de balanceamento seja utilizada em vários jogos que utilizem o *framework,* ou que interfaces sejam construídas para fazer balanceamento de jogos específicos. Esse protocolo apresenta todas as funções possíveis para a manipulação dos atributos das entidades dentro do jogo, são elas: listar os atributos expostos do jogo, salvar uma configuração de atributos, carregar uma configuração salva, mudar diretamente um atributo de uma entidade, visualizar

os valores dos atributos e visualizar os tipos de atributos expostos. Essas funções serão melhores detalhadas nas próximas subseções, quando será explicado como o protocolo foi implementado.

5.2.1 O Protocolo

Inicialmente, é necessário entender como é feita a comunicação entre o jogo e a ferramenta de interface de balanceamento usada pelo *game designer*. Como trata-se de duas aplicações diferentes, é necessário fazer uma ponte que facilite a comunicação entre essas estruturas. Essa ponte é o protocolo de comunicação, elaborado a partir das funções que se fazem necessárias para que a ferramenta de interface de balanceamento consiga "enxergar" o que foi exposto pelo programador dentro do jogo.

Na ciência da computação, é comum a utilização de protocolos para diversos fins: conexão entre máquinas, comunicação na rede, transferência de dados, entre outros usos, e sempre seguindo um conjunto de regras para efetuar uma comunicação eficaz entre estruturas diferentes e independentes (PERLMAN, 2001). Pensando nessas características, foi construído um protocolo para comunicar a estrutura do jogo com a estrutura da ferramenta de interface de balanceamento, ou das ferramentas, pois com a facilidade de ter estruturas independentes, é possível então criar várias ferramentas de interface diferentes, para se comunicar com o mesmo jogo, desde que estas obedeçam as mesmas regras estabelecidas pelo protocolo.

O protocolo pensado para o Xpose *Framework* tem as seguintes regras de comunicação:

- Listagem de atributos expostos – Possibilita à interface gráfica saber quais os atributos que foram expostos pelo programador utilizando o *framework*.
- Gravação de uma configuração de atributos – Possibilita salvar uma configuração de valores para os atributos feita pelo *game designer*. Uma configuração de atributos consiste no conjunto dos valores de todos os atributos expostos num instante de tempo em um jogo.
- Carga de uma configuração de atributos – Possibilita que uma configuração de atributos previamente salva, seja atribuída aos respectivos atributos de um jogo.
- Mudança de um determinado atributo – Possibilita que um atributo seja modificado diretamente, sem intervenção direta ao código fonte.

- Recuperação do valor de um atributo – Recupera o valor de um atributo específico.
- Recuperação do tipo de um atributo – Recupera o tipo de dado ao qual os valores dos atributos pertencem.

Para tornar a comunicação mais amigável ao universo do *game designer*, cada atributo pode ter um nome fantasia, que é diferente do nome da variável que o programador coloca no código. Isto foi estabelecido pois os nomes das variáveis dentro do código geralmente não são os mesmos que o *game designer* estabeleceu no GDD. Na maioria das vezes, até o idioma utilizado para a nomenclatura das variáveis dentro do código é diferente do idioma usado pelo *game designer* para definir um atributo da entidade. Além das abreviações comumente feitas pelos programadores nas variáveis, o que tornaria complicado para o *game designer* entender qual atributo ele está balanceando. Por exemplo, uma variável no código para o atributo "velocidade do soco" pode ser simplesmente escrita como "pVel" ou "velocity" ou "v", deixando muitas vezes incompreensível para alguém que não está acostumado com a leitura do código fonte. Por esse motivo, sempre que o protocolo vai se referir a um atributo, ou a uma lista de atributos, ele se utiliza do nome fantasia para troca de informações. O *framework* se responsabiliza por mapear os nomes fantasia nos nomes reais das variáveis, na seção 5.2.2 será mostrado como é feito esse mapeamento.

O protocolo de comunicação poderia ser implementado utilizando varias tecnologias diferentes, como: socket através de uma comunicação de rede, na própria linguagem das ferramentas (Actionscript 3), ou com uma linguagem diferente como javascript, java, C. Para este trabalho, escolheu-se utilizar javascript como linguagem de implementação do protocolo, baseando-se no mesmo princípio da escolha da linguagem de programação do *framework*, que é o que as empresas que serviram como candidatas a experimentação do *framework* estão mais acostumadas a trabalhar.

Um jogo construído em Actionscript 3 pode ser facilmente distribuído na *web,* e funciona bem em todos os navegadores para computadores pessoais, sendo incorporado no código html de uma página web. A utilização de javascript como linguagem para implementação do protocolo se faz possível pois, como o jogo e a interface gráfica estão no mesmo ambiente, o mesmo navegador e a mesma página html, então este possibilita que, através do javascript, mensagens sejam facilmente passadas entre os objetos presentes no html. Através da função:

document.getElementById(nome_do_jogo), é possível encontrar o jogo dentro do html e chamar as funções expostas pelo *framework* para comunicação com o protocolo. Na Tabela 5.1, estão as funções utilizadas no protocolo que implementa as possibilidades de comunicação listadas anteriormente nesta seção:

Funções	Descrição
getExposedParameters()	Retorna a lista dos nomes fantasia dos atributos expostos pelo framework. Esta lista servirá para que a interface saiba quais atributos foram expostos pelo programador utilizando o framework Xpose.
prepareParametersToSave()	Prepara uma estrutura em formato JSON, para ser salvo pela interface gráfica. Pega todos os atributos expostos e seus valores atuais para serem salvos.
loadParameters(data)	Passa uma os dados salvos anteriormente numa estrutura JSON, em uma estrutura de objetos reconhecida pelo framework.
changeParameter(fantasyName, value)	Muda o valor de um atributo específico passando-se o nome fantasia e o novo valor.
getParameterValue(fantasyName)	Retorna o valor de um determinado atributo ao se passar o nome fantasia correspondente.
getParameterType(fantasyName)	Retorna o tipo do valor de um atributo ao se passar o nome fantasia correspondente.

Tabela 5.1 –Implementação das funções do protocolo

No Anexo 1 : Protocolo de comunicação em javascript está a implementação completa do protocolo usado na implementação.

5.2.2 O Framework

Como discutido anteriormente, a linguagem que deve ser usada para a implementação do framework é a linguagem que os jogos que vão consumi-lo estão

implementados. Por esse motivo, não será dada ênfase a detalhes da linguagem na implementação, mas sim, às técnicas usadas para que este trabalho seja facilmente replicado em outras implementações.

Com a separação do *framework,* da ferramenta de interface gráfica de balanceamento e do jogo propriamente dito, a implementação do núcleo do framework ficou bastante simples como pode ser observado na arquitetura de classes na Figura 5.3. Basicamente, existem três estruturas (classes) que compõem o *framework*: *Xpose, ParametersCollection e ExposedObjects*. Na seção seguinte, serão apresentados os detalhes dessas estruturas e como elas contribuem para o funcionamento do framework.

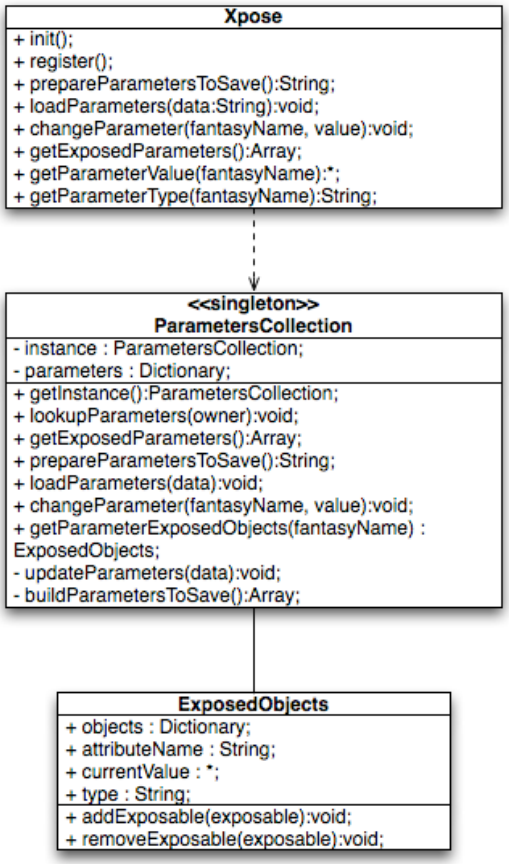


Figura 5.3 – Arquitetura atual do framework Xpose

5.2.2.1 Xpose

A classe Xpose é a entrada principal de acesso ao *framework*. É a partir dela que o usuário expõe os atributos das entidades e que o protocolo faz a comunicação com as ferramentas de interface de balanceamento. Além das funções que estão expostas no

protocolo, a classe Xpose apresenta duas outras funções que são de extrema importância para bom funcionamento do *framework*, a função *init* e a função *register*.

Como o próprio nome sugere, a função *init*, inicializa o *framework*. Ela deve ser chamada pelos jogos que desejam expor atributos de suas entidades. É nela que o *framework* declara as funções que irá expor para manipular os atributos expostos, via protocolo de comunicação. Como nesta implementação está sendo usado o protocolo em *javascript*, é na função *init* que são adicionados os *call-backs*, ou seja, as chamadas das funções que o *javascript* poderá acessar no *framework*. Esse processo é feito usando a interface de comunicação entre *javascript* e *Actioscript* 3, *ExternalInterface.* Por exemplo, para expor a função *changeParameter* do framework é adicionada a seguinte linha de código na inicialização do Xpose:

```
ExternalInterface.addCallback("changeParameter", Xpose.changeParameter);
```

Nesta chamada, a classe estática *ExternalInterface*, que faz parte da biblioteca de Actionscript 3, adiciona uma "porta" de comunicação externa, e informa que o nome a ser chamado externamente será "*changeParameter*" e, ao ser invocado esse método, o *Xpose.changeParameter* será executado. Este mesmo procedimento é usado para expor todas as funções que farão parte do protocolo de comunicação listados na tabela 5.1.

Se não for possível utilizar essa forma de implementação do protocolo em outras linguagens, no caso, por exemplo, de uma linguagem em que a comunicação via ExternalInterface e javascript não exista, é perfeitamente possível utilizar outras formas de expor funções para que outros programas consigam acessá-los. Por exemplo, estabelecendo uma conexão via *socket* entre o jogo e a ferramenta de interface gráfica, onde, através de mensagens na rede as funções podem ser acessadas. Outras formas de comunicação entre programas são facilmente encontradas na literatura dependendo da linguagem que o framework e o jogo estão sendo implementados.

Já a função *register* é a responsável por registrar as variáveis de uma classe para que estas sejam expostas para o balanceamento. Por exemplo, digamos que num jogo de naves espaciais existe a classe *Player,* que é responsável por controlar as variáveis da entidade que representa a nave principal do jogador dentro do jogo e, nesta entidade, o *game designer* deseja mexer no atributo relativo a velocidade desta nave.

Neste caso, o programador precisa registrar a classe *Player* no *framework* Xpose para que está seja acessível, via protocolo de comunicação, na ferramenta de interface gráfica de balanceamento. Esse registro é feito sendo chamada a função estática *Xpose.register(this),* geralmente a chamada desta função é feita logo após a inicialização da classe que deseja ser registrada.

Ao fazer o pedido de registro da classe através da função Xpose.register, entra em ação a classe mais importante em termos de operação do *framework*, a *ParametersCollection,* que será detalhada a seguir.

5.2.2.2 ParametersCollection

A classe *ParametersCollection* é responsável por implementar todas as funções que a fachada Xpose expõe e por registrar de fato as entidades que desejam ter seus atributos expostos através da função *lookupParameters*. A função *lookupParameters* é responsável por registrar as entidades no *framework* e é chamada dentro da implementação do *Xpose.register*. O conceito geral por trás da função *lookupParameters* é receber a instancia da classe que deseja se registrar, ver todos as variáveis dessa classe que foram marcadas para serem expostas e guardar uma referência para cada uma dessas variáveis a fim de conseguir fazer as operações de mudança de variável, listagem das variáveis expostas, listagem dos tipos, e todas as outras funções já discutidas anteriormente no protocolo de comunicação.

