UNIVERSIDADE ESTADUAL PAULISTA
Faculdade de Ciências - Bauru

Bacharelado em Ciência da Computação

Felipe Teixeira de Avelar

Desenvolvimento de um jogo educacional de cunho sócio-histórico

UNESP

2012

Felipe Teixeira de Avelar

Desenvolvimento de um jogo educacional de cunho sócio-histórico

Prof. Dr. Marco Antônio Rahal Sacoman

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, Unesp, campus de Bauru, como parte do Trabalho de Conclusão de Curso.

UNESP

2012

Felipe Teixeira de Avelar

Desenvolvimento de um jogo educacional de cunho sócio-histórico

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, Unesp, campus de Bauru, como parte do Trabalho de Conclusão de Curso.

BANCA EXAMINADORA

Marco Antônio Rahal Sacoman
Professor Doutor
DCo - FC - UNESP - Bauru
Orientador

Simone das Graças Domingues Prado
Professora Doutora
DCo - FC - UNESP - Bauru

Andréa Carla Gonçalves Vianna
Professora Doutora
DCo - FC - UNESP - Bauru

Bauru, 29 de Novembro de 2012.

Agradecimentos

Não é possível expressar a minha gratidão a todos que quero nesse pequeno espaço, mas tentarei fazê-lo da melhor forma possível.

Agradeço a todos os meus amigos e colegas, que me deram força para seguir em frente e, em grande parte, formaram meu caráter e meus ideais, direcionando-me por esse caminho inusitado e desconexo que é a vida.

Agradeço, também, a todos os professores que compartilharam um pouco do seu conhecimento e foram pacientes em transmitir um pouco do que sabiam e que sempre foram solícitos em aumentar a minha bagagem teórica e cultural, indicando leituras, auxiliando com dúvidas, muitas vezes alheias ao tema que tratavam em sala de aula, além das diversas vezes que permaneceram após as aulas para conversar informalmente sobre os mais diversos tópicos.

Quero agradecer, em especial, ao professor Sacoman, que teve a paciência de me orientar, prover um grande suporte nas horas de dúvida e complicações com o projeto e, ainda, tornar possível o progresso adequado desse trabalho.

Ainda agradeço e dedico esse trabalho a minha família, que tornou possível, sob todos os aspectos, que esse trabalho fosse terminado, desde o auxílio financeiro, até o psicológico. Dedico em especial a minha irmã, Fernanda, pelos momentos que foram sacrificados por não estar ao seu lado buscando finalizar esse projeto.

Resumo

Com a evolução dos meios tecnológicos voltados para escola, faz-se necessário a modificação e criação dos paradigmas pedagógicos aplicados em sala de aula. Tendo isso em vista, esse trabalho tenta explanar de forma clara e sucinta como desenvolver um jogo completo, seguindo os preceitos adequados de pedagogia e Engenharia de Software, explicitando os principais artifícios lúdicos e gráficos adequados para o desenvolvimento intelectual do aluno.

Para tanto, é estudado, ao longo dessa monografia, o processo de desenvolvimento por completo, fazendo uso da linguagem de desenvolvimento *ActionScript 3.0*, auxiliada por alguns outros tipos de tecnologias, mostrando como o uso de artigos tecnológicos na sala de aula é possível e altamente indicado.

Palavras-chave: ActionScript 3.0, Engenharia de Software, Pedagogia, Jogos educacionais.

Abstract

With the technology evolution for school proposals, became necessary the creation and modification of the pedagogical paradigms applied in the classroom. With this in mind, this monograph attempts to explain clearly and succinctly how to develop a complete game, following the precepts of appropriate pedagogy and software engineering steps, explaining the main entertainment devices and graphics suitable for the student's intellectual development.

Therefore, it is studied throughout this academic work, the complete process of development, making use of the development language *ActionScript 3.0*, aided by some other technologies, showing that the use of technology articles in the classroom is possible and highly indicated.

Keywords: ActionScript 3.0, Software engineering, Pedagogics, Educational games.

Lista de Algoritmos

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

A indústria de jogos eletrônicos cresceu muito nos últimos cinquenta anos. Com o passar dos anos, esse mercado se popularizou cada vez mais entre os jovens e se tornou uma das principais frentes de entretenimento existentes no mundo moderno.

Como ferramenta pedagógica, os videogames se mostraram extremamente poderosos. Sendo utilizados tanto para treinamentos específicos, como por exemplo, *softwares* de gestão (*SimCity*, *Theme Series*) e simuladores (*Flight Simulator*, *Rail Works*), como para o ensino de matérias básicas do currículo fundamental.

Com a expansão desse mercado, cada vez mais empresas entram na competição para desenvolver seus jogos voltados para o ensino, tanto específico como básico. Em franca expansão, essa indústria deve tomar certos cuidados, tendo a sensibilidade de desenvolver um *software* condizente com a faixa etária dos estudantes que farão uso desse instrumento, atentando para os objetivos indiretos daquilo que o jogo deve proporcionar (PASSERINO, 1998), aliado aos padrões comuns de qualidade de *software* (MORATORI, 2003).

Desde que esses parâmetros sejam atingidos, o educador terá uma poderosa ferramenta em suas mãos. Caberá também aos desenvolvedores explicitarem, muitas vezes, através de treinamento, como o jogo pode ser adequado ao paradigma pedagógico da instituição.

Se feito de forma correta, os jogos pedagógicos podem ser utilizados de diversas formas, podendo desde passar a matéria ao aluno pela primeira vez até uma revisão mais aprofundada do conteúdo. A partir daí será papel do educador implementar a ferramenta no momento adequado e de forma correta, além de receber o retorno dos alunos quanto ao conhecimento relativo a matéria, podendo identificar e sanar possíveis deficiências do estudante em partes específicas da matéria.

Para desenvolver a ferramenta proposta nesse trabalho, será utilizada a linguagem *ActionScript 3*. Apesar de existirem diversas ferramentas adequadas para atingir o objetivo proposto, essa linguagem mostrou-se a forma mais eficiente em termos de tempo e qualidade, mostrando o melhor custo benefício para a criação de jogos educativos devido à alta qualidade do trabalho final que a ferramenta oferece, desde que usada adequadamente, além da rápida curva de aprendizagem.

1.1 Problema

Com o avanço do uso de tecnologia nas escolas públicas, tais como a compra de computadores e a implementação do acesso à rede mundial de computadores (Internet), torna-se necessário propôr novos meios para a educação.

Faz-se necessário utilizar essas tecnologias para que elas ampliem vigorosamente a taxa de aprendizado do aluno sem que eles percam o foco perante aos novos desafios, além de utilizar esses novos equipamentos para auxiliá-los em sala e, assim, facilitar a absorção do conteúdo abordado.

Aliado a todos esses fatores, a concentração dos alunos vem diluindo-se, tendo em vista a quantidade de tarefas que ele realiza ao mesmo tempo durante o dia. Eles fazem os deveres de casa, enquanto conversam com amigos e assistem a um vídeo via *streaming* na Internet. Isso torna a atenção e concentração do aluno muito frágil e pequena, sendo perdida em questão de minutos (KROEFF, 2009).

Assim, torna-se necessário encontrar novos meios de prender a atenção desses alunos. Uma proposta bastante aceita é associar as atividades pedagógicas à tarefas comuns ao cotidiano deles, tornando a seleção e absorção de informação uma atividade natural e de fácil execução.

1.2 Justificativa

A ideia de utilizar jogos na sala de aula para otimizar o aprendizado dos alunos não é nova. Seguindo essa linha, aliado ao fato da modernização dos equipamentos escolares, muitas empresas estão desenvolvendo jogos eletrônicos voltados para educação. Para isso, muitas empresas tem utilizado a linguagem *ActionScript 3.0*, por oferecer uma melhor integração com a parte de *design*, visto que essa linguagem tem integração direta com o pacote *Adobe*, uma taxa de velocidade de aprendizagem relativamente alta, além do suporte que lhe é oferecida.

Muitos professores preferem adotar esses jogos como atividades complementares à aula, visto que, fugindo do paradigma tradicional de didática, aplicar o conteúdo através de jogos tem se mostrado uma forma eficiente de captar a atenção do aluno e melhorar o seu desempenho acadêmico.

Com o intuito de aproveitar esse mercado aquecido, pretende-se desenvolver um jogo de cunho sócio-histórico, tentando associar diversão e educação em uma atividade que aborda história e sociologia, associando essas matérias com uma atividade de lazer, comum a grande parte dos jovens da atualidade.

1.3 Objetivos

A finalidade é desenvolver um jogo em *ActionScript 3.0* utilizando as técnicas de Engenharia de Software e pedagogia adequadas com o intuito de torná-lo implementável e possível de ser realizada a manutenção de forma simples e intuitiva. Esse jogo abordará temas de história republicana brasileira e bases de política sociológicas, com o intuito de, além da revisão dos tópicos de história, estimular o aluno a ser mais crítico e questionador dos paradigmas éticos e morais na política nacional.

1.3.1 Objetivo geral

Desenvolver um jogo eletrônico com o intuito de abordar assuntos sócio-históricos relativos a história política recente do Brasil.

1.3.2 Objetivos específicos

- Estudar as especificações da linguagem *ActionScript 3.0*;
- Adequar um jogo à estrutura pedagógica vigente na escola pública;
- Estudar bibliotecas auxiliares e verificar a sua aplicação no projeto;
- Desenvolver o jogo utilizando *ActionScript 3.0*.

1.4 Organização do trabalho

No capítulo 2 são tratados os assuntos que suportam esse trabalho. Para isso, serão abordados brevemente os temas relativos aos jogos eletrônicos, educação utilizando jogos e uma breve introdução à engenharia de *software*.

Já no capítulo 3, são tratadas todas as fases do projeto. Tendo em vista que ele é subdividido em três grandes partes, nesse capítulo são abordados desde as ferramentas e os métodos selecionados até o processo de documentação final do trabalho.

Para finalizar, no capítulo 4 são abordados os problemas enfrentados e as soluções encontradas, os trabalhos futuros relacionados a esse projeto e as considerações finais acerca do produto desenvolvido ao longo dessa monografia.

Capítulo 2

Referencial teórico

A história dos jogos eletrônicos sempre foi diretamente entrelaçada com a educação. Eles sempre serviram para trabalhar as técnicas de desenvolvimento de jovens estudantes de computação. Com o tempo, o segmento de jogos se tornou industrialmente lucrativo, com grandes empresas e desenvolvedores independentes surgindo no cenário de desenvolvimento de jogos, consolidando-se a cada dia que passa como um mercado amplo e dinâmico.

Com o passar do tempo, muitos trabalhos passaram a tratar da influência dos jogos eletrônicos (videogames) na vida dos jovens, seus efeitos positivos e negativos e como seria possível tirar proveito dessa forma de entretenimento, que consegue captar a atenção do usuário devido ao seu alto nível de interatividade nas atividades, provendo aos professores uma poderosa ferramenta pedagógica.

2.1 Evolução dos jogos eletrônicos

A maioria dos historiadores consideram que ela iniciou em 1958, sem que o criador tivesse a intenção de torná-lo algo comercial ou voltado para outros fins além da diversão. Como descreve Reis (2005), a história desse tipo de jogo é relativamente recente e se segue a seguinte ordem de fatos.

O primeiro jogo foi desenvolvido pelo norte-americano Willi Higinbothan, com a intenção de atrair a atenção do público que visitava o complexo de laboratórios da *Brookhaven Nacional Laboratories*.

Conhecido como *Tennis for Two*, esse jogo era uma espécie de tênis. Ele era mostrado em um osciloscópio e processado em um computador analógico como pode ser visto na Figura 1. O criador declarou que seria possível integrá-lo a uma televisão, o que tornaria viável uma versão doméstica do seu invento, que nunca foi patenteado.

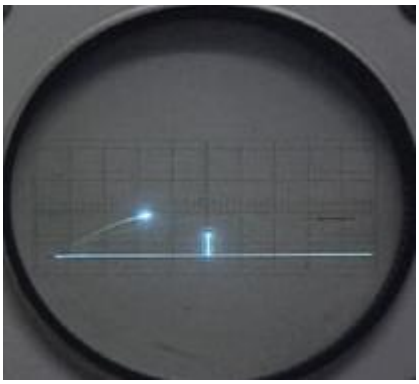

Figura 1: Imagem de um osciloscópio executando o jogo *Tennis for Two*.
Fonte: Imagem extraída do site Gamersquarter[1]

Em 1962, mais um jogo que tinha como intenção atrair a atenção do público foi criado no Instituto de Tecnologia de Massachusetts (MIT). Com o único motivo de tornar a visita mais agradável e destacar a criação do primeiro minicomputador já produzido, o *Spacewar* era um jogo de batalha espacial. Ele foi desenvolvido, principalmente, pelo estudante Stephen Russel e teve a ajuda de Dan Edwards, Robert Sanders, Steve Piner, Peter Samson, Martin Graetz e Alan Kotok e implementava conceitos reais de física como, por exemplo, gravidade e aceleração. Além de atrair a atenção do público, o outro intuito do jogo era induzir os jovens a ingressassem na área de computação.

No ano de 1968, surge a primeira patente relacionada a videogames. Ralph Baer, um alemão formado em engenharia eletrônica, patentou a *Brown Box*, o qual pode ser visto na Figura 2. Esse aparelho, que foi o primeiro protótipo de videogame patenteado, executava jogos de futebol, tênis de mesa, voleibol e tiro quando ligado a uma televisão.

Figura 2: Imagem da *Brown Box*.
Fonte: Imagem extraída do site Pongmuseum[2]

Em 1971, Nolan Bushnell desenvolveu o *Computer Space*, uma versão *arcade* do *Spacewar*, que foi o primeiro videogame explorado comercialmente (BATTAIOLA, 2000).

