centro universitário senac

Luis Henrique Menezes Mauruto  
Nilson Calazans Dias Filho  
Willians Schallemberger Schneider

IRTaktiks – Jogo de RPG Tático para Interfaces Multi-toque

São Paulo  
2008

Luis Henrique Menezes Mauruto  
Nilson Calazans Dias Filho  
Willians Schallemberger Schneider

IRTaktiks – Jogo de RPG Tático para Interfaces Multi-toque

Trabalho de conclusão de curso apresentado ao Centro Universitário Senac – Campus Santo Amaro, como exigência parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Fábio Roberto de Miranda

São Paulo  
2008

|  |
| --- |
| Mauruto, Luis Henrique Menezes Filho, Nilson Calazans Dias Schneider, Willians Schallemberger  IRTaktiks – Jogo de RPG Tático para Interfaces Multi-toque / Luis Henrique Menezes Mauruto, Nilson Calazans Dias Filho e Willians Schallemberger Schneider – São Paulo, 2008.  fls, il. color.  Orientador: Prof. Fábio Roberto de Miranda  Trabalho de Conclusão de Curso – Centro Universitário Senac – Campus Santo Amaro – São Paulo, 2008. |

Alunos: Luis Henrique Menezes Mauruto, Nilson Calazans Dias Filho e Willians Schallemberger Schneider.

Título: IRTaktiks – Jogo de RPG Tático para Interfaces Multi-toque

Trabalho de conclusão de curso apresentado ao Centro Universitário Senac – Campus Santo Amaro, como exigência parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Fábio Roberto de Miranda

A banca examinadora dos Trabalhos de Conclusão em sessão pública realizada em \_\_\_/\_\_\_/\_\_\_\_\_ considerou os candidatos:

( ) Aprovados ( ) Reprovados

Examinador

Examinador

Presidente

Agradecimentos

Agradecemos aos nossos familiares e companheiras, pelos quatro anos de apoio, compreensão e incentivo nesta etapa de nossas vidas.

A todos os professores que através do empenho em transmitir seus conhecimentos, contribuíram com nossa formação acadêmica e profissional.

Ao nosso orientador: Prof. Fábio Roberto de Miranda, pela ajuda, críticas e sugestões; de grande ajuda no desenvolvimento deste trabalho.

A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original.

Albert Einstein

Resumo

Com o constante avanço da tecnologia, a interação do homem com a máquina vem se tornando cada vez mais natural e intuitiva. O uso de superfícies sensíveis ao toque se populariza a cada dia.

Este trabalho propõe o uso desta tecnologia a fim de desenvolver um jogo de *RPG* tático para superfícies multi-toque em que os jogadores irão interagir simultaneamente e de maneira natural sobre a superfície de projeção do jogo. Esta superfície utiliza o princípio da reflexão total interna frustrada da luz para que com dispositivos de custo acessível seja possível identificar os múltiplos pontos tocados pelos usuários e permitir a interação multi-toque com as aplicações.

Palavras-chave: jogos eletrônicos, sistemas projetor-câmera, computação gráfica, multi-toque, interação humano-computador.

Abstract

A fazer...

Sumário

[1. IRTaktiks 15](#_Toc201055283)

[1.1. Introdução 15](#_Toc201055284)

[1.2. Interação Multi-toque 15](#_Toc201055285)

[1.2.1. História 16](#_Toc201055286)

[1.3. Objetivo 17](#_Toc201055287)

[2. Bases Teóricas e Tecnologias Empregadas 20](#_Toc201055288)

[2.1. Dispositivos Multi-toques 20](#_Toc201055289)

[2.1.1. Microsoft Surface 20](#_Toc201055290)

[2.1.2. ReacTable 21](#_Toc201055291)

[2.1.3. iPhone 22](#_Toc201055292)

[2.2. Jogos e Interatividade 23](#_Toc201055293)

[2.2.1. Jogos de Estratégia 23](#_Toc201055294)

[2.2.2. Jogos de RPG 24](#_Toc201055295)

[2.2.3. RPGs Eletrônicos 26](#_Toc201055296)

[2.3. Realidade Virtual 29](#_Toc201055297)

[2.3.1. Realidade Aumentada 29](#_Toc201055298)

[2.4. Implementações de Superfícies Multi-toque 30](#_Toc201055299)

[2.4.1. Iluminação Traseira (Rear Illumination) 30](#_Toc201055300)

[2.4.2. Reflexão Total Interna Frustrada da Luz (FTIR) 30](#_Toc201055301)

[2.5. Tecnologias Utilizadas 31](#_Toc201055302)

[2.5.1. OSC 31](#_Toc201055303)

[2.5.2. TUIO 31](#_Toc201055304)

[2.5.3. ReacTIVision 32](#_Toc201055305)

[2.5.4. Touchlib 33](#_Toc201055306)

[2.5.5. Microsoft XNA 35](#_Toc201055307)

[3. Projeto 36](#_Toc201055308)

[3.1. Adequação da Mesa 36](#_Toc201055309)

[3.1.1. Estrutura 36](#_Toc201055310)

[3.1.2. Visão Computacional 40](#_Toc201055311)

[3.1.3. Testes e Dificuldades Encontradas 41](#_Toc201055312)

[3.2. Jogo 44](#_Toc201055313)

[3.2.1. Protótipo 46](#_Toc201055314)

[3.2.2. Versão Final 47](#_Toc201055315)

[4. RESULTADOS 77](#_Toc201055317)

[4.1. Trabalhos Futuros 77](#_Toc201055319)

[4.2. Conclusão 77](#_Toc201055320)

[5. Referências bibliográficas 78](#_Toc201055321)

[6. APÊNDICES 79](#_Toc201055322)

[6.1. Fórmulas dos Atributos Calculados 79](#_Toc201055323)

[6.2. Tabela de Fatores 80](#_Toc201055324)

[6.3. Fórmulas das Ações 81](#_Toc201055325)

[6.3.1. Ataque 81](#_Toc201055326)

[6.3.2. Itens 82](#_Toc201055327)

[6.3.3. Habilidades 82](#_Toc201055328)

Lista de Figuras

Figura 1 - Lemur Input Device 17

Figura 2 - Microsoft Surface 18

Figura 3 - ReacTable 19

Figura 4 - iPhone 20

Figura 5 - Exemplo de campanha em andamento 26

Figura 6 - Final Fantasy - Square (1987) 27

Figura 7 - Final Fantasy VII - Squaresoft (1997) 27

Figura 8 - World of Warcraft - Blizzard (2004) 28

Figura 9 - Final Fantasy Tactics - Squaresoft (1999) 29

Figura 10 - Arquitetura do sistema 31

Figura 11 - ReacTIVision reconhecendo um fiducial 34

Figura 12 - Componentes elétricos utilizados 37

Figura 13 - Mesa após reestruturação da parte elétrica 37

Figura 14 - Representação das malhas do circuito elétrico da mesa 38

Figura 15 - Placa de circuito impresso com os resistores de 56Ω e 5,6Ω 39

Figura 16 - Microsoft LifeCam VX-6000 40

Figura 17 - Toque antes e depois da reestruturação 41

Figura 18 - Canhão de led usado na iluminação 41

Figura 19 - Fiduciais sobre papel vegetal e plástico 42

Figura 20 - Visão geral do módulo de comunicação 43

Figura 21 - Protótipo 44

Figura 22 - Versão final 45

Figura 23 - Arquitetura da versão final 46

Figura 24 - Visão do módulo Listener 47

Figura 25 - Exemplo de eventos do módulo Input 48

Figura 26 - Exemplo de utilização de efeitos hlsl 49

Figura 27 - Exemplo de utilização de fonte XML 50

Figura 28 - Exemplo de fonte-textura 50

Figura 29 - Exemplo de uso de fonte-textura 51

Figura 30 - Representação da área visível da cena 52

Figura 31 - Exemplo de sobreposição de sprites 52

Figura 32 - Software Vue xStream 6 53

Figura 33 - Mapa utilizando arquivo de geometria (40Mb) 54

Figura 34 - Mapa usando heightmap e efeito hlsl de mesclagem 55

Figura 35 - Exemplo de utilização de áreas 56

Figura 36 - Áreas com e sem suavização 56

Figura 37 - Fluxo de execução de uma animação 58

Figura 38 - Exemplo de efeito de partículas 59

Figura 39 - Exemplo de exibição de informações 59

Figura 40 - Estrutura organizacional do jogo 61

Figura 41 - Generalização de telas 64

Figura 42 - Menu do jogador e da unidade 66

Figura 43 - Itens e seus respectivos subitens 67

Figura 44 - Fluxo de execução de uma ação através do menu 68

Figura 45 - Unidade movendo-se dentro da área especificada 69

Figura 46 - Máquina de estados do submódulo Mover 70

Figura 47 - Mira sobre uma unidade inimiga 71

Figura 48 - Máquina de estados do submódulo Aim 72

# IRTaktiks

## Introdução

Os meios de interação entre usuários e dispositivos são objetos de constantes avanços e buscas por funcionalidades e diversificação. A cada dia surgem novas idéias e modelos de interação assim tornando os mundos virtuais criados pelas aplicações interativas cada dia mais realista e atraente. Idéias e propostas de formas de interação que surgiram através de filmes ou seriados, como o *Holodeck*, presente no seriado *Star-Trek*, ou o monitor de operações do filme *Minority Report*, hoje se encontram em vias de estarem presentes no cotidiano das pessoas; através de interfaces em celulares, restaurantes, hotéis, cassinos, exibições artísticas, na indústria, em pesquisas, em museus e em entretenimento.

Os jogos têm um papel essencial nessa evolução, tanto na parte de hardware, como por exemplo, processadores, placas de vídeo, memórias e dispositivos de interação; quanto na parte financeira, arrecadando bilhões de dólares todos os anos para a indústria do entretenimento de jogos eletrônicos.

Tendo como motivação um trabalho realizado por ex-alunos do Centro Universitário Senac, este trabalho pretende desenvolver um jogo para superfícies multi-toque, buscando aproveitar as possibilidades dessa interação; e auxiliar novas pesquisas na área de interatividade com jogos, desenvolvimento e expansão da área de jogos eletrônicos.

## Interação Multi-toque

É uma técnica de interação homem-computador com utilização de dispositivos periféricos. O multi-toque consiste no reconhecimento de múltiplos toques simultâneos em uma superfície, que pode ser uma tela ou uma mesa com projeção, por exemplo, e sua interpretação através de software. Esse reconhecimento pode ser de posição, pressão ou ângulo (conforme o dispositivo de captura), permitindo que diversos dedos, mãos ou pessoas (dependendo do tamanho do dispositivo) interajam, provendo uma forma rica e intuitiva de interação, como por exemplo, o monitor de operações apresentado no filme *Minority Report*.

Esta tecnologia se popularizou com a ajuda do *YouTube*, em 2006, com vídeos do evento *Technology Entertainment Design Conference*, em *Monterey* na *Califórnia*. Nele, o pesquisador do instituto de ciências matemáticas *Courant*[[1]](#footnote-2), Jeffenson Y. Han; demonstrou seu trabalho de pesquisa de interação multi-toque utilizando uma superfície com display gráfico interativa, permitindo a interação de múltiplos usuários; apresentando implementações elegantes de diversas técnicas e aplicações. Os protótipos de J. Han despertaram o interesse de diversas vertentes de pesquisa sobre essa nova alternativa de interação, populando a Internet com diversos tutoriais e *weblogs* que trocam experiências entre estes pesquisadores e fomentam o desenvolvimento de protótipos.

### História

O multi-toque teve seu início em 1982, com *tablets* feitos na universidade de Toronto e com telas dos laboratórios Bell. Nos anos 90 a universidade de Delaware desenvolveu um sofisticado sistema de reconhecimento de gestos e escrita, base para o *mouse-pad* *iGesture* e teclados *TouchStream*, comercializados pela *FingerWorks* em 2001. Estes teclados são reconhecidos pela sua ergonomia: aponte e arraste com um ou mais dedos (impressas sobre uma superfície macia), eliminando totalmente a necessidade de um dispositivo apontador, como o mouse.

O primeiro dispositivo multi-toque com display visual integrado comercializado foi o *Lemur Input Device*, um controlador multimídia profissional da companhia francesa *JazzMutant* lançado em 2005. Em julho de 2007, a *Apple* registrou cerca de 300 patentes com seu produto *iPhone*. Meses depois, a *Microsoft* anunciou seu futuro produto, a *Microsoft Surface*, como sua grande inovação.



Figura - Lemur Input Device

## Objetivo

O objetivo deste trabalho é desenvolver para uma mesa multi-toque um *RPG* tático semelhante ao famoso *Final Fantasy Tactics*, onde o jogador controlará vários personagens com características diferentes, com o objetivo de derrotar o inimigo através de ataques, magias e itens. A composição do cenário de batalha e a disposição dos personagens, juntamente com suas características influenciam no decorrer da batalha, tentando mesclar a um *RPG*, elementos base dos jogos de estratégia.

O jogo desenvolvido deve permitir o uso de táticas, como por exemplo, se beneficiar de uma determinada composição de personagens no campo de batalha para obter vantagens sobre o inimigo. O jogo será jogado por dois jogadores sobre uma superfície multi-toque, em que o toque dos jogadores será utilizado para definir as ações que os personagens devem executar e o posicionamento dos elementos no jogo. As interações do jogador com seus personagens se darão através de toques.

{imagem: irtaktiks}

Cada jogador controlará uma quantidade definida de personagens, sendo que cada um destes personagens possui características próprias e únicas que o diferencia dos demais, dando-lhe algumas vantagens e desvantagens. A tática do jogo fica a cargo de utilizar estas características, em conjunto com o cenário, da melhor maneira possível em beneficio próprio a fim de derrotar o adversário.

A superfície multi-toque que será utilizada no desenvolvimento do projeto foi desenvolvida por ex-alunos do Centro Universitário Senac, no ano de 2007, sendo tema de um trabalho de conclusão de curso. Trata-se de uma mesa de acrílico que se baseia no princípio da reflexão total interna frustrada da luz, desenvolvido por Jeffenson Y. Han, para permitir a detecção de toques por softwares de visão computacionais.

As ações que os usuários executarem sobre a mesa, toques, por exemplo; é reconhecida através de um software que analisa imagens enviadas por uma *webcam*. O software processa estas imagens e envia as informações extraídas para o jogo, que, com base nestas, atualiza o seu estado. Por fim, projeta-se o jogo sob a superfície da mesa, para que o usuário possua a impressão de estar interagindo diretamente com o jogo.

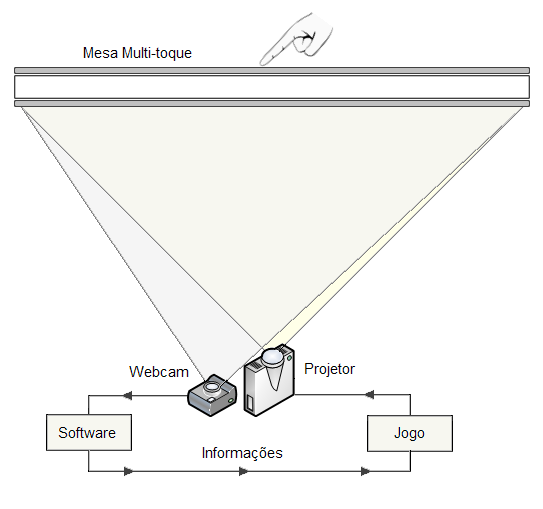


Figura - Visão geral do sistema

Este trabalho está dividido da seguinte maneira: no capítulo a seguir, apresentaremos as bases teóricas utilizadas neste trabalho. Nele falaremos sobre os dispositivos multi-toques existentes e suas principais características, os tipos de jogos que abrangem este projeto e sobre as ferramentas utilizadas em seu desenvolvimento. O terceiro capítulo descreve o projeto, sua concepção e desenvolvimento, bem como problemas e soluções encontradas. O quarto capítulo encerra o trabalho, apresentando os resultados obtidos e possíveis trabalhos futuros.

## Organização do Trabalho

Este tra

# Bases Teóricas e Tecnologias Empregadas

Neste capítulo serão apresentadas as bases teóricas e as tecnologias empregadas no desenvolvimento do projeto. Exemplos de dispositivos multi-toques comercialmente distribuídos hoje em dia serão exibidos.