Existe a necessidade da criação de um "nome fantasia" para as variáveis de código que serão expostas, pois nem sempre o nome dado a um atributo pelo game designer dentro do GDD é igual ao nome da variável que o programador especifica no código. Então, esse mapeamento entre nome fantasia e nome da variável também precisa ser feito no *ParametersCollection,* dentro da função *lookupParameters*.

A implementação da classe *ParametersCollection,* na linguagem de programação escolhida para este trabalho, simplificou bastante a forma que o programador consegue expor as variáveis do código sem precisa mudar muito o código fonte. Duas características da linguagem foram essências para que existisse essa simplificação na maneira do programador trabalhar: *Annotation* e *Reflection*[7]. Com o uso de

[7] Annotation e Reflection são características de implementação presente na maioria das linguagens de programação orientadas a objetos hoje em dia. Mais detalhes sobre o uso dessas características podem ser encontradas na especificação de Actionscript 3 (http://www.adobe.com/devnet/actionscript/documentation.html)

Annotation é possível indicar nas classes as variáveis que precisam ser expostas e adicionar algumas informações como: o nome fantasia da variável e o seu valor inicial. Por exemplo, voltando a nossa classe *Player* citada na subseção anterior, se existe um atributo velocidade que precisa ser exposto, o programador precisa apenas indicar no código que essa variável é diferenciada das demais, para que seja exposto apenas o que o *game designer* especificou no GDD. Para isso é usado *Annotation* da seguinte forma:

```
[expose(fantasy="velocidade da nave",defaultValue=100)]
public var velocity:Number;
```

Tudo que está fora dos colchetes ([ ]) é o código que o programador já teria de toda forma que implementar, dentro dos colchetes está a indicação feita pelo programador que esta variável está sendo exposta. Nesse caso, a variável *velocity,* é indicada utilizando *Annotation*, com a característica *expose*, que é uma palavra reservada para o *framework* Xpose, e tem como parâmetros o nome fantasia (*fantasy*) "velocidade da nave" e o valor inicial, ou padrão (*defaultValue*) 100.

A partir dessas informações, quando o *lookupParameters* for chamado para a classe *Player,* vai ser possível, dessa vez usando *Reflection*, passar por todas as variáveis da classe e sempre que for encontrada uma variável com a *Annotation* "[expose..." essa variável e os respectivos parâmetros (nome fantasia e valor padrão) serão armazenados na *ParametersCollection*, para que toda manipulação de variáveis seja possível, via protocolo de comunicação. Dentro da classe *ParametersCollection* todas as variáveis expostas são guardadas numa tabela *Hash,* indexada pelos nomes fantasia, não basta apenas guardar a variável, é necessário guardar também as instancias e classes das quais elas fazem parte, por esse motivo foi criada uma estrutura que encapsula essas informações, que é a ultima classe a ser descrita do *framework* Xpose, a *ExposedObjects.*

5.2.2.3 ExposedObjects

A classe *ExposedObjects* é responsável por guardar as características fixas e temporais dos atributos que estão sendo expostos. Ela guarda o nome real da variável (*attributeName*), o tipo da variável (*type*), o valor atual da variável (*currentValue*), que inicia com o valor padrão indicado na *Annotation,* e as instâncias dos objetos criados da entidade (classe) exposta.

Com essas informações guardadas e gerenciadas pela classe *ExposedObjecs,* fica fácil a manipulação dos valores dos atributos das entidades por parte do *framework* e, como visto, o esforço do programador para expor esses atributos foi o de adicionar *Annotation* às variáveis que deseja expor e registrar a classe do jogo que tem atributos para serem expostos.

5.2.3 Ferramenta de Interface - XposeGUI

A ferramenta de interface gráfica de balanceamento desenvolvida para esse trabalho é bem simples, com o intuito de provar o conceito defendido, ela foi chamada de XposeGUI. Esta ferramenta poderia ter sido implementada das mais diversas formas e linguagens de programação, desde que respeitasse o protocolo de comunicação especificado anteriormente. Na Figura 5.4, pode ser observada a simplicidade da implementação desta ferramenta, que foi desenvolvida em Actionscript 3.

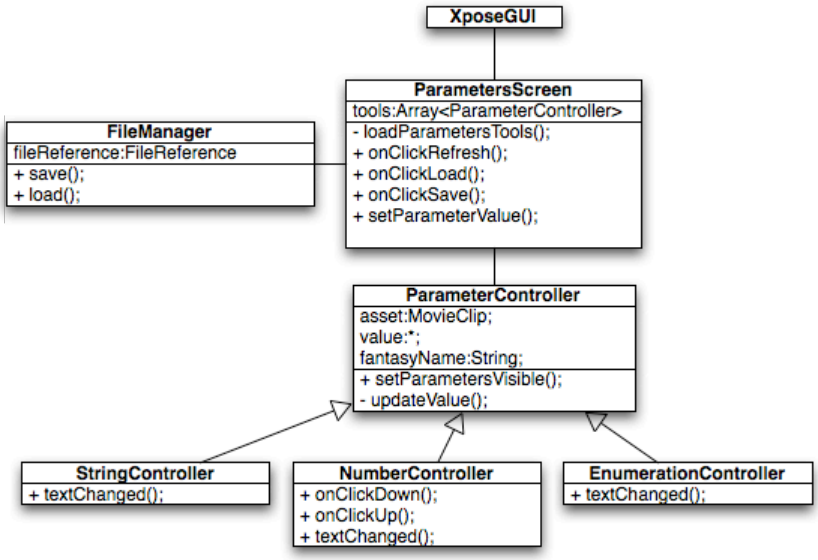


Figura 5.4 – Arquitetura da interface gráfica XposeGui

A ferramenta é composta de uma única tela onde o usuário pode acessar as funcionalidades especificadas pelo protocolo. A classe *ParametersScreen* é a responsável por representar essa tela e através de controles criados na interface consegue: listar os atributos expostos, salvar uma determinada configuração de valores dos atributos em um arquivo, carregar uma configuração salva anteriormente e mudar o valor de um atributo específico.

Cada atributo é representado por um *ParameterController* que é uma classe abstrata. Para cada tipo de atributo diferente, é instanciado um controlador específico para este tipo. Por exemplo, um atributo numérico (velocidade da nave, força do personagem, distancia entre plataformas, etc.) é representado pela classe *NumberController* que herda da classe *ParameterController*.

Essa é uma implementação possível para a ferramenta gráfica de balanceamento que será utilizada pelo *game designer*, mas como informado nas seções anteriores, nada impede que outras implementações sejam construídas, pois há um desacoplamento entre a interface gráfica e o jogo, facilitado pelo protocolo já descrito. Como a implementação da XposeGUI e da Xpose foram feitas usando a linguagem Actionscript 3 e o protocolo implementado em javascript, ficou simples a integração de todas as ferramentas num mesmo ambiente, nesse caso num *browser*. Abaixo, na Figura 5.5, pode ser observada uma imagem da ferramenta sendo utilizada em conjunto com um jogo implementado utilizando o *framework*.

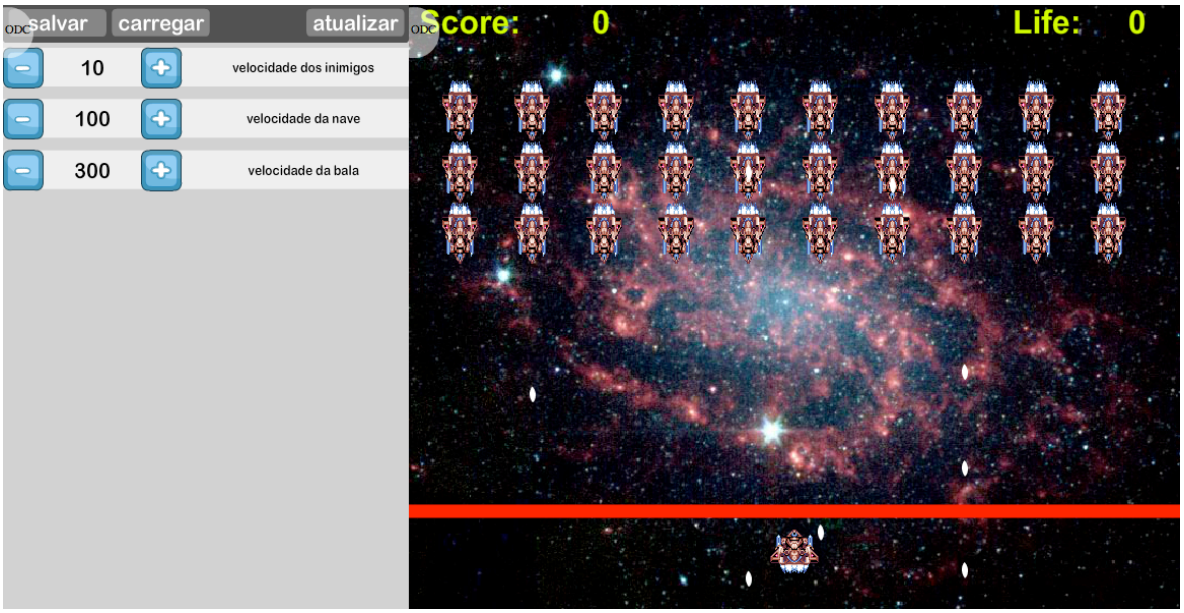


Figura 5.5 – Jogo implementado utilizando o framework Xpose e a ferramenta de interface gráfica XposeGui

Como pode ser visto na Figura 5.5, do lado direito está o jogo sendo executado em tempo real e do lado esquerdo temos a ferramenta XposeGUI apresentando os atributos expostos e os botões de controle para que o *game designer* consiga alterar as variáveis enquanto o jogo está em execução.

6 Avaliação

Com o *framework* (Xpose), a interface gráfica de balanceamento dos atributos (XposeGui) e o protocolo de comunicação entre eles implementados, se faz necessário medir a eficiência deste sistema em um ambiente real de desenvolvimento de jogos.

Para analisar a veracidade da hipótese de trabalho, foi escolhido o método cientifico de experimento controlado. Um experimento controlado é um método de investigação de hipóteses testáveis em que uma ou mais variáveis independentes são manipuladas para medir seus efeitos em uma ou mais variáveis dependentes (SINGER; STOREY; DAMIAN, 2002).

Um estudo de caso poderia ser implementado para esta pesquisa, algumas empresas se mostraram interessadas em usar o *framework* Xpose em seus projetos. Porém, um dos fatores que nos motivou a usar um experimento controlado, é poder obter uma maior generalização dos resultados. De fato, usar variáveis controladas torna mais fácil a reprodução do experimento em outros ambientes ou outras empresas que desejarem implantar o *framework* em seu processo de desenvolvimento.

Por se tratar de um método científico e formal, um experimento controlado precisa seguir alguns procedimentos de forma a poder ser replicado em outros ambientes. A seguir, será mostrado como esses procedimentos foram aplicados e planejados.

6.1 Planejamento do experimento

Existem alguns passos e cuidados que precisam ser respeitados para a execução de um experimento formal, são eles: concepção, projeto, preparação, execução, análise, disseminação e tomadas de decisões (PFLEEGER, 1995). Para o planejamento deste experimento, todos os passos foram seguidos, como pode ser visto a seguir.

6.1.1 Concepção

Na etapa de concepção, o que se busca é descobrir quais os objetivos do experimento e o que se quer responder no final dele.

Para o experimento em questão o que se busca é comprovar a hipótese de trabalho proposta inicialmente. O objetivo do experimento, ou objetivo experimental, é descobrir se utilizando a ferramenta e o *framework* construídos (XposeGUI + Xpose),

o *game designer* conseguirá ser mais eficiente no balanceamento dos jogos, ou se usando a forma tradicional, em que o *game designer* junto com o programador faz as alterações, ainda é a maneira mais eficiente de executar o balanceamento de jogos.

