

CENTRO UNIVERSITÁRIO SENAC

**Willians Schallemberger Schneider
Nilson Calazans Dias Filho
Luis Henrique Menezes Mauruto**

IRTaktiks – Jogo de RPG Tático para Interfaces Multi-toque

São Paulo
2008

**WILLIANS SCHALLEMBERGER SCHNEIDER
NILSON CALAZANS DIAS FILHO
LUIS HENRIQUE MENEZES MAURUTO**

**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

Schneider, Willians Schallemberger
Filho, Nilson Calazans Dias
Mauruto, Luis Henrique Menezes

IRTaktiks – Jogo de RPG Tático para Interfaces Multi-toque / Luis Henrique
Menezes Mauruto, Nilson Calazans Dias Filho e Willians Schallemberger Schneider – São
Paulo, 2008.

109 fls, il. color.

Orientador: Prof. Fábio Roberto de Miranda

Trabalho de Conclusão de Curso – Centro Universitário Senac – Campus Santo
Amaro – São Paulo, 2008.

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

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

Este trabalho apresenta o desenvolvimento do jogo *IR Taktiks*, *RPG* tático para interfaces multi-toque baseadas na arquitetura de J.Y. Han: Reflexão Total Interna Frustrada da Luz (FTIR) e com a utilização de dispositivos acessíveis.

Palavras-Chave: Jogos Eletrônicos, Sistemas Projetor-Câmera, Computação Gráfica, Multi-Toque, Interação Humano-Computador.

ABSTRACT

This work presents the development of the game *IRTaktiks*, tactical RPG for multi-touch interfaces; based on JY Han's architecture: Frustrated Total Internal Reflection (FTIR) and with the use of low-cost devices.

Palavras-Chave: Electronics Games, Projector-Camera Systems, Computer Graphics, Multitouch, Human-Computer Interaction

SUMÁRIO

1.	IRTAKTIKS	14
1.1.	Introdução.....	14
1.2.	Interação Multi-toque	14
1.2.1.	História	15
1.3.	Objetivo	16
2.	BASES TEÓRICAS E TECNOLOGIAS EMPREGADAS	19
2.1.	Dispositivos Multi-toques	19
2.1.1.	Microsoft Surface.....	19
2.1.2.	ReacTable.....	21
2.1.3.	iPhone	22
2.2.	Jogos e Interatividade.....	24
2.2.1.	Jogos de Estratégia	25
2.2.2.	Jogos de RPG	25
2.2.3.	Jogos de RPG Eletrônicos.....	27
2.2.4.	Realidade Virtual	31
2.3.	Implementações de Superfícies Multi-toque.....	32
2.3.1.	Iluminação Difusa (Diffused Illumination).....	32
2.3.2.	Reflexão Total Interna Frustrada da Luz (FTIR).....	34
2.4.	Tecnologias Utilizadas	36
2.4.1.	OSC.....	37
2.4.2.	TUIO	37
2.4.3.	ReacTIVision	38
2.4.4.	Touchlib	39
2.4.5.	Microsoft XNA.....	42
3.	PROJETO.....	44
3.1.	Concepção	44
3.2.	Adequação da Mesa	47
3.2.1.	Estrutura	47
3.2.2.	Visão Computacional	51
3.2.3.	Testes e Dificuldades Encontradas	55
3.3.	Jogo.....	58
3.3.1.	Protótipo	59
3.3.2.	Versão Final.....	60
4.	RESULTADOS.....	93
4.1.	Trabalhos Futuros.....	94
4.1.1.	Dinamismo em Visão e Projeção	94
4.1.2.	Processos Interativos em Superfícies Interativas	95
4.2.	Conclusão.....	96
5.	REFERÊNCIAS BIBLIOGRÁFICAS	97
6.	APÊNDICES.....	101
6.1.	Fórmulas dos Atributos Calculados	101
6.2.	Tabela de Fatores	102
6.3.	Fórmulas das Ações	103
6.3.1.	Ataque	103
6.3.2.	Itens	104
6.3.3.	Habilidades	104