## Dispositivos Multi-toques

### Microsoft Surface

Idealizada desde 2001, pela *Microsoft Hardware* em parceria com a *Microsoft Research*, o *Microsoft Surface Computer* foi apresentada ao público em abril de 2007 pela *Microsoft* *Corp* em parceria com a *AT&T.*

Permite através de uma mesa multi-toque a manipulação de imagens, mapas, aplicativos, vídeos e jogos através do toque. Possui a capacidade de reconhecer objetos sobre sua superfície, como celulares, *palmtops*, cartões de crédito, entre outros. Utiliza uma interface de comunicação com tecnologia *Wi-Fi* e *Bluetooth 2.0*.

Revelou-se depois que as capacidades de tratar interação multi-toques demonstradas na *Microsoft Surface* estarão presentes na próxima versão do Windows (Windows 7), e que os fabricantes de *notebooks* passarão a oferecer produtos com esta funcionalidade.



Figura - Microsoft Surface

#### Especificações

Possui uma tela de 30 polegadas disposta como uma mesa, 22 polegadas de altura (56 cm), 21 polegadas de profundidade (53 cm), e 42 polegadas de largura (106 cm). Sua superfície é de acrílico e o interior de aço com pintura eletrostática. Utiliza o sistema operacional *Microsoft Windows Vista* e possui conectividade *Ethernet* 10/100, *wireless* 802.11 b/g e *Bluetooth* 2.0.

Vem preparado para reconhecer vários aparelhos com tecnologia *Bluetooth* e *wireless*, bastando colocar o aparelho sobre a tela que o sistema reconhecerá e irá mostrar as opções de interação com o [periférico](http://pt.wikipedia.org/wiki/Perif%C3%A9rico).

A visão computacional é formada por canhões de leds infravermelhos direcionados à superfície, juntamente com cinco câmeras de resolução 1280 x 960 que permitem a detecção do toque.

#### Interação

A interface possui quatro componentes principais: interação direta, contato multi-toque, multiusuário e reconhecimento de objetos. Permite que diversas pessoas interajam simultaneamente com as aplicações de forma simples e intuitiva; com a capacidade de reconhecer a presença de dispositivos sobre sua superfície, e trocar informações a partir de uma de suas opções de conectividade.

### ReacTable

A *ReacTable* é um instrumento musical colaborativo, desenvolvido pela Universidade Pompeu Fabra, situada em Barcelona, que permite o reconhecimento de objetos e tem a possibilidade de detectar interação multi-usuário e sintetizar sons gerados através de fontes, filtros e osciladores. Cada objeto colocado sobre sua superfície é classificado por um software a partir de marcadores fiduciais[[2]](#footnote-3) situados em sua superfície e capturados por uma câmera. Assim, cada objeto é classificado como um dos geradores e filtros de uma aplicação musical obtendo-se como resultado um som único, resultado da interação destes objetos. Este instrumento utiliza como base o software de detecção de fiduciais *ReacTIVision*, que reconhece dos objetos sobre a mesa. Foi utilizado em shows e apresentações da cantora islandesa Björk, no *Coachella Festival*, em 2007 na Califórnia.



Figura - ReacTable

#### Especificações

É uma mesa redonda e translúcida, utilizada numa sala escura, aparentando um display iluminado por trás. Sob a mesa, há uma câmera, utilizada na visão computacional, e um projetor conectado a um computador. A parte tangível são peças de acrílico com fiduciais impressos em sua superfície.

#### Interação

Os diversos tipos de objetos tangíveis representam diferentes módulos de um sintetizador analógico, alguns objetos podem interferir no funcionamento daqueles que estão à sua volta. Quando um objeto é posicionado sobre a mesa, em alguns casos, aparecem controles que permitem a mudança, com um toque, do comprimento de onda, volume, tons e padrões musicais.

### iPhone

O *iPhone* é um *smartphone* desenvolvido pela *Apple Inc*. que mescla funções de *iPod*, câmera digital e Internet. A interação é feita através de uma tela sensível ao toque.



Figura - iPhone

#### Especificações

Possui um processador de 620 MHz ARM 1176[2], operando em 412MHz, 128MB DRAM, armazenamento flash de 8 ou 16 GB, conectividade *headphone*, *USB*, *Firewire*, *Wi-Fi* (802.11b/g), *Bluetooth* 2.0+EDR e uma tela multi-toque.

#### Interação

O dispositivo possui três sensores: um sensor de proximidade, que previne que o rosto ou orelhas interajam com o dispositivo enquanto uma chamada é atendida; um sensor de luz ambiente que altera automaticamente o brilho do display, e um acelerômetro de 3 eixos, que, de acordo com a orientação do aparelho, muda o posicionamento das informações exibidas em sua tela. Os toques são reconhecidos por mapeamento do campo elétrico em uma película que fica entre uma tela protetora e o display *LCD*.

## Jogos e Interatividade

### Jogos de Estratégia

Do grego *stratègós*[[3]](#footnote-4),possui o significado de liderança do exército. Por isso, grande parte dos jogos de estratégia envolve embates militares e gerenciamento de recursos entre nações, povos, condados, planetas, ou qualquer história que justifique divergência.

Existem duas características básicas para os jogos de estratégia. Se são em tempo real ou turnos; e se são abstratos ou simuladores.

Na dimensão temporal, não existe divisão de tempo ou de permissão para jogadores terminarem suas jogadas. Todos podem agir assim que tomadas às decisões, sem a necessidade de esperar sua vez, como ocorre em jogos baseados em turnos; onde cada jogador deve esperar seu momento de agir. Em alguns jogos, porém, podemos encontrar ambas as características. As ações são em tempo real e quando os jogadores entram em combate, as ações mudam para um ambiente de turno.

Em termos de realidade, trata-se da pureza de estratégia do jogo em relação ao ambiente. Enquanto alguns jogos tentam reproduzir fielmente as guerras napoleônicas, ou as conquistas do império romano; outros não possuem ligação com o mundo real, como por exemplo: Go, damas, xadrez, entre outros.

### Jogos de RPG

*RPG* é a sigla de *Role Playing Game*, podendo ser traduzido como jogo de interpretação de papéis, ou representação de personagens. É um tipo de jogo em que os participantes assumem papéis de personagens, criam uma história colaborativamente e seu progresso segue a partir da improvisação dos jogadores dentro de um sistema de regras predeterminado, determinando o progresso do jogo.

Os *RPGs* são tipicamente mais colaborativos e sociais do que competitivos. Une seus participantes em um único time que aventura-se como um grupo. Raramente têm ganhadores ou perdedores, tornando-o fundamentalmente diferente de outros jogos de tabuleiro, jogos de cartas, esportes, ou qualquer outro tipo de jogo. Por esses motivos a *NASA*[[4]](#footnote-5), em longas missões espaciais, utiliza o *RPG* como forma de entretenimento, evitando o conflito entre os tripulantes. Os russos, como alternativa, utilizaram o jogo de xadrez, reduzindo o tédio, estresse e depressão que estas viagens podem causar.

O *RPG* é um jogo pouco convencional quando comparado aos jogos habituais. Em um teatro, os atores recebem seu *script*, um conjunto de suas ações, gestos e falas, com tudo o que seus personagens devem saber e fazer. O jogador interpreta um personagem de ficção, seguindo o enredo definido por um roteiro. Em um jogo de estratégia, por outro lado, o jogador segue um conjunto de regras onde, para vencer, precisam-se vencer os desafios impostos pelos adversários. Cada partida é única, uma vez que é impossível prever seus movimentos durante o jogo. No *RPG*, esses dois universos se unem.

Assim como um jogo de estratégia, há regras que o definem, e definem aquilo que o seu personagem pode ou não pode fazer. A esse conjunto de regras chama-se sistema. No teatro, todos os personagens têm uma história, e deve ser interpretado assim como os atores fazem. Em um RPG, os jogadores, não lutam apenas contra um adversário específico, mas vivem aventuras em um mundo imaginário.

Assim como romances e filmes, *RPGs* alimentam a imaginação sem limitar o comportamento do jogador a um enredo específico. Pode ser orientado a um livro de *RPG*, como o famoso *Dungeons & Dragons*. Além disso, os jogadores podem desenvolver seus próprios enredos e sistemas, onde boa é baseada em livros. Como exemplo, temos a obra de J. R. R. Tolkien, O Senhor dos Anéis, que influência até hoje a criação de diversos *RPGs*.

#### Dungeons & Dragons

*Dungeons & Dragons* (abreviado como *D&D* ou *DnD*) foi o primeiro *RPG* comercial, publicado em 1974 nos EUA. É um livro que retrata um universo de fantasia medieval e suas características. Cada jogador controla um único personagem e este possui uma área de atuação (bardo, monge, guerreiro, entre outros dentre o universo retratado). Atuam em grupos e as partidas geralmente possuem várias sessões, que são chamadas de aventuras e esse conjunto de sessões é chamada campanha.

Os resultados das escolhas do grupo e sua historia básica é determinada pelo Mestre, de acordo com as regras e sua devida interpretação. O Mestre escolhe e descreve os vários Personagens do Mestre (PDMs), o cenário, as relações e os resultados destes encontros. As extensas regras do jogo cobrem diversas áreas como interações sociais, usos de magia, combate e o efeito do ambiente nos personagens. O Mestre pode escolher quais regras publicadas ele vai utilizar e até mesmo criar novas, se achar necessário.



Figura - Exemplo de campanha em andamento

### RPGs Eletrônicos

Os primeiros jogos de RPG para computador surgiram no início dos anos 70 baseados na série *Dungeons & Dragons*. Ganharam popularidade durante a década de 80 e hoje é um dos gêneros de jogos mais populares de todo o planeta.

Segundo um estudo[[5]](#footnote-6) de Gwendolyn Kestrel, Ph.D. em educação, os jogos de *RPGs* podem ampliar a capacidade de leitura, interpretação e raciocínio lógico de seus jogadores, além de estimular o pensamento criativo

Os primeiros *RPGs* criados para computadores eram jogados em modo texto, onde cada comando determinava uma ação que o personagem deveria executar. A interface era baseada em caracteres e as ações que poderiam ser feitas, limitadas a movimentos e ataques a monstros imaginários. Um exemplo é o *Zork*, criado em 1979, descendente do *Colossal Cave Adventure*.

Com o início dos videogames em formato console durante a década de 80, o gênero passou a se popularizar e ganhar jogos mais inteligentes e visualmente mais interessantes. O personagem destes jogos agora é representado graficamente na tela e os objetos com os quais ele interage também. No final da década, séries hoje em dia populares começaram a ser lançadas, como por exemplo, *Final Fantasy* e *Might and Magic*.



Figura - Final Fantasy - Square (1987)

Já na década de 90, os jogos de *RPG* deixaram ser *2D* e passaram a ser *3D*. Cada vez mais bem trabalhados e com personagens mais realistas e ações mais complexas, ganharam mais adeptos ao gênero. Grande exemplo desta popularização são os jogos *Final Fantasy VII* e *Final Fantasy VIII*, sendo que o primeiro ganhou uma continuação em filme.



Figura - Final Fantasy VII - Squaresoft (1997)

Juntamente com a popularização da internet, um novo gênero de *RPGs* foi criado: o MMORPG (*Massively Multiplayer Online Roling Playing Game*). Nesse gênero vários jogadores conectados pela internet interagem seus personagens uns com os outros dentro de um mesmo mundo virtual. Devido à grande possibilidade de interação com jogadores de qualquer parte do mundo, esse gênero de jogo é sucesso hoje em dia. Exemplos mais recentes são os jogos: *Ragnarok Online*, *World of Warcraft*, *Cabal Online*, *Lineage*, entre outros.



Figura 9 - World of Warcraft - Blizzard (2004)

#### Final Fantasy Tactics

O *Final Fantasy Tactics* é um *RPG* para o *Playstation* desenvolvido pela *Squaresoft*, em 1999, cuja inovação está a cargo de seu sistema de batalhas. Diversos fatores externos como o terreno, condições climáticas, disposição e classe dos personagens são fatores decisivos durante a batalha. Com isso, todas as ações devem ser pensadas, de maneira estratégica, para que o adversário consiga ser superado.

Cada personagem possui seu próprio campo de ação baseado em seus atributos, e os inimigos somente podem ser atingidos se nele estiverem. Cada atributo do personagem e sua respectiva evolução são baseados em sua classe. Com isso, as possibilidades de estratégia são imensas, pois cada combinação de classes em diferentes níveis de evolução produz um resultado diferente, com pontos fortes e fracos. A batalha é ganha após a derrota de todos os inimigos da tela.

Seria interessante pegar um exemplo concreto de batalhar, com uns dois screenshots, para explicar melhor a idéia. Entendi o que vocês conseguiram dizer mas ainda não tenho o bastante para imaginar como seria.



Figura - Final Fantasy Tactics - Squaresoft (1999)

## Realidade Virtual

Realidade Virtual (RV) é uma interface avançada para aplicações computacionais, onde o usuário pode navegar e interagir, em tempo real, em um ambiente tridimensional gerado por computador, usando dispositivos multisensoriais. (KIRNER & TORI, 2004). A realidade virtual é um sistema imersivo, ou seja, com ajuda de dispositivos multisensoriais, como capacetes e luvas digitais, o utilizador tem a sensação de estar dentro do mundo virtual e é capaz de manipular os objetos ali presentes.

### Realidade Aumentada

Um subconjunto da Realidade Virtual, a Realidade Aumentada (RA) consiste em modificar o mundo real com a sobreposição de objetos virtuais (UFRJ 2006). Azuma define Realidade Aumentada como sendo uma variação dos ambientes virtuais (VE). Enquanto VE mergulha o usuário completamente no mundo virtual (realidade virtual imersiva - RVI), que pode ser alcançada com a utilização de alguns equipamentos, como capacetes e luvas (KIRNER & TORI, 2004); na Realidade Aumentada existe um contraste entre real e virtual. Melhor que substituir completamente a realidade, a RA permite compor o real sobrepondo o virtual, assim ambos podem coexistir no mesmo espaço (AZUMA, 1997), (AZUMA et. al, 2001).

Uma das formas de se incluir objetos virtuais no mundo real é por meio da utilização de imagens projetadas sobre objetos reais, denominada Realidade Aumentada Espacial (RAE) (BIMBER & RASKAR, 1997). A RAE utiliza técnicas de Visão Computacional para rastrear e orientar a câmera constantemente para fundir imagens do mundo real e virtual corretamente.

## Implementações de Superfícies Multi-toque

### Iluminação Traseira (Rear Illumination)

### Reflexão Total Interna Frustrada da Luz (FTIR)

A reflexão da luz é o fenômeno físico em que um feixe de luz incide sobre uma superfície e é redirecionada para o mesmo meio de propagação de onde veio, podendo ser total ou parcial. Ocorrendo a reflexão parcial, obrigatoriamente ocorre a refração, que é o nome dado ao fenômeno quando a luz passa de um meio de propagação para outro.

A reflexão total interna da luz é um fenômeno óptico que ocorre quando um raio de luz atinge o limiar entre os meios de propagação em um ângulo maior do que o ângulo crítico, em relação à sua normal. Se o índice de refração do meio atual é menor o do meio externo, a luz é totalmente refletida.

*{imagem - ângulos}*

Quando a luz atravessa uma interface entre materiais com diferentes índices de refração, o feixe luminoso será parte refratado e parte refletido na interface da superfície..

No entanto, se o ângulo de incidência é maior (o raio de luz está mais para paralelo à interface) do que o ângulo crítico, o ângulo de incidência da luz refratada é tal que se desloca ao longo da interface, então a luz vai iluminar totalmente a interface ao invés de ser refletido de volta.

Quando se toca com o dedo tal superfície, alteramos o meio em que o feixe de luz iria se propagar assim alterando o ângulo critico empurrando então o feixe de luz para o lado oposto ao toque. Para isto se da o nome de Reflexão Total Interna Frustrada da Luz.

Procurar um texto mais científico que explique isto. Eu achei que não tinha nada a ver com o índice de refração do dedo, e sim que o dedo é um meio difusor, por isso manda luz em todas as direções.