Definir o que vem a ser um balanceamento eficiente não é uma tarefa tão simples. O que de fato vai dizer se o balanceamento foi feito de forma eficiente ou não é a aceitação pelo público do jogo. Infelizmente, essa aceitação é algo complicado de ser medido, já que exigiria que muitas pessoas jogassem, e com o passar do tempo avaliassem se gostaram ou não dele. É uma resposta que pode ser fácil de determinar com o longo do tempo, após a publicação do jogo, porém é difícil e custosa de se saber durante o desenvolvimento. Torna-se então inviável para esta pesquisa ter um número considerável de pessoas testando os jogos balanceados com e sem o Xpose para descobrir qual o método é o mais eficiente.

A forma que foi encontrada para resolver o problema, de descobrir qual o método mais eficiente, foi então medir a quantidade de mudanças nas variáveis realizadas pelo *game designer* em um determinado intervalo de tempo. Como a hipótese experimental sugere, existe a suspeita que haverá uma maior quantidade de mudanças por tempo no caso em que o *game designer* trabalhe de forma autônoma. Se essa suspeita se confirmar, pode ser deduzido que mais testes foram feitos pelo *game designer* e, consequentemente, mais configurações foram testadas, aumentando assim a eficiência do balanceamento.

6.1.2 Projeto do experimento

Tendo claro o objetivo do experimento controlado a ser implementado, chega o momento da construção do projeto do experimento. Inicialmente, parte-se das hipóteses levantadas. Existem duas hipóteses que guiarão todo o experimento, a **hipótese nula**, ou seja, aquela que afirma que não há diferença entre usar a ferramenta ou usar o método tradicional para balancear o jogo, e a nossa **hipótese experimental,** que foi introduzida no início deste trabalho, que é: se o *game designer* usar uma ferramenta que exponha os atributos para balanceamento, ao invés de sempre recorrer ao programador para executar mudanças de valores desses atributos, ele conseguirá se concentrar e se dedicar mais à atividade e, com isso, experimentar várias configurações diferentes e ser mais eficiente em seu trabalho.

Como o público alvo deste trabalho são os *game designers*, estes foram escolhidos como os **indivíduos experimentais** para a realização do experimento. O experimento

foi realizado com dois *game designers* de empresas diferentes do polo de tecnologia do Porto Digital[8] em Recife. O nível de experiência dos *game designers* escolhidos é bem semelhante, os dois têm por volta de 5 anos de experiência na área e já participaram do desenvolvimento de cerca de 40 jogos, já trabalharam com jogos desde a fase de concepção até as fases finais de balanceamento.

Os jogos utilizados para a realização dos experimentos são jogos casuais. Estes são os **objetos experimentais** da pesquisa. Por serem jogos mais simples, a quantidade de atributos a serem balanceados também é reduzida em relação a jogos mais complexos. Porém, apesar da simplicidade dos jogos, o desafio do balanceamento é comparável ao de jogos maiores no que se diz respeito a forma como o balanceamento é feito, logo, o experimento torna-se válido para o propósito desta pesquisa.

Foram utilizados 3 jogos para a realização do experimento: o Resgate, Rota Certa e Space War, Figura 6.1. O Resgate é de um jogo rítmico, em que o objetivo do jogador é salvar os animais em uma locomotiva que está pegando fogo, e chegar até a próxima estação garantindo a segurança da locomotiva. Rota Certa é um jogo de perguntas e respostas musicais, em que o jogador precisa responder às perguntas corretamente, guiando, com isso, um barco para a sua "rota certa". O Space War é um jogo de naves espaciais. O objetivo do jogador é destruir todos os inimigos antes que estes o destruam. O jogo Resgate e Rota Certa são jogos que fazem parte de uma plataforma de ensino para crianças, a Turma do Som[9], desenvolvidos pela empresa Daccord. O jogo Space War foi desenvolvido exclusivamente para este trabalho, no intuito de provar o conceito da ferramenta. Detalhes sobre cada um desses jogos, como, público alvo, ação do jogador, objetivos e mecânica de jogo podem ser vistos em seus respectivos GDDs resumidos que estão disponíveis no Anexo 2 desta pesquisa.

---

[8] O Porto Digital é o nome dado ao conjunto de empresas de tecnologia, definido como o Arranjo Produtivo de Tecnologia da Informação e Comunicação e Economia Criativa, que está situado no Recife, capital de Pernambuco, no nordeste brasileiro. (www.portodigital.org)

[9] Turma do Som é uma plataforma de ensino de música para crianças, desenvolvido pela empresa Daccord, que está situada no Porto Digital, em Recife, Pernambuco, mais detalhes sobre a plataforma de ensino e a empresa pode ser encontrado em www.daccord.com.br

Figura 6.1 – Jogos utilizados no experimento.

Fonte: www.turmadosom.com.br

As variáveis que foram utilizadas no balanceamento destes jogos, ou seja as **variáveis independentes** do experimento, estão na tabela 6.1. O tempo total de execução do balanceamento foi de 30 minutos. Porém, caso fosse atingido uma nível interessante antes desse tempo, o *game designer* poderia parar o experimento. A **variável dependente**, ou seja, aquela que foi analisada posteriormente em relação à eficiência do balanceamento foi a quantidade de mudanças feitas por tempo. Outro aspecto observado foi a variação de valores de uma mesma variável. Essa variação de valores informa o quanto o *game designer* teve a liberdade de flutuar entre diferentes valores de variáveis até chegar ao valor desejado. Esta ultima variável não apresenta um valor numérico que a caracterize, mas é bastante importante para mostrar que o nível de liberdade do *game designer* durante o balanceamento usando ou não o *framework*.

Para a realização do experimento os jogos foram balanceados por dois **métodos** diferentes. No método tradicional, em que o *game designer* joga, faz suas anotações

do que precisa ser mudado para o balanceamento, e passa para o programador, que, por sua vez, realiza as alterações e devolve novamente ao *game designer*, e do método definido neste trabalho, em que o *game designer* indica quais as variáveis que precisam ser balanceadas, o programador anota no código e depois, o *game designer* fica livre para executar o balanceamento sem a intervenção do programador.

Não fazia sentido que um mesmo *game designer* utilizasse os dois métodos para um mesmo jogo, pois, dessa maneira, após realizar o balanceamento usando um dos métodos, por exemplo, utilizando ao método tradicional, quando fosse realizar o balanceamento utilizando o outro método, já saberia previamente quais os valores que precisaria atribuir às variáveis. Isso reduziria o tempo de testes e ajustes para cada jogo, que é intrínseco do balanceamento, e dessa maneira, a segunda vez que fosse realizar o balanceamento seria muito mais rápida que a primeira.

Para que isso não ocorresse, como temos dois *game designers* com experiência similar, colocamos um *game designer* para balancear um jogo usando o *framework* Xpose, e o outro para balancear o mesmo jogo usando o método tradicional. Para um outro jogo, o papel se inverte, aquele *game designer* que balanceou o primeiro jogo usando o *framework* agora vai balancear outro jogo usando o método tradicional, e vice versa. Na Figura 6.2, está o esquema da distribuição de jogos e formas de balanceamento entre os *game designers*.

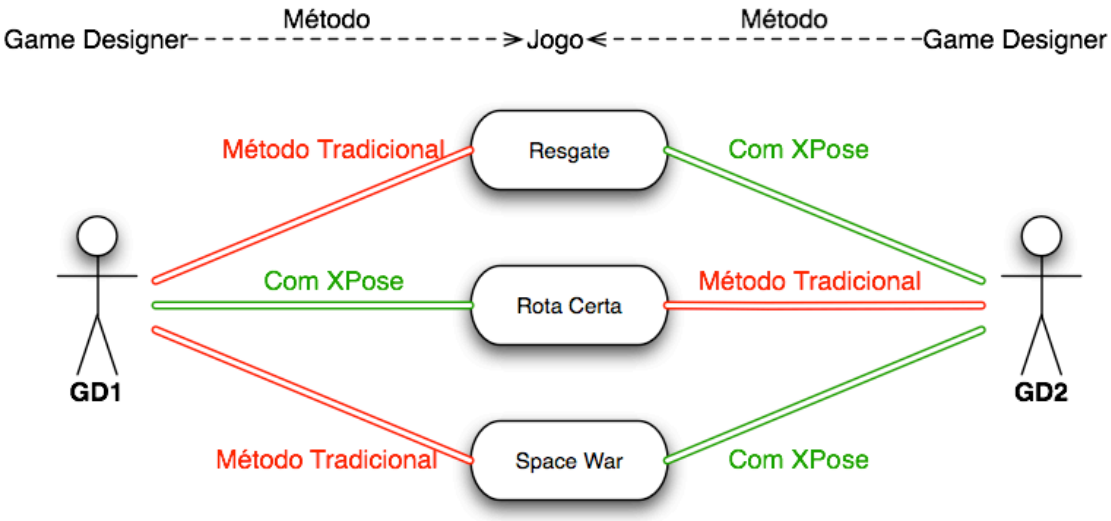


Figura 6.2 – Distribuição de jogos e métodos entre os *game designers*

Para cada jogo, algumas variáveis foram escolhidas para serem manipuladas. Isso foge um pouco da realidade de um balanceamento no método tradicional, pois,

geralmente, o *game designer* não tem essa indicação antes de iniciar um trabalho de balanceamento. O que ocorre no geral é que o *game designer*, ao jogar pela primeira vez, indica ao programador quais atributos deseja mudar, e a partir daí, o programador vai em busca das variáveis que representam estes atributos no código.

Usando o Xpose, o *game designer* deve definir já no momento de concepção quais atributos ele está interessado em modificar, também podendo adicionar novos atributos para serem expostos durante a fase de balanceamento.

O que foi verificado é que, ter claramente quais os atributos que podem ser mudados, facilita a organização do trabalho dos *game designers*.

Os atributos que foram alterados no experimento para cada um dos jogos está na tabela 6.1 abaixo.

Jogo	Atributos Expostos
Resgate	Tempo para apagar fogo
	Tempo para vagão pegar fogo
	Energia perdida ao queimar
	Batidas para quebrar gelo
	Energia ganha ao pegar item de HP
Rota Certa	Tempo inicial de jogo
	Tempo ganho ao acertar
	Tempo perdido ao errar
	Tempo por questão
Space War	Velocidade da nave
	Velocidade do inimigo
	Velocidade da bala do inimigo
	Velocidade da bala do player
	Linhas de inimigos
	Inimigos por linha
	Vidas iniciais do jogador

Tabela 6.1 – Atributos expostos no experimento por jogo

6.1.3 Preparação do experimento

Para a preparação do experimento, foi preciso treinar os *game designers* participantes para usar as ferramentas implementadas neste trabalho. O XposeGUI foi

mostrado a cada um dos *game designers* que participaram, assim como detalhes de como funcionaria a comunicação entre o XposeGUI e o jogo, mostrando as trocas de informações através do protocolo. Eles também tiveram acesso ao GDD dos jogos que iriam balancear.

Para simular o método tradicional, foi colocado ao lado do *game designer* um programador que ficaria à sua disposição para efetuar as modificações requeridas imediatamente. Claro, esse acesso direto entre programador e *game designer* não retrata o que acontece na vida real, onde o programador não está 100% do tempo disponível para o *game designer*. Isto quer dizer que nosso experimento favorece o método convencional, sem uso do Xpose. É, portanto, um cenário extremo para teste da ferramenta desenvolvida. Se a eficiência do método for mesmo assim comprovada, então o impacto positivo do Xpose será ainda maior no mundo real.

6.2 Resultados

A **análise** dos resultados foi dividida em duas partes. A primeira parte foi para rever todos os dados encontradas no experimento, ou seja, recuperar o valor das variáveis independentes e, com elas, calcular as variáveis dependentes e verificar se estes dados são válidos. A segunda parte foi para analisar os dados válidos, organizando seus valores de forma a descobrir se a hipótese nula é suportada ou refutada.