[1]Endereço eletrônico da imagem: http://www.gamersquarter.com/tennisfortwo/
[2]Endereço eletrônico da imagem: http://www.pongmuseum.com/history/baer-videogameinvasion.php

No ano seguinte, surge o primeiro aparelho doméstico de videogame. Baer apresentou a sua *Brown Box* para grandes empresas e a *Magnavox* lançou o *Odyssey 100*, também conhecido como o primeiro videogame da história.

Figura 3: Imagem do *Odyssey 100*.
Fonte: Imagem extraída do site Playerschoicegames[3]

Ainda segundo Reis (2005), esse sistema permitia a troca de cartuchos. Inicialmente foram lançados 12 cartuchos, em sua grande maioria jogos de esportes. O cenário dos jogos eram cartões plásticos coloridos que deveriam ser fixados na tela da televisão para simular a textura de fundo. Ainda trazia consigo um rifle rudimentar que era opcional para os jogos de tiro.

Ainda nesse ano, foi fundada a empresa *Atari*, uma empresa específica para o desenvolvimento de jogos. Seus idealizadores foram Bushnell e Ted Dabney. Seu primeiro lançamento foi uma versão *arcade* de um jogo de tênis de mesa, conhecido como *Pong*.

Graças a sua simplicidade e grande diversão, *Pong* alcançou um grande sucesso entre os jogadores, o que fez com que a *Atari* lançasse uma versão feita para jogar em casa, chamada *Home Pong* (Figura 4), em 1974, pois foi o precursor do grande sucesso que a indústria de videogames viria a se tornar.

[3]Endereço eletrônico da imagem: http://www.playerschoicegames.com/largerlistofsystems.html

Figura 4: Imagem do *Home Pong* desenvolvido pela *Atari*.
Fonte: Imagem extraída do site Playerschoicegames[4]

Com a introdução do *Zircon/Fairchild Channel F* no mercado em 1976, o sistema rudimentar de trocas de cartuchos do *Odyssey 100* foi aperfeiçoado, sendo ele citado como o primeiro console programável da história da indústria dos jogos eletrônicos.

A intenção desse console era possibilitar a reutilização do mesmo *hardware* para executar outros *softwares*, permitindo que os usuários gastassem menos com o aparelho e mais com os jogos. Apesar de ser uma boa ideia, o *Zircon* tinha jogos de baixíssima qualidade e sua vida útil foi relativamente curta.

A partir de 1977, o crescimento da indústria dos *games* passou a ficar clara, fazendo a *Atari* ser comprada pela *Warner*. Além disso, surgiram algumas grandes desenvolvedoras de jogos, conhecidas até hoje, como a *Sega* e a *Konami*. Nos anos seguintes foram lançados diversos consoles e atualizações de *hardware*, podendo-se destacar o lançamento da segunda geração do *Odyssey*, o primeiro portátil, conhecido como *Microvision*, do *Atari 2600*, do *Intelevision*, desenvolvido pela *Matell Eletronics*, além do *Nintendo Famicom*, desenvolvido pela *Nintendo*.

O *Nintendo Famicom* (Figura 5) foi lançado no Japão em 1983, um ano antes do *crash* dos videogames no mercado americano. Esse problema na indústria de jogos foi causado principalmente pelos lançamentos de jogos de baixa qualidade, sendo o carro chefe dessa crise o *Atari 5200*, que além de possuir jogos de qualidade extremamente inferior, tinha um controle de difícil manipulação.

[4]Endereço eletrônico da imagem: http://www.playerschoicegames.com/2011/05/15/atari-pong/

Figura 5: *Nintendo Famicom*.
Fonte: Imagem extraída do site Kotaku[5]

O "salvador" do mercado de jogos eletrônicos nos Estados Unidos chegou no ano de 1985, com o lançamento do *Nintendo Entertainment System* (NES), a versão ocidental do *Nintendo Famicom* que tem a imagem de um exemplar na Figura 6. A *Nintendo* usou uma técnica de propaganda que mostrava seu sistema como "um centro de entretenimento e não uma máquina para rodar(sic) jogos"(REIS, 2005).

Figura 6: *Nintendo Entertainment System* com um *design* renovado para se adequar aos padrões ocidentais.
Fonte: Imagem extraída do site Thegamersdungeon[6]

[5]Endereço eletrônico da imagem: http://www.kotaku.com.au/2010/10/report-why-is-the-nintendo-famicom-red/
[6]Endereço eletrônico da imagem: http://www.thegamersdungeon.com/2012/01/buying-nintendo-8-bit-nes-and-16-bit.html

Esse sistema dominou o mercado de videogames até o final da década de 80, pois em 1989 a *Sega* lançava o primeiro console 16-bits da história. Conhecido como *Megadrive*, esse console impressionou pela grande evolução gráfica em relação aos videogames 8-bits. Esse console contou com a ajuda das maiores *softhouses* da época: *Konami*, *Capcom*, *Eletronic Arts*, entre outras.

O *Megadrive* deu início ao que muitos jogadores chamam de "geração de ouro" dos videogames. Ainda em 1989, a *Nintendo* lançou o *Gameboy*, seu primeiro portátil. Ele foi um enorme sucesso e teve alguns clássicos associados ao seu nome. Dentre eles pode-se citar, principalmente, o jogo *Tetris*, um grande sucesso do mundo dos *games*, que ficou sob a patente da *Nintendo* por quatro anos e a franquia de jogos *Pokémon*, também pertencente à *Nintendo*, uma das franquias de maior sucesso no mundo dos *games* (KENT, 2001).

Para combater o *Megadrive*, a *Nintendo* lança, em 1990, o sucessor do *NES*, conhecido como *Super Famicom* no Japão. O *Super Nintendo Entertainment System* (SNES), como ficou conhecido no ocidente, tinha uma qualidade gráfica e sonora um pouco superior aos seus concorrentes, entretanto seu processador era mais fraco.

Figura 7: Do lado esquerdo, um exemplar de *Megadrive* e à direita um *SNES*.
Fonte: Imagem extraída do site Legacy.Gamelib[7]

Com uma lista base de lançamento muito forte, o *SNES* foi um estrondoso sucesso de vendas em todo o mundo. O console emplacou diversos clássicos no mercado dos *games*, tais como: *Super Mario World*, *The Legend of Zelda: Link to the Past*, *Top Gear*, *Chrono Trigger*, entre outros. Esse console e o *Megadrive* fizeram o mercado de *games* se consolidar de vez, tendo uma competição extremamente disputada e com lançamentos de jogos de altíssima qualidade técnica relativo à parte de *Game Design* e jogabilidade.

[7]Endereço eletrônico da imagem: http://legacy.gamelib.com.br/stories/read/a-uniao-faz-a-forca-2d-e-3d.html

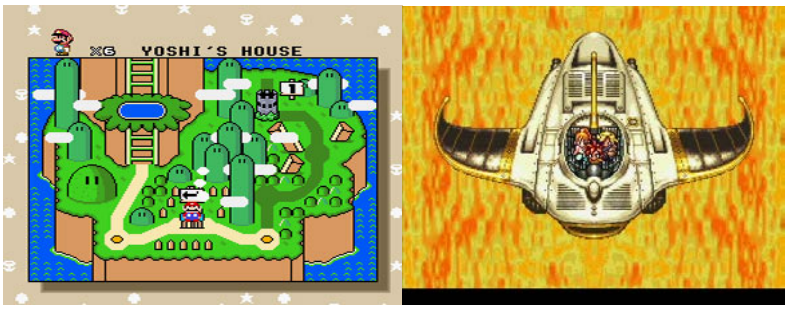

Figura 8: *Super Mario World* e *Chrono Trigger*.
Fonte: Imagens extraídas do site Nintendo[8]

Os clássicos lançados para esses videogames serviram de base para praticamente qualquer tipo de jogo e jogabilidade existente hoje. Essa geração também foi responsável pela consolidação da padronização dos processos de *game design*, que ainda eram muito precários nas gerações anteriores.

A partir dessa consolidação, o mercado dos *games* se desenvolveu sempre buscando atender a uma qualidade gráfica e sonora superior. Isso pode ser notado no lançamento da geração 32/64-bits, que permitiu o uso do paradigma de jogos 3D, possibilitando ao jogador ter noções mais próximas do mundo real, implementando a profundidade como um fator a ser usado nos jogos. Um bom exemplo de implementação desse paradigma é o jogo *Final Fantasy VII*, um sucesso do gênero *RPG*, que pode ser visto na Figura 9.

Figura 9: *Final Fantasy VII*.
Fonte: Imagens extraídas do site Twinfinite[9]

[8]Super Mario World: http://www.nintendo.com/games/detail/OnTm1QccFa_Ht39i-dKiI-Af8WRu2Cje e Chrono Trigger: http://www.nintendo.com/games/detail/Sle1SM12gBdPli4jNLSQMAzfoc5aYuL2

[9]Imagem extraída de http://www.twinfinite.net/blog/2012/06/28/featurama-why-do-we-need-final-fantasy-vii-hd/

Houve também, posteriormente, modificações na forma de jogar. As mais recentes trocaram os tradicionais *joysticks* para inserir o conceito de movimentação ao mundo dos jogos eletrônicos. Os maiores exemplos vieram no console *Wii*, cujo controle capta o movimento do jogador a partir de uma espécie de bastão e no *Xbox360*, console da *Microsoft*. Ele possui uma câmera, *Kinect*, que captura a imagem do jogador e realiza os movimentos dele com o personagem, dispensando o uso de um controle físico. Além desses dois, a *Sony* também tentou implementar, sem muito sucesso, um controle baseado em movimentos no seu console *PlayStation 3*. Todos os três controles podem ser visto na Figura 10.

Figura 10: *Kinect* (*Microsoft*), *Wiimote* (*Nintendo*) e *Move* (*Sony*).
Fonte: Imagens extraídas do site Zonga[10]

2.2 Os jogos na educação

Como abordado na seção anterior, a indústria de jogos eletrônicos é uma das mais fortes quando se trata de entretenimento. Vislumbrando essa forma de diversão, muitos estudiosos da área de pedagogia tentaram associar medidas pedagógicas a jogos eletrônicos, procurando tornar a atividade de aprendizagem mais interessante para os alunos, podendo ter efeito de revisão de conteúdo ou de ensino propriamente dito.

> O jogo pode ser considerado como um importante meio educacional, pois propicia um desenvolvimento integral e dinâmico nas áreas cognitiva, afetiva, linguística, social, moral e motora, além de contribuir para a construção da autonomia, criticidade, criatividade, responsabilidade e cooperação das crianças e adolescentes.(MORATORI, 2003, p.9)

Desde que o educador tenha objetivos bem definidos em como utilizar os jogos na educação, eles podem auxiliar em qualquer espécie de atividade que o professor queira desenvolver, desde

[10]Imagem retirada de http://www.zonga.com.br/controles-de-movimento-o-embate-xbox-360-kinect-playstation-move-wii-remote-plus/

um conhecimento superficial das dificuldades dos alunos, até aplicar o conteúdo de forma divertida e agradável para o estudante, passando por uma rápida abordagem de revisão ou introdução para uma nova temática a ser trabalhada.

Para tanto, o jogo deve conter algumas características básicas para se tornarem úteis para o professor, como trata Passerino (1998) em seu trabalho. Segundo a autora, um jogo deve:

- Apresentar as informações de diversas formas, sendo claro, direto e lógico;
- Promover um certo nível de concentração e coordenação do usuário;
- Permitir que o usuário desenvolva a criatividade sem se preocupar com os erros;
- Realizar a integração homem-máquina de forma gráfica e que essa representação gráfica seja coerente;
- Possibilitar ao usuário receber os resultados de suas ações imediatamente, permitindo a autocorreção rapidamente e aumentando a autoestima do aluno;
- Ser capaz de repetir os exercícios infinitas vezes.

Para cumprir essas especificações, um *game* deve se apresentar ao jogador de forma cuidadosa, não se preocupando apenas com o conteúdo, mas, também, com a forma como apresentá-lo e abordá-lo, sempre motivando o usuário a desenvolver seu intelecto.

Passerino (1998) também ressalta a importância de considerar os objetivos indiretos do jogo, que são:

- A memória visual, auditiva e cinestésica;
- Orientação espacial, em duas ou três dimensões, e temporal;
- A coordenação visual e manual;
- Percepção auditiva e visual;
- O raciocínio lógico-matemático;
- A expressão linguística, tanto oral, como escrita;
- O planejamento e a organização.

Por isso, um jogo eletrônico não pode ser analisado apenas pela sua qualidade de *software*, mas é preciso verificar, também, os aspectos pedagógicos do jogo e o que ele vislumbra passar ao estudante ao fim da atividade.

Relacionando o trabalho de Passerino (1998) com o de Moratori (2003), chega-se à conclusão de que o desenvolvimento intelectual da criança para a resolução de problemas se dá a partir da criatividade. Sendo assim, é essencial que o estudante seja inserido em um contexto que utilize desde a imaginação até a abstração de diversificadas técnicas para a solução de tipos específicos de problemas. Logo, o jogo seria a ferramenta ideal, já que ele se alia à imaginação, podendo representar situações imaginativas muitas vezes, de forma interativa e divertida.

Tendo em suas mãos uma ferramenta tão poderosa, caberá ao educador orientar seus alunos para que usem essa ferramenta de forma adequada, realizando assim o principal objetivo do ensino-aprendizagem, que é o desenvolvimento cognitivo.

Além disso, o jogo deve proporcionar uma experiência agradável para o usuário. Sendo assim, segundo Moratori (2003), o programa deverá respeitar as seguintes características básicas de qualidade relacionadas à Engenharia de Software:

- Execução rápida;
- Não existirem erros na execução;
- Interface amigável;
- Tempo suficiente para a exibição dos *frames* de cada tela;
- A possibilidade de ajuda;
- Um bom nível de interatividade;
- Possibilidade de sair do jogo a qualquer momento.