Explicar brevemente como OSC, OSCpack e TUIO se relacionam ao trabalho para justificar porque se fala deles aqui

## Tecnologias Utilizadas

### OSC

O *Open Sound Comunication (OSC)* é um protocolo desenvolvido para a comunicação entre computadores, sintetizadores de som e outros dispositivos multimídia. É utilizado em diversas áreas, como Realidade Virtual, Interfaces Web e meio de transporte para outros protocolos que não possuem facilidade de comunicação.

#### OSCpack

É um conjunto de classes em C++ responsáveis por criar e ler pacotes do protocolo OSC, incluindo as funcionalidades mínimas para a comunicação utilizando UDP nas plataformas *Windows* e *POSIX*.

Atualmente é utilizada em diversos projetos, como o *ReacTIVision*, *Touchlib*, *AudioMulch*, entre outros; principalmente pela capacidade de prover a comunicação entre as plataformas *Windows*, *Linux* e *Mac*.

### TUIO

É um protocolo de comunicação desenvolvido com a finalidade de atender os requisitos de comunicação entre interfaces tangíveis. Define propriedades comuns de baseado no controle de objetos, toques e gestos. Foi criado pela equipe de desenvolvimento do projeto *ReacTable* e implementado sobre o protocolo de comunicação *OSC*. Hoje, possui diversas implementações nas linguagens Java, C, C++, Flash, entre outras.

As mensagens são divididas em *profiles*, baseados na interação com a interface tangível. Atualmente, possui *profiles* para interfaces 2D, 3D e customizadas. Cada um, por sua vez, possui dois tipos de mensagens diferentes, usadas na representação da interação de objetos e toques com o dispositivo.

A mensagem carrega diversas informações sobre a interação. As principais são: sessão, identificador da interação, posição no espaço 2D ou 3D, ângulo, vetor de movimento, vetor de rotação, aceleração de movimento, aceleração de rotação.

Qual a vantagem em se usar OSC/ OSCPack / TUIO versus outras soluções, por exemplo:

- inventar seu próprio protocolo sobre UDP ou TCP/IO

- invocação remota de métodos

- usar APIs voltadas a construir jogos multiplayer

### ReacTIVision

O *ReacTIVision* é um aplicativo open source de visão computacional voltado para reconhecimento de marcadores fiduciais. Criado pelo *Music Technology Group* da universidade *Pompeu Fabra* de *Barcelona*, como parte do projeto *ReacTable*; foi desenvolvido em *C++*, com a finalidade de obter imagens de um dispositivo de captura (no caso uma webcam), e reconhecer marcadores fiduciais. Baseado na arquitetura cliente-servidor atua como servidor, obtendo as informações dos fiduciais, como identificador, posição e orientação; enviando-os em seguida às aplicações cliente.

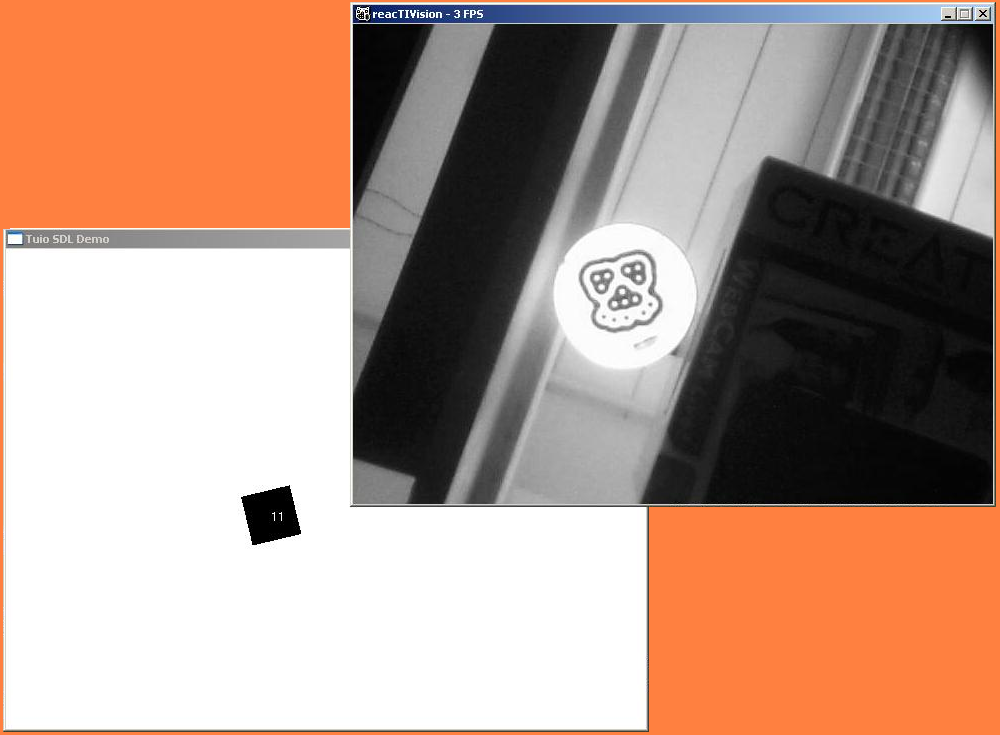


Figura - ReacTIVision reconhecendo um fiducial

Os marcadores fiduciais reconhecidos pelo *ReacTIVision*, são gerados por um algoritmo. O software, através de imagens enviadas por uma câmera, aplica diversos filtros, um algoritmo de segmentação e em seguida um algoritmo reverso ao de criação dos fiduciais. Com isso, o marcador é identificado e sua posição, inclinação e direção são obtidas.

Estes dados, juntamente com informações históricas, são utilizados na formação de mensagens *TUIO*. Estas mensagens, padrão de comunicação entre dispositivos multi-toque, são enviadas para as aplicações cliente, através do protocolo *OSC*, utilizando as bibliotecas do *oscpack*.

### Touchlib

O *Touchlib* é uma biblioteca que permite a detecção de toques em superfícies multi-toque que utilizam o princípio da *Reflexão Total Interna Frustrada da Luz*, *Iluminação Traseira* ou *Iluminação Frontal*. Desenvolvido pela *Natural User Interface Group[[6]](#footnote-7)* em parceria com a *White Noise Audio[[7]](#footnote-8)*; é bastante utilizada em aplicações multi-toque devido ao grande número de funcionalidades.

Através de algoritmos de divisão e comparação, detecta realces no histograma das imagens enviadas por uma *webcam*; transformando-os em informações sobre cursores, e disparando eventos que podem ser tratados em aplicações *C/C++*. Estes eventos são disparados quando um dedo toca, percorre ou é retirado da superfície multi-toque. Esta biblioteca permite a integração através do protocolo *TUIO*, sobre o protocolo OSC, utilizando a biblioteca *oscpack*.

Esta biblioteca aplica filtros nas imagens recebidas, a fim de melhorar a percepção de toques. Atualmente esta biblioteca trabalha apenas na plataforma *Windows*, porém existem esforços sendo realizados para portá-la para outras plataformas, como *Mac* e *Linux*.

#### Calibração e Configuração

Perguntas que \_\_precisam\_ ser respondidas:

- Que parâmetros são estipulados na calibração?

-Que tipo de ajuste é feito nas coordenadas de dados de entrada ou da imagem de saída uma vez que se tenha estes parâmetros?

A calibração e configuração do software devem ser feitos sempre que a câmera ou superfície multi-toque mudar de posição. Ao executar o aplicativo de calibração, diversas janelas exibem as imagens com os diversos filtros aplicados. Alterando-se os valores dos filtros é possível obter uma imagem nítida apenas do toque.

Coisas que não estão claras mas precisam ficar:

- o utilitário de calibração é uma ferramenta pronta

Seria interessante incluir uma figurinha deste utilitário

Se vocês não entendem muito do que acontece durante a calibração e ela para vocês é só um procedimento mecânico, não deve ser tão enfatizada como se fosse um fundamento teórico que é importante para a compreensão do trabalho de vocês – que é o objetivo desta seção como um todo. Pode ser mencionado depois, ou na parte de projeto ou num apêndice

O *Touchlib* trabalha em uma escala que vai de zero até um. O canto superior esquerdo da imagem é o ponto (0, 0), enquanto o inferior direito é o (1, 1). A calibração subdivide a imagem que o software processa em 20 imagens menores. Projetando essa divisão sobre a superfície multi-toque e sem seguida tocando os vértices dessas imagens, o software gera um arquivo *XML* de configuração, com os valores destes pontos e as configurações usadas nos filtros.

A touchlib limiariza e segmenta a imagem? Se sim, como se parametriza isto? Vi vocês fazendo isto manualmente e é preciso documentar aqui

Durante a execução do *Touchlib*, o arquivo é lido e os filtros aplicados na imagem enviada pela *webcam*. Quando um toque é detectado, sua posição é calculada através de uma interpolação linear apenas na respectiva fatia de imagem, obtida através da calibração. Isso permite que a câmera não necessite estar perpendicular à superfície de projeção, pois a distorção provocada é compensada via software.

### Microsoft XNA

A fazer...

# Projeto

**Incluir uma seção de concepção, ou idéia geral, em que vocês explicam como é o produto ou projeto do ponto de vista dos jogadores. Está faltando isto ser feito a contento e boas opções são dar uma visão geral na introdução e detalhar aqui, no início do capítulo de projeto.**

Para o reconhecimento de toques o software *Touchlib* foi utilizado, e para o reconhecimento de fiduciais, o software *Reactivision* foi utilizado. Ambos os softwares utilizam o protocolo *TUIO* juntamente com o protocolo *OSC* para se comunicar com o jogo. O conjunto de bibliotecas *oscpack*, que implementa o protocolo *OSC*, será utilizado para efetuar a comunicação softwares envolvidos.

O desenvolvimento do jogo se dará através do desenvolvimento de um protótipo inicial e uma versão final. Para agilizar o desenvolvimento do jogo, facilitar a integração e deixá-lo mais robusto, confiável e rápido, a linguagem de domínio específico, *Microsoft XNA 2.0* foi utilizada.

As ações que os usuários executarem sobre a mesa, como o posicionamento de um fiducial ou o toque de um ou mais dedos sobre sua superfície será reconhecida pelos softwares através da análise das imagens enviadas por uma *webcam*. O software processa as informações e envia uma mensagem *TUIO* para cada objeto ou dedo sobre a mesa, contendo as informações como posição, ângulo de movimentação, velocidades calculadas entre outras informações. Estas mensagens *TUIO* são empacotadas em envelopes *OSC* e enviadas via *UDP* para as aplicações clientes, no caso deste projeto, o jogo. O jogo lê estas informações e atualiza o estado do jogo, projetando-o através de um projetor sobre a mesa. Com isso o usuário possui a impressão de estar manipulando diretamente os objetos do jogo.

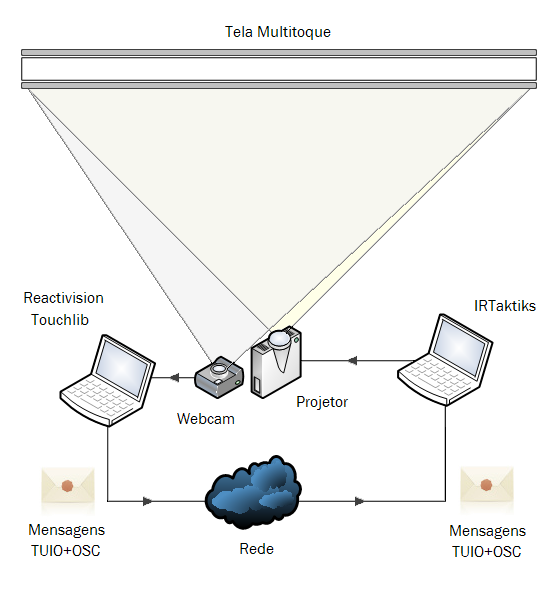


Figura - Visão geral do sistema

O desenvolvimento do projeto foi dividido em duas frentes. A primeira foi responsável pela adequação da mesa às necessidades do jogo, enquanto a segunda, responsável pela criação do jogo.

As melhorias na atual mesa iniciaram com a reestruturação de sua parte elétrica, de modo a facilitar sua posterior manutenção. A incorporação de uma superfície de projeção, alteração da webcam, responsável pela captura de imagens para a visão computacional, e adição e configuração de softwares responsáveis pelo reconhecimento de toques e fiduciais.

O desenvolvimento do jogo aconteceu através de um protótipo inicial, com a finalidade de validar a arquitetura, as tecnologias empregadas, softwares e frameworks utilizados; seguido da versão final onde os requisitos e objetivos foram atendidos.

## Adequação da Mesa

### Estrutura

A mesa multi-toque é formada por uma superfície de acrílico transparente de aproximadamente 1,2m x 1,6m, acoplada a um suporte de madeira sobre rodas, facilitando seu deslocamento. Possui 47 entradas para leds infravermelhos, de modo que a luz percorra o interior do acrílico capturada de acordo por causa do princípio da reflexão total interna da luz. Estas entradas são dispostas pelos quatro lados da mesa, intensificando a propagação da iluminação dos leds, em direção ao centro do acrílico.

Nenhuma foto da mesa????

Anteriormente, a mesa não atendia todas as necessidades do projeto, pois o reconhecimento do toque somente era possível pressionando o dedo com força considerável sobre sua superfície, próximo às suas laterais. Isso acontecia devido à intensidade da iluminação dos LEDs ser insuficiente para percorrer toda a superfície do acrílico. A mesa possuía 47 leds, que eram alimentados por uma fonte de apenas 9V. Para cada LED existia um resistor de 150Ω, limitando a corrente elétrica em torno de 52mA.

Vocês tiveram um trabalhão ao longo do semestre que não está refletido aqui. Ainda temos alguma imagem da iluminação do circuito pré-reestruturação para evidenciar que havia pouco realce infravermelho nos pontos de toque.

Para atender às necessidades, a parte elétrica foi completamente reestruturada. Inicialmente, tínhamos o objetivo de apenas trocar os componentes eletrônicos utilizados, aumentando a corrente elétrica nos leds e conseqüentemente a intensidade da iluminação.

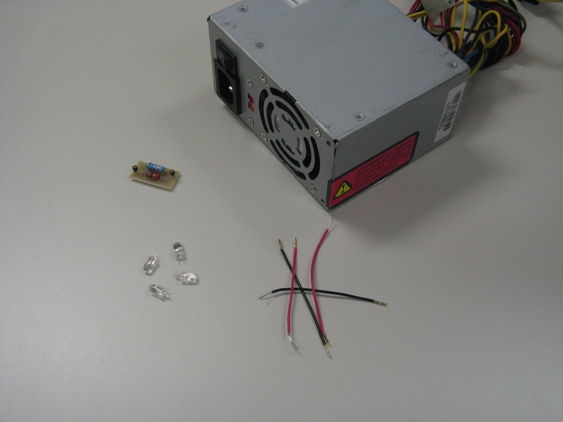


Figura - Componentes elétricos utilizados

Devido a problemas de destruição por parte de terceiros **não-identificados que esporadicamente freqüentavam o laboratório em que o projeto era desenvolvido**, fios e conexões também tiveram que ser trocados. Para facilitar a manutenção, evitando dificuldades de reparos após eventuais novas depredações, decidimos tornar todas as ligações completamente modulares, de fácil substituição, pois não seria utilizada nenhuma solda ou cola na fixação dos componentes.

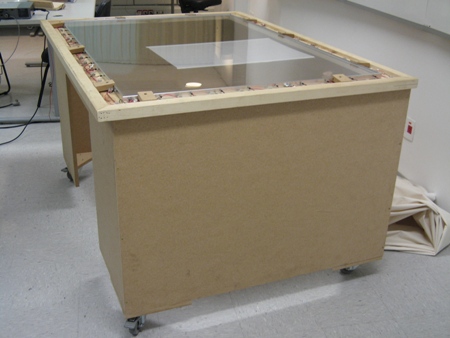


Figura - Mesa após reestruturação da parte elétrica

Atualmente, a mesa conta com 47 leds infravermelhos de alto brilho, com corrente elétrica de trabalho de 100mA e tensão de barreira de potencial de 1,2V, subdivididos em 10 malhas. Cada malha possui dois resistores, um de 56Ω e outro de 5,6Ω, ligados em série, limitando a corrente de 5 leds, também ligados em série. O último ramo, por não possuir 5 leds, possui dois resistores de 56Ω.