Os dados colhidos nos experimentos foram registrados de duas formas. Para os experimentos usando o Xpose, os valores dos atributos foram gravados automaticamente a cada mudança feita pelo *game designer* e, no final do balanceamento, a ferramenta gerou um relatório com os valores alterados para cada atributo em cada momento do experimento. Já para o experimento no modo tradicional (programador + *game designer*), a cada pedido de mudança feito pelo *game designer* os atributos mudados foram anotados manualmente, registrando-se o momento da mudança e o valor do atributo.

Com isso, para cada um dos jogos foram registrados todas as alterações feitas nos atributos das entidades, tanto para o balanceamento usando o método tradicional quanto para o balanceamento usando o Xpose. Um gráfico comparativo para cada um dos atributos modificados foi traçado. O eixo x representa o número de alterações realizadas e o eixo y o valor dos atributos em cada uma das alterações, como pode ser visto na Figura 6.3. A linha azul no gráfico representa as alterações dos atributos

usando o método tradicional, e a linha vermelha representa as alterações usando o *framework* Xpose.

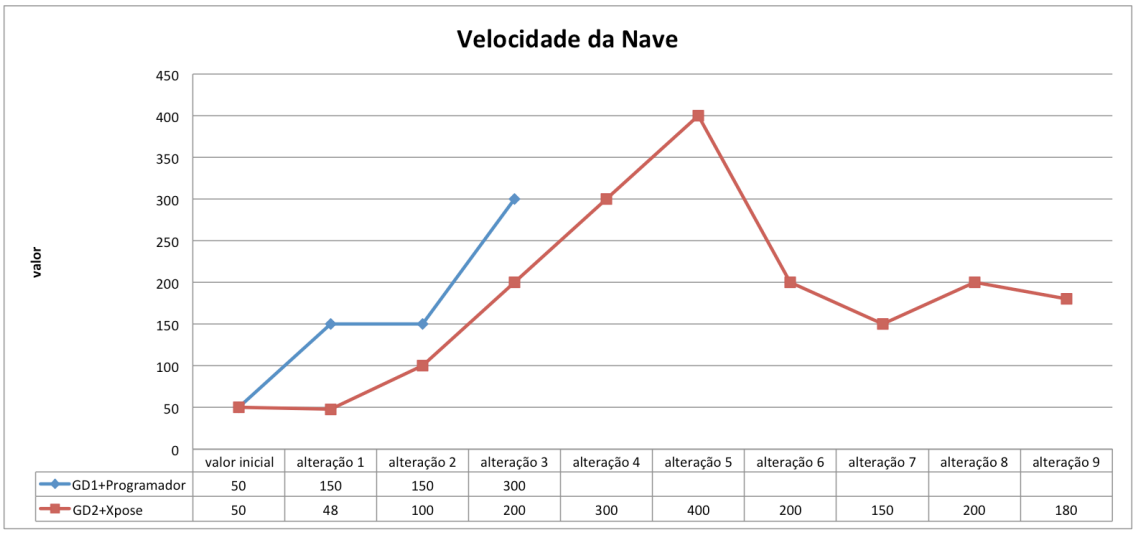
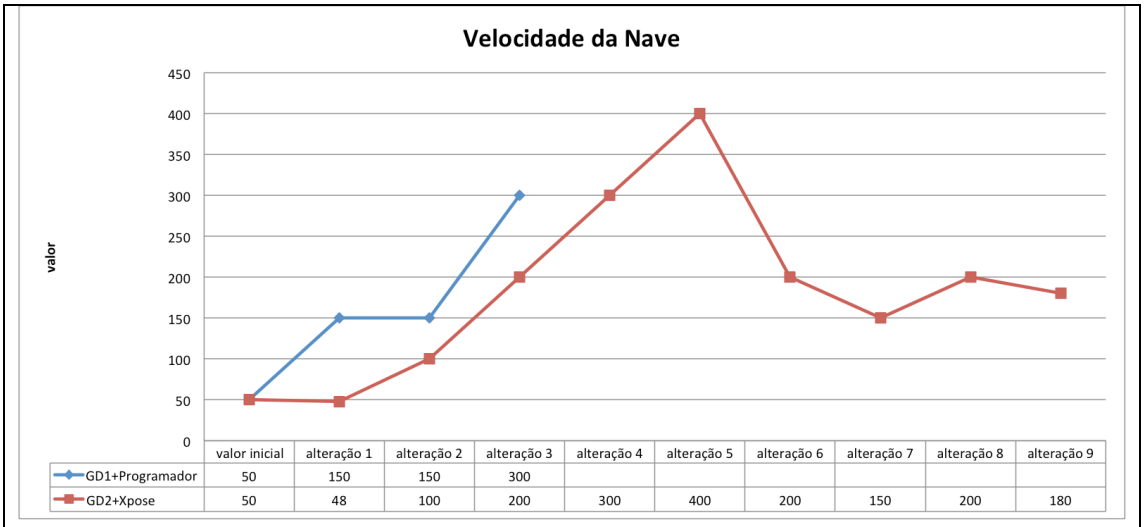


Figura 6.3 – Gráfico das alterações efetuadas no atributo velocidade da nave, para o jogo Space War.

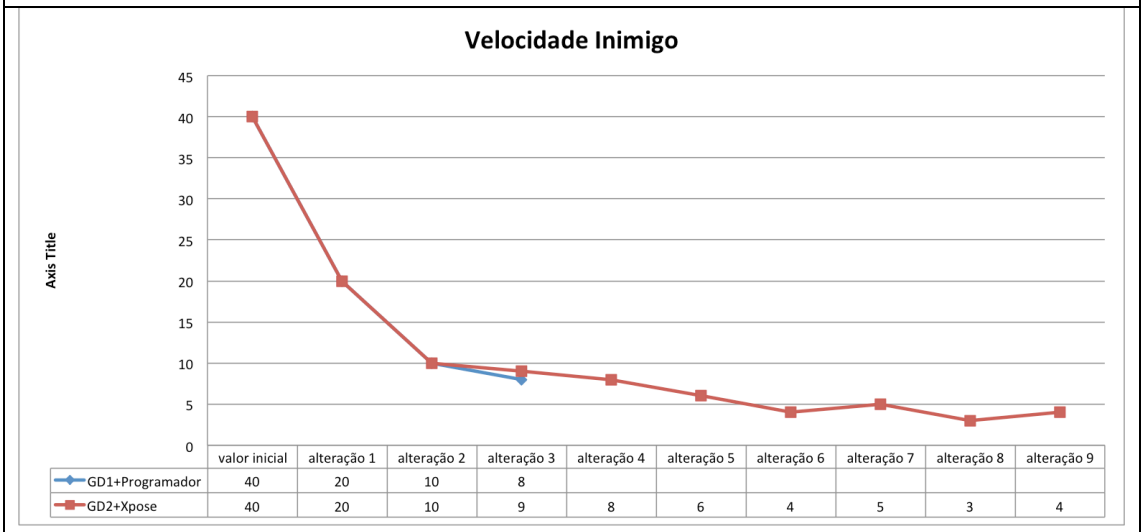
6.2.1 Apresentação dos resultados

Foram analisados 16 atributos considerando os três jogos usados no experimento. Verificando a variável independente, quantidade de alterações, em apenas um dos atributos foi observado que a quantidade de alterações foi maior usando o método tradicional, em que o programador e o *game designer* precisam se comunicar para executar a atividade. Para os outros 15 atributos a quantidade de alterações sempre foi maior quando usado o *framework* Xpose e a ferramenta de interface de balanceamento XposeGui.

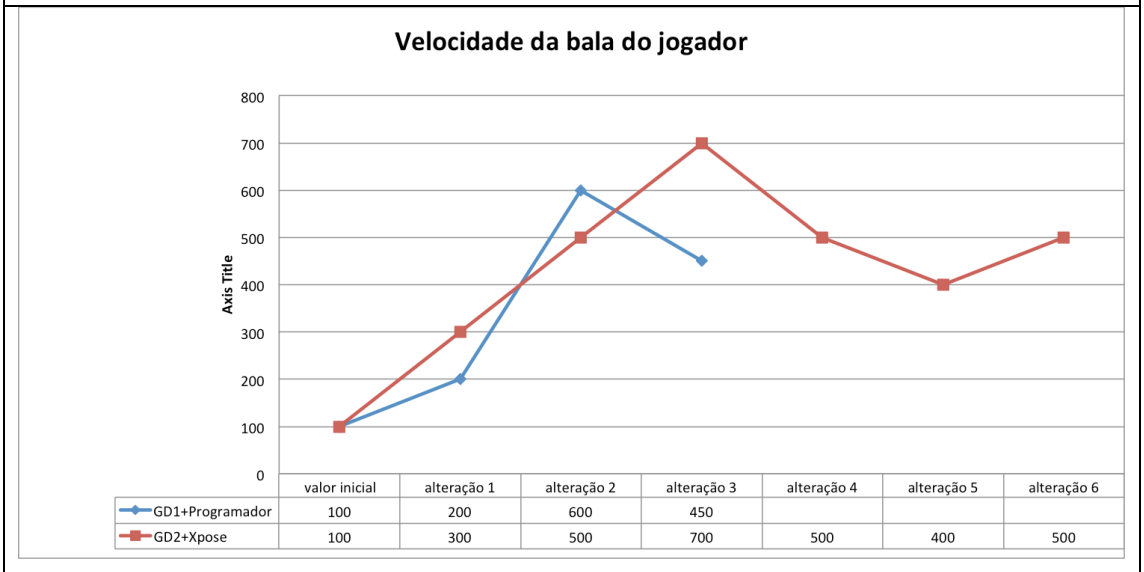
Para o jogo Space War, que foi o jogo com maior quantidade de atributos a serem balanceados, sete no total, os valores de cada atributo em cada alteração pode ser verificado nos gráficos da Figura 6.4.



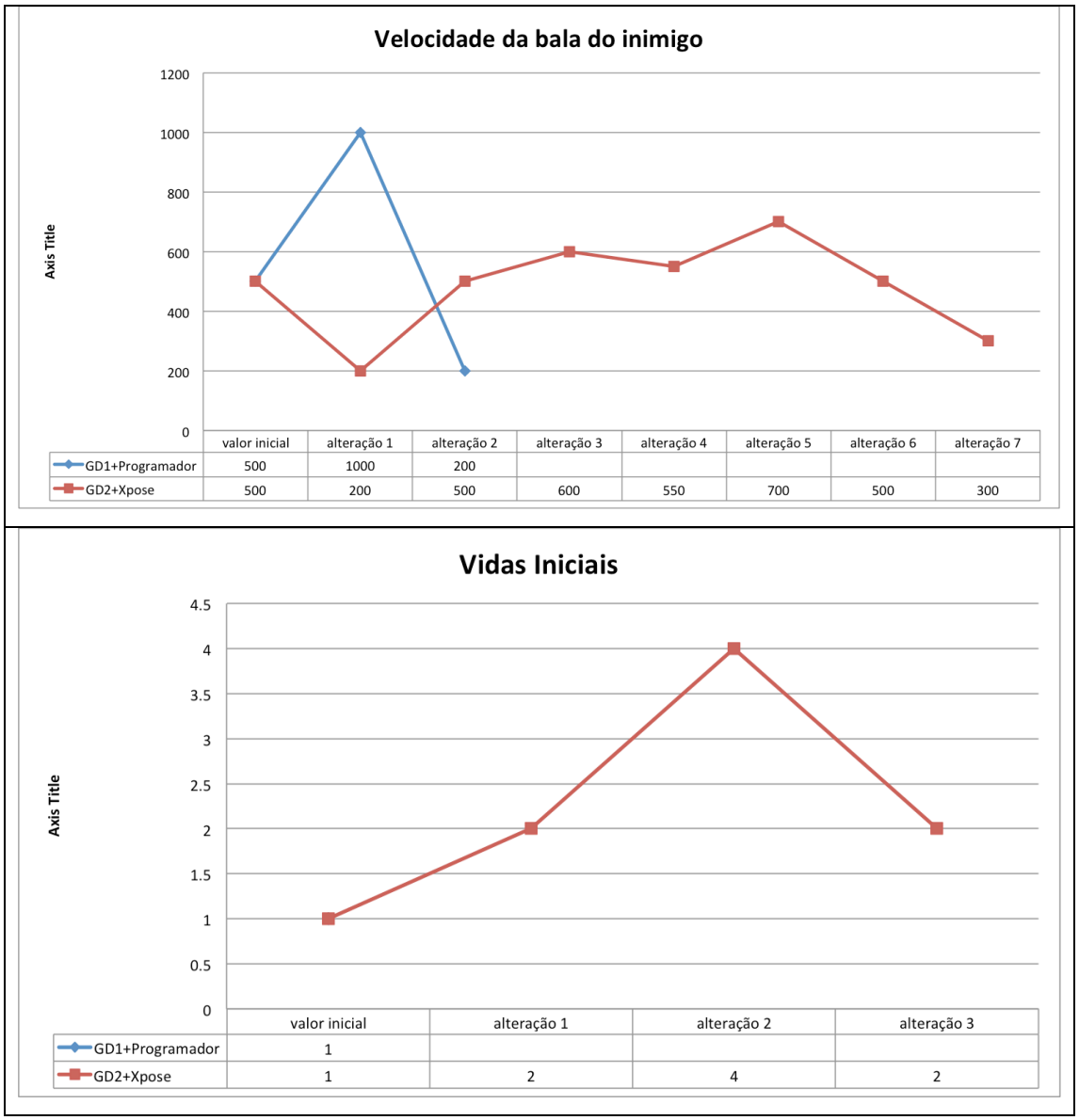
	valor inicial	alteração 1	alteração 2	alteração 3	alteração 4	alteração 5	alteração 6	alteração 7	alteração 8	alteração 9
GD1+Programador	50	150	150	300						
GD2+Xpose	50	48	100	200	300	400	200	150	200	180