Em resumo, um jogo deve atender a diversas questões pedagógicas, enquanto cumpre os tradicionais requisitos de Engenharia de Software relativos à qualidade de qualquer tipo de *software*, para que se desempenhe as funções básicas de desenvolvimento intelectual do aluno. O jogo deverá ser uma atividade que foque no processo de construção cognitiva do aluno e não um mero reprodutor de conteúdo que procura apenas o resultado final da aprendizagem.

2.3 Engenharia de Software

Nos últimos 50 anos, o desenvolvimento de *softwares* tornou-se cada vez mais importante no dia-a-dia das pessoas. Arquiteturas mais sofisticadas, profundas melhorias realizadas na parte de *hardware*, além do aumento da capacidade de armazenamento e de memória, aliada aos mais variados tipos de entrada e saída, que a área possibilitou, tornaram os computadores sistemas mais sofisticados e complexos (PRESSMAN, 2006).

Tendo isso em vista, tornou-se necessário desenvolver uma técnica que organizasse e proporcionasse um controle do fluxo gigantesco de informação que começou a circular com esses novos adventos tecnológicos. Este fato levou a indústria a fazer algumas perguntas que, segundo Pressman (2006), mostram a sua preocupação com a forma que o *software* é desenvolvido, o tempo que ele leva para ser terminado, porquê é difícil realizar a manutenção e calcular o progresso de seu desenvolvimento.

Levando tudo isso em conta, percebeu-se a necessidade de criar uma área que fosse capaz de suprir as carências burocráticas e processuais do desenvolvimento de *software*, ou seja, torná-lo mais procedimental, possibilitando a reprodução dos procedimentos utilizados anteriormente como uma forma de experiência e tratamento de problemas durante o processo. Assim surgiu a engenharia de *software*.

Engenharia de Software é definida como:

> Engenharia de software é metodologia de desenvolvimento e manutenção de sistemas modulares, com as seguintes características: processo (roteiro) dinâmico, integrado e inteligente de soluções tecnológicas; adequação aos requisitos funcionais e seus respectivos procedimentos pertinentes; efetivação de padrões de qualidade, produtividade e efetividade em suas atividades e produtos; fundamentação na Tecnologia da Informação disponível, viável, oportuna e personalizada; planejamento e gestão de atividades, recursos, custos e datas.(REZENDE, 2005, p. 2)

A Engenharia de Software tenta padronizar a forma de produção, manutenção e evolução do *software*, podendo, a partir disso, estabelecer prazos e custos plausíveis com o processo adotado, visando, principalmente, um resultado positivo ao final de todo o processo de desenvolvimento e/ou manutenção. (PRESSMAN, 2006; REZENDE, 2005)

Para obter esse resultado, é necessário que a Engenharia de Software seja aplicada desde o início do processo, no qual se faz uma análise dos requisitos do cliente, qual metodologia será utilizada para o desenvolvimento, utilização de meios gráficos para representar como possivelmente ficará o *software* ao final do seu desenvolvimento, até o final do processo, parte que organizará toda a documentação gerada ao longo do projeto, criação de arcabouços de manutenção e testes, além de suporte ao cliente.

Basicamente, a Engenharia de Software tenta resolver problemas que foram encontrados nos primórdios do desenvolvimento de *software*, tornando-o algo padronizado que, mesmo com uma certa burocracia inicial, agiliza o processo ao longo das etapas em que se perderiam mais tempo para se trabalhar, pois o projeto já está inteiramente organizado e planejado. Esse procedimento faz o número de modificações ao longo do processo diminuírem, além de prever, aproximadamente, quanto tempo será gasto em cada etapa, podendo, assim, controlar algum setor que não esteja dentro do prazo ou notar alguma desordem ao longo do projeto.

Capítulo 3

Projeto e desenvolvimento

Esse projeto foi dividido em três fases: pré-projeto, desenvolvimento e pós-projeto.

Foi na primeira fase que, obedecendo os paradigmas citados na seção 2.3, coletou-se e foi feita a análise prévia dos requisitos do projeto, organizando-o de um modo adequado; selecionou-se um modelo de desenvolvimento e de testes, foram escolhidas ferramentas a serem utilizadas, além de realizar um estudo de cada ferramenta a ser usada ao longo de todo o desenvolvimento do projeto.

A documentação dessa fase é necessária para evitar possíveis erros futuros por falta de preparo do projeto e agilizar a correção ao longo do processo de produção do *software*. Junto a isso, essa documentação também serve para dar uma noção geral de como ficará o projeto ao final de seu desenvolvimento. Para isso foi utilizado o diagrama UML de caso de uso.

Após essa fase inicial, entrou-se na etapa de desenvolvimento. Nela iniciou-se a programação a partir dos requisitos analisados, utilizando as ferramentas e paradigmas estudados inicialmente. Nessa etapa, também aplicou-se todos os testes e avançou-se para a fase final do projeto apenas quando todos os testes foram efetuados e, comprovadamente, não apresentaram erros críticos. Para isso utilizou-se as teorias de curva para testes apontadas por Bastos et al. (2007) e que são abordadas na seção 3.2.2.

Durante o desenvolvimento, toda a documentação de código foi feita internamente, usando-se uma ferramenta específica que é tratada na seção 3.1.1.3. Desenvolveram-se, ainda, comentários internos em inglês para possibilitar a manutenção do código em qualquer lugar do mundo.

Pela mesma razão, a programação do jogo foi feita utilizando-se o inglês, como é abordado na seção 3.1.1.2.

Terminada a fase de desenvolvimento, começou a parte de pós-projeto. Essa fase é apenas uma formalização e documentação de tudo que foi produzido ao longo do projeto, visando facilitar a manutenção e evolução do *software*, segundo os paradigmas de Engenharia de Software, já previamente estudados e abordados.

Assim como na última fase do projeto, as outras etapas também possuíram documentação própria, apesar do intuito da documentação final ser mais abrangente que a produzida nas fases precedentes.

3.1 Pré-projeto

Nessa fase foram definidos, como apresentado no Capítulo 2, o tema do jogo, as práticas de Engenharia de Software para um melhor progresso do projeto, um estudo das ferramentas utilizadas.

O texto que segue, aborda cada uma dessas etapas, seus detalhes e características básicas, possibilitando uma melhor compreensão de como funcionou todo o processo trabalhado na seção 3.2, facilitando, assim, a análise do trabalho de desenvolvimento ao longo de todo o procedimento e o porquê de cada passo subsequente.

Em primeiro lugar, para tornar todas as fases possíveis, incluindo o pré-projeto, fez-se necessário organizar um cronograma de desenvolvimento que tivesse em seu escopo o tempo que cada tarefa deveria ocorrer, evitando que as tarefas não fossem realizadas em tempo hábil para entregar o projeto conforme proposto inicialmente.

O cronograma proposto foi definido junto a proposta e a versão final dele pode ser visto na Tabela 1.

Tabela 1: Cronograma proposto para desenvolvimento de atividades.

		Mês				
Fase	Atividade	Jun	Jul	Ago	Set	Out
1. Pré-projeto	1.1 Definição de tema, ferramentas e paradigmas	X				
	1.2. Estudo de ferramentas, tecnologias e paradigmas necessários	X				
	1.3. Documentação	X				
2. Desenvolvimento	2.1 Programação		X	X	X	X
	2.2. Testes			X	X	X
	2.3. Documentação			X	X	X
3. Pós-projeto	3.1. Documentação					X

Definido o cronograma, o trabalho a ser desenvolvido ficou com objetivos mais claros e tornou todo o processo de desenvolvimento bem mais rápido, como é visto nas próximas seções.

3.1.1 Definição do tema, das ferramentas e dos paradigmas

Inicialmente definiu-se o tema e qual seria a melhor abordagem para ele. Feito isso, definir os paradigmas e, subsequentemente, as ferramentas foi um processo relativamente natural. Ao longo dessa seção é visto como foram estabelecidos cada um desses elementos.

3.1.1.1 Definição do tema

Foi a primeira fase de definição realizada durante o trabalho, escolheu-se um tema que combinava muito bem com a ideia do jogo, de ter um cidadão que revolta-se com o estado da política brasileira e procura melhorar o seu país, de forma honesta, com seus próprios esforços. Então, definiu-se que seria abordado o início da vida política de um candidato: a corrida eleitoral.

Foram diversas razões que convergiram para esse tema. Primeiro o fato da sua grande recorrência, a cada dois anos tem-se eleições, alternando entre a eleição municipal (prefeitos e vereadores) e a estadual/federal (governador, deputados estaduais e federais, presidente e senadores). Segundo pela multidisciplinaridade que daria ao jogo. Usando esse tema é possível, através de meios lúdicos e outros artifícios, abordar temas de história e sociologia, formando não só um aluno intelectualmente superior, mas, também, um aluno crítico e capaz de associar a história de seu país com os

devidos problemas e, assim, procurar um meio de evitar que os mesmos erros sejam cometidos novamente.

A abordagem sociológica no jogo foi definida para ser feita a partir de metáforas com elementos do jogo como, por exemplo, itens para serem coletados pelo jogador. Todas essas metáforas são devidamente explicadas ao jogador, que passa a associar elementos de uma corrida eleitoral com diversos elementos sócio-historicos desse tipo de disputa. É possível citar, entre esses elementos, o populismo e a afiliação política, como principais exemplos.

Já para tratar dos temas históricos, foi proposta uma técnica mais direta de pedagogia. Ao invés de fazer uso dos meios lúdicos que, indiretamente auxiliam em parte da história brasileira também, definiu-se como uma boa forma de tratar a revisão dos temas históricos a partir de perguntas esporádicas que abordem os tópicos passados em sala de aula.

Com essas definições, possibilitou-se a criação de um documento de *game design*, anexado ao final desse trabalho, especificando melhor como deveriam se comportar os elementos do jogo, seu significado, objetivos, entre outros. Isso serviu para esclarecer alguns tópicos relativos a Engenharia de Software envolvida no trabalho.

3.1.1.2 Definição de paradigmas

Parte dos paradigmas já estavam definidos. Esses paradigmas seguem um padrão para todos os jogos pedagógicos e são discutidos na seção 2.2 desse trabalho.

Decidido que o projeto seguiria os modelos pedagógicos propostos por Passerino (1998), faltou definir apenas os paradigmas de Engenharia de Software que seriam utilizados nesse projeto. Depois de analisados alguns modelos de desenvolvimento, incluindo modelos ágeis, devido ao tempo reduzido e a concisão dos requisitos propostos, optou-se por adotar o modelo *RAD* (*Rapid Application Development*), já que ele prestigia o desenvolvimento rápido. Ainda que seja uma adaptação do modelo cascata, esse modelo se enquadra como um modelo incremental. Pode-se definir o *RAD* como:

> O *RAD* (*Rapid Application Development*, desenvolvimento rápido de aplicação) é um modelo de *software* incremental que enfatiza um ciclo de desenvolvimento curto. O modelo RAD é uma adaptação 'de alta velocidade' do modelo cascata, no qual o desenvolvimento rápido é conseguido com o uso de uma abordagem de construção baseada em componentes. Se os requisitos forem bem compreendidos e o objetivo do projeto for restrito, o processo RAD permite a uma equipe de desenvolvimento criar um 'sistema plenamente funcional', dentro de uma período de tempo muito curto (por exemplo, 60 a 90 dias).(PRESSMAN, 2006, p.41)

Também definiu-se os diagramas UML que foram julgados necessários pela equipe de desenvolvimento. Respeitando ao máximo o conceito de redução de documentação proposto pelo modelo *RAD*, optou-se apenas por dois diagramas, o diagrama de caso de uso, como uma prévia do que seria o sistema e o diagrama de classes para fins de manutenção.

Ainda definiu-se como seria o padrão de documentação, interna e externa, do código. A parte interna de documentação do código foi toda feita em inglês, visando facilitar a manutenção do código. Já a parte externa foi feita em português, com o intuito de promover a utilização dessas classes em outros projetos nacionais. Com isso em vista, é possível explicar o porquê de algumas classes não terem sido documentadas.

A intenção da documentação externa foi para as classes genéricas e, por esse motivo, não foram documentadas as classes usadas especialmente para o desenvolvimento desse produto, mas apenas para as classes que poderiam ser utilizadas em outros projetos que venham a ser semelhantes a esse.

Ainda nessa etapa definiu-se os testes que seriam usados. Os testes escolhidos foram os testes de unidade, integração, estresse e desempenho. Teste de unidade é o teste mais básico dentre os tipos de testes; ele verifica a funcionalidade de um bloco de código específico. Já o teste de integração verifica se blocos reunidos funcionam da forma esperada. O teste de estresse tem a intenção de avaliar o *software* em condições críticas e verificar qual seu limite, além de seu desempenho nessas situações. Já o teste de desempenho verifica se o sistema se comporta de uma forma agradável para os usuários (BASTOS et al., 2007).

Como esses testes foram aplicados é explicado no subcapítulo direcionado ao desenvolvimento.

3.1.1.3 Definição de ferramentas

Com o tema e os paradigmas definidos, a documentação feita, faltou apenas definir as ferramentas que seriam utilizadas. Seguindo um processo natural, a escolha de ferramentas dividiu-se em duas partes.

Na primeira parte foram estabelecidas as ferramentas que auxiliariam na programação, linguagem e programas auxiliares de desenvolvimento e na segunda foram definidas as ferramentas para documentação. O texto que segue aborda, brevemente, as ferramentas escolhidas para o desenvolvimento.