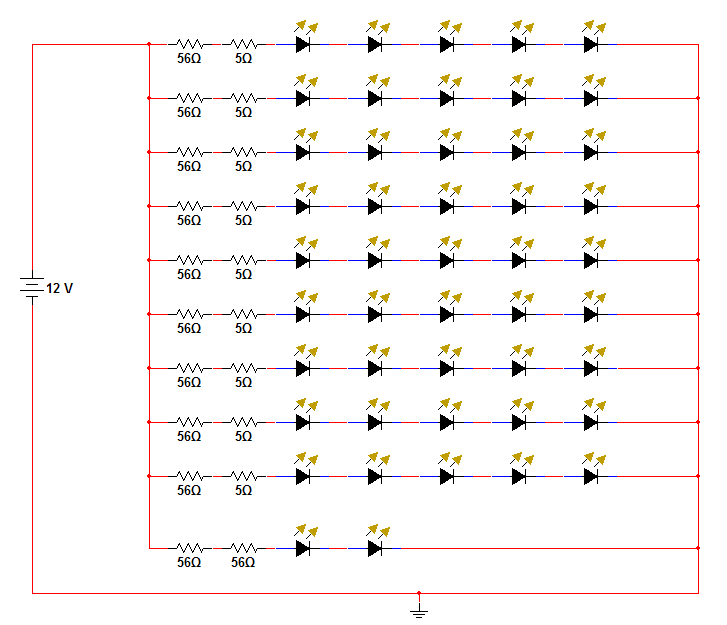


Figura - Representação das malhas do circuito elétrico da mesa

Na malha com 5 leds, a corrente elétrica aumentou para 97mA, enquanto na de 2 leds, a corrente é de 84mA. Como a intensidade de iluminação de um led, aumenta de forma linear com o aumento de sua corrente elétrica, podemos considerar que a iluminação da mesa teve um aumento de 53%.

|  |  |
| --- | --- |
|  |  |

Para cada malha, os resistores em série foram montados sobre uma placa de circuito impresso, de modo a facilitar seu acoplamento lagem à mesa e eliminar ligações feitas através de soldagem.

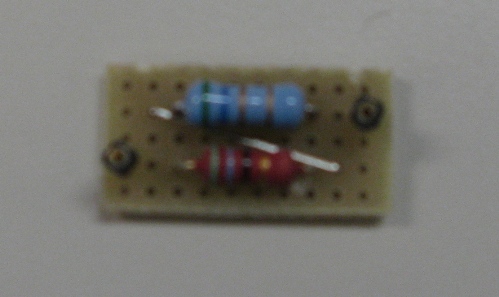


Figura - Placa de circuito impresso com os resistores de 56Ω e 5,6Ω

### Visão Computacional

Para obter as imagens dos toques e fiduciais foi utilizada uma *webcam* *Microsoft LifeCam VX-6000*. A escolha desta *webcam* se deu ao fato de possuir ângulo de visão 71º, sensor CCD (*charge coupled device)* com resolução de 800px por 600px e taxa de quadros de 30 fps (*quadros por segundo*).

Como esta *webcam* possui um filtro que inibe a captação da luz infravermelha, este teve que ser removido. Após sua remoção, um filtro que permite somente a passagem da luz infravermelha foi adicionado, permitindo que se utilize a mesa, mesmo em ambientes iluminados.

A webcam fica posicionada abaixo de mesa, olhando para o acrílico, de modo a obter as imagens dos toques e fiduciais. Por possuir grande ângulo de visão, a câmera pode ser colocada a uma distância menor que uma câmera convencional e mesmo assim cobrir uma área da mesa maior ou equivalente.



Figura - Microsoft LifeCam VX-6000

A projeção é feita utilizando um projetor de resolução máxima de 1280px por 960px e um espelho com fator de ampliação 2. A imagem é direcionada sobre a superfície inferior do acrílico. Para que a projeção possa ser vista pelo usuário da mesa é necessário um material difusor posicionado sob a superfície do acrílico.

O material ideal para este tipo de mesa é um polímero para projeções, fabricado pela *Rosco*. Este material não foi encontrado facilmente no Brasil, sendo sua utilização descartada. Foram realizados dois testes, utilizando papel vegetal e sacolas plásticas. Testes realizados indicaram que sacos plásticos forneceram maior nitidez na detecção de toques e fiduciais, em relação ao papel vegetal. Como este material não foi encontrado no tamanho necessário, o papel vegetal foi adotado como anteparo de projeção.

### Testes e Dificuldades Encontradas

Antes da reestruturação da mesa, a detecção de toques era difícil, pois o contraste entre o acrílico e os dedos era baixo. Após a mudança do circuito elétrico, o contraste ficou maior, permitindo uma leitura mais precisa por parte do software *TouchLib*.



Figura - Toque antes e depois da reestruturação **Este DEPOIS está usando só um segmento com poucos LEDSs, ficou melhor que isso!! Vocês têm alguma foto da mesa usando todos???**

Uma grande dificuldade no desenvolvimento da mesa foi o reconhecimento de marcadores fiduciais da Reactivision, que acabaram sendo eliminados da concepção do projeto. A projeção sobre o acrílico causa muitas variações de iluminação, o que inviabilizava a detecção destes fiduciais em imagens de uma câmera captando luz no espectro visível. Para resolver este problema, os fiduciais seriam eram iluminados por LED infravermelho, de maneira a ser discernível na imagem da câmera mesmo quando há incidência de luz do projetor sobre ele. Um circuito com um LED infravermelho foi encaixado no fundo de um copo de isopor, que funciona como difusor da luz e auxilia na iluminação. O fiducial fica sobre a mesa, dentro do copo, sendo iluminado pelo led, através de um furo no topo do copo.



Figura - Canhão de led usado na iluminação

Informações que não ficaram claras a respeito destes testes:

- Desde a óbvia: “por que preciso de um material como anteparo”?

- Até: qual propriedades vocês gostariam que o material tivesse

- Passando por: como foi o setup de testes. É preciso mostrar um esqueminha explicando como vocês colaram este material para fazer as coisas e que teste foi feito

- Incluindo: por que os testes não foram refeitos com a VX 6000? Imagino que seja porque vocês já tinham desistido dos fiduciais por causa dos maus resultados obtidos com a camerazinha da Creative

Utilizando este canhão, os testes utilizando anteparos de projeção para ficar junto ao acrílico de maneira a que os usuários possam ver as imagens do jogo puderam ser iniciados. A detecção do fiducial era apenas possível nas imagens captadas pela webcam quando foi usado como anteparo material de sacos plásticos brancos (polietileno), enquanto o toque era detectável sob os dois materiais. Outra limitação dos fiduciais foi a distância que a câmera deveria estar do fiducial, que era em torno de 25 cm . A essa distância é impossível executar qualquer aplicação, pois a área usável da mesa é muito restrita. Com base nestes testes e nestas limitações, o uso de fiduciais foi descartado do projeto.



Figura - Fiduciais sobre papel vegetal e plástico

## Jogo

A primeira etapa do desenvolvimento do jogo foi escolher qual seria o ambiente de desenvolvimento. **Não! A primeira etapa deveria ser ter uma visão geral ou concepção do jogo que se quer fazer! Em termos de monografia, depois viria explicar do ponto de vista do usuário e da aplicação! Inclusive esta deveria ser a primeira informação a constar neste capítulo e está fazendo uma grande falta.** A escolha deveria ser baseada nas funcionalidades de comunicação das bibliotecas utilizadas para o reconhecimento dos toques e objetos sobre a mesa.

Como utilizamos o software *Touchlib* para o reconhecimento de toques e o *Reactivision* para o reconhecimento de fiduciais, e pelo fato de ambos utilizarem a mesma arquitetura de comunicação, bem como a mesma biblioteca, *oscpack*, o jogo poderia ser desenvolvido em praticamente qualquer ambiente. Dessa forma a escolha foi baseada apenas em qual ambiente a produtividade seria maior e qual teria mais recursos. Dentre *frameworks* existentes, escolhemos o *Microsoft® XNA 2.0*, devido à enorme variedade de recursos disponíveis, documentação, desempenho e ganho de produtividade, uma vez que a linguagem de programação adotada seria *C#.*

O jogo desenvolvido é um *RPG* tático semelhante ao conhecido *Final Fantasy Tactics*, onde o jogador controla vários personagens com características diferentes, cujo objetivo é derrotar o inimigo através de ataques, magias e itens, utilizando táticas, como por exemplo, se beneficiar de uma determinada posição no campo de batalha para obter vantagens sobre o inimigo.

Transpondo a idéia do jogo para uma mesa multi-toque, decidimos utilizar objetos físicos para representar os personagens no campo de batalha e o toque seria utilizado para interagir com as ações que os personagens deveriam executar. Como para a detecção de objetos utilizaríamos fiduciais, e devido a problemas em sua detecção, o controle dos personagens também ficou a cargo do toque.

Com isso, o jogo ganhou portabilidade, uma vez que não é necessário utilizar uma mesa para jogá-lo. Qualquer superfície multi-toque em qualquer posição é suficiente para interagir com o jogo.

O jogo deve ser jogado por dois jogadores, sendo que cada jogador terá várias unidades de combate. Cada uma possui diversos atributos que quando configurados tornam-na única e diferente das demais em vários aspectos. Além de atributos, as unidades possuem classes que lhes dão características, vantagens, desvantagens e ações diferentes ampliando as possibilidades de estratégia de cada um dos times. O objetivo é derrotar todas as unidades do jogador adversário, utilizando as características de cada unidade de combate e suas respectivas ações.

Faz muuuuuita falta uma figura do jogo para apontar estas questões de exércitos, unidades, etc, tipos de personagens, etc

O projeto do jogo, como dito anteriormente, consistiu no desenvolvimento de um protótipo seguido da versão final. Antes do desenvolvimento do protótipo, um módulo de comunicação entre o jogo e a mesa foi projetado e desenvolvido. Dessa forma, futuros problemas de integração seriam eliminados, uma vez que a construção do jogo levaria este módulo de comunicação em consideração.

Foi decidido que este módulo utilizaria eventos para representar as interações dos usuários com a mesa. Com isso, o projeto do jogo foi simplificado e modularizado. O serviço que lê as mensagens *TUIO*, e dispara os eventos, é executado em uma *thread* apartada, aumentando o desempenho do sistema de comunicação.

O diagrama abaixo seria melhor digerido se antes vocês mostrassem um diagrama de blocos em que se pudesse ter uma idéia da arquitetura do sistema como um todo. Fornecer detalhe antes de esclarecer a visão geral é uma boa estratégia pra confundir quem lê.

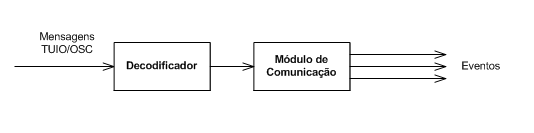


Figura - Visão geral do módulo de comunicação

### Protótipo

O protótipo foi desenvolvido com o intuito de validar as tecnologias empregadas na concepção do projeto. Teve como foco de desenvolvimento a decodificação de mensagens *TUIO*, geração de eventos no software em decorrência de ações realizadas na mesa pelos usuários, e criação de um simples jogo utilizando o *Microsoft XNA*. Desta forma, após o desenvolvimento do protótipo, estaríamos seguros quanto à escolha dos softwares escolhidos para o reconhecimento dos toques e fiduciais, bem como da tecnologia empregada para o desenvolvimento do jogo.

O desenvolvimento deste protótipo foi de extrema importância, pois possibilitou uma visão geral do problema que seria desenvolver um jogo com inúmeras regras -> estas regras são regras de gameplay ou regras de desenvolvimento de software? Esclarecer melhor o que isto quer dizer e fornecer exemplos. Que tipo de esclarecimento ou problema o protótipo permitiu ver melhor?. Com isso, sua arquitetura deveria ser bem planejada, ou então teríamos um jogo de fraco desempenho e de grande tempo de resposta.

Os testes realizados em cima do protótipo se mostraram bastante satisfatórios, uma vez que o tempo de resposta de uma ação foi praticamente instantâneo, que se trata de um requisito muito importante a este tipo de sistemas, já que a sensação de estar manipulando o objeto diretamente tem que ser sentida pelos usuários de qualquer aplicação em superfícies multi-toque.



Figura - Protótipo

A arquitetura do protótipo doe similar à adotada no segundo protótipo, que se manteve até a arquitetura final e que será comentada no item X.y.z.

Este trecho do texto de vocês ficou *chato.* Vocês não explicaramdireito como é o protótipo e o que ele fazia, nem incluíram uma foto dele projetado na mesa, por exemplo. Se eu não tivesse visto alguma coisa no lab diria que parece que vocês não obtiveram resultados com ele

### Versão Final

O desenvolvimento da versão final teve como foco principal sua arquitetura. Foi trabalhada de modo a deixar o jogo o mais rápido possível, sem comprometer a qualidade e os requisitos propostos. A arquitetura foi dividida em diversos módulos, a fim de facilitar a implementação e extensão de funcionalidades, uma vez que com padrões definidos, a adição de novas funcionalidades é bastante fácil e ágil.



Figura - Versão final

A arquitetura interna no *XNA* é centralizada na classe *Game*, que provê métodos para atualização e desenho de objetos, além de possuir uma lista de *GameComponents* e *Services*, que são atualizados e desenhados automaticamente pela classe *Game*. Internamente, o *XNA* cria uma *thread* para cada componente e serviço, não havendo, portanto, uma ordem prevista de execução. A vantagem desta arquitetura é a velocidade na execução, uma vez que várias *threads* executando paralelamente se beneficiam dos processadores *multi-core,* bastante comuns hoje em dia.

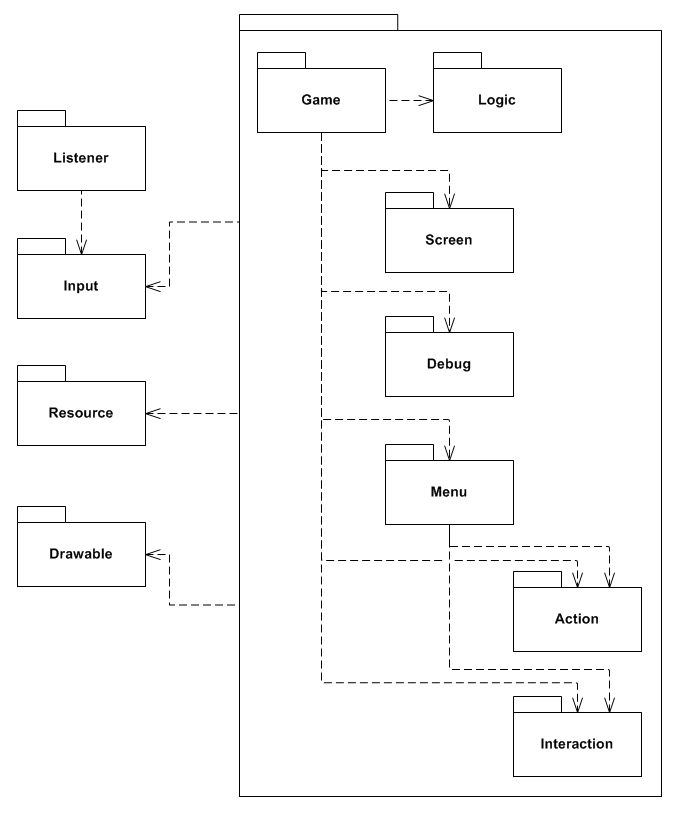


Figura - Arquitetura da versão final

Coloquem aqui um ou dois parágrafos fazendo referência à figura 23 e comentando em linhas gerais a arquitetura, antes de entrar na discussão de cada módulo como vem a seguir

#### Módulo Listener

Construído utilizando as bibliotecas do *oscpack*, é responsável por obter as mensagens *TUIO* enviadas pela mesa, decodificá-las e transformá-las em entradas para o jogo através da comunicação com o módulo Input. Baseia-se em uma arquitetura cliente-servidor, exercendo a função de cliente.