	valor inicial	alteração 1	alteração 2	alteração 3	alteração 4	alteração 5	alteração 6	alteração 7	alteração 8	alteração 9
GD1+Programador	40	20	10	8						
GD2+Xpose	40	20	10	9	8	6	4	5	3	4



	valor inicial	alteração 1	alteração 2	alteração 3	alteração 4	alteração 5	alteração 6
GD1+Programador	100	200	600	450			
GD2+Xpose	100	300	500	700	500	400	500


Velocidade da bala do inimigo
Axis Title
1200
1000
800
600
400
200
0
valor inicial | alteração 1 | alteração 2 | alteração 3 | alteração 4 | alteração 5 | alteração 6 | alteração 7
GD1+Programador | 500 | 1000 | 200
GD2+Xpose | 500 | 200 | 500 | 600 | 550 | 700 | 500 | 300
Vidas Iniciais
Axis Title
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
valor inicial | alteração 1 | alteração 2 | alteração 3
GD1+Programador | 1
GD2+Xpose | 1 | 2 | 4 | 2

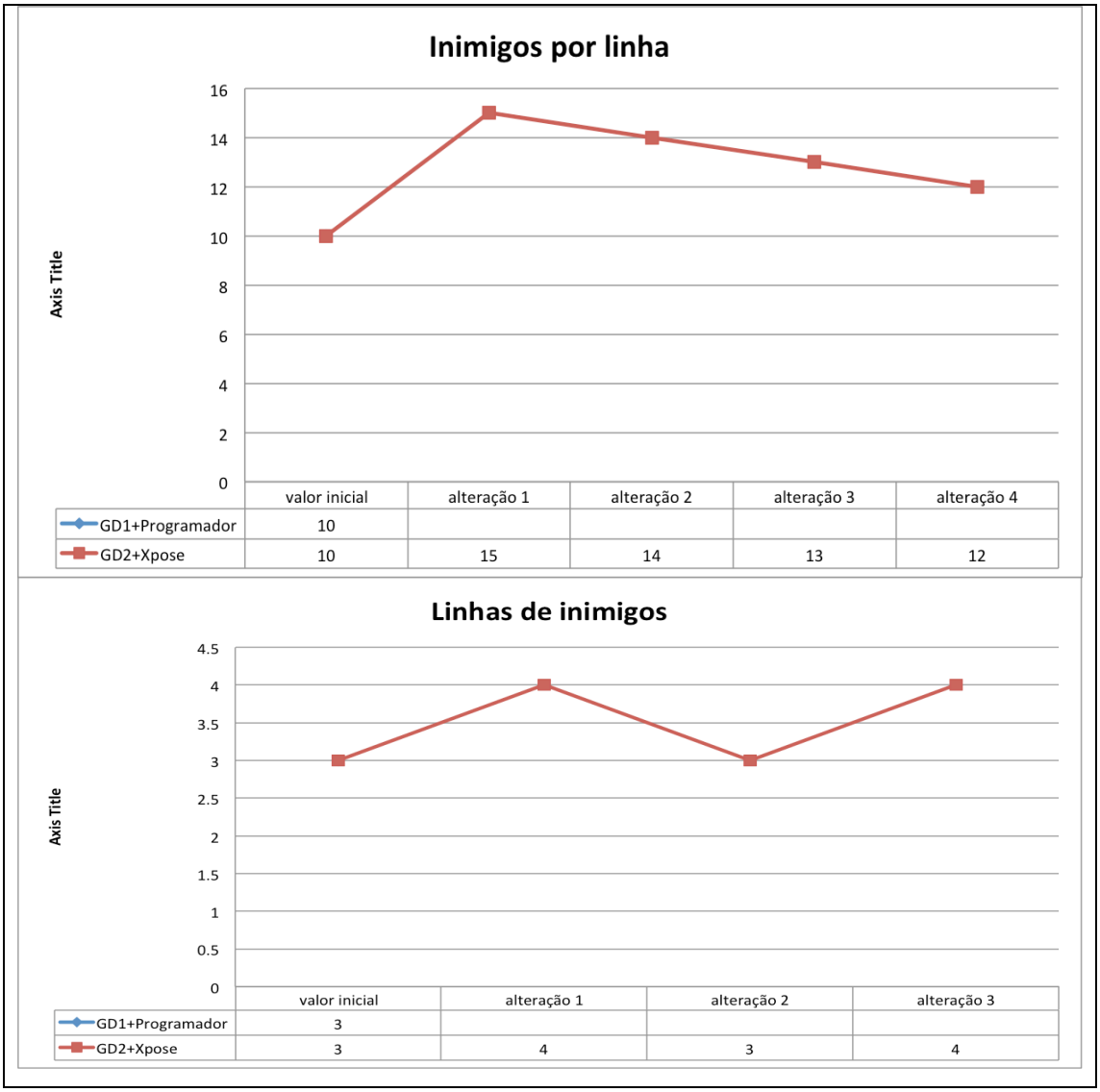
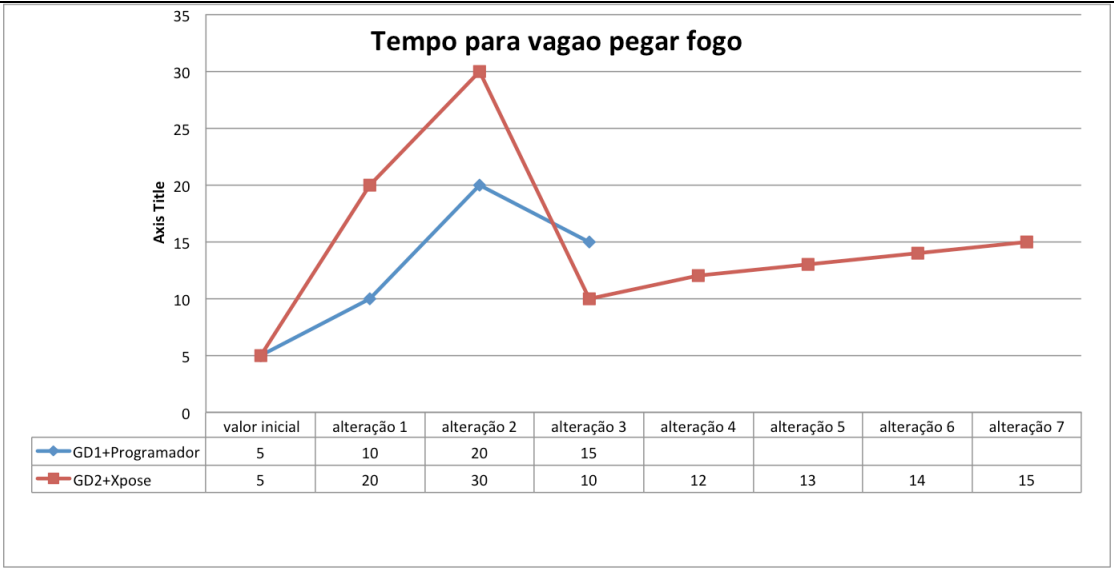
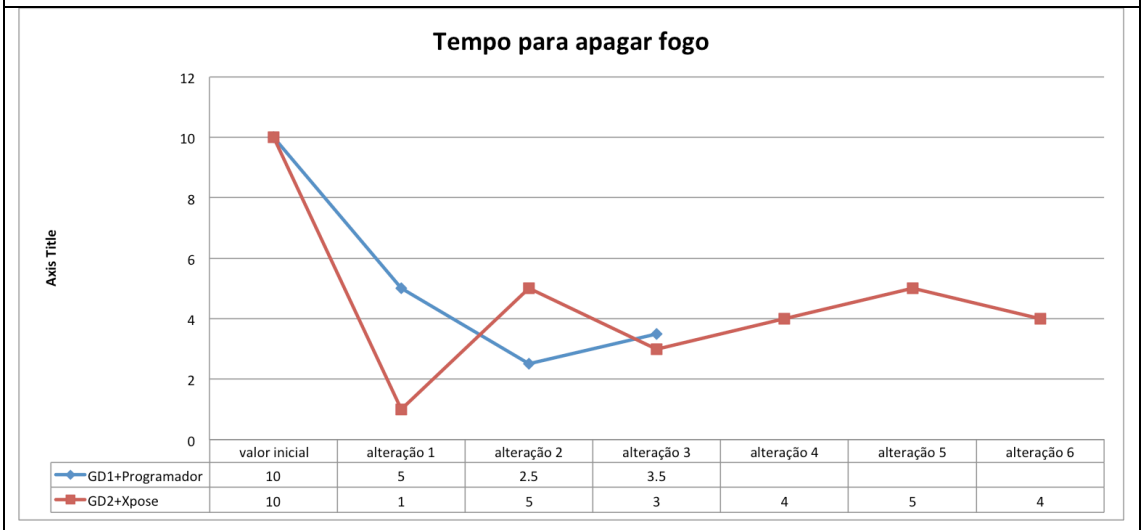


Figura 6.4 – Gráficos de alterações dos atributos no jogo Space War durante o experimento