A linguagem ActionScript 3.0 e o framework Adobe Flash Professional CS 5.5

ActionScript 3.0 (AS3) é a versão mais recente da linguagem voltada para animação em *Flash*, desenvolvida pela *Adobe*.

Desenvolvida para se trabalhar com animações de forma mais simples e natural, a AS3 permite que o programador trabalhe tanto com codificação como de forma gráfica, sendo necessário a integração com outros *softwares* no caso da segunda.

Além disso, essa linguagem fornece ao programador paradigmas recentes e muito potentes, tendo em vista que ela é orientada a objetos e eventos. Isso permite que o programador faça uso de técnicas sofisticadas tanto relativas à programação como à Engenharia de Software.

Com diversos módulos prontos, ela oferece um suporte enorme quando se trata de animações. Graças a esse suporte, aliado ao suporte para *web browsers* para executar essas animações, a AS3 ficou extremamente popular, sendo usada para diversos tipos de aplicações *web*, dentre elas, o desenvolvimento de jogos.

Surgiram vários portais que permitiram que desenvolvedores independentes com conhecimentos em AS3 mostrassem seus trabalhos, tornando possível a qualquer um tornar-se um desenvolvedor de jogos mundialmente conhecido. Alguns dos portais de jogos que podem ser citados são: *Kongregate*, *ArmorGames* e *Miniclip*.

Um dos *softwares* que possibilita a integração do AS3 graficamente é o *Adobe Flash Professional* e pode ser visto na Figura 11. Esse *software* agiliza exponencialmente o desenvolvimento de animações e jogos, tendo em vista que tudo pode ser trabalhado visualmente, fazendo uso do conceito de *timeline* e *library*. Além disso, é oferecido um suporte em XML e *javascript*, permitindo que a linguagem seja muito forte e flexível.

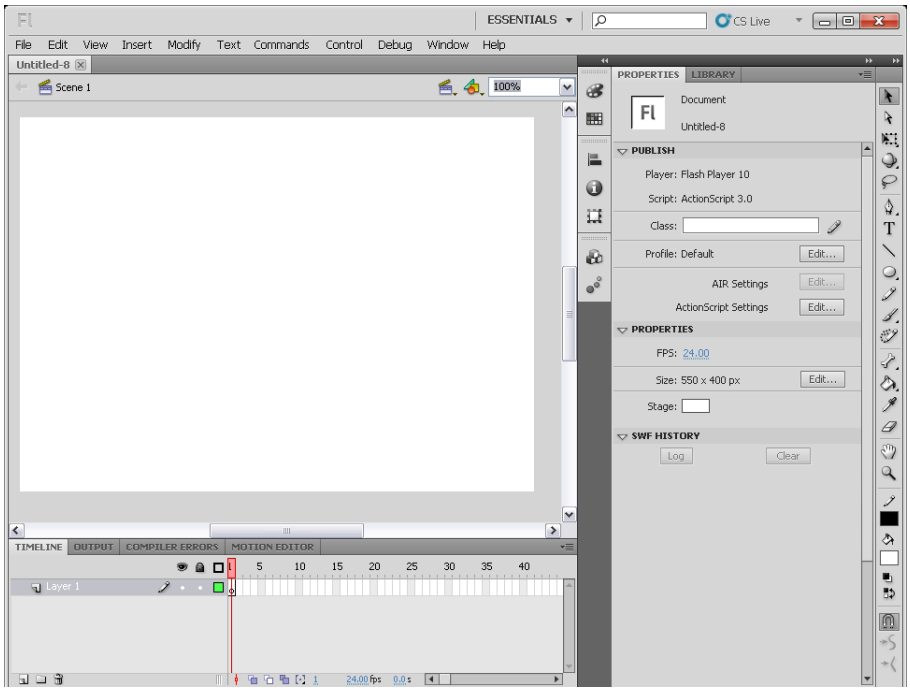

Figura 11: Interface do *Flash Professional*.
Fonte: Imagem extraída do site da *Adobe*[1]

Pode-se também criar *scripts* que automatizem tarefas no *Adobe Flash Professional*, usando *javascript*, de forma que trabalhos repetitivos dentro do *software* da *Adobe* se tornem rápidos e menos exaustivos durante a programação.

Junto a tudo isso, atualmente a *Adobe* trabalha em uma forma de converter código AS3, de forma que ele execute nativamente em HTML5, integrando as duas ferramentas e possibilitando a conversão de antigos "filmes" *flash* em animações HTML5, evitando a perda de algum trabalho devido ascensão de alguma nova tecnologia.

Tendo isso em vista, a linguagem *ActionScript 3.0* ainda pode ser considerada uma das melhores opções quando se trata do desenvolvimento de jogos para plataformas *web*. Sua flexibilidade e o suporte fornecido pela *Adobe* permitem que ela seja uma linguagem com a qual ainda se trabalhe bastante. Ainda deve-se considerar que não haverá perda de trabalho, já que a empresa que a desenvolveu procura se adequar a novas tecnologias.

Junto a isso tudo, as facilidades relativas à parte de *design* permite que trabalhos com uma maior qualidade visual seja feita e, como foi visto na seção anterior, jogos educacionais requerem um cuidado especial quanto à relação visual com o aluno.

Sendo assim, a linguagem AS3, aliada ao *framework* da *Adobe*, destaca-se como melhor opção para o desenvolvimento de jogos educacionais na atualidade. Mesmo havendo novas tecnologias *web* em ascensão, elas ainda são muito instáveis para se trabalhar, além de não possuírem um padrão para todos os *web browsers*, como o *flash player*. E, caso a tecnologia realmente venha ser substituída, a desenvolvedora tomou o cuidado de dar suporte para a migração de objetos antigos para o molde novo, sem que, como já foi citado, se tenha perda do trabalho feito.

[1]Endereço eletrônico da imagem: http://www.adobe.com/devnet/flash/articles/create-first-flash-document.html

FlashDevelop

O *FlashDevelop* é uma iniciativa *Open Source* criada para agilizar o processo de desenvolvimento de programas que utilizem *Flash*. Consiste de um editor de textos de fácil instalação e utilização, esse programa é fruto de um trabalho feito para desenvolvedores *Flash* e que foi liberado em 2005 (FLASHDEVELOP, 2012).

Esse editor possui diversas ferramentas que ajudam na velocidade de desenvolvimento, entre e, elas, pode-se citar as operações de texto, *snippets*, completação de código, integração com o *ASDoc*, entre outras.

Dessas ferramentas, a integração com o *ASDoc* é uma das mais interessantes, tendo em vista que ela é utilizada dentro da interface do programa. Quando o usuário pretende utilizar um método de uma classe definida por ele, caso ela tenha documentação, a própria interface mostra o que faz cada parâmetro e o que aquele método retorna, entre outros. O programa ainda oferece suporte para a geração da documentação a partir do *ASDoc* com uma interface amigável, evitando complicações em utilizar o gerador *inline*, como propõe a *Adobe*, mas sem eliminar as diretivas que são oferecidos como opção.

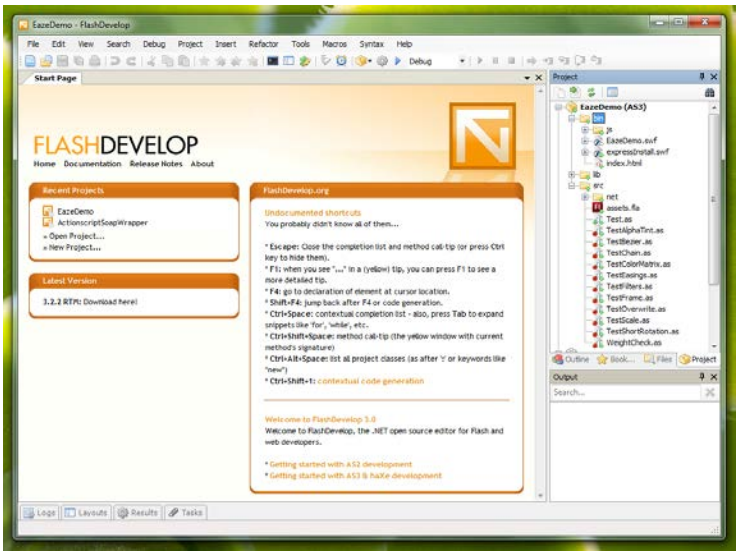

Figura 12: Interface do *FlashDevelop*.
Fonte: Imagem extraída do site flashdevelop[2]

ASDoc

O *ASDoc* é uma ferramenta desenvolvida pela própria *Adobe* que analisa comentários, no código, em um formato específico e, a partir deles, gera uma documentação HTML para todos os métodos e variáveis com níveis de encapsulamento “public” e “protected”. Também gera um índice que torna possível o acesso a tudo que foi documentado via código. Ele permite que sejam adicionadas ou removidas classes em um diretório usando diretivas. (ASDOC..., 2012)

O *FlashDevelop* oferece todo um suporte para utilizar essa ferramenta, tornando seu uso simples e intuitivo, além de dar um aspecto muito bom para a simplicidade que a ferramenta oferece,

[2]Endereço eletrônico da imagem: http://flashdevelop.org/wikidocs/images/tour/0a-interface.png

tornando a documentação de código algo simples, interessante e rápido para qualquer tipo de projeto que utilize *ActionScript 3.0*.

Optou-se por essa ferramenta para a geração de toda a documentação do código gerado pelo projeto, pois, além de ser uma ferramenta extremamente potente, ela foi desenvolvida pela própria *Adobe*, ou seja, fornece um ótimo suporte à documentação de sua linguagem.

StarUML

Outra ferramenta de iniciativa *Open Source*, o *StarUML* é uma ferramenta que conta com diversas características interessantes. Além de tornar possível a geração gráfica de diagramas UML, esse programa permite ao usuário realizar engenharia direta (diagrama gera código) e engenharia reversa (código gera diagrama) para as linguagens *Java*, *C++* e *C#*.

Ele também permite a criação de diagramas UML segundo a padronização 2.0 e fornece ao usuário uma interface simples que permite um trabalho rápido para o desenvolvimento de documentação UML (STARUML, 2012).

Essa ferramenta foi selecionada para criar o diagrama de caso de uso, tornando visual o jogo, mostrando de forma mais clara aos desenvolvedores como deveria ficar o jogo em seus estágios finais.

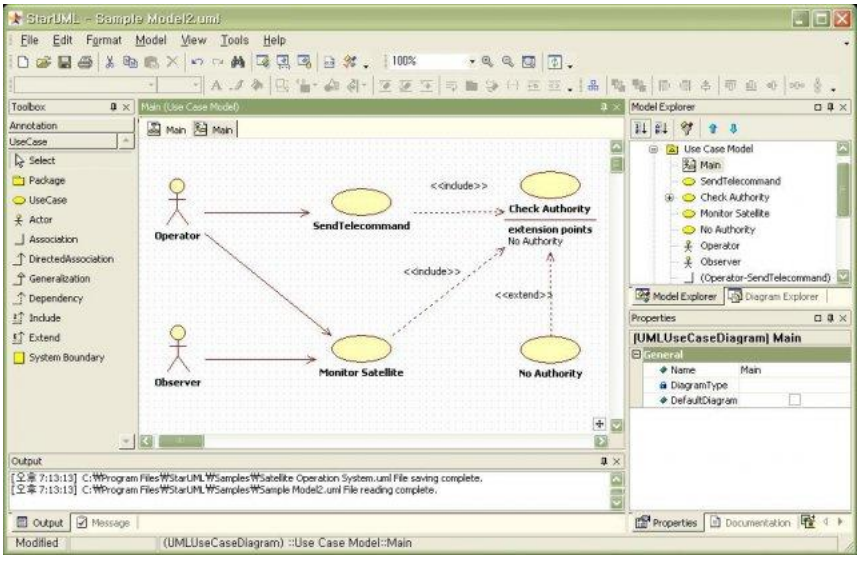

Figura 13: Interface do StarUML.
Fonte: Imagem extraída do site sourceforge.net/projects/staruml[3]

Crocus Modeller

Apesar da grande capacidade do *StarUML*, ela não oferece suporte nativo à *ActionScript 3.0*. Para resolver esse problema optou-se pelo *Crocus Modeller*.

Desenvolvido utilizando a interface *Adobe Air*, o *Crocus Modeller* é uma ferramenta UML designada especialmente para a plataforma *Flash*. Assim como o *StarUML*, ele possui engenharia

[3]Endereço eletrônico da imagem: http://sourceforge.net/projects/staruml/#screenshots

direta e reversa. No entanto ele oferece suporte apenas para diagramas de classe. Também oferece suporte ao Flex e à *ActionScript 3*, além de uma interface simples e amigável (CROCUS..., 2012).

Essa ferramenta foi selecionada para realizar uma documentação visual do código, após seu término, tornando, assim, a manutenção do código mais simples e direta para os desenvolvedores que voltem a trabalhar no código, futuramente, com extensão desse projeto.

3.1.2 Documentação

Definido todos os aspectos de desenvolvimento do projeto, foi possível gerar a documentação de *game design*, que se encontra anexado a esse trabalho, assim como foi possível criar o diagrama UML de caso de uso, como pode ser visto na Figura 14.