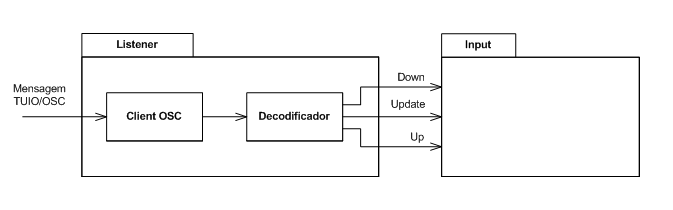


Figura - Visão do módulo Listener : **esta notação híbrida está um pouco estranha. Isto que vocês precisam representar seria melhor atendido por um diagrama de implantação ou por um diagrama de colaboração (o irmão do diag. Seqüência). As setas num diagrama em que os pacotes são mostrados significariam relações entre os pacotes, e não os tipos de métodos que são chamados. Ou vocês poderiam partir para diagramas 100% criados só para suas necessidades e usar uma notação de blocos comum**

As mensagens *TUIO* possuem informações sobre cada um dos toques e objetos que estão sobre a mesa. Estas mensagens tratam toques sobre a mesa como cursores e objetos identificados por fiduciais como objetos do TUIO. Cada cursor ou objeto possui um identificador, servido de base para o reconhecimento de ações mais complexas como funcionalidades *drag-and-drop*, ou simplesmente arrastar e soltar. Além de identificadores, cada cursor e objeto possuem três tipos de mensagens diferentes: *Down*, *Update* e *Up*.

As mensagens *Down* são enviadas quando o objeto ou o cursor são criados, ou seja, quando o objeto é colocado sobre a mesa ou quando o dedo encosta sua superfície. As mensagens *Update* são enviadas para informar que o cursor ou o objeto estão ativos, em outras palavras, servem para informar que o objeto continua sobre a mesa, parado ou em movimento, ou ainda para informar que o mesmo dedo encontra-se sobre a mesa, também parado ou em movimento. Já as mensagens to tipo *Up* são enviadas quando o objeto ou o cursor são removidos, ou seja, quando removidos da superfície da mesa. Com estes três tipos de mensagens é possível rastrear qualquer tipo de movimento sobre a mesa, seja ele usando objetos, toques, ou até mesmo uma combinação de ambos.

#### Módulo Input

É responsável por gerenciar as entradas de ações por todo o jogo. Para isso, utiliza uma arquitetura que distribui eventos comuns a todos os componentes do jogo. Dessa forma, não existem problemas de integração, uma vez que a comunicação entre mesa e jogo é centralizada. Após a definição dos eventos e dos dados que estes enviam a quem os trata, bastou apenas utilizar este módulo para obter as informações sobre toques e objetos sobre a mesa.

Inicialmente foram projetados seis eventos. Três representariam as ações possíveis com objetos sobre a mesa e três, os toques. Com o descarte do uso de fiduciais e conseqüentemente de objetos, o modelo final dispõe apenas de três eventos: *CursorDown*, *CursorUpdate* e *CursorUp*, todos estes enviando em seus argumentos o identificador do cursor, ou dedo, e a posição em que se encontra.

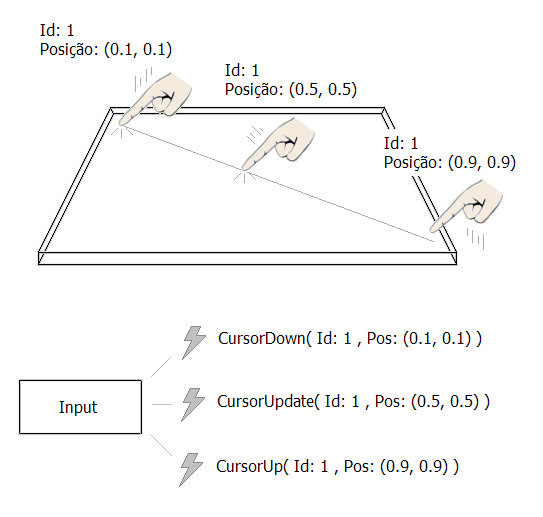


Figura - Exemplo de eventos do módulo Input

O identificador trata-se de um número inteiro gerado automaticamente pelo software *Touchlib* enviado dentro da mensagem *TUIO*. A posição também é enviada nesta mensagem, porém escalonada em valores entre zero e um, permitindo fácil conversão. Com estes dois parâmetros, é possível rastrear seqüências de movimentos, como arraste e rotacionamento. O movimento de arrastar é dado como um evento *CursorDown*, seguido de inúmeros eventos *CursorUpdate*, encerrando com um evento *CursorUp*.

#### Módulo Resource

É o módulo responsável por gerenciar os recursos utilizados pelo jogo, como texturas, fontes, mapas, imagens e efeitos. Baseia-se em gerenciadores, que são responsáveis por carregar os recursos a partir de arquivos e transformá-los em objetos manuseáveis dentro do *framework* *XNA*. Existem três tipos de recursos utilizados pelo jogo: texturas, efeitos e fontes.

Texturas são imagens, em diversos formatos e codificações. São utilizadas no desenho das estruturas dos menus, imagens de fundo, personagens do jogo entre outros. São usadas também na geração do mapa e na criação de partículas.

Efeitos, por sua vez, são códigos em *HLSL* (*High Level Shader Language*), utilizados para aplicar efeitos específicos, píxel a píxel, durante a renderização de uma cena. A aplicação destes efeitos acontece dentro da própria placa de vídeo, não consumindo assim tempo de *CPU* da máquina que executa o jogo, mas por sua vez, requer uma placa de vídeo que possua suporte.

Por utilizar na renderização do mapa mesclagem de texturas, a fim de obter um mapa mais realista, instruções *HLSL* mais complexas foram utilizadas. Com isso, a placa de vídeo necessária deve ser capaz de compilar estas instruções utilizando *PixelShader[[8]](#footnote-9)* *3.0* e *VertexShader[[9]](#footnote-10)* *3.0*. Placas de vídeo com suporte nativo ao *DirectX[[10]](#footnote-11) 9.0c*, possuem esta característica.

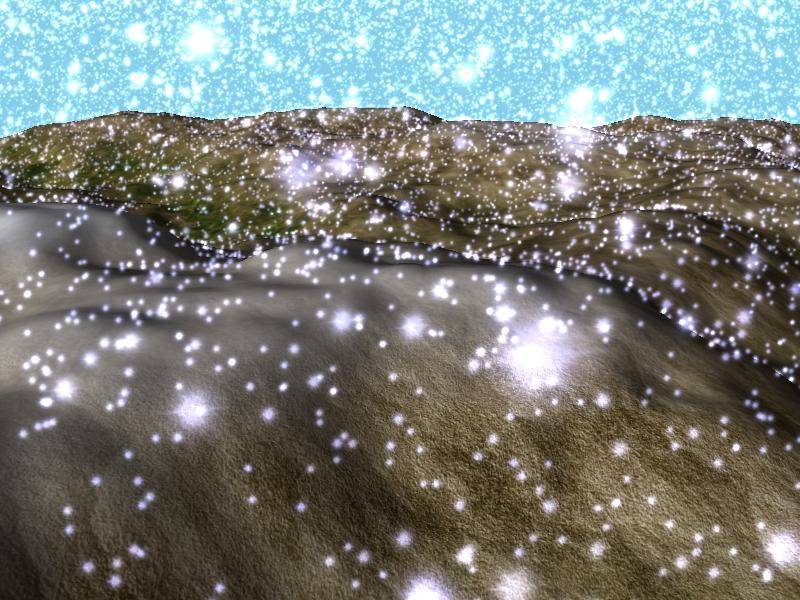


Figura - Exemplo de utilização de efeitos hlsl

Já as fontes são utilizadas na escrita de textos no jogo. Existem dois tipos de fontes utilizadas no desenvolvimento do jogo. A primeira é baseada em uma estrutura XML, onde as propriedades como espaçamento vertical, espaçamento horizontal, tamanho, cor e fonte são informados. Este tipo de fonte é a que consome menos memória, porém não é possível aplicar nenhum efeito.

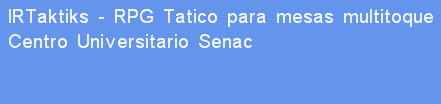


Figura - Exemplo de utilização de fonte XML

Quando se deseja que o texto escrito possua efeitos, como sombreamento, brilho, chanfros, contornos e texturas; é necessário o uso de fontes-textura. A fonte-textura trata-se de uma imagem que possui a seqüência os caracteres ASCII, já com os efeitos desejados, do número 32 aos 127. Cada caractere deve estar dentro de uma área de fundo totalmente transparente e entre elas deve existir em todas as direções, no mínimo, um píxel na cor magenta (R: 255; G: 0; B: 255; A: 255), para definir a separação dos caracteres, pelo *ContentProcessor[[11]](#footnote-12)* do framework *XNA*.

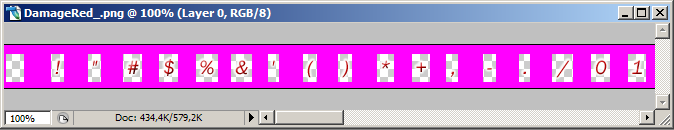


Figura - Exemplo de fonte-textura

Com isso, é possível desenvolver os caracteres em qualquer software gráfico, como por exemplo, o *Adobe Photoshop*; e em seguida aplicar as regras acima de forma a utilizar a fonte no jogo. Como a aplicação das regras é algo trabalhoso e repetitivo, foi desenvolvido um programa que aplica as regras tendo como fonte o arquivo com os caracteres sobre um fundo transparente, facilitando o desenvolvimento de texturas para utilização no jogo.

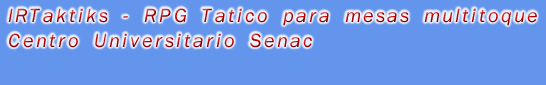


Figura - Exemplo de uso de fonte-textura

Os gerenciadores de recursos trabalham com *cache*. O carregamento dos recursos é feito apenas quando a primeira solicitação acontece, e a partir disso, qualquer uso ao recurso será imediato, pois o mesmo estará já estará alocado em memória, sendo desalocado apenas ao fim do jogo.

#### Módulo Drawable

É o módulo responsável por gerenciar todas as operações de desenho do jogo, desde textos e texturas a mapas e animações. É subdividido em seis submódulos, cada um responsável pelo gerenciamento de uma entidade: *Camera*, *Sprite*, *Map*, *Area*, *Animation* e *Damage*.

Parte desse gerenciamento é feita utilizando uma lista de tarefas. Como a execução da atualização dos componentes não segue uma ordem específica, utilizando o mesmo gerenciador, é possível ordenar os componentes, através de uma prioridade, quando necessário.

##### Submódulo Camera

Representa a câmera do jogo, provendo matrizes de visão e projeção, ângulo de visão e posição do observador dentro da cena. Cada objeto 3D, como o mapa, áreas e animações, necessitam de uma matriz de projeção e visão para serem projetadas na tela de forma correta. A matriz de projeção define a área visível da cena, ou *frustum*; enquanto a matriz de visão representa as transformações que devem ser aplicadas para os objetos serem projetados corretamente na tela.

A área visível é representada de maneira semelhante a uma pirâmide, cortada por dois planos. O plano mais próximo da posição do observador é chamado de *near plane*, enquanto o mais afastado de *far plane*. Qualquer objeto que após a aplicação de suas transformações encontra-se fora desta “pirâmide” é automaticamente removido da renderização da cena.

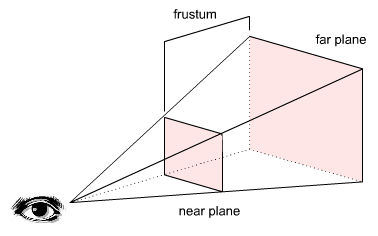


Figura - Representação da área visível da cena

A altura e largura do *viewport* da tela de jogo coincidem com as do *near plane*. Devido às especificidades de cada objeto 3D, decidiu-se por utilizar matrizes de projeção e visão individuais.

##### Submódulo Sprite

Gerencia o desenho de *sprites*, ou seja, textos e imagens, utilizando prioridades para ordenar os componentes que serão desenhados. Quando um objeto necessita se desenhar na tela, ele transfere esta responsabilidade ao gerenciador, que armazena o que deve ser desenhado, e a cada atualização do jogo, a lista é ordenada e os itens armazenados são desenhados.



Figura - Exemplo de sobreposição de sprites

##### Submódulo Map

Responsável pela criação e desenho do mapa sobre o qual o jogo acontece. O mapa, diferentemente dos *sprites*, trata-se de um objeto 3D, fazendo uso do submódulo *Camera*.

Não há gerenciadores que controlam o seu desenho, pois não existe a possibilidade de existirem dois mapas sendo desenhados ao mesmo tempo. O mapa é sempre o primeiro item a ser desenhado, uma vez que todos os personagens devem estar sobre ele.

Inicialmente, optou-se por utilizar um arquivo de geometria para representar o mapa com suas texturas. Para sua criação, utilizou-se geradores automáticos de terrenos, como o *Terragen 2*, *L3DT*, *Nem's Mega 3D Terrain Generator* e o *Vue xStream 6*. Estes programas exportam arquivos em diversos formatos, com texturas embutidas, que são facilmente reconhecidos por diversos softwares de modelamento, como o *Autodesk 3ds Max* ou *Softimage XSI*.

O framework *XNA*, por trabalhar intimamente com o *Microsoft DirectX*, apenas é capaz de reconhecer arquivos de geometria de formato *X*. Para a geração destes arquivos foi utilizado o software *Autodesk 3ds Max*, que possui um *plugin* otimizado para a integração com o *XNA*. O software escolhido para gerar os terrenos foi o *Vue xStream 6*, pois além de exportar a cena para o formato próprio do *Autodesk 3ds Max*, possui alta qualidade de geração, diversas texturas e algoritmos de geração.

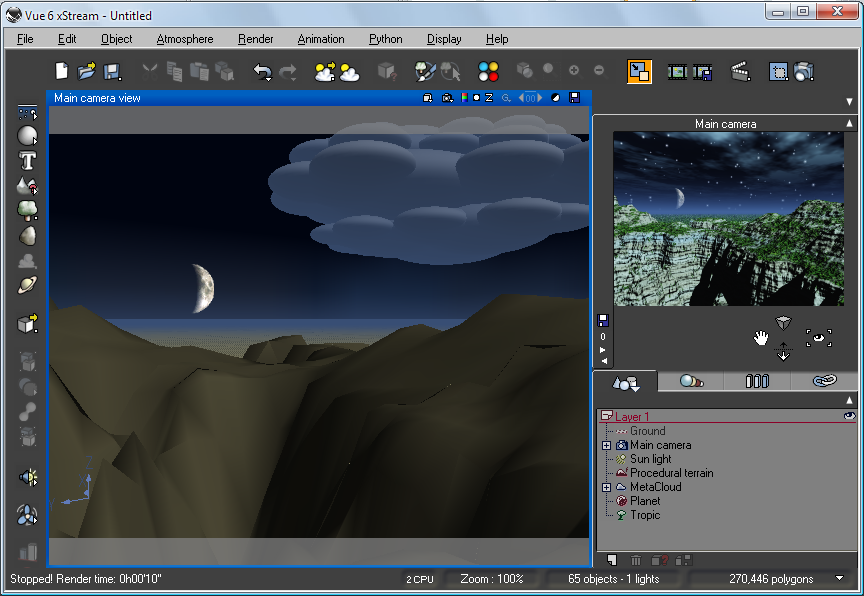


Figura - Software Vue xStream 6

Após a criação do mapa, este foi importado para dentro do jogo. Apesar de o arquivo exportado conter as texturas e estas estarem referenciadas o XNA não conseguia interpretá-las. Tentativas de aplicação da textura por código dentro no jogo também não surtiram efeitos, e outra abordagem para a geração do mapa teve que ser pensada. Outro fator que nos levou a tomar esta decisão foi a quantidade de memória consumida (aproximadamente 50Mb) e o tamanho do arquivo de geometria (aproximadamente 40Mb).