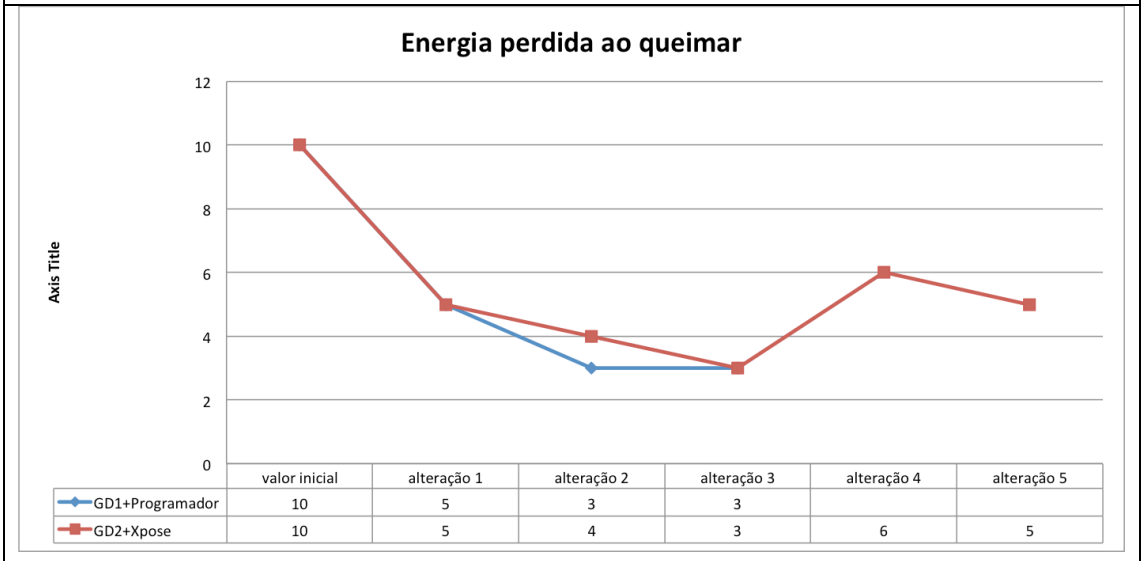
Para o jogo Resgate, os valores de cada atributo em cada alteração pode ser verificado nos gráficos da Figura 6.5. Para o jogo do Resgate, 5 atributos foram expostos aos *game designers*. Nesse jogo o atributo "energia ganha ao pegar HP" sofreu mais alterações usando o método tradicional.



	valor inicial	alteração 1	alteração 2	alteração 3	alteração 4	alteração 5	alteração 6	alteração 7
GD1+Programador	5	10	20	15				
GD2+Xpose	5	20	30	10	12	13	14	15



	valor inicial	alteração 1	alteração 2	alteração 3	alteração 4	alteração 5	alteração 6
GD1+Programador	10	5	2.5	3.5			
GD2+Xpose	10	1	5	3	4	5	4



	valor inicial	alteração 1	alteração 2	alteração 3	alteração 4	alteração 5
GD1+Programador	10	5	3	3		
GD2+Xpose	10	5	4	3	6	5

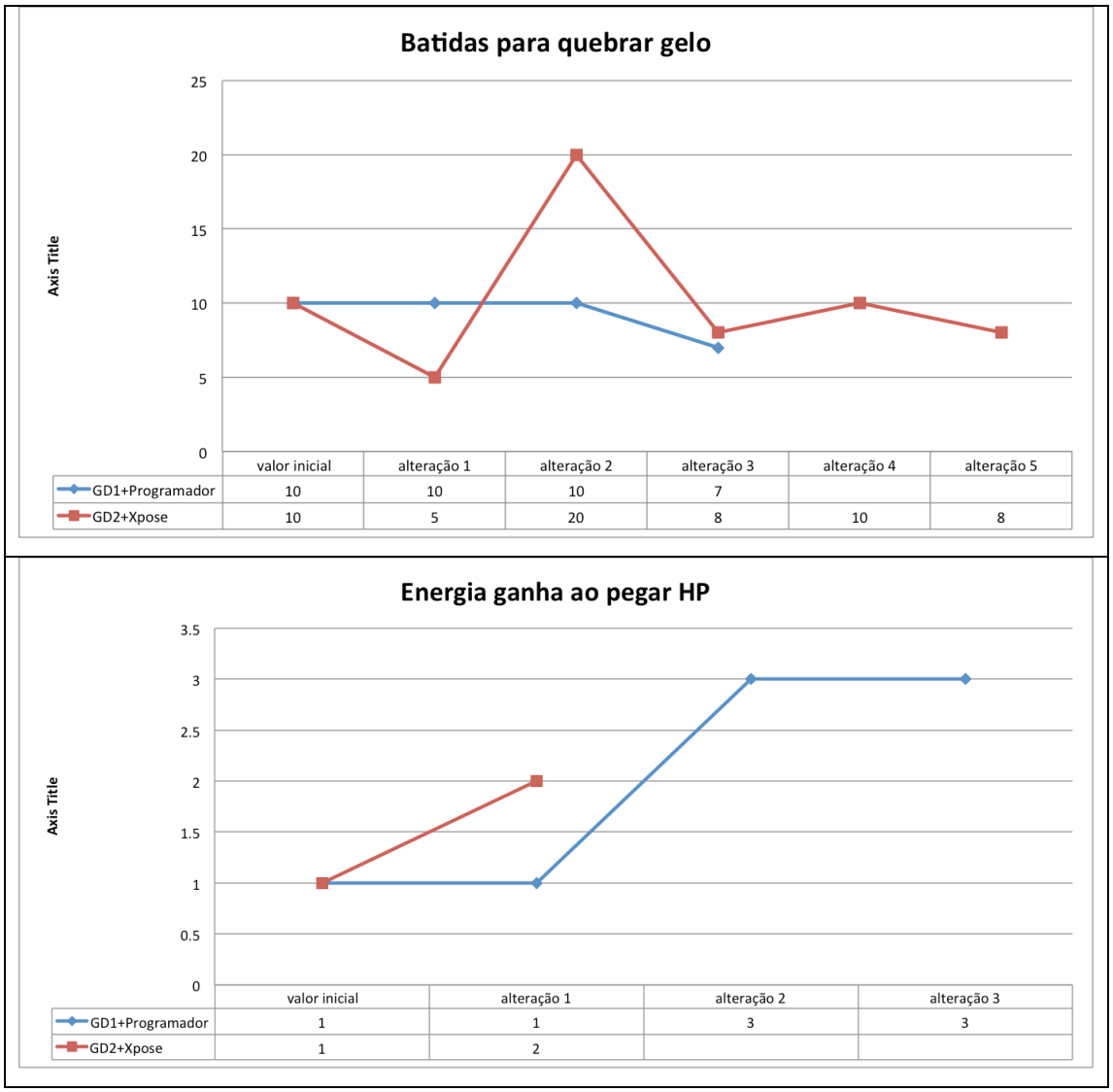
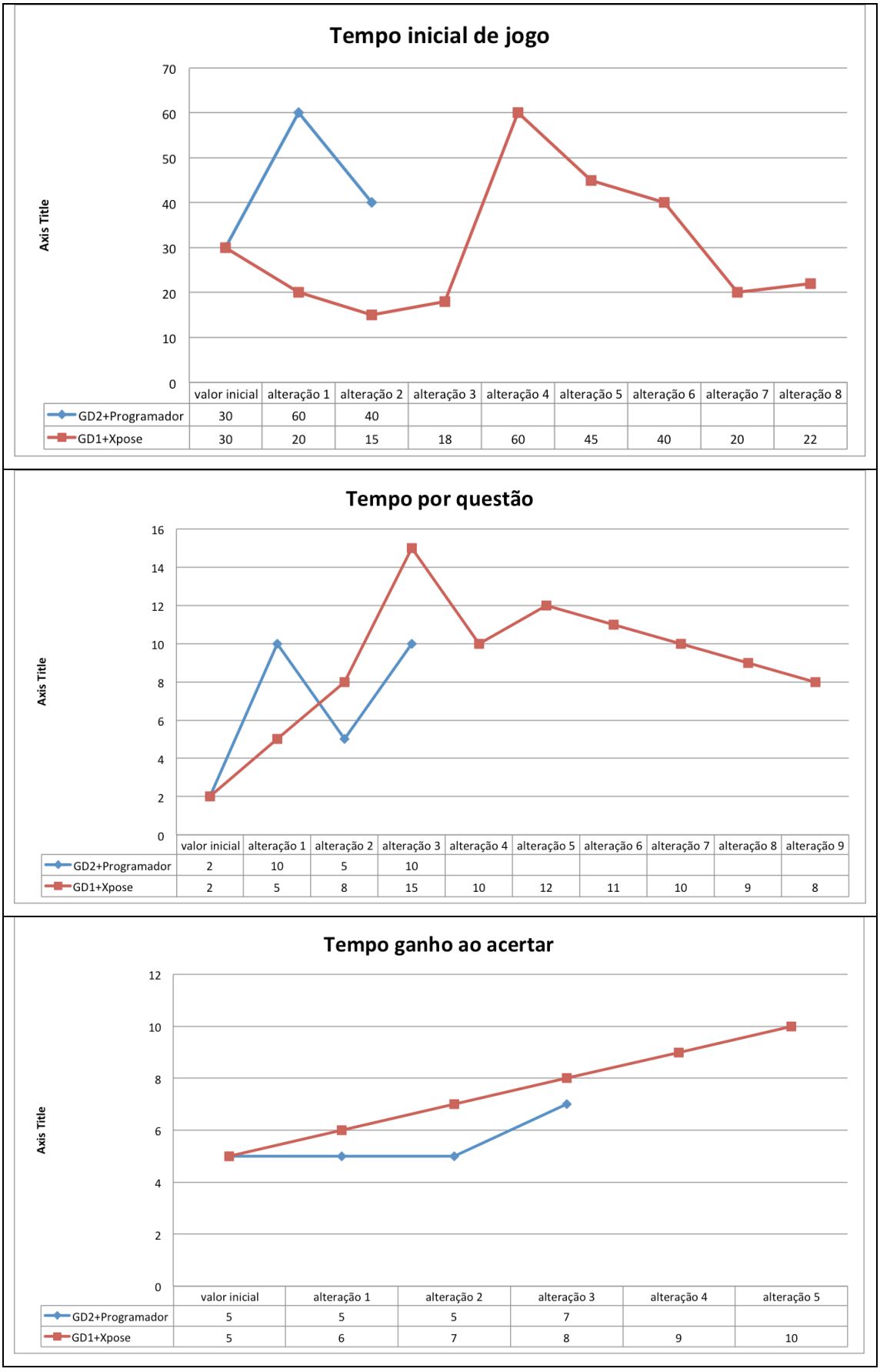


Figura 6.5 - Gráficos de alterações dos atributos no jogo Resgate durante o experimento

Para o jogo Rota Certa, os valores de cada atributo em cada alteração pode ser verificado nos gráficos da Figura 6.6. Neste jogo, apenas 4 atributos foram expostos para balanceamento.


Tempo inicial de jogo
Axis Title
70
60
50
40
30
20
10
0
valor inicial | alteração 1 | alteração 2 | alteração 3 | alteração 4 | alteração 5 | alteração 6 | alteração 7 | alteração 8
GD2+Programador | 30 | 60 | 40
GD1+Xpose | 30 | 20 | 15 | 18 | 60 | 45 | 40 | 20 | 22
Tempo por questão
Axis Title
16
14
12
10
8
6
4
2
0
valor inicial | alteração 1 | alteração 2 | alteração 3 | alteração 4 | alteração 5 | alteração 6 | alteração 7 | alteração 8 | alteração 9
GD2+Programador | 2 | 10 | 5 | 10
GD1+Xpose | 2 | 5 | 8 | 15 | 10 | 12 | 11 | 10 | 9 | 8
Tempo ganho ao acertar
Axis Title
12
10
8
6
4
2
0
valor inicial | alteração 1 | alteração 2 | alteração 3 | alteração 4 | alteração 5
GD2+Programador | 5 | 5 | 5 | 7
GD1+Xpose | 5 | 6 | 7 | 8 | 9 | 10

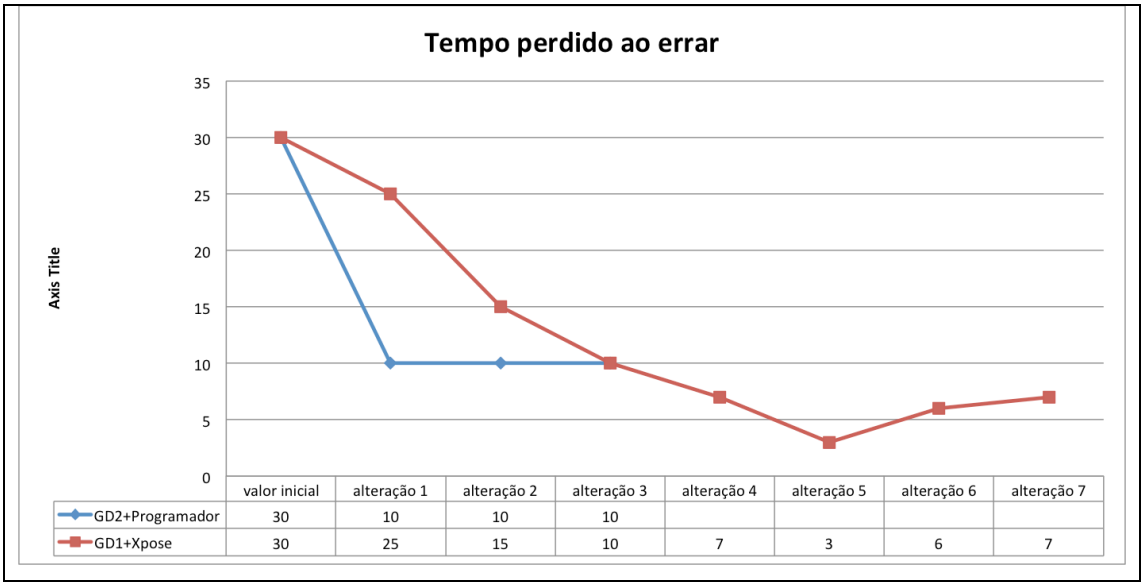


Figura 6.6 - Gráficos de alterações dos atributos no jogo Rota Certa durante o experimento

6.2.2 Discussão sobre os Resultados

Como foi observado na seção anterior, a quantidade de modificações nos atributos foi bem maior quando usado o *framework* Xpose. Em um dos casos, no atributo "energia ganha ao pegar hp" do jogo Resgate, aconteceu de, na primeira alteração, o *game designer* já ficar satisfeito com o valor escolhido, o que acarretou a esse atributo não ser mais mexido durante o experimento. O que não é o comum, como pode ser visto nos demais gráficos.