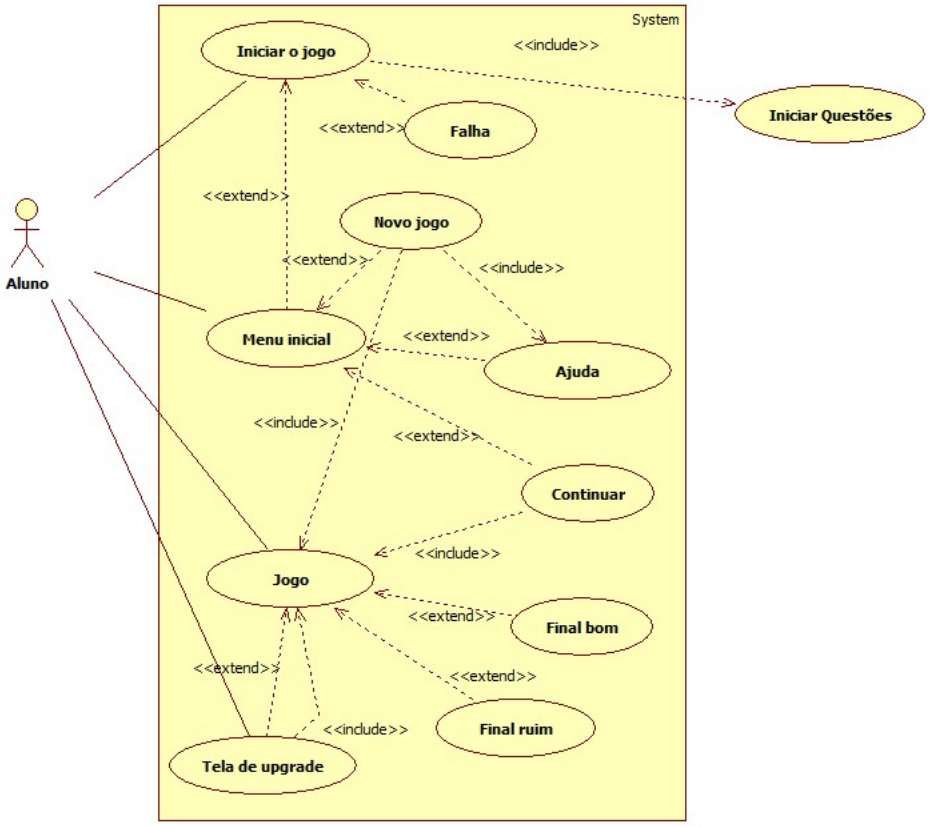

Figura 14: Diagrama de caso de uso.

A partir dessa documentação, foi possível partir para a segunda fase do projeto: a fase de desenvolvimento.

3.2 Desenvolvimento

A fase de desenvolvimento foi dividida em três partes: programação, testes e documentação.

A parte de programação foi onde o jogo realmente foi desenvolvido, onde foram criadas suas classes e eventos, como se comportava cada tela e também foi a etapa que adequou os aspectos

pedagógicos do jogo, no qual se desenvolveu praticamente toda a parte de documentação de código, deixando apenas para as ferramentas adequadas gerarem.

Nessa etapa também foi realizada a diagramação das telas e inseridas as imagens do jogo, sendo que, para isso, foi contratado um *designer* que trabalha na área de desenvolvimento de jogos pedagógicos, para desenvolver os meios lúdicos de forma adequada e condizente com a ideia do projeto.

Já na fase de testes, o código foi testado nos quatro padrões citados. Para isso foram realizados testes com técnicas específicas que são explicadas de maneira mais clara na seção destinada a essa etapa.

Ao final, gerou-se a documentação HTML das classes utilizando a ferramenta proposta na seção anterior.

3.2.1 Programação

O desenvolvimento se dividiu em dois paradigmas, orientação a objetos e orientação a eventos. A orientação a objetos é um paradigma de programação que envolve conceitos mais complexos que a programação não orientada. Entre esses conceitos os mais importantes são classe e objetos; também são importantes os conceitos de herança e polimorfismo, que tornam este um paradigma extremamente poderoso e que, a cada dia, torna-se mais comum no mercado.

Classe pode ser definida como uma forma de descrever um conjunto de objetos com características e ações semelhantes, ou seja, um grupo de objetos que faz a mesma coisa. Unido-a ao conceito de herança e polimorfismo, tem-se uma capacidade de reutilização e de generalização de código que torna esse tipo de paradigma fantástico para aumentar a velocidade de desenvolvimento, além de permitir que a reutilização de código seja facilitada.

Já o paradigma de orientação a eventos se baseia em um laço de repetição que escuta quando um evento ocorre. Ao ocorrer esse evento, ele é disparado para que todos os "escutadores" disparem a ação desejada. Esse é um paradigma muito comum em diversas linguagens com interação usuário máquina, assim como é comum, também, em sistemas operacionais.

É possível dizer que o paradigma de orientação a objetos estruturou a forma do código, enquanto a orientação a eventos realiza a comunicação entre esses módulos, realizando uma combinação forte e que permite sanar quase todos os problemas relacionados aos paradigmas de desenvolvimento de jogos eletrônicos que antes não era possível.

Seguindo esses paradigmas, a programação acabou se subdividindo em três partes: controle de tela, elementos do jogo e controle de eventos. A seguir, é apresentada de forma mais detalhada como funciona cada um desses elementos.

3.2.1.1 Controle de tela

As classes designadas para essa tarefa são classes que vão tratar diretamente com o usuário, manipular elementos de tela, realizar *swaps* entre telas, decidir qual tela aparecerá em qual momento. Basicamente toda a interação dos objetos com o usuário será através das classes que foram designadas para essa tarefa. Em alguns casos pode realizar também trabalhos realizados pelo núcleo do programa como, por exemplo, carregamento de arquivos.

Por esse motivo, durante o escopo de definição proposto na seção 3.1.1.2, optou-se por deixar essas classes fora da documentação a ser gerada pelo *ASDoc*, já que a reusabilidade delas é baixíssima, devido ao alto nível de especificação das tarefas designadas.

As classes designadas recebem o nome das telas a qual elas controlam, exceto a classe *Main*, como é explicado mais a frente, que foi designada especificamente para realizar o *swap* de telas. As classes contidas nesse conjunto de tarefas são:

BeginScreen.as

Encarregada pelo controle da tela inicial, é a primeira tela do jogo. Ela introduz ao nosso jogador o personagem principal da história, João, além de ajudar na contextualização do aluno diante do tema do jogo, a corrida eleitoral. Para isso, é usado como meio lúdico uma urna eletrônica, cuja foto do nosso personagem está estampada como candidato a ser votado. Os botões de controle de jogo foram posicionados sobre os botões de ação da urna original, ou seja, ao invés de termos os botões: "branco", "corrige" e "confirma", temos os botões: "novo jogo", "continuar" e "ajuda".

É importante ressaltar que os botões foram posicionados nessa ordem para seguir uma ordem lógica de desenvolvimento, além de obedecer os padrões já inseridos nos jogos desenvolvidos. A ordem lógica geralmente é novo jogo, continuar e outras opções. Como a leitura é, em geral, da esquerda para direita, a ordem proposta para os botões é exatamente essa.

Figura 15: Tela inicial.

Ainda é tarefa dessa classe carregar o arquivo XML das questões propostas e o *shared object* de *save* do jogador, tratar possíveis erros no carregamento desses arquivos e cuidar de subtelas, como a tela de ajuda.

Game.as

Encarregada de posicionar e controlar todos os elementos do jogo, é nessa classe que são instanciados objetos das classes mais genéricas que são explicados em texto que segue. Essa classe também

é encarregada por processar as respostas visuais dadas ao jogador, atualizando a tela a cada *loop* de jogo e controla, ainda, os efeitos de cada item no jogador, escutando os eventos de item disparados ao longo da rodada.

Ao fim da rodada faz a verificação da posição do jogador e toma a decisão de terminar o jogo ou chamar a subtela de *upgrade*, como previsto na documentação de *game design*. Nela também é chamada a subtela de questões. Essa tela aparece quando o item "debate" é pego.

A parte de controle dos elementos do jogo é explicado em texto que segue, na seção destinada a este tópico, tornando desnecessário tratar de como o personagem, o fundo e os outros elementos do jogo se comportam, neste tópico. A classe *Game* faz apenas o controle de objetos dessas classes de controle de elementos do jogo, além de tratar alguns eventos relacionados a pontuação, velocidade e dinheiro do personagem. Na Figura 16 é possível ver a tela de *game* e suas subtelas.

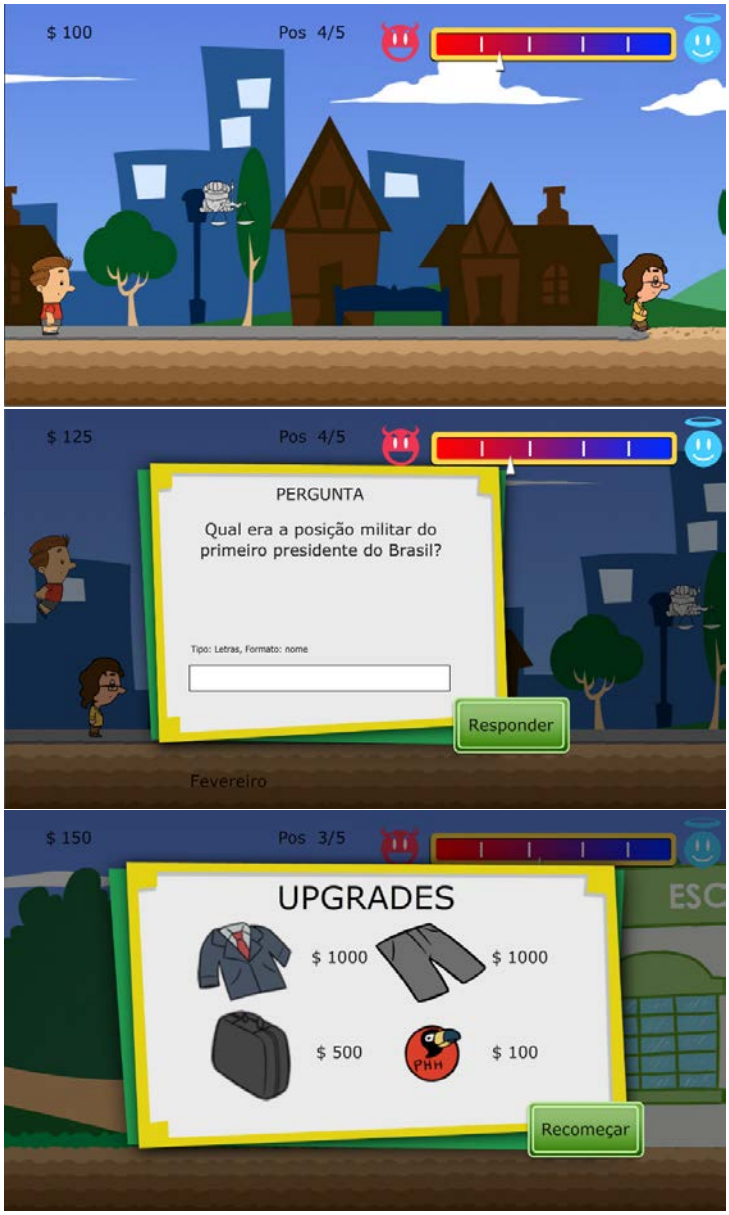

Figura 16: Tela do *game*, tela de questão e tela de *upgrade*.

Context.as e EndGame.as

Telas com o intuito de contextualizar o jogador diante de uma história.

No caso da classe *Context*, é exibido uma animação sempre que o jogador optar por iniciar um novo jogo. Apesar de não possuir texto, os quadrinhos são explicados durante a ajuda, tornando o texto desnecessário e redundante. A tela de contexto pode ser vista na Figura 17.

Figura 17: Tela de contexto.

Já a classe *EndGame* aparecerá quando o jogador terminar a corrida na primeira posição. Ela apresenta o fim a que levaram as decisões do jogador, o que lhe dá o livre-arbítrio ao jogador de escolher o final, sabendo que, caso ele termine com uma baixa honestidade, terá um final ruim, já com uma boa taxa de honestidade, receberá o final bom. Essas duas telas podem ser vistas na Figura 18.

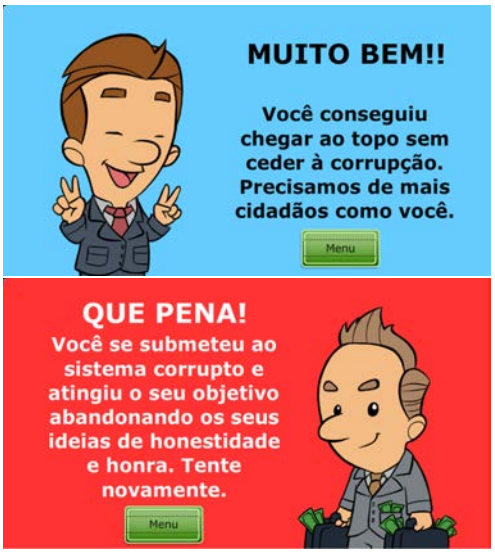

Figura 18: Tela dos finais bom e ruim.

Subtelas

Essas são telas que aparecem ao longo do jogo, sem ser uma tela propriamente dita, mas um *pop-up* que surge sobre a tela corrente. Essas subtelas tem o intuito de trazer informações adicionais, como no caso da ajuda, ou completar elementos do jogo, como no caso das telas de *upgrade* e de questão. Todas esses *pop-ups* utilizam o mesmo fundo e o mesmo botão, apesar de que, no caso do botão, o texto dele será alterado.

Figura 19: Fundo e botão dos *pop-ups*.

Como é possível ver na Figura 19, o fundo faz uma referência a pátria. As tiras amarela e verde são um meio lúdico de referenciar a bandeira, enquanto o fundo branco estimula o aluno a ter uma sensação de limpeza que deve surgir na pátria. Enquanto isso, o botão remete a uma cédula monetária, referenciando não só a corrupção, mas todo o dinheiro envolvido no meio eleitoral, o que inclui a corrida eleitoral.