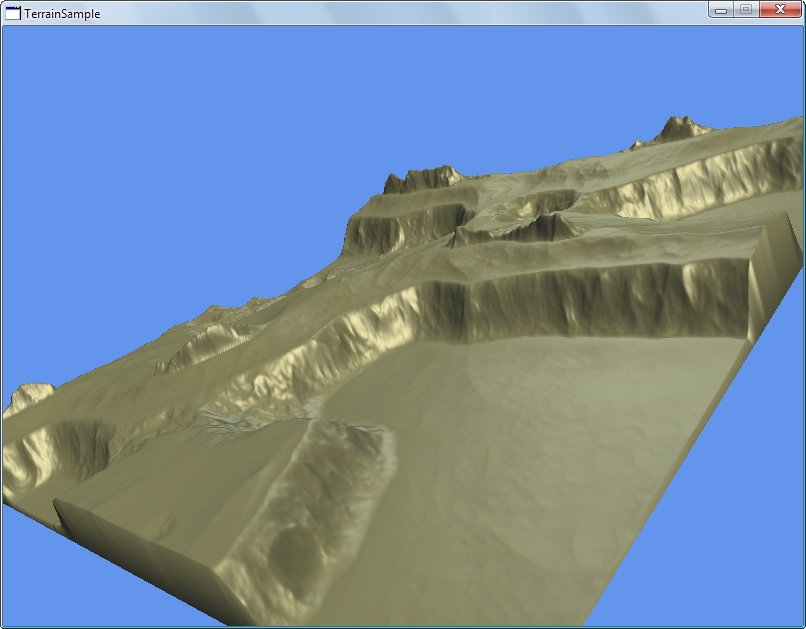


Figura - Mapa utilizando arquivo de geometria (40Mb)

A segunda alternativa para a geração do terreno seria a utilização de um arquivo de imagem, monocromático chamado de *heightmap*, ou mapa de altura. Este arquivo também pode ser gerado através do *Vue xStream 6*[[12]](#footnote-13). O tamanho do arquivo determina o tamanho do terreno que é gerado, enquanto cada píxel determina um vértice do terreno. A altura de cada um destes vértices é dada pela intensidade da cor branca do píxel em questão. O jogo lê o arquivo, criando os vértices para cada píxel lido e um efeito *hlsl* aplica uma textura sobre cada um destes vértices. Para que o terreno tivesse bastante realismo, foi utilizado um efeito que mescla quatro texturas, baseando-se na altura do vértice. Para que este efeito rode corretamente, a placa de vídeo deve possuir suporte nativo ao *DirectX 9.0c*.

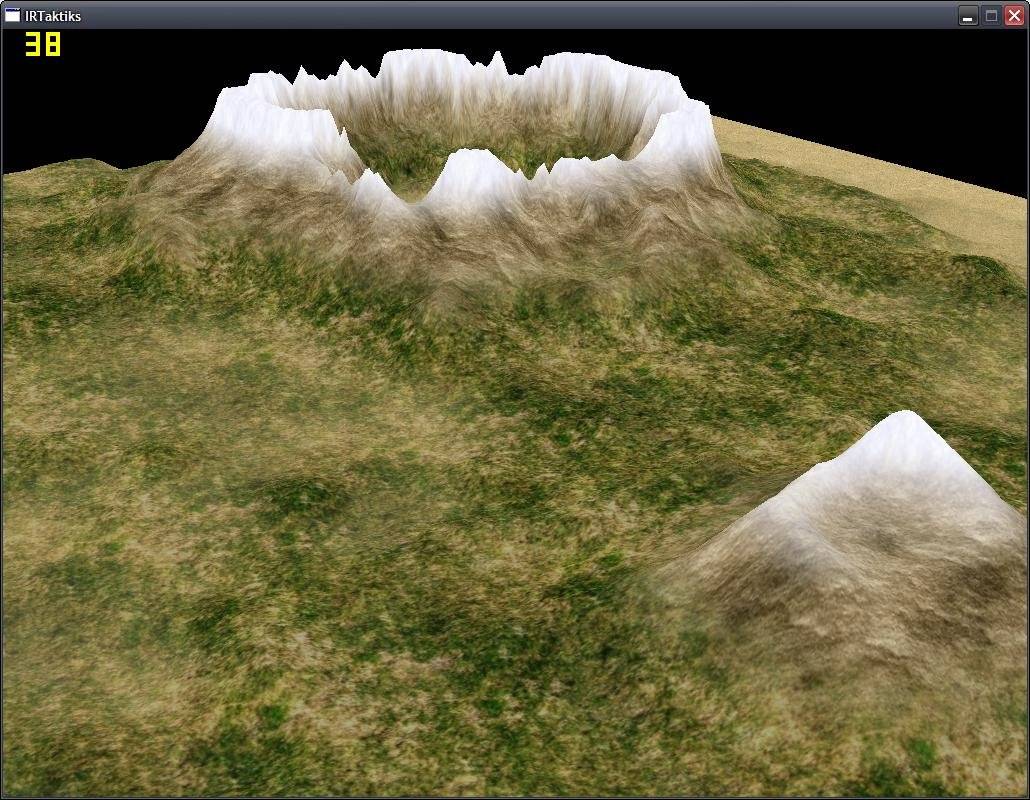


Figura - Mapa usando heightmap e efeito hlsl de mesclagem

##### Submódulo Area

Responsável pela criação e representação de uma área circular sobre o mapa. É utilizado para impor limites nas ações dos jogadores. Assim como o mapa, também se trata de um objeto 3D, porém com uma das dimensões igual a 1.

Apesar de uma área circular não ser 3D, isso se deu ao fato do *framework* *XNA* não desenhar nenhuma primitiva simples, retas, elipses e retângulos; além de triângulos, dada uma lista de pontos no espaço.

Da mesma maneira que o submódulo *sprite*, há um gerenciador para controlar o desenho das áreas, uma vez que pode existir mais de uma área sendo desenhada ao mesmo tempo. Este gerenciador também trabalha com uma fila de prioridades. Por decisão, todas as áreas são desenhadas após o desenho do mapa e antes de qualquer *sprite*, para que os personagens, textos e menus não sejam sobrepostos pelas áreas.

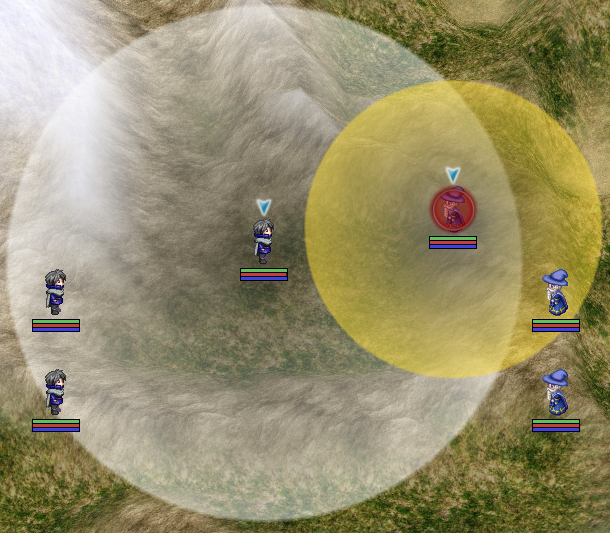


Figura - Exemplo de utilização de áreas

Da mesma maneira que o mapa, também utiliza um efeito *hlsl* para definir uma cor para a área, e aplicar uma suavização no serrilhados do desenho de sua geometria. O preenchimento da cor acontece de maneira interpolada das bordas para o centro, aplicando um efeito de degradê. Já a suavização foi feita utilizando um algoritmo de *anti-aliasing* com o canal *alpha* da cor da área.



Figura - Áreas com e sem suavização

##### Submódulo Animation

É o responsável por executar animações em uma determinada posição no espaço. As animações são um ou mais efeitos de partículas executados ao longo do tempo. Efeitos de partículas são diversos pontos (partículas) que de movimentam no espaço, baseadas em fórmulas matemáticas que regem seu movimento.

A cada ação executada no jogo, uma animação é invocada, de modo a responder ao usuário um *feedback* visual sobre a sua interação. Foram codificadas cerca de vinte animações diferentes, de modo que cada ação possua sua respectiva animação, permitindo a associação ação-animação por parte do jogador.

De modo a não comprometer a performance do jogo, cada animação é executada em uma *thread* separada. Com isso, é possível iniciar efeitos em tempos diferentes, uma vez que enquanto a animação espera antes de iniciar um novo efeito, o jogo continua sua execução normalmente.

Como cada animação pode ser composta por um ou mais efeitos de partículas diferentes, o controle de cada um destes efeitos exigiu o desenvolvimento de um gerenciador de partículas. Este gerenciador é responsável por desenhar todas as partículas existentes no jogo em um determinado momento. A animação somente invoca os efeitos na ordem correta e no período de tempo calculado, enquanto o próprio efeito se encarrega de solicitar o desenho de suas partículas ao gerenciador.

Cada uma das partículas de um determinado efeito possui suas características próprias, como posição, cor, tempo de vida atual e tempo total de vida. Quando o tempo de vida de uma partícula atinge o tempo total de vida, ela automaticamente se destrói, não sobrecarregando o jogo, calculando posições de partículas de efeitos que não existem.

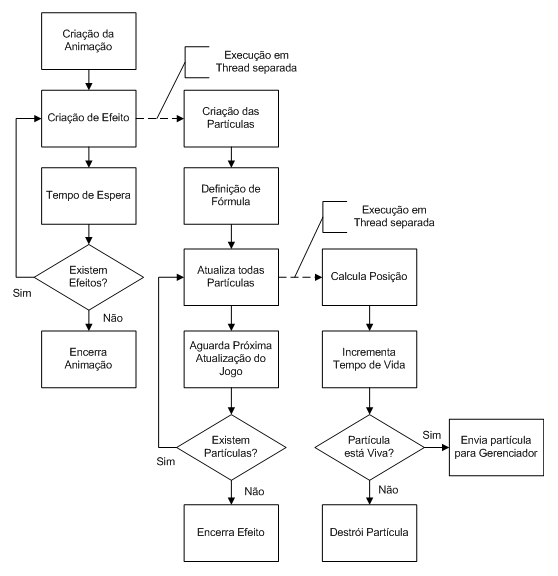


Figura - Fluxo de execução de uma animação

O gerenciador de partículas utiliza ainda, um efeito *hlsl* para determinar a cor de cada uma das partículas e sua escala. De acordo com o tempo de vida da partícula, a cor desta vai de tornando transparente, de modo a fazer uma transição suave durante o fim da animação. A escala é baseada na distância da partícula em relação ao observador da câmera. Quanto mais próxima a partícula estiver do observador, maior ela será, enquanto mais distante do observador, menor ela será; tornando o efeito o mais realístico possível.

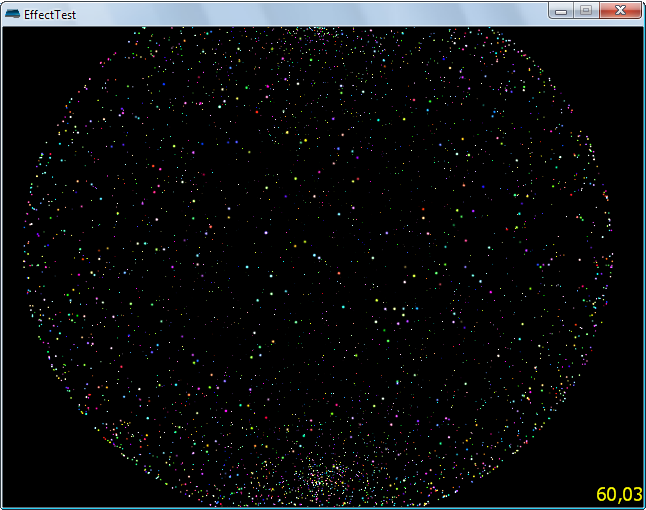


Figura - Exemplo de efeito de partículas

##### Submódulo Damage

Responsável por exibir informações sobre os efeitos das ações entre as unidades. Exibe informações como: quantidade de dano sofrido, quantidade de mana utilizada, quantidade de pontos de vida recuperados e atributos que tiveram seu valor aumentado.

A exibição destas informações ajuda na visualização do andamento do jogo e da quantização por parte do jogador, da força que sua unidade possui. Todas as informações benéficas à unidade são informadas na cor verde, enquanto as maléficas, na cor vermelha.



Figura - Exemplo de exibição de informações

#### Módulo Game

É o módulo central do jogo. Contém as classes que representam o jogo (*IRTGame*), os jogadores (*Player*) e as unidades (*Unit*), que são os personagens que o jogador controla. A classe que representa o jogo herda da classe *Game*, do *framework XNA*, possuindo funcionalidades de atualização e desenho. As classes que representam os jogadores e as unidades herdam da classe *GameComponent*. Com isso, ambas podem ser componentes da classe *IRTGame*.

Cada jogador possui nome, e uma lista de unidades que este irá controlar no jogo. Uma unidade possui diversas características, como nome, pontos de vida, pontos de mana, tempo de espera após a execução de uma ação, classe, atributos e ações. Os atributos foram separados em outro módulo, devido ao fato de serem utilizados em todos os cálculos do jogo. As ações, por sua vez, se subdividem em três tipos: Ataques, Habilidades e Itens. Cada unidade possui uma lista de cada um destes tipos de ações.

As ações, assim como os atributos, também foram implementados em outro módulo, responsável pela criação das ações, e pelo gerenciamento dos acontecimentos que a execução da ação provoca entre as unidades.

Ambos, jogadores e unidades, possuem menus, que informam o jogador real o estado atual de suas unidades, e propiciam a interação com o jogo, através da execução de ações. Todos os menus, assim como os atributos e ações, também foram implementados em um modulo separado, com seus respectivos gerenciadores.

Cada unidade possui indicadores, que auxiliam o jogador a ter uma visão geral sobre o jogo de maneira rápida e eficiente. Existem quatro indicadores, no total, para cada unidade dentro do jogo. Três delas se encarregam de mostrar, em forma de barras, a quantidade de pontos de vida, mana e tempo de espera após uma ação; enquanto a quarta se encarrega de indicar se a unidade está selecionada ou não. Quando o jogador seleciona uma unidade, ela pode executar ações; desde que o tempo de espera tenha de passado.

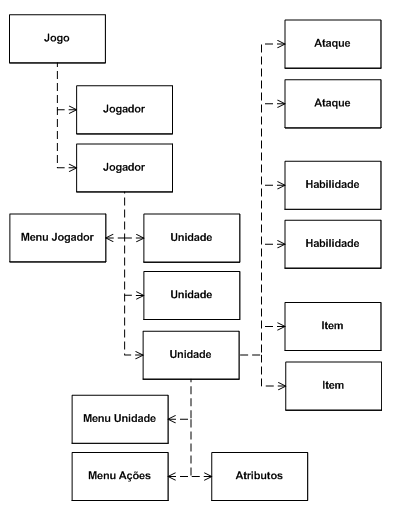


Figura - Estrutura organizacional do jogo

#### Módulo Logic

Responsável pela representação lógica das unidades, como orientação, atributos e classes.

A orientação indica em qual direção a unidade está olhando. Pode possuir os valores: *Norte*, *Sul*, *Leste* ou *Oeste*.

Os atributos determinam as características da unidade, e são divididos em dois tipos: fixos e calculados. Os atributos fixos são formados por valores entre 1 e 99 e não dependem de nenhum outro fator para determinar seu valor. Os atributos calculados, por sua vez, são regidos por fórmulas baseadas nos atributos fixos e sofrem influências da classe a qual a unidade pertence. Uma classe que beneficia um determinado atributo terá neste um valor maior, do que uma classe que não o bonifica; para os mesmos valores de atributos fixos.

Os atributos fixos de uma unidade são:

* Nível
* Força
* Agilidade
* Vitalidade
* Inteligência
* Destreza

Os atributos calculados são:

* Pontos de vida total
  + Quantidade máxima de pontos de vida de uma unidade.
* Pontos de mana total
  + Quantidade máxima de pontos de mana de uma unidade.
* Ataque
  + Utilizado no cálculo do dano de ataques físicos.
* Defesa
  + Utilizado no cálculo da defesa contra ataques físicos.
* Ataque mágico
  + Utilizado no cálculo do dano de ataques mágicos.
* Defesa mágica
  + Utilizado no cálculo da defesa contra ataques mágicos.
* Taxa de desvio
  + Porcentagem de desviar de um ataque físico.
* Taxa de acerto
  + Porcentagem de acertar um ataque físico.
* Alcance de Ataque
  + Tamanho da área onde um ataque pode ser desferido.
* Alcance de Habilidade
  + Tamanho da área onde uma habilidade pode ser solta.
* Alcance de Movimento
  + Tamanho da área para onde a unidade pode se mover.
* Tempo de espera
  + Tempo de espera após a execução de uma ação.