Em alguns gráficos, pode-se observar que não houve nenhuma modificação de alguns atributos, por parte do *game designer*, durante o experimento usando o método tradicional. Isto pode ser observado nos três últimos atributos do balanceamento do jogo Space War: vidas iniciais, inimigos por linha e linhas de inimigos. Ao perguntar ao *game designer* que fez o balanceamento, ele informou que o motivo dessa não alteração, foi por conta do tempo que não foi suficiente para detalhar esses atributos. Como o tempo do experimento foi fixado em 30 minutos e o jogo Space War foi o que possuía maior número de atributos a serem balanceados, o *game designer* se limitou a alterar as variáveis que ele achou mais importantes nesse tempo. A falta de tempo para balancear alguns detalhes no jogo é algo apontado por *game designers* da indústria (MACHADO et al., 2012), e foi observado na prática durante o experimento realizado.

Outro fator que merece destaque na análise dos gráficos apresentados, é que houve uma certa liberdade, por parte dos *game designers*, para extrapolar, em alguns

momentos, os valores dos atributos, quando estavam usando o Xpose. Podem ser observados vários picos que fogem dos valores finais dos atributos em alguns casos, como nos gráficos dos seguintes atributos: tempo por questão e tempo inicial do jogo, no jogo Rota Certa (ver Figura 6.6), batidas para quebrar gelo no Resgate (ver Figura 6.5), e velocidade da nave no Space War (ver Figura 6.4). Esses picos são mais discretos nos experimentos em que o método tradicional foi usado. Isso indica que o *game designer* passou a ter maior liberdade para experimentar valores que podem em alguns momentos parecer absurdos para o valor de um atributo, porém, como há uma facilidade de mudança rápida desses valores, esse pontos fora do normal podem ajudar a descobrir valores que acrescentem um diferencial ao jogo.

Na maioria dos gráficos, pode ser observado que o *game designer* que usou o *framework*, testou os mesmos valores utilizados pelo *game designer* que usou o método tradicional. Em vários gráficos, pode ser visto um ajuste fino nas alterações finais, quando usando o *framework*, como é o caso das alterações feitas no atributo "velocidade do inimigo" do jogo Space War. Esse ajuste fino seria bastante complicado usando o método tradicional, pois várias idas e vindas seriam adicionadas ao processo, e usando o *framework*, numa só jogada, o *game designer* consegue executar essas modificações rapidamente.

7 Conclusões

Este trabalho surgiu a partir de um problema encontrado na indústria de jogos digitais, que é a dificuldade de comunicação entre *game designer* e programador numa das fases finais de desenvolvimento, o balanceamento dos atributos das entidades do jogo. Por ser uma atividade que tipicamente é pensada pelo *game designer* e executada pelo programador, alguns problemas nessa execução, e na comunicação entre as partes, foram identificados e atacados por esta pesquisa. Levantamos a hipótese de trabalho de que, se o *game designer* usar uma ferramenta que exponha os atributos para balanceamento, ao invés de sempre recorrer ao programador para executar mudanças de valores desses atributos, ele conseguirá se concentrar e se dedicar mais a atividade e, com isso, experimentar várias configurações diferentes e ser mais eficiente em seu trabalho.

Baseando-se nessa hipótese, desenvolvemos um *framework*, o Xpose, que facilita o trabalho do programador na exposição dos atributos das entidades durante o desenvolvimento do jogo. Atributos estes indicados pelo *game designer* ainda na fase de concepção. Dessa forma o programador não precisa mudar a forma que já está acostumado a trabalhar, usando as *engines* e outras bibliotecas que já está habituado, pois o Xpose passa a ser mais um componente acoplado ao código e foi desenvolvido para não interferir na estrutura dos projetos dos jogos, como mostrado no decorrer deste trabalho.

Com o uso do *framework* Xpose no desenvolvimento, as variáveis de código que representam os atributos das entidades, apontados pelo *game designer*, ficam expostas para serem capturadas por uma ferramenta de interface gráfica de balanceamento, que se comunica com o código do jogo usando um protocolo de comunicação, também desenvolvido neste trabalho (ver Anexo 1). Essa ferramenta de interface gráfica de balanceamento é o que o *game designer* usará para manipular os atributos, e pode ser implementada de diversas formas, desde que obedeça o protocolo definido. Com isso, o *game designer* pode escolher a melhor forma de interferir nas variáveis do jogo e efetuar o balanceamento sem depender do programador.

Durante os experimentos foi observado que a eficiência no balanceamento foi melhorada com Xpose. Os testes indicaram que o *game designer* consegue ter mais

liberdade para efetuar as mudanças dos valores dos atributos. Foi medida a quantidade de mudanças realizadas para cada uma das variáveis expostas em três jogos. Em aproximadamente 94% dos casos analisados nos experimentos, a quantidade de alterações por tempo, efetuada para cada variável, foi maior usando o Xpose. Isso indica que, se o *game designer* consegue efetuar mais alterações por tempo, ele pode experimentar mais, testar vários valores diferentes, até chegar ao valor mais próximo do ideal de balanceamento da variável em questão.

7.1 Principais contribuições

A principal contribuição desta pesquisa foi a formalização de uma maneira de expor as variáveis de um jogo, de forma que o programador não precise mudar a forma que está acostumado a trabalhar no desenvolvimento, e o *game designer* não precise ir diretamente ao código, ou pedir ajuda ao programador para manipular os valores das variáveis. Essa formalização de como esse processo é feito, em termos de engenharia de software, possibilita que o *framework* e as ferramentas apresentadas neste trabalho possam ser replicados para diversos jogos em linguagens de programação diferentes, pois, na implementação mostrada, não há nada que seja específico de apenas uma linguagem de programação. A arquitetura formalizada é independente de linguagem.

Outro ponto que merece ser ressaltado é a comprovação da agilidade adquirida pelo *game designer* durante o balanceamento, desde que este consiga ser autossuficiente em sua atividade. Por se tratar de uma atividade que o *game designer* é o principal responsável, quanto mais autonomia ele tenha para executá-la, mais eficiente passa a ser seu trabalho.

7.2 Trabalhos Futuros

Este trabalho apresenta a possibilidade de extensão para diversos caminhos que foram identificados durante a pesquisa e que podem melhorar bastante a ideia inicial discutida.

A inclusão de atributos compostos é uma das possibilidades de extensão deste trabalho. O normal é que os atributos das entidades sejam mapeados diretamente a uma variável no código fonte, mas é possível que, na descrição de algumas entidades de jogo, o *game designer* aponte atributos que não são representados por apenas uma

variável no código fonte. Por exemplo, em um jogo do gênero de corrida (ex. Formula 1, Grand Turismo, Top Gear, etc.), um atributo possível para uma personagem pode ser a sua agressividade em uma corrida. Para que o atributo agressividade passe a fazer sentido dentro do código do jogo é necessário que algumas variáveis sejam alteradas, por exemplo, digamos que a agressividade seja um valor que vai de 1 até 10, mas que para compor este valor as variáveis alteradas sejam: (a) a aceleração do carro, (b) velocidade máxima atingida e (c) tempo de reação ao ser ultrapassado. Nesse caso, é preciso compor uma função em que as variáveis a, b e c resultem na variável composta agressividade. A criação desta função e a ligação com as variáveis do jogo não é suportada por esta versão do Xpose. Estudando a possibilidade de incluir atributos compostos no framework atual, foi visto que o esforço maior seria na elaboração de uma interface gráfica interessante para o *game designer*. Por falta de tempo para experimentar alguns modelos de interface gráfica para executar tal trabalho esta funcionalidade não foi incluída nesta versão da pesquisa.

Outra funcionalidade que também tem muito a ver com a facilidade e liberdade de controle por parte do *game designer* no balanceamento do jogo, é o controle temporal do jogo. Nas pesquisas de Bret Victor, como visto na Figura 1.3, ele propõe um controle temporal que facilita o entendimento do funcionamento de determinado atributo. Esse controle temporal é um adicional interessante para a ferramenta XposeGui desenvolvida nesta pesquisa. Para adicionar o controle temporal no XposeGui seria necessário a adição de um controle externo aos estados do jogo, para que idas e vindas entre os frames fossem facilmente acessadas. Por limitações de tempo para implementação e testes, o controle temporal não foi inserido na pesquisa, porém, entendemos que deixaria bem mais interessante a experiência do *game designer* para balancear o jogo de forma ainda mais eficiente. Algumas ideias de como seria a implementação foram testadas, porém como não obtivemos resultados satisfatórios essa funcionalidade não foi inserida para esta versão.

Os experimentos com o Xpose apontaram um aumento na efetividade do balanceamento feito pelo *game designer*, porém, uma crítica aos experimentos realizados é que estes só foram executados com jogos simples e casuais. Como trabalhos futuros também destacamos a necessidade de testes em jogos maiores e com mais variáveis para serem balanceadas. Usar o *framework* em outras linguagens de programação e com ambientes diferentes, que possuam mais conexões com outras

engines também deixaria os resultados dos experimentos mais conclusivos, e provavelmente outras ideias surgiriam para melhorias na arquitetura pensada.

Estas são algumas das melhorias e extensões do *framework* e das ferramentas de apoio que imaginamos para este trabalho. Outras podem ser pensadas e incorporadas para evolução desta área de pesquisa que é bastante útil às empresas no mercado de jogos, mas que ainda precisa evoluir em termos acadêmicos.

Referências

ADAMS, E. **The Designer's Notebook: Why Design Documents Matter**. Disponível em: <http://www.gamasutra.com/view/feature/1522/the_designers_notebook_why_.php>. Acesso em: 18 jul. 2012.

ANDRADE, G. D. DE et al. **Online Adaptation of Computer Games Agents : A Reinforcement Learning Approach**Recife, 2004.

ANDRADE, G. D. DE. **Balanceamento Dinâmico de Jogos : Uma Abordagem Baseada em Aprendizagem por Reforço**. [s.l.] UFPE, 2006.

APONTE, M. V.; LEVIEUX, G.; NATKIN, S. Scaling the Level of Difficulty in Single Player Video Games. 2009.

BRATHWAITE, B. **Types of Game Designers**. Disponível em: <http://bbrathwaite.wordpress.com/2007/11/20/types-of-game-designers/>. Acesso em: 11 jun. 2012.