Outro ponto relevante desses *pop-ups* são os de questão. Eles contém campos de texto dinâmico, que, ao ter um XML analisado, coloca cada elemento na *textbox* designada para ele.

```xml
<?xml version="1.0" encoding="utf-8"?>
<questoes>

    <questao>
        <pergunta>A pergunta em si</pergunta>
        <tiporesposta>Tipo: Número/Nome/Etc, Modelo: Forma que deve ser dada a resposta</tiporesposta>
        <resposta>Resposta da pergunta</resposta>
        <referencia>link de referência para o aluno clicar e ir direto a página do tema.</referencia>
        <tema>Tema da questão, será clicável com o link escondido.</tema>
    </questao>

</questoes>
```

Figura 20: Estrutura do documento XML.

A estrutura de XML apresentada na Figura 20 foi escolhida pois, além de permitir fácil edição do banco de questões pelos professores, ela é bastante simples, permitindo facilmente a qualquer um realizar sua manutenção. A forma deve seguir o modelo apresentado anterior, ou seja, um nó principal que contem todas as questões, chamado de "questoes". Dentro desse nó maior, existem nós "questao" que representa, cada um, os elementos da questão. Internamente a "questao" existem os seguintes elementos: "pergunta", "tiporesposta", "resposta", "referencia", "tema".

Os nós "pergunta" e "resposta" dispensam definições, pois um guarda a pergunta que será feita na tela e o outro armazena a resposta. Já "tiporesposta" salva qual o tipo de resposta e como ela deve ser dada, por exemplo, caso a resposta seja uma data, esse campo pode conter: "data, dd/mm/aaaa". É importante lembrar que, no caso de respostas que contenham letras do alfabeto, não importa a forma que o aluno vai escrever, pois a verificação da cadeia de caracteres é feita com ela toda em maiúsculo.

Já os campos "referencia" e "tema" são utilizados nos *feedbacks*, que pode ser visto na Figura 21, em que um possui um *link* de referência para a matéria abordada pela questão e o outro possui o tema da questão, o qual será o objeto clicável. Para fazer isso, usou-se uma *tag* HTML que ficou da seguinte forma: *<u>tema</u>*.

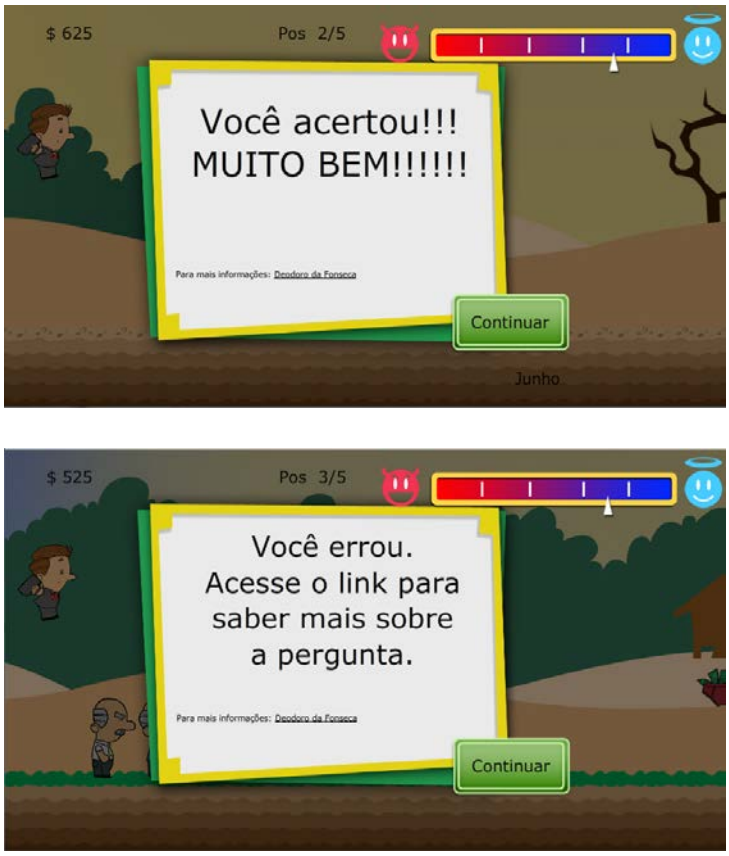

Figura 21: Telas de *feedback* positivo e negativo.

Ainda relacionado à parte de questões, apesar da facilidade de se inserir as questões manualmente no arquivo XML de perguntas, ao longo do projeto, notou-se a necessidade da criação de um meio para facilitar a inserção, edição e remoção de perguntas do arquivo. Para isso, foi desenvolvido um aplicativo utilizando a tecnologia *AIR* da própria *Adobe*, para tornar a edição do banco de questões ainda mais simples para os professores. A interface desse aplicativo pode ser visto na Figura 22.

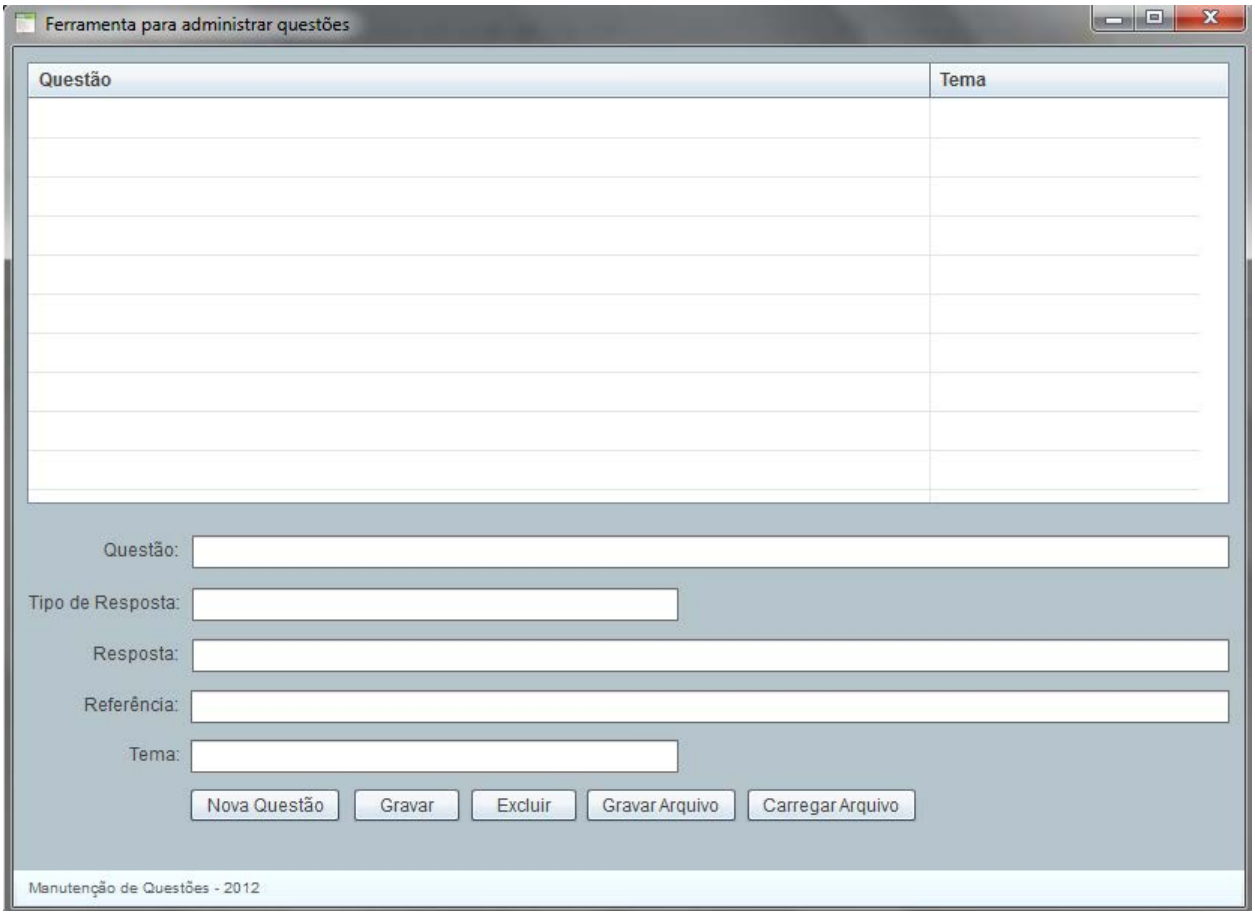

Figura 22: Aplicativo para edição do banco de questões.

Já a subtela de ajuda fica encarregada de introduzir o enredo do jogo ao aluno, além de explicar o que significa cada meio lúdico usado. Isso tem o efeito de situar o estudante em qual contexto ele está inserido para, a partir daí, começar a desenvolver os seus próprios conceitos e possibilitar a revisão de conceitos já explanados na sala de aula.

3.2.1.2 Elementos do jogo

Várias classes ficaram responsáveis por cuidar dos elementos que aparecerem na tela de *game*, ou seja, foram feitas classes específicas para cuidar dos itens que aparecem na tela, dos personagens, tanto o herói, como os adversários, do "honestômetro" proposto no documento de *game design* e do fundo. Com o intuito de permitir o reuso, proposto pelos paradigmas de orientação a objetos e Engenharia de Software, essas classes foram concebidas da forma mais genérica possível e documentadas adequadamente, com o auxílio do *ASDoc*, ferramenta proposta na seção 3.1.1.3.

A seguir serão tratadas de forma mais profunda como essas classes trabalham e como elas respeitam os paradigmas educacionais propostos por Passerino (1998). Serão abordadas as suas principais características, uma breve descrição de como é o processo de atualização de cada uma, já que são elementos que aparecerão na tela dinamicamente, além dos principais métodos e propriedades.

BackGround.as

Responsável por atualizar e controlar o fundo, essa classe se divide, basicamente, em duas fases: construção e atualização.

Na fase de construção insere-se fundos até que não haja mais espaço em branco. O procedimento para isso pode ser visto no Algoritmo 3.1.

Algoritmo 3.1 Construção do fundo

```
1  i = 0;
2  vetorFundo.insere_final(fundos[i]);
3  i = i + 1;
4  while(soma_largura(vetorFundo) < LARGURA_DA_TELA){
5          vetorFundo.insere_final(fundos[i]);
6          if(i < NUMERO_DE_FUNDOS)
7                  i = i + 1;
8          else
9                  i = 0;
10 }
```

Basicamente o que faz o algoritmo é inserir um fundo inicial e enquanto a tela não for completamente preenchida por fundos, ele insere um novo *background*. A cada novo fundo, incrementa-se a variável de qual fundo será usado. Caso o último fundo seja atingido, então reinicia-se esse contador.

Já a fase de atualização ficou responsável por três tarefas. A primeira, movimentar o fundo da direita para esquerda, dando a sensação que o herói está percorrendo a tela. A segunda foi o controle para dar o efeito de "animação cilíndrica", ou seja, não deixar que o jogador visse os fundos se completando. Isso permite a reutilização de fundos e uma padronização na atualização dele. A terceira tarefa é eliminar os fundos que já saíram da tela do vetor que guarda as imagens de *background.* O Algoritmo 3.2 representa, de forma simplificada, como funciona a atualização do fundo.

Algoritmo 3.2 Atualização do fundo

```
1  if(vetorFundo[0].x + soma_largura(vetorFundo) - passo <
       LARGURA_DA_TELA){
2              vetorFundo.insere_final(fundos[i]);
3              if(i < NUMERO_DE_FUNDOS)
4                      i = i + 1;
5              else
6                      i = 0;
7  }
8  for(j = 0; j < vetorFundo.length; j++)
9              vetorFundo[j].x = vetorFundo[j].x -
                   velocidadeDoPersonagem;
10 if(vetorFundo[0].x + vetorFundo[0].largura < 0)
11             vetorFundo.remove(vetorFundo[0])
```

Essa classe pode ser implementada em qualquer jogo que utilize a técnica de *infinite scroll* para jogos que movimentem a tela horizontalmente de forma repetitiva.

Nessa seção também é importante abordar os aspectos de *design* do fundo. Ele foi concebido de tal forma para que o jogador tenha a sensação de estar "percorrendo" o Brasil inteiro em busca de votos. Para isso o cenário foi dividido em cinco partes, cada uma representando uma região brasileira. Os fundos podem ser vistos na Figura 23.

Figura 23: Fundos: Norte, Nordeste, Sudeste, Sul e Centro-oeste.

Character.as e Enemy.as

Essas classes ficaram responsáveis pela inserção e movimentação dos personagens na tela. No caso da classe *Character*, ela controla a atualização vertical do herói e suas principais propriedades como, por exemplo, velocidade horizontal. Também na fase de construção do herói, ela verifica quais são os elementos que foram comprados e os faz "aparecer" no personagem, que vai sendo personalizado conforme os itens que você compra, apresentando ao jogador as mudanças visualmente, mostrando como o herói está criando seu caminho, além de ser um *feedback* direto da tela de *upgrade* para o jogador.

Já a classe *Enemy* ficou responsável pela locomoção dos inimigos. Em essência, ela é bastante parecida com a atualização da classe *Character*, exceto pelo fato de locomover o personagem no eixo x, o que não acontece com o personagem principal. Além disso, o método *update* retorna o quanto aquele objeto já andou, tornando possível a comparação do herói em relação aos seus adversários.

Ambas as classes instanciam seus elementos dinamicamente, a partir de um *linkage* realizado através das interfaces declaradas na seção 3.1.1.3. Isso, principalmente para o escopo da *Enemy* tornou mais genérico o código, tornando possível sua reutilização para todos os inimigos, passando-lhe como parâmetro apenas o índice do inimigo que aquele objeto se referia.

PointerController.as

Essa classe ficou responsável pelo "honestômetro". É uma classe que serve para qualquer tipo de *slider*, basta passar ou indicar onde o ponteiro deve iniciar e ele realiza uma operação percentual para posicioná-lo em relação à barra.