As classes definem as habilidades que a unidade poderá utilizar e influenciam nos valores dos atributos calculados. Cada atributo calculado possui fatores que, de acordo com a classe, podem aumentar ou diminuir o seu valor calculado. Com isso, unidades que possuam os mesmos valores de atributos físicos podem ter atributos calculados diferentes, aumentando a diversidade de estratégias que o jogador pode formar. Há seis classes disponíveis no jogo:

* Cavaleiro
  + Possui habilidades de ataques físicos
  + Bônus em ataques físicos e pontos de vida.
* Paladino
  + Possui habilidades de cura e sacrifício.
  + Bônus em pontos de vida, defesa física e mágica.
* Bruxo
  + Possui habilidades de ataques mágicos.
  + Bônus em ataques mágicos, destreza e pontos de mana.
* Sacerdote
  + Possui habilidades de proteção e cura.
  + Bônus em pontos de mana, defesa física e mágica e ataques mágicos.
* Assassino
  + Possui habilidades de ataque físico e críticos.
  + Bônus em ataques físicos e tempo de espera.
* Monge
  + Possui habilidades de ataque físico e mágico mesclados.
  + Bônus em ataques físicos e mágicos.

#### Módulo Screen

Módulo responsável por gerenciar os estados do jogo. Decidiu-se utilizar telas para representar cada um dos estados possíveis do jogo: *Título*, *Configuração*, *Jogo* e *Fim*. A qualquer momento, o jogo pode solicitar a mudança de estado. Quando isso ocorre, todos os componentes que pertencem ao estado anterior são removidos, ou seja, é como se o jogo recomeçasse. Dessa forma, ao mudar de estado, nenhum componente desnecessário é mantido, consumindo tempo de processamento e memória.

A tela de título é o estado inicial do jogo. Exibe o nome do jogo e as informações iniciais. Quando o jogo se inicia, o jogo muda seu estado para a tela de configuração, onde os jogadores e as unidades são criados. Após a configuração de todas as unidades, o jogo muda seu estado para a tela de jogo, onde o jogo, de fato, acontece. Quando um jogador vence a partida, o estado do jogo é novamente alterado, dessa vez para a tela de fim, onde o resultado do jogo é exibido.

Cada uma destas telas, é uma implementação da classe *IScreen*. Esta classe é responsável por controlar os componentes que o jogo possui em sua própria tela, e eliminar os componentes de outra tela. A utilização desta arquitetura fornece bastante agilidade ao jogo, pois não é necessário se preocupar com a reciclagem de componentes mortos, durante a execução do jogo.

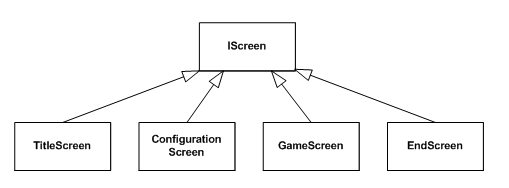


Figura - Generalização de telas

#### Módulo Debug

Responsável por exibir um indicador de performance e auxílio à detecção de toques.

A cada toque que acontece na mesa, uma animação de curta duração é executada, de forma a verificar a qualidade da calibração do software *TouchLib*. Em uma boa calibração, a animação acontece exatamente sob o dedo fez o toque.

O indicador de performance é exibido na parte inferior da tela. Este indicador calcula a taxa de *frames* por segundo (*fps*). Um valor aceitável deve possuir média em torno de 40, uma vez que abaixo deste valor, o jogo apresenta alguns travamentos.

#### Módulo Menu

É o módulo que cria, desenha e gerencia todos os menus do jogo. Cada jogador possui um menu, enquanto cada unidade possui dois menus. O do jogador exibe seu nome e a quantidade de unidades vivas que ele possui. O primeiro menu da unidade é responsável por exibir seu estado atual, enquanto o segundo se encarrega de exibir as ações que podem ser executadas.

O menu que exibe o estado da unidade é dividido em duas partes. A primeira exibe as informações mais importantes de maneira mais detalhada que os indicadores. Exibe o nome da unidade e a quantidade de pontos de vida, mana e do tempo de espera após a execução de uma ação; no formato de barras e números. Dessa forma, uma medição menos precisa pode ser obtida através dos indicadores, enquanto uma mais precisa, através deste menu.

A cor deste menu se altera conforme o estado dos pontos de vida da unidade. Quando ela possui mais de 50% dos pontos de vida totais, este menu é verde, indicando que tudo está bem. Quando o total de pontos está entre 10% e 50%; a cor se altera para amarelo, indicando atenção. Quando o total de pontos está abaixo dos 10%, ele se altera para vermelho, indicando que a unidade está prestes a morrer.

Ao tocar sobre este menu, a segunda parte é expandida. Esta exibe os atributos fixos e calculados da unidade atual. Em conjunto com a primeira parte, permite que o jogador tenha um “raio-x” sobre a unidade, tendo acesso às principais informações que a unidade possui.

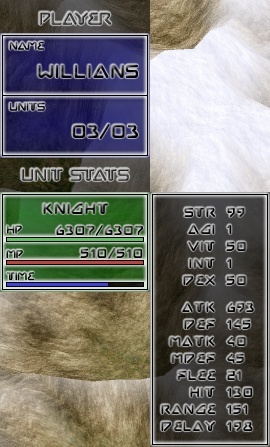


Figura - Menu do jogador e da unidade

O menu responsável por exibir as possíveis ações de uma unidade é controlado por um gerenciador, por ser mais complexo e possuir a necessidade de se comunicar diretamente com o módulo *Action*, responsável por controlar a execução das ações. Este menu apenas está ativo quando o tempo total de espera da unidade transcorreu completamente.

É composto por uma lista de itens que possuem uma lista de subitens, representados por instâncias da classe *ActionMenu* e *CommandMenu*, respectivamente. O menu possui quatro itens: *Mover*, *Atacar*, *Habilidades* e *Itens*. Os subitens estão ligados às ações da unidade que possui o menu, permitindo sua construção de forma dinâmica.

Durante o projeto do menu, decidiu-se que os subitens seriam especializações dos itens do menu. Isso facilitaria a codificação, uma vez que ambos devem ser capazes de avisar ao restante do módulo quando foram tocados, através de eventos.

Durante a construção do menu, os ataques, habilidades e itens da unidade são lidos, e com base neles, os subitens são criados. Cada subitem exibe ao lado do nome da ação, um parâmetro, que representa, no caso das habilidades, a quantidade de mana gasta na execução, ou, no caso dos itens, a quantidade de itens que a unidade possui. Ataques, por serem ilimitados, são representados com o número zero.

Cada subitem possui uma propriedade que verifica automaticamente se a unidade pode executar a ação relacionada ao menu. Caso ela não possa ser executada, o submenu não responderá aos toques do usuário. O item *Mover* não possui subitens, já que não existem tipos de movimentação diferentes.



Figura - Itens e seus respectivos subitens

Ações que têm como alvo a própria unidade não necessitam de informações extras para serem executadas. Porém, nem todas as ações têm como alvo o próprio jogador. Dois exemplos são: o item *Mover* e os subitens do item *Atacar*. O primeiro necessita que o jogo determine uma área pela qual a unidade pode se mover e movimente-a; enquanto os segundos necessitam de um alvo ou uma posição para desferir o ataque.

Com o intuito de auxiliar nas obtenções dessas informações, o módulo *Interaction* foi desenvolvido. Apesar de ser explicado em detalhes mais a frente, uma introdução se faz necessária para compreender a lógica de execução do menu. Este módulo é dividido em duas partes: a primeira é responsável por auxiliar na movimentação de uma unidade enquanto a segunda auxilia a na obtenção de alvos e posições.

Quando os subitens são tocados, verifica-se se este necessita de informações extras. Caso afirmativo, o respectivo item do módulo *Interaction* é iniciado e o subitem aguarda a obtenção das informações através de uma mensagem via evento. Quando a mensagem é recebida, as informações são passadas ao subitem e a ação pode enfim, ser executada.

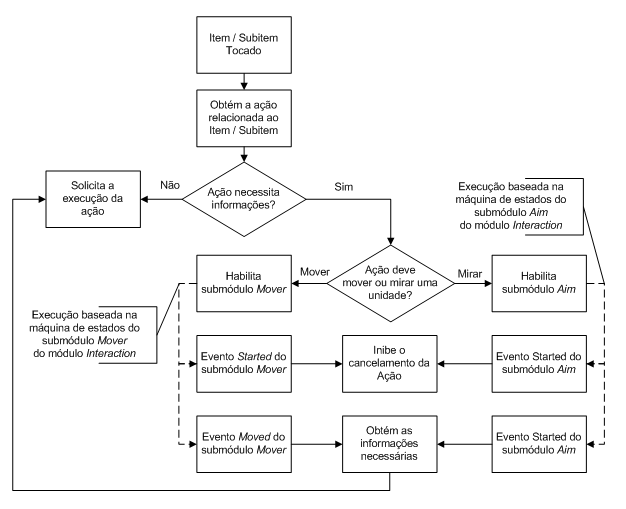


Figura - Fluxo de execução de uma ação através do menu

#### Módulo Interaction

É o módulo responsável por auxiliar o menu na execução de ações. Divide-se em dois submódulos: *Mover* e *Aim*, ambos baseados em uma máquina de estados. Comunica-se com o menu através de mensagens passadas via eventos.

O estado inicial dos submódulos aguarda a ativação por parte do menu para mudar de estado. Assim que um item ou subitem é tocado no menu, o menu ativa o respectivo submódulo e este fica aguardando a interação do jogador. Caso o jogador não interaja com o submódulo, ele é desativado pelo menu, retornando ao seu estado inicial.

##### Submódulo Mover

O movimento de uma unidade pelo jogador é realizado arrastando-a para a posição desejada, dentro do limite imposto pelo jogo. Após arrastar a unidade, o jogador deve escolher para qual direção a unidade deve olhar.

Ao ativar este submódulo, uma instância do submódulo *Area* do módulo *Drawable* é desenhada, com tamanho baseado no atributo calculado: *Alcance de Movimento*. A esfera desenhada indica a área pela qual a unidade pode se mover. Caso o jogador tente arrastar a unidade para fora desta área, a unidade permanecerá na última posição válida.



Figura - Unidade movendo-se dentro da área especificada

Ao tocar sobre a unidade, o evento *CursorDown* é disparado pelo módulo *Input*, servindo de gatilho para a mudança de estado. Ao mudar de estado, o identificador relacionado ao dedo é armazenado e o evento *Started* é disparado; avisando o menu que a unidade está em movimento, portanto, não é possível mais mudar a escolha.

Ao arrastar a unidade, o evento *CursorUpdate* é disparado repetidas vezes. Verifica-se se o identificador do dedo que está se movimentando é igual ao armazenado, para evitar que outros dedos se movendo sobre a mesa no mesmo momento influenciem na movimentação da unidade. Se forem iguais, verifica-se se a posição do dedo está dentro da área imposta e em caso afirmativo, a posição da unidade é atualizada. O estado atual do submódulo não muda, pois a unidade só deixa de se mover quando o dedo é retirado da mesa.

Ao retirar o dedo da mesa, o evento *CursorUp* é disparado. O identificador novamente é comparado para evitar que outros dedos que estejam sendo retirados, influenciem no movimento da unidade. Se o identificador coincidir, a unidade encerra o seu movimento e inicia-se a orientação da unidade.

A orientação é feita de maneira análoga à movimentação da unidade, repetindo o ciclo de eventos *CursorDown*, *CursorUpdate* e *CursorUp* anteriormente explicados. Quando o jogador retira o dedo da mesa, terminando de orientar a unidade, o evento *Moved* é disparado avisando o menu que a unidade terminou de se mover. Após a finalização do movimento da unidade, o submódulo é desativado, retornando-o ao seu estado inicial.

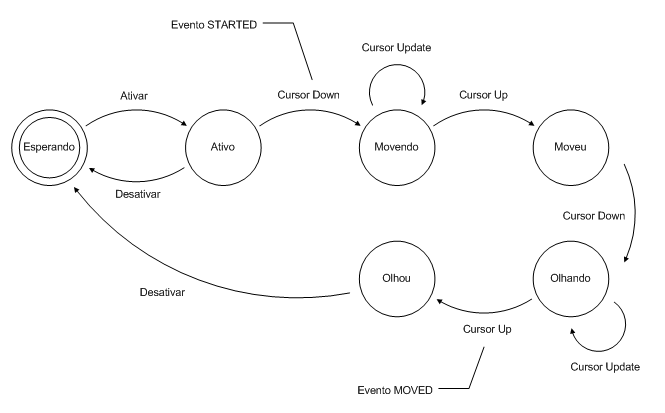


Figura - Máquina de estados do submódulo Mover

#### Submódulo Aim

A obtenção de um alvo para a execução de um determinado ataque, habilidade ou item é realizado com o auxílio de uma mira. O jogador arrasta a mira para a posição onde a ação deve ser executada, dentro do limite imposto pelo jogo. Com a posição definida, a ação é executada.

Assim como o submódulo *Mover*, o módulo *Aim* também faz uso de uma área para determinar o limite em que a mira pode se mover. Os atributos calculados usados para a definição do tamanho são: *Alcance de Ataque* e *Alcance de Habilidade*, usados para ataques e habilidades, respectivamente. Caso o jogador tente mover a mira para fora desta área, a mira mantém a última sua posição válida.

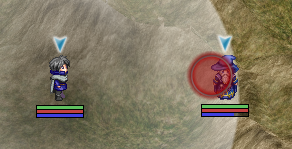


Figura - Mira sobre uma unidade inimiga

A ordem de execução dos eventos *CursorDown*, *CursorUpdate* e *CursorUp* é muito semelhante à utilizada no submódulo *Mover*, não necessitando de maiores aprofundamentos. De maneira análoga, quando o jogador começa a mover a mira, o evento *Started* é disparado, avisando o menu que o jogador iniciou a execução da ação e esta não pode mais ser cancelada ou trocada.

Durante a movimentação da mira, é verificado, cerca de quatro vezes por segundo, se a mira está posicionada sobre alguma unidade. Se sim, e a unidade for uma unidade aliada, a mira muda para a cor verde. Se a unidade for inimiga, a cor adotada foi a vermelha. Quando a mira está sobre nenhuma unidade, ou seja, não está mirando em ninguém, sua cor é cinza.

Toda unidade que está sendo mirada é armazenada, de forma a evitar um processamento desnecessário para descobrir qual unidade está na última posição válida de mira. Quando a mira não está mirando sobre nenhuma unidade, a unidade armazenada é nula.

Quando a mira é liberada pelo jogador, retirando o dedo da mesa, o evento *Aimed* é disparado, informando o menu, sobre qual unidade que a mira se encontra e sua última posição válida. De posse destas informações juntamente com a unidade invocadora, é possível executar qualquer ação.

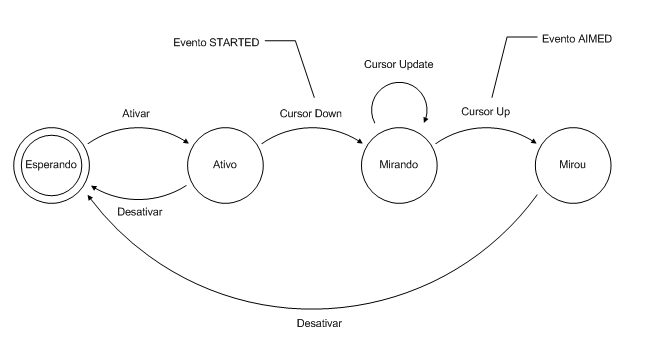


Figura - Máquina de estados do submódulo Aim

#### Módulo Action

É o módulo responsável por criar, gerenciar e executar as ações das unidades em jogo. Este módulo possui três gerenciadores, responsáveis por gerenciar cada tipo de ação existente, no caso, ataques, habilidades e itens.