BRATHWAITE, B.; SCHREIBER, J. **Challenges for Game Designers**. [s.l.] Paperback, 2009.

BURGUN, K. **Understanding Balance in Video Games**. Disponível em: <http://www.gamasutra.com/view/feature/134768/understanding_balance_in_video_.php>. Acesso em: 27 abr. 2013.

COOK, D. **Game Design Logs**. Disponível em: <http://www.lostgarden.com/2011/05/game-design-logs.html>. Acesso em: 15 jul. 2012.

FURTADO, A. **DSL Tools Melhore sua produtividade através de linguagens visuais de domínio-específico no Visual Studio.NET**, 2005.

FURTADO, A. W. B. **SHARPLUDUS: IMPROVING GAME DEVELOPMENT EXPERIENCE THROUGH SOFTWARE FACTORIES AND DOMAIN-SPECIFIC LANGUAGES**. [s.l.] Universidade Federal de Pernambuco, 2006.

FURTADO, A. W. B. **Domain-Specific Game Development**. [s.l.] Universidade Federal de Pernambuco, 2012.

FURTADO, A. W. B.; SANTOS, A. **Linguagem de Domínio Especifico: o que são e quando e como utilizar**, 2010.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1. ed. [s.l.] Addison-Wesley, 1994. p. 395

GOVONI, D. **Java Application Frameworks**. 1. ed. [s.l.] Wiley, 1999. p. 432

ISAACSON, W. **Steve Jobs a biografia por Walter Isaacson**. [s.l.] Companhia das Letras, 2011. p. 607

JANSSEN, C. **Game Balance**. Disponível em: <http://www.techopedia.com/definition/27041/game-balance>. Acesso em: 4 maio. 2013.

LANG, T. **Four Ways to Write Your Design Docs**. Disponível em: <http://www.gamecareerguide.com/features/737/four_ways_to_write_your_design_.php?page=1>. Acesso em: 9 jun. 2012.

MACHADO, T. L. DE A. **Guidelines Para a Criação de Jogos : Boas Práticas Para Reduzir Conflitos Entre o Design e o Desenvolvimento Guidelines Para a Criação de Jogos : Boas Práticas Para Reduzir Conflitos Entre o Design e o Desenvolvimento**. [s.l.] Universidade Federal de Pernambuco, 2009.

MACHADO, T. L. DE A. et al. The Game Development Conflicts According to the Game Industry. p. 43–51, 2012.

NEWHEISER, M. **Playing Fair: A Look at Competition in Gaming**. Disponível em: <http://www.strangehorizons.com/2009/20090309/newheiser-a.shtml>. Acesso em: 30 abr. 2013.

PERLMAN, R. **Protocol Design Folklore**. Disponível em: <http://www.informit.com/articles/article.aspx?p=20482>. Acesso em: 9 ago. 2013.

PFLEEGER, S. L. Experimental design and analysis in software engineering. **Annals of Software Engineering**, v. 1, n. 1, p. 219–253, dez. 1995.

PIRANHA, E. **Game Configuration with JSON**. Disponível em: <http://davidwalsh.name/game-json>. Acesso em: 2 jul. 2013.

ROCHA, E. J. T. S. **Forge 16V: Um Framework para Desenvolvimento de Jogos Isométricos**. [s.l.] Universidade Federal de Pernambuco, 2003.

ROMERO, M. **Using Configuration Files in UnrealScript**. Disponível em: <http://romerounrealscript.blogspot.com.br/2012/03/using-configuration-files-in.html>. Acesso em: 2 jul. 2013.

SAUVÉ, J. **Frameworks. O que é um framework?** Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm>. Acesso em: 2 jul. 2013.

SCHREIBER, I. **Game Design Concepts**. Disponível em: <http://gamedesignconcepts.wordpress.com/2009/08/20/level-16-game-balance/>. Acesso em: 28 jun. 2013.

SHEFFIELD, B. **What Went Wrong? Learning From Past Postmortems**. Disponível em: <http://www.gamasutra.com/view/feature/4001/what_went_wrong_learning_from_.php>. Acesso em: 27 maio. 2012.

SINGER, J.; STOREY, M.; DAMIAN, D. Selecting Empirical Methods for Software Engineering Research. 2002.

SIRLIN. **Balancing Multiplayer Games**. Disponível em: <http://www.sirlin.net/articles/balancing-multiplayer-games-part-1-definitions.html>. Acesso em: 27 abr. 2013.

VICTOR, B. **No Title**. Disponível em: <http://worrydream.com/#!/InventingOnPrinciple>. Acesso em: 4 maio. 2013.

VIVIER, C. **Live Scratchpad**http://neonux.github.io/LiveScratchpad/, , 2012. Disponível em: <neonux>

Anexos

Anexo 1 : Protocolo de comunicação em javascript

```
//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);
}
```

Anexo 2: Jogos do Experimento

Resgate

Local: Montanhas.

Objetivo: sincronização de andamento.
Conteúdo: andamento; imitação rítmica.
Modalidades de comportamento musical: apreciação; performance, técnica.
Habilidades: percepção auditiva; reação e prontidão; sincronização; controle motor fino.
Público Alvo: Crianças do ensino fundamental I e II

Estrutura do jogo:
Música com variações de andamento (ex.: edição do *playback* da canção *Maria Fumaça*). O usuário irá apagar o fogo nos vagões salvando os passageiros do trem em chamas. Para tanto, deverá apertar teclas para acelerar ou *rallentar* de modo a sincronizar o seu andamento com o trem.

O fogo deverá ser apagado nos vagões antes do final da trilha. Para o resgate de cada personagem, ele ouvirá uma sequência de andamento. Termina quando chegar ao final do percurso com os passageiros salvos.

Três níveis de dificuldade progressivos. Nos níveis mais difíceis, colocar mais subidas e descidas, onde o trem vai *rallentar* e acelerar.

Itens
Alguns itens ajudarão ou atrapalharão o jogador durante o jogo. São eles:
- Arvores – As árvores aparecem na frente do personagem para que esse não consiga enxergar os vagões, com isso ele deverá se guiar apenas com a música
- Carteiro – Em alguns instantes aparecerá um carteiro entregando itens que podem ser bons ou ruins. Os itens são:

- Primeiros Socorros – ao receber o jogador ganhará um acréscimo no seu HP, caso tenha deixado algum vagão pegar fogo.

- Gelo – Ao receber gelo o jogador ficará paralisado, e deverá aumentar o andamento para conseguir se livrar do gelo.

Controles

O jogador controlará o personagem através das setas do teclado, direita e esquerda, em movimentos alternados, para conseguir entrar no andamento da música.

Imagem de referência

Abaixo uma referencia de como o jogo deve se apresentar visualmente. Mudanças são permitidas.

Fases

O jogo terá 9 fases sequenciais. Segue uma possível configuração inicial para as fases. Precisará passar por ajustes na etapa de balanceamento.

Fase 1

- Mecânica básica
- Musica A

- Cenário A

Fase 2
- Mecânica básica
- Musica B
- Cenário A
- Poucas árvores são inseridas

Fase 3
- Mecânica básica
- Musica C
- Cenário A
- Carteiro surge 3 vezes (apenas com Primeiros Socorros)
- Poucas árvores são inseridas

Fase 4
- Mecânica básica
- Musica D
- Cenário B
- Carteiro surge 4 vezes (Primeiros Socorros 60%, Gelo 20%)
- Aumenta quantidade árvores

Fase 5
- Mecânica básica
- Musica E
- Cenário B
- Carteiro surge 8 vezes (Primeiros Socorros 50%, Bloco de Gelo 50%)

Fase 6
- Mecânica básica
- Musica F
- Cenário B

Fase 7

- Mecânica básica
- Musica G
- Cenário C
- Carteiro surge 8 vezes (Primeiros Socorros 10%, Bloco de Gelo 90%)

Fase 8

- Mecânica básica
- Musica H
- Cenário C

Fase 9

- Mecânica básica
- Musica I
- Cenário C
- Carteiro surge 8 vezes (Bloco de Gelo 100%)

Rota Certa

Local: Oceano.

Objetivo: game educacional que busca criar automações de determinados conhecimentos musicais.
Conteúdo: Timbres, estilos e alturas.
Habilidades: percepção auditiva; reação e prontidão; teoria musical.

Gênero: puzzle, relacione a coluna da esquerda com a direita.
Público-alvo: crianças do Ensino Fundamental I e II.
Plataforma: flash, web.

O jogo faz parte do conjunto de games que compõe a plataforma de educação musical Turma do Som.

Setup:
Tela dividida em 4 partes:
- *HUD*: na parte superior da tela com os campos para:
- Tempo total: inicia-se com 10 segundos;
- Pontuação: igual ao número de levels avançados
- Área de pergunta: logo abaixo da HUD.
- Box onde serão feitas as perguntas por extenso
- Área de resposta: centro da tela o Leme com as possibilidades de resposta em suas "empunhadura"
- Tempo da pergunta: parte inferior da tela o Representada por um veleiro que se desloca da esquerda para a direita.

Ações do jogador:
- Escolher resposta: clicar sobre uma das opções do timoneiro

- Clicando na empunhadura correta:
HUD: tempo é adicionado em 2 segundos

Timoneiro: feedback visual é ativado com + 2 subindo e som de acerto é tocado
Área do veleiro: vento é criado impulsionando a embarcação e ela avança mais rapidamente para a próxima marcação

- Clicando na empunhadura errada:
HUD: tempo é reduzido em 5 segundos
Timoneiro: feedback visual é ativado com - 5 subindo e som de erro é tocado.
Área do veleiro: nuvenzinha de tempestade é criada retardando a embarcação e ela avança lentamente para a próxima marcação

Condição de derrota:

Quando o tempo for igual a zero, o jogador perde a partida. Marcações das perguntas: sempre que o barco passar por uma das marcações do cenário (bandeirinhas), uma pergunta será feita Dependendo do estágio, existem mais ou menos perguntas.
O jogador terá então que responder no intervalo entre as marcações. Caso não responda, ele perde 5 segundos quando o barco toca nessas marcações.
Passando de estágio: quando o barco chegar no final de seu percurso (porto), o jogador passa de fase. Medida de pontuação: quando o usuário passar de fase, ele ganhará uma estrela. À cada 5 acertos, trocar o objeto de pontuação para ele mesmo +5. Depois por 10, 15, 20...

Progressão:

Cada nível terá uma quantidade estipulada de perguntas. Apresentar sempre "FASE X" antes de começar de fato o estágio.
À cada 5 fases passadas, mudar o feedback sonoro e visual dando a sensação de vitória parcial + período de descanso para o jogador. Essas 5 fases são marcos para o jogador.
Se possível termos opções variadas de imagens de background em uma ordem determinada e quando chega-se ao último, entra em loop

Ciclo 1

- Level 1: 1 pergunta
- Level 2: 1 pergunta
- Level 3: 2 perguntas
- Level 4: 2 perguntas

- Level 5: 3 perguntas

Ciclo 2 em diante

- Level A: level A do ciclo anterior + 1
- Level B: level B do ciclo anterior + 1
- Level C: level C do ciclo anterior + 1
- Level D: level D do ciclo anterior + 1
- Level E: level E do ciclo anterior + 1

Space War

Local: Espaço.

Conceito: Jogo de Aventura no estilo de Space Invaders. Onde o jogador deverá eliminar todos os inimigos em uma única fase. O jogador é representado por uma nave especial e poderá atirar nos seus inimigos. Os Inimigos também atiram contra o jogador e vão se aproximando com o passar do tempo.

Plataforma: flash, web.

Público-alvo: faixa etária entre 15 e 25 anos.

Movimentos do Jogador

- Deslocamento na horizontal, direita e esquerda
- Atirar para cima

Movimento das naves inimigas

- deslocamento na vertical, para baixo.
- Atirar para baixo

Quantidade de inimigos - inicialmente 30 inimigos (necessita ajustes).

Quantidade de Vidas – 3 vidas

Imagem de referência