Ele possui duas operações básicas: movimentar para a direita e movimentar para esquerda. Além disso, possui duas propriedades constantes relativos as quantidades máxima e mínima do *slider*.

Ambos os métodos são análogos. Cada método verifica se atingiu o limite do *slider* (inferior no caso de movimentação para a esquerda e superior no caso de movimentação para a direita), caso não tenha atingido, move na direção proposta pela método. Apesar de sua simplicidade, essa classe pode controlar qualquer tipo de *slider*, com qualquer tipo de limite, já que ele movimenta percentualmente o ponteiro.

Já no caso do *slider* utilizado pelo jogo, ele respeita as considerações primárias de *feedback* propostas por Passerino (1998), em que ele deve ser imediato a ação do jogador. O honestômetro faz uso de meios lúdicos para representar a corrupção como algo ruim, quanto mais corrupto mais próximo do "diabinho", enquanto quanto mais próximo do anjo, menos o jogador se corrompeu.

Figura 24: Honestômetro.

Item.as

Essa classe é pai de todas as classes de itens (Baby.as, Doc.as, Money.as, Themis.as e TV.as) e é responsável por controlar a geração, movimentação e remoção dos itens na tela de jogo, no entanto não tem relação com os itens de *upgrade*.

O método de remoção dessa classe deve ser sobrecarregado por todas suas filhas, para que cada uma tenha um efeito ao ser obtido pelo jogador. Para isso foram usados os conceitos de herança e polimorfismo, propostos pelo paradigma de orientação a objetos.

Cada item tem um efeito e uma explicação, os quais podem ser encontrados no documento de *game design* proposto pelo projeto. Na Figura estão a representação de todos os itens.

Figura 25: Itens do jogo.

Esses são os principais meios lúdicos relacionados a parte sociológica do jogo, tendo em vista que cada um representa uma metáfora relativo a como se comporta a política no Brasil e como cada um pode ter características boas e ruins para o jogador. Eles também são o principal meio de escolha que o aluno tem, já que ele pode escolher pegar ou não pegar o item, arcando com as consequências de suas decisões.

3.2.1.3 Controle de eventos

Foram as classes responsáveis por controlar os eventos, tantos os seus efeitos, armazenando o que ocorria, como o conjunto de constantes de eventos que são tratados pelo próprio jogo. Foram utilizadas duas classes: *Global.as* e *Events.as*.

A classe *Global* ficou encarregada de salvar todos os dados e transmiti-los através de todas as classes e para isso guarda esses valores em suas propriedades e, quando necessário, são acessadas, transferindo as informações requeridas.

Já a classe *Events* é filha da *Event* criada pela própria *Adobe* para o tratamento de eventos. Ela armazena todas as constantes referentes aos eventos do jogo e é utilizada para dispará-los para que todos os escutadores sejam capazes de tratar os dados, conforme a necessidade daquele evento.

3.2.2 Testes

Como supracitado, os testes realizados foram: unidade, integração, estresse e desempenho. Esses testes se dividiram em duas partes: durante o desenvolvimento e pós-programação. Os testes de unidade e integração foram realizados durante o processo de criação do jogo, já que eles buscam

sanar problemas que estão relacionados diretamente com o momento de produção, verificando falhas diretamente no código, ou seja, são testes de caixa-branca.

Já os testes de estresse e desempenho foram realizados depois da produção do *software* por completo e nele o código não foi levado em conta, procurando erros sem "conhecer" o código, ou seja, testes de caixa-preta. A cada erro que ocorria nesses testes, voltava-se para a fase de produção, o *software* era corrigido e todos os testes eram aplicados novamente, tanto os de caixa-branca, quanto os de caixa-preta.

A quantidade de testes tentou respeitar os gráficos de exaustão de testes, propostos por Bastos et al. (2007), que podem ser vistos na Figura 26. Neles é possível o ponto ótimo para parar testes, relacionando o custo-benefício de se continuar testando um *software*. O projeto tentou respeitar ao máximo esse ponto, aliando-o ao prazo reduzido, mesmo que a proposta do modelo *RAD* seja produzir de forma rápida, muitas vezes relegando práticas adequadas de teste.

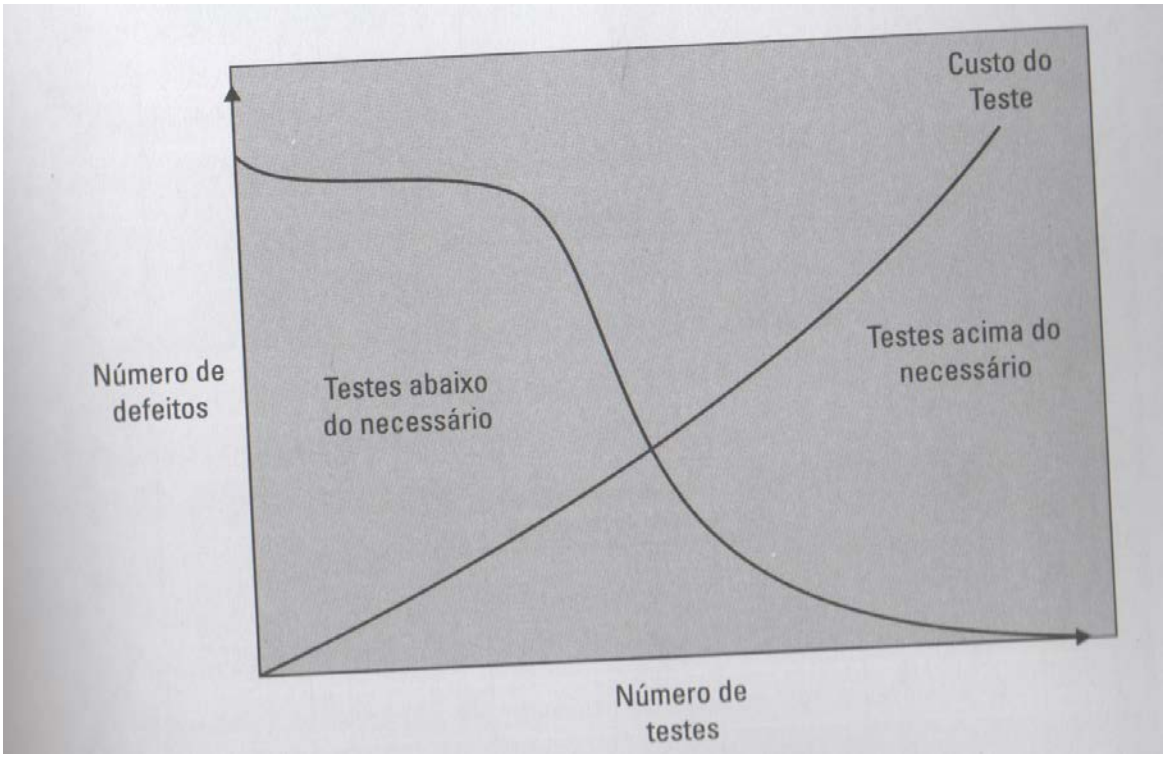

Figura 26: Gráficos de exaustão de testes.
Fonte: Bastos et al., 2007, p. 25.

Ainda é importante especificar que, para testes de estresse, foram elevadas a velocidade do herói, para verificar em que ponto o excesso de velocidade se tornaria desagradável para o jogador, gerando algum tipo de *slowdown* no sistema pela velocidade com que se moveria o *background*. Ainda relacionado com esse tipo de teste, verificou-se o quão grande poderia ser o banco de questões, sem que este prejudicasse o desenvolvimento da atividade e o processamento do jogo.

Já para os testes de desempenho, foram utilizados os computadores que ainda são bastante usados em escolas públicas, ainda que o projeto para implementação desses computadores nas escolas públicas tenha sido descontinuado. O jogo foi testado nos computadores portáteis "Magalhães", cuja especificação segundo o distribuidor (PORTÁTIL..., 2012) é:

- Processador: Intel Celeron M 900 MHz;

- Memória: 1GB;
- Disco rígido: 30 GB particionados – 1/3 para Windows; 1/3 para Linux CM; 1/3 para dados do usuário;
- Tela de 9 polegadas, resolução: 1024x600 pixeis;
- Wi Fi: 802.11b/g;
- Porta RJ-45;
- *Webcam*;
- Caixas de som;
- Microfone embutido;
- Duas portas USB 1.0;
- Entrada para cartões SD;
- Bateria de 3 células com duração média de 3 horas;

Figura 27: Computador portátil Magalhães
Fonte: Imagem extraída do site do Portátil Magalhães[4]

Mesmo com as restrições de um *hardware* um pouco mais simples, o jogo foi aprovado em todos os testes aos quais foi submetido, não sendo encontrado novos erros até a sua homologação e documentação.

3.2.3 Documentação

A documentação dessa fase foi feita inteiramente via código, respeitando o formato proposto pelo *ASDoc*. Ainda é importante ressaltar que toda a codificação foi feita em inglês, permitindo a manutenção por qualquer pessoa que entenda inglês ou português, já que a documentação das classes genéricas foram feitas em português brasileiro.

[4]Endereço eletrônico da imagem: http://mediablog.portatilmagalhaes.com/pmag3.png

Mesmo que todos os comentários no formato proposto pelo gerador de documentação tenham sido feitos ao longo do desenvolvimento do projeto, nenhuma documentação foi gerada nessa fase, sendo deixada para a fase de pós-projeto a criação de toda a documentação relativo a manutenção do trabalho desenvolvido.

3.3 Pós-projeto

Ao final do projeto, viu-se a necessidade de gerar mais documentos relativos ao projeto. Portanto, essa fase foi composta de apenas uma tarefa: a documentação de todo o trabalho desenvolvido ao longo do projeto.

Para isso, utilizou-se a ferramenta da *Adobe*, *ASDoc*, gerando a documentação no formato HTML, com uma página *index* como menu principal que dá acesso a todas as classes documentadas. Fazem parte desse documento:

- Auxiliares:
  - Item.as
  - BackGround.as
  - Enemy.as
  - Character.as
  - PointerController.as
  - Events.as
  - Global.as
- Auxiliares.Items:
  - Baby.as
  - Doc.as
  - Money.as
  - Themis.as
  - TV.as

Foi gerado, ainda, um diagrama de classes, para auxiliar na manutenção e relação dos elementos entre si. Esse diagrama pode ser visto na Figura 28.

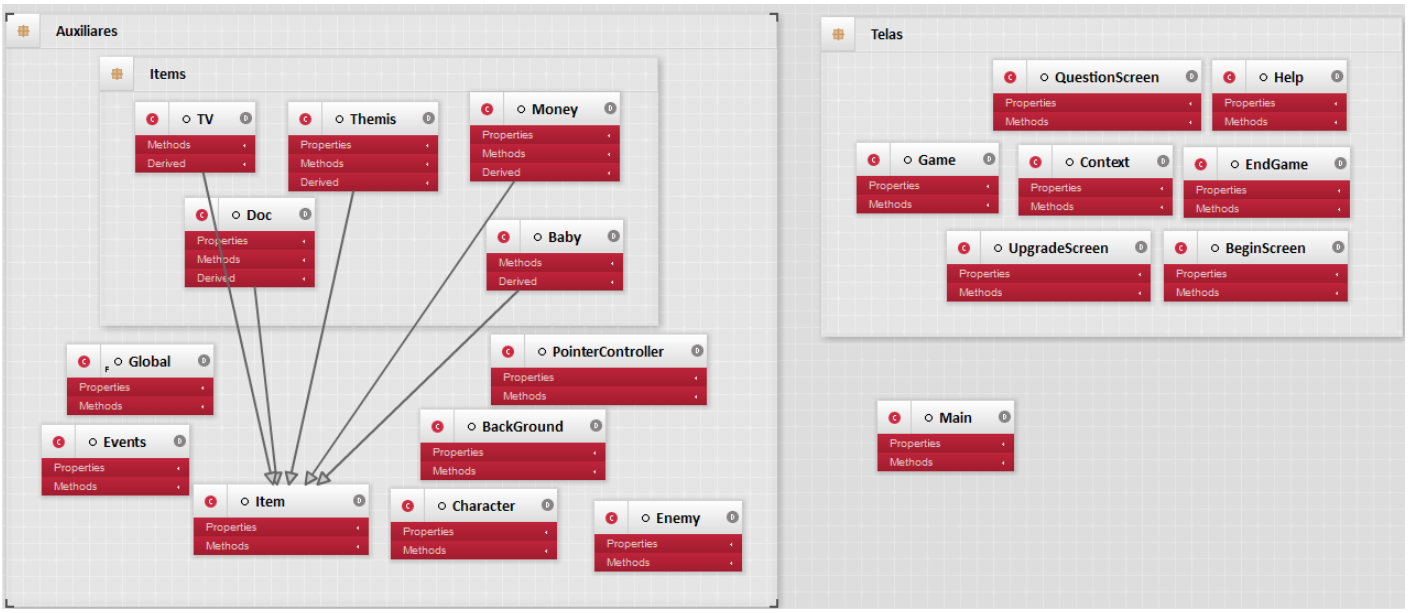

Figura 28: Diagrama de Classes.

Capítulo 4

Conclusão

4.1 Problemas e soluções

Devido aos poucos recursos disponíveis para sua produção, foram encontrados problemas que obrigaram a cortar algumas práticas adequadas relativas à Engenharia de Software, além de reduzir algumas características do programa a serem implementadas em trabalhos futuros.

Exemplos desses problemas são a impossibilidade de fazer a documentação de testes e do *software* auxiliar para alteração do banco de questões, além do fato de não ser possível contratar um *designer* de som para desenvolver os arquivos de mídia necessários sob os aspectos pedagógicos de áudio. Para solucionar esses problemas, propõem-se que o projeto continue, assim tornando possível a sua implementação por completo futuramente.