Dentro do jogo, quaisquer ações de uma unidade herdam de uma mesma classe, que define quais as características comuns entre elas, como por exemplo, nome ou quantidade de mana gasta para sua execução.

Quando o menu de uma unidade é exibido, cada item ou subitem que possui ligação com uma ação, verifica através da propriedade *Enabled* da ação, se esta pode ser executada. Quando a ação não pode ser executada, o item / subitem se desabilita automaticamente, evitando que o jogador consiga executar algo que sua unidade não tenha pontos de mana suficientes.

Devido ao fato desta propriedade ser abstrata, cada tipo de ação pode implementá-la da melhor maneira. Um ataque não gasta pontos de mana e nem possui restrições de uso, tornando o valor destra propriedade sempre verdadeira. Já uma habilidade, que consome pontos de mana da unidade, verifica se a quantidade de mana que vai ser utilizada é menor ou igual à quantidade que a unidade possui. Um item, por sua vez, apenas verifica se a quantidade em estoque que a unidade possui é maior que zero.

A quantidade de mana ou de itens em estoque é armazenada em uma propriedade chamada *Attribute*. Trata-se de um valor numérico que pode ser utilizada para armazenar qualquer atributo relevante ao tipo de ação que será executada.

Durante o projeto deste módulo, foi decidido que as execuções de todas as ações de um determinado tipo deveriam estar agrupadas dentro de seu respectivo gerenciador. Para que isso fosse possível, foram criados *delegates*, que de maneira análoga, se comportam como propriedades que armazenam métodos.

Quando uma ação é criada, além de informações básicas, é passada a assinatura de um método, que será armazenado neste *delegate*. Dessa forma, quando o método *Execute* da ação for executado, este, por sua vez, chama o método armazenado que foi escolhido durante a criação da ação.

Cada gerenciador de ações utiliza o *design pattern Factory*, sendo ele o responsável pela criação das ações deste tipo. Para cada classe de unidade, uma lista de ações diferentes é criada, diversificando o jogo, e ampliando as possibilidades de táticas. Quando o gerenciador cria a ação, ele informa qual o método de sua classe que será executado quando a ação for executada. Isso permite que duas ações diferentes, executem o mesmo método, simplificando a codificação e manutenção.

*{imagem : diagrama execução}*

O *delegate* criado exige que os métodos que ele armazenará possuam quatro atributos: a ação que está sendo executada, a unidade que invocou a ação, a unidade alvo da ação e a última posição válida da mira. Estas restrições foram feitas, pois garante que qualquer ação possa ser executada de posse destes atributos. O módulo *Interaction* foi criado com esta finalidade, de maneira a garantir a obtenção de todas as informações necessárias para qualquer ação possível, antes de qualquer execução.

Toda execução de uma ação segue um padrão. Primeiramente, aplicam-se os efeitos da ação sobre a unidade que a invocou, como por exemplo: diminuição dos pontos de mana, execução de animações e exibição de informações sobre estes efeitos.

Em seguida, caso exista uma unidade alvo, as conseqüências da ação são aplicadas. O cálculo dos pontos de dano ou de recuperação é realizado, a execução da respectiva animação é iniciada e as informações são exibidas. Caso a unidade seja nula, isso indica que o jogador utilizou a ação sobre o terreno, sem objetivo. Neste caso, apenas a animação é executada, e nenhum cálculo é processado.

Quando as conseqüências da ação devem ser aplicadas à unidade alvo durante um determinado período de tempo, uma nova *thread* é criada para a aplicação destes efeitos. Desta forma o jogador pode controlar outras unidades e efetuar outras ações, enquanto a unidade permanece sob o efeito da ação.

*{imagem : seqüência de execução}*

Cada ação possui uma propriedade que indica se ela pode ser executada em outra unidade ou é apenas de uso próprio. É esta propriedade que define que a ação necessitará ou não de informações extras quando o menu ligado à ela for tocado.

# RESULTADOS

Descrever da melhor maneira possível como o jogo roda de todos os pontos de vista: desempenho, “jogabilidade”, praticidade ou efetividade da montagem mecânica/física, como cada componente (webcam, fonte, leds, PCs, projetores) contribuiu ou poderia ser melhorado. Incluir fotos que documentem bem as diversas funcionalidades do jogo em execução na mesa de verdade.

## Trabalhos Futuros

A fazer...

## Conclusão

Revisar os objetivos, listá-los e dizer quanto foram atendidos. Explicar por que alguns eventualmetne não foram atendidos.;

# Referências bibliográficas

BIMBER, Oliver; RASKAR, Ramesh. **Spatial Augmente Reality**: merging real and virtual worlds. Wellesley, MA: A K Peters, 1997.

KIRNER, Cláudio; TORI, Romero**. Realidade Virtual**: Conceitos e Tendências. São Paulo: Editora Mania de Livro, 2004.

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO. **Augmented Reality**: Realidade Aumentada e Visão Computacional.  Disponível em: <<http://www.lamce.ufrj.br/grva/realidade_aumentada/>> Acesso em: 15 de maio de 2006.

AZUMA, Ronal T. A survey of Augmented Reality. **In: Presence: Teleoperators and Virtual Environments**. v. 6. p 355-385. ago 1997

# APÊNDICES

## Fórmulas dos Atributos Calculados

As fórmulas dos atributos calculados utilizam como parâmetros os atributos fixos de uma unidade e um fator relacionado à sua classe, que quantificará a importância deste atributo para a classe. A legenda utilizada na representação das fórmulas encontra-se a seguir:

Valor do atributo relacionado ao nível.

Valor do atributo relacionado à força.

Valor do atributo relacionado à agilidade.

Valor do atributo relacionado à vitalidade.

Valor do atributo relacionado à inteligência.

Valor do atributo relacionado à destreza.

Valor do fator relacionado à classe.

Baseada nesta legenda, as fórmulas dos atributos calculados são:

* Pontos de vida total - (*maxlife*)
* Pontos de mana total - (*maxmana*)
* Ataque - (*atk*)
* Defesa - (*def*)
* Ataque mágico - (*matk*)
* Defesa mágica - (*mdef*)
* Taxa de desvio - (*flee*)
* Taxa de acerto - (*hit*)
* Alcance de Ataque - (*arange*)
* Alcance de Habilidade - (*hrange*)
* Alcance de Movimento - (*mrange*)
* Tempo de espera - (*time*)

## Tabela de Fatores

São os fatores relacionados às classes, utilizados nas fórmulas dos atributos calculados para quantificar a importância deste atributo para uma determinada classe. Isso permite que um Paladino tenha mais pontos de vida que um Bruxo, enquanto este tenha mais ataque mágico que o Paladino, embora ambos possuam os mesmos valores de atributos calculados.

Estes parâmetros, como verificado nas fórmulas, influenciam apenas a quantidade máxima de pontos de vida e mana, ataque e defesa, e ataque e defesa mágicos.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Pontos de Vida | Pontos de Mana | Ataque | Defesa | Ataque Mágico | Defesa Mágica |
| Cavaleiro | 0.85 | 0.50 | 1.00 | 0.60 | 0.40 | 0.40 |
| Paladino | 1.00 | 0.75 | 0.70 | 1.00 | 0.85 | 1.00 |
| Bruxo | 0.55 | 0.90 | 0.50 | 0.80 | 1.00 | 0.85 |
| Sacerdote | 0.60 | 1.00 | 0.60 | 0.85 | 0.90 | 0.95 |
| Assassino | 0.75 | 0.55 | 0.90 | 0.55 | 0.50 | 0.55 |
| Monge | 0.70 | 0.65 | 0.80 | 0.65 | 0.60 | 0.65 |

Tabela – Valores dos fatores, em relação às classes do jogo e os atributos calculados.

## Fórmulas das Ações

Existem diversas ações dentro do jogo, cada uma respectiva a uma classe de unidades. Cada unidade possui três tipos de ação, Ataque, Habilidade e Item. No jogo, todas as classes possuem as mesmas ações de ataque e item; deixando a diferenciação entre as classes a cargo das habilidades.

Todos os ataques e algumas habilidades podem falhar, ou seja, a unidade pode errar a sua execução, ou o alvo pode desviar. Esta chance é calculada baseada nos atributos calculados *Taxa de Acerto* (*Hit*) e *Taxa de Desvio* (*Flee*) das unidades invocadora e alvo. A fórmula utilizada segue[[13]](#footnote-14):

Outra característica no cálculo das ações é a aleatoriedade nos valores calculados. O resultado das fórmulas das ações é escolhido aleatoriamente dentro de uma faixa de 10% pra mais ou pra menos, do valor de base calculado.

### Ataque

Utilizado para atacar o inimigo, sem consumo de pontos de mana. Possui dois tipos: Ataque curto e longo. O ataque curto (*short*) é baseado na força da unidade e possui área de ação fixa igual a 100. Já no ataque longo (*long*), o dano é baseado na destreza da unidade, e o alcance é dado pelo atributo calculado *Alcance de Ataque*. A ambos os ataques, são aplicadas a fórmula de aleatoriedade.

* Ataque curto - (*short*)
* Ataque Longo - (*long*)

### Itens

Ao iniciar o jogo, cada unidade inicia com 10 poções (*potion*), 5 éteres (*ether*) e 1 elixir. As poções são itens responsáveis por recuperar pontos de vida perdidos e os éteres os pontos de mana. Ambos podem ser usados em outras unidades. Já o elixir recupera todos os pontos de vida e mana de uma unidade, mas é um item de uso próprio, não podendo ser usado em outra unidade.

As poções e os éteres recuperam valores fixos de vida e mana. Uma poção recupera 500 pontos de vida e um éter 350 pontos de mana. A todos os itens não são aplicados as taxas de acerto e de aleatoriedade.

### Habilidades

Cada classe de unidade possui três habilidades sendo possível que duas classes possuam uma mesma habilidade. No total foram implementadas 17 habilidades diferentes, de modo a exemplificar o compartilhamento de habilidades. Cada habilidade com suas características e peculiaridades estão a seguir.

#### Stealth

Dobra a destreza da unidade invocadora por 60 segundos. É utilizada pela classe de assassinos, não pode ser usada em outra unidade e consome 50 pontos de mana. Nunca erra e não sofre aleatoriedade em seu valor.

#### Ambush

Inflige dano equivalente a 500% de um ataque curto a uma unidade alvo; porém a taxa de acerto final calculada cai pela metade. É utilizada pela classe de assassinos e consome 65 pontos de mana. Pode errar e sofre aleatoriedade no valor final.

#### Curse

Mata instantaneamente a unidade alvo. A porcentagem de sucesso é igual a 5% da taxa de acerto final calculada. É utilizada pela classe de assassinos e consome 110 pontos de mana.

#### Quick

Dobra a agilidade da unidade invocadora por 30 segundos. É utilizada pela classe de cavaleiros, não pode ser usada em outra unidade e consome 40 pontos de mana. Nunca erra e não sofre aleatoriedade em seu valor.

#### Impact

Inflige dano equivalente a 300% de um ataque curto a uma unidade alvo. Possui ainda 15% da taxa de acerto final calculada de dobrar o dano, totalizando 600%. É utilizada pela classe de cavaleiros e consome 70 pontos de mana. Pode errar e sofre aleatoriedade em seu valor.

#### Revenge

Inflige dano equivalente a 50% de um ataque curto a uma unidade alvo. Porém o alvo desta habilidade são os pontos de mana. É utilizada pela classe de cavaleiros e consome 75 pontos de mana. Pode errar e sofre aleatoriedade em seu valor.

#### Warcry

Dobra a força da unidade invocadora por 30 segundos. É utilizada pela classe de monges, não pode ser utilizada em outra unidade e consome 45 pontos de mana. Nunca erra e não sofre aleatoriedade em seu valor.

#### Insane

Aplica cinco ataques consecutivos à unidade alvo. O dano de cada ataque é calculado com base no ataque mágico da unidade invocadora. É utilizada pela classe de monges e consome 90 pontos de mana. Cada ataque é independente, pode errar e sofre aleatoriedade em seu valor.

#### Reject

Aplica grande quantidade de dano baseada no ataque normal, mágico, nível, força, inteligência e pontos de mana. É utilizada pela classe de monges e consome todos os pontos de mana restantes da unidade invocadora. Pode errar e sofre aleatoriedade em seu valor.

#### Might

Dobra a vitalidade da unidade invocadora por 60 segundos. É utilizada pela classe de paladinos, não pode ser utilizada em outra unidade e consome 60 pontos de mana. Nunca erra e não sofre aleatoriedade em seu valor.

#### Heal

Recupera pontos de vida da unidade alvo. É utilizada pela classe de paladinos e sacerdotes, sendo a habilidade compartilhada. Consome 85 pontos de mana, nunca erra, mas sofre aleatoriedade em seu valor.

#### Unseal

Reduz 50% dos pontos de vida da unidade alvo ao custo de 50% dos seus pontos de vida. É utilizada pela classe de paladinos e é necessário que a unidade invocadora possua mais de 500 pontos de mana, apesar de não gastá-los. Nunca erra e não sofre aleatoriedade em seu valor.

#### Barrier

Aumenta a defesa física e mágica de uma unidade alvo por 60 segundos. É utilizada pela classe de sacerdotes e consome 80 pontos de mana. Nunca erra e não sofre aleatoriedade em seu valor.

#### Holy

Inflige dano à unidade alvo, baseado no ataque mágico da unidade invocadora. É utilizada pela classe de sacerdotes e consome 180 pontos de mana. Nunca erra, mas sofre aleatoriedade em seu valor.

#### Drain

Rouba pontos de mana da unidade alvo. A quantidade é baseada na inteligência das unidades invocadora e alvo. É utilizada pela classe de bruxos e consome 50 pontos de mana. Nunca erra, mas sofre aleatoriedade em seu valor.

#### Flame

Inflige dano à unidade alvo, baseado no ataque mágico da unidade invocadora. É utilizada pela classe de bruxos e consome 210 pontos de mana. Nunca erra, mas sofre aleatoriedade em seu valor.

#### Frost

Aplica 10 ataques à unidade alvo, baseado no ataque mágico da unidade invocadora. É utilizado pela classe de bruxos e consome 250 pontos de mana. Nunca erra e a aleatoriedade de cada ataque é de 50% pra mais ou pra menos.

1. Divisão de pesquisas e treinamento avançado de Ciência da Computação e Matemática da Universidade de Nova Iorque, Estados Unidos da América. [↑](#footnote-ref-2)
2. Imagens únicas, geradas através de algoritmo usado na visão computacional para obter informações sobre o mundo real. [↑](#footnote-ref-3)
3. Stratos: Exército, Ago: Liderança. [↑](#footnote-ref-4)
4. Agência espacial norte-americana [↑](#footnote-ref-5)
5. http://www.newhorizons.org/strategies/literacy/kestrel.htm [↑](#footnote-ref-6)
6. http://nuigroup.com [↑](#footnote-ref-7)
7. http://whitenoiseaudio.com [↑](#footnote-ref-8)
8. Operações executadas pelo processador de uma placa de vídeo, sobre a cor de cada píxel de uma imagem, com o objetivo de aplicar texturas, efeitos, sombras e explosões. [↑](#footnote-ref-9)
9. Operações matemáticas executadas pelo processador de uma placa de vídeo, sobre os vértices de um objeto 3D, com o objetivo de adicionar efeitos e alterar sua aparência. [↑](#footnote-ref-10)
10. Coleções de *Application Programming Interfaces* (API) que auxiliam na execução de tarefas multimídia, como jogos, áudio e vídeos; na plataforma *Microsoft*. [↑](#footnote-ref-11)
11. Módulo do *framework* XNA responsável por carregar arquivos externos, como texturas, sons, modelos 3D, entre outros, para dentro do jogo. [↑](#footnote-ref-12)
12. http://www.e-onsoftware.com/products/vue/vue\_6\_xstream/ [↑](#footnote-ref-13)
13. Para a representação das unidades invocadoras (*caster*) e das unidades alvo (*target*) nas fórmulas, serão utilizados os prefixos “*c*” e “*t*”, respectivamente. [↑](#footnote-ref-14)