Outro problema encontrado foram as limitações do uso da ferramenta *ActionScript 3.0*. Muitas implementações ficaram demasiadamente lentas, quando inseridas com conceitos de segurança. Para resolver esse problema, tornou-se necessário desistir de algumas verificações de segurança, além da otimização do código, para que, assim, o produto final parasse de apresentar tais problemas.

4.2 Trabalhos futuros

Seguindo o caminho natural, as próximas etapas desse projeto seriam a implementação adequada de artigos sonoros e a introdução do sistema para testes e avaliação na escola. Para tanto, faz-se necessário a escolha de uma escola, reunião com docentes que sejam favoráveis a essa implementação, além de um estudo de como realizá-lo de forma adequada e um treinamento relativo ao uso da ferramenta, em sua totalidade, para os docentes que irão aplicá-la em sala.

Feito isso, é possível relacionar os dados da implementação com as informações obtidas na sala de aula, permitindo avaliar a evolução dos estudantes em determinado aspecto, como a introdução do jogo melhorou a revisão e potencializou a absorção de conteúdo, verificando se os resultados obtidos foram os desejados pelos pesquisadores e professores.

4.3 Considerações finais

O intuito do desenvolvimento do jogo foi atingido, tornando possível ao professor associar metáforas a conceitos avançados passados em aula. Aliou-se a isso, o poder de revisão que as questões

permitem ao professor, podendo abordar um ponto específico da matéria ou uma questão geral, permitindo infinitos usos para a ferramenta. Além disso, a interatividade foi alcançada a partir dos dispositivos lúdicos propostos pelo jogo, possibilitando ao jogador desenvolver a criatividade, realizando suas próprias escolhas.

No entanto, nota-se a necessidade de dar continuidade ao projeto, para que ele atinja a sua proposta total de avaliar a evolução dos alunos em relação ao material proposto a ser trabalhado. Além disso, é necessário completar componentes não produzidos como, por exemplo, documentação ou elementos sonoros. Ou seja, obteve-se um produto final, mas é necessário verificar como ele funcionará na prática.

Bibliografia

AMORY, A. et al. The use of computer games as and educational tool: identification of appropriate game types and game elements. *British Journal of Educational Technology*, v. 30, p. 311–321, 1999.

ASDOC tool. In: ADOBE: Help Resource Center. Adobe, 2012. Disponível em: <http://livedocs.adobe.com/flex/3/html/help.html?content=asdoc_1.html>. Acesso em: 07 out. 2012.

BASTOS, A. et al. *Base de conhecimento em testes de software*. [S.l.]: Martins, 2007.

BATTAIOLA, A. L. Jogos por computador: Histórico, relevância tecnológica, tendências e técnicas de implementação. In: *Anais da SBC*. [S.l.: s.n.], 2000. v. 2.

BLOW, J. Game development: Harder than you think. *Queue*, v. 1, p. 28–37, 2004.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: guia do usuário*. [S.l.]: Elsevier, 2006.

BRAUNSTEIN, R. *ActionScript 3.0 Bible*. [S.l.]: Wiley Publishing, 2010.

CROCUS Modeller. In: CROCUS Modeller. Crocus Modeller, 2012. Disponível em: <http://crocusmodeller.com/>. Acesso em: 07 out. 2012.

DERDENGER, T. Technological tying and the intensity of competition: An empirical analysis of the video game industry. Revisado e re-enviado para análise no periódico "Quantitative Marketing and Economics Quantitative Marketing and Economics". 2011.

FLASHDEVELOP. In: FLASHDEVELOP. FlashDevelop, 2012. Disponível em: <http://www.flashdevelop.org/>. Acesso em: 19 nov. 2012.

KENT, S. L. *The Ultimate History of Video Games: From Pong to Pokémon and Beyond : the Story Behind the Craze that Touched Our Lives and Changed the World*. [S.l.]: Prima, 2001.

KROEFF, V. N. Novas tecnologias, novos sujeitos aprendentes: o desafio pedagógico em tempos de múltiplos aprendizados. *Innovación Educativa*, v. 19, p. 15–23, 2009.

MORATORI, P. B. *Por que utilizar jogos educativos no processo de ensino aprendizagem?* Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, 2003.

PASSERINO, L. M. Avaliação de jogos educativos computadorizados. In: *Taller Internacional de Software Educativo - TISE' 98. Anais.* [S.l.: s.n.], 1998.

PORTÁTIL Magalhães. In: PORTÁTIL Magalhães. Portátil Magalhães, 2012. Disponível em: <http://www.portatilmagalhaes.com/>. Acesso em: 16 out. 2012.

PRESSMAN, R. *Engenharia de software*. [S.l.]: MCGRAW HILL - ARTMED, 2006.

RANDEL, J. M. et al. The effectiviness of games for educational purposes: A review of recent research. *Simulation and Gaming*, v. 23, p. 261–276, 1992.

REIS, G. dos. *Videogame: história, gêneros e diálogo com o cinema.* Dissertação (Mestrado) — Universidade de Marília, 2005.

REZENDE, D. A. *Engenharia de Software e Sistemas de Informação*. [S.l.]: Brasport, 2005. ISBN 9788574522159.

ROSENZWEIG, G. *Actionscript 3.0 game programming university*. [S.l.]: Que Corp., 2007.

SALEN, K.; ZIMMERMAN, E. *Rules of Play: Game Design Fundamentals*. [S.l.]: MIT Press, 2003.

SANDERS, W. B.; CUMARANATUNGE, C. *Actionscript 3.0 design patterns*. [S.l.]: O'Reilly, 2007.

SCHUYTEMA, P. *Design de games: uma abordagem prática*. [S.l.]: Cengage Learning, 2008.

STARUML. In: STARUML. StarUML, 2012. Disponível em: <http://crocusmodeller.com/>. Acesso em: 07 out. 2012.

TAROUCO, L. M. R. et al. Jogos educacionais. *Renote*, v. 2, 2004.

Anexos

Anexo A - Documento de *Game Design*

Corrida Eleitoral

Plataforma

Computadores pessoais que tenham *Flash* versão 10 ou superior.

Público-Alvo

Crianças a partir de 14 anos.

Objetivo

Do jogador

Ganhar a maratona, melhorando seu personagem, pegando itens na caminhada e respondendo corretamente às questões.

Do jogo

Ensinar e sedimentar conhecimentos sobre história da república no Brasil e introduzir e/ou sedimentar uma opinião sócio-crítica no jogador.

Enredo

João, um adulto de 37 anos, é um professor de história e está cansado de ver toda noite a corrupção que assola os mais diversos níveis de poderes no Brasil. Ele resolve que vai tentar mudar o sistema a partir de dentro.

Ele, então, torna-se elegível e participará da corrida eleitoral. Repleta de armadilhas, adversários inescrupulosos e patrocinadores corruptos, João tenta seguir a sua caminhada até Brasília de forma honesta, sem sucumbir à corrupção do sistema. Será que ele conseguirá atingir seu objetivo sem se corromper?

Personagem

João tem 37 anos e é um típico cidadão brasileiro. Ele leciona em escolas particulares e públicas para ganhar a vida.

Cansado da corrupção que vê todo dia no jornal noturno, aliado a sua formação acadêmica em história e especializações em sociologia, João tenta mudar o mundo ensinando seus pupilos.

Mesmo não vendo mudança, João acredita na transformação e melhora do nosso país, graças ao amor que ele tem por essas terras.

Aspectos técnicos

Resumo

O jogo será uma corrida entre os candidatos e na tela aparecerão itens aleatoriamente que poderão melhorar ou piorar a condição do jogador. O jogador poderá realizar *upgrades* no seu personagem ao final de cada corrida com o dinheiro conquistado durante a partida.

Desenvolvimento

Será desenvolvido usando *ActionScript 3.0* e *Adobe Flash Professional* CS 5.5.

Cenário

No piso passarão os meses, representando a corrida eleitoral em si. A chegada será identificada por um colégio eleitoral, que terá a "votação" em seu topo. Ao fundo existirá uma imagem que se repetirá ao longo da corrida. Essa imagem deverá ser composta por cinco imagens, sendo que pode ou não existir uma transição entre elas.

As cinco imagens deverão representar as cinco regiões brasileiras: Sul, Sudeste, Centro-Oeste, Nordeste e Norte. Elas estarão "conectadas" e ficarão em *loop* no *background*. Para suavizar a transição entre o fundo, poderão ser feitas imagens transitivas, desde que haja condições técnicas para o designer desenvolver essa característica.

Competidores

Serão os adversários do personagem. Terão as características de desenvolvimento do personagem, sendo que o melhor competidor será semelhante ao personagem do jogador em seu estado final e o pior competidor será equivalente ao estado inicial. Os outros competidores estarão em estados intermediários.

Haverá um total de sete competidores e oito jogadores por partida (os sete competidores e o jogador).

Os competidores não serão afetados pelos itens e nem os tirarão da tela, eles apenas correrão até a linha de chegada.

Atributos do personagem jogável

O personagem jogável será composto de três atributos:

- Honestidade: Atributo mais importante, representa o nível de honestidade do jogador. Será representado visualmente por um "termômetro" de honestidade (honestômetro) que deverá ficar no canto superior direito da tela e que degrada de vermelho a branco (0%~50%) e branco a azul (50%~100%). O nível de honestidade será o responsável pelo final que aparecerá quando o jogador terminar a corrida na primeira posição. O honestômetro se iniciará em 50%.

- Dinheiro: Será representado no canto superior esquerdo da tela, representa o valor monetário que o jogador tem em caixa. Será responsável pela compra dos *upgrades* do jogador ao final de cada partida.

- Popularidade: Sem representação visual, mostra a velocidade do jogador na corrida eleitoral. Quanto maior a popularidade, mais rápido ele correrá. Poderá ser aumentada permanentemente, com os *upgrades*, ou temporariamente, com itens que aparecem na tela.

- Posição: Representará a posição do jogador na corrida eleitoral. Será apresentado abaixo da representação do dinheiro.

Itens

Serão responsáveis pela melhora ou piora do *status* do jogador. Os itens de *upgrade* farão "parte" da composição gráfica do jogador conforme forem comprados. Os itens aparecerão aleatoriamente na parte direita da tela e sumirão na parte esquerda ou ao atingir o personagem.

- Itens da corrida:

  - Themis: Deusa da justiça, ela representá o respeito do jogador pelas leis brasileiras. Ela aumentará permanentemente o honestômetro. Graficamente será a cabeça da deusa.
  - Dinheiro na cueca: Representação da corrupção no Brasil, este item dará ao jogador dinheiro, mas, em troca, reduzirá a honestidade permanentemente. Graficamente será uma cueca com notas de dinheiro saindo dela.
  - Bebê: Em campanhas eleitorais é comum atitudes populistas como beijar bebês. Sendo assim, esse item aumentará temporariamente a popularidade do jogador. Graficamente será um bebê.
  - Burocracia: Principal entrave para se implementar novas medidas no Brasil, esse item reduzirá temporariamente a popularidade do jogador. Graficamente será um homem velho, com uma pasta que terá escrito "Burocracia". Obs: Este será o único item não "flutuante" da tela.
  - Debate: Item especial que poderá ter efeito positivo ou negativo em TODAS as características do jogador, será temporário para popularidade e permanente para os outros dois atributos. Graficamente será uma tevê com debate escrito nela. Esse item abrirá uma tela com uma pergunta sobre história republicana do Brasil, terá um campo de *edit* para o jogador escrever a resposta. Caso a resposta esteja correta, aumentará as características do jogador, caso contrário, diminuirá.

- Itens de *upgrade*:

–Maleta: Aumentará os pontos de honestidade recebido pelo jogador, pois está preparado para provar sua honestidade, se defender de ataques injustos e passar pelo julgamento da "Ficha limpa".

–Paletó: Aumentará a popularidade permanentemente. Um político bem vestido causa uma melhor impressão.

–Calça Social: Aumentará a popularidade permanentemente. Mesma explicação do paletó.

–Broche do partido: Dará a habilidade do jogador conseguir pular no ar (Pulo duplo). Essa habilidade é decorrente do alcance político que aumenta ao se estar em um partido.

Telas

O jogo será composto de quatro telas principais e três telas dinâmicas. As telas dinâmicas aparecerão sobre a tela atual.

•Telas principais:

–Inicial: Será a primeira tela. Com o visor e teclado de uma urna eleitoral, no visor estará o personagem do jogo e o nome do jogo "Corrida Eleitoral". Os botões de ação da urna (Confirmar, Corrigir e Branco) serão substituídos por: Começar, Instruções e Créditos.

–Abertura: Uma introdução da história de João. O jogo começará ao clicar no botão de iniciar.

–Jogo: Tela em que terá a corrida. Já foi previamente descrita no cenário.

–Final: Apresentará como terminou a história. Existirão dois finais:

∗Bom: Final no qual o honestômetro é maior ou igual a 51%.

∗Ruim: Final no qual o honestômetro está abaixo de 51%.

•Telas dinâmicas:

–Instruções: Explicará quais os botões do jogo e as propriedades, devidamente justificadas, de cada item.

–Pergunta: Pausará o jogo principal e aparecerá com uma pergunta e um campo de resposta, além de um botão de aceitar a resposta.

–*Upgrade*: Onde o jogador comprará os itens do personagem, clicando nos botões que representarão o *upgrade*.

Ações do personagem

O personagem correrá automaticamente e somente o pulo será controlado. O pulo será realizado pela tecla espaço, caso tenha pulo duplo, o segundo clique realizará o pulo duplo.

Considerações finais

Haverá um protótipo que representará a parte programacional do jogo. A este protótipo serão aplicadas posteriormente, as imagens descritas acima, com auxílio de um designer externo (Vinicius Giacomini) ao Departamento e Curso de Ciência Computação, que será contratado com os recursos fornecidos ao projeto.