7.	ANEXOS.....	109
7.1.	Arquivo de Configuração Touchlib	109

LISTA DE FIGURAS

Figura 1 - Lemur Input Device	16
Figura 2 - IRTaktiks	17
Figura 3 - Visão geral do sistema	18
Figura 4 - Microsoft Surface	19
Figura 5 - Estrutura interna da Microsoft Surface	20
Figura 6 - ReacTable	21
Figura 7 - iPhone	22
Figura 8 - Estrutura da tela multi-toque do iPhone	23
Figura 9 - Gesture de redimensionamento no iPhone	24
Figura 10 - Sistema GPS no iPhone	24
Figura 11 - Exemplo de campanha em andamento	27
Figura 12 - Zork (1979)	28
Figura 13 - Final Fantasy - Square (1987)	28
Figura 14 - Final Fantasy VII - Squaresoft (1997)	29
Figura 15 - World of Warcraft - Blizzard (2004)	29
Figura 16 - Cenário tridimensional isométrico	30
Figura 17 - Personagem e sua área de atuação	30
Figura 18 - Personagem efetuando um ataque	31
Figura 19 - Rear Illumination	33
Figura 20 - Exemplo da detecção de toques utilizando Rear Illumination	33
Figura 21 - Front Illumination.....	34
Figura 22 - Exemplo da detecção de toques utilizando Front Illumination	34
Figura 23 - Exemplos de reflexão com refração e reflexão total da luz	35
Figura 24 - Reflexão total interna frustrada da luz.....	35
Figura 25 - Exemplo da detecção de toques utilizando FTIR	36
Figura 26 - ReacTIVision reconhecendo um fiducial	38
Figura 27 - Marcadores fiduciais	39
Figura 28 - Exemplo de interpolação no cálculo da posição do toque	41
Figura 29 - Demonstração do software de calibração	41
Figura 30 - Representação das camadas do framework XNA.....	43
Figura 31 - Elementos do jogo	44
Figura 32 - Arquitetura do sistema	46
Figura 33 - Mesa multi-toque utilizada no projeto.....	47
Figura 34 - Contraste do toque na mesa antes da reestruturação	48
Figura 35 - Parte elétrica após a reestruturação	49
Figura 36 - Representação do circuito elétrico da mesa	49
Figura 37 - Contraste do toque na mesa após reestruturação	50
Figura 38 - Placa de circuito impresso com os resistores de 56Ω e $5,6\Omega$	51
Figura 39 - Conector com LED	51
Figura 40 - Microsoft LifeCam VX 6000	52
Figura 41 - Filtro inibidor da luz infravermelha	52
Figura 42 - Toque com e sem o filtro inibidor da luz infravermelha	52
Figura 43 - Microsoft LifeCam VX 6000 desmontada e filtro inibidor da luz visível ...	53
Figura 44 - Toque sem e com o filtro inibidor da luz visível	53
Figura 45 - Sistema de projeção	54
Figura 46 - Papel vegetal como anteparo de projeção	54

Figura 47 - Comparativo do toque antes e depois da reestruturação.....	55
Figura 48 - Copo e circuito com LED usado na iluminação do fiducial.....	56
Figura 49 - FTIR utilizando anteparo difusor para projeção	56
Figura 50 - Fiduciais sobre papel vegetal e saco plástico	57
Figura 51 - Toque sobre papel vegetal e saco plástico	58
Figura 52 - Protótipo.....	59
Figura 53 - Versão final	61
Figura 54 - Arquitetura da versão final	62
Figura 55 - Relação entre os módulos Listener e Input.....	63
Figura 56 - Exemplo de eventos do módulo Input	64
Figura 57 - Exemplo de utilização de efeitos hsl	66
Figura 58 - Exemplo de utilização de fonte XML	66
Figura 59 - Exemplo de fonte-textura	67
Figura 60 - Exemplo de uso de fonte-textura	67
Figura 61 - Representação da área visível da cena	68
Figura 62 - Exemplo de sobreposição de sprites	69
Figura 63 - Software Vue xStream 6	70
Figura 64 - Mapa utilizando arquivo de geometria (40Mb)	71
Figura 65 - Mapa usando heightmap e efeito hsl de mesclagem	72
Figura 66 - Exemplo de utilização de áreas	73
Figura 67 - Áreas com e sem suavização	73
Figura 68 - Fluxo de execução de uma animação.....	75
Figura 69 - Exemplo de efeito de partículas	76
Figura 70 - Exemplo de exibição de informações.....	76
Figura 71 - Estrutura organizacional do jogo.....	78
Figura 72 - Tela de título.....	80
Figura 73 - Generalização de telas	81
Figura 74 - Menu do jogador e da unidade	82
Figura 75 - Itens e seus respectivos subitens	84
Figura 76 - Fluxo de execução de uma ação através do menu.....	85
Figura 77 - Unidade movendo-se dentro da área especificada	86
Figura 78 - Máquina de estados do submódulo Mover	87
Figura 79 - Mira sobre uma unidade inimiga	88
Figura 80 - Máquina de estados do submódulo Aim	89
Figura 81 - Diagrama de criação e execução de ações	91
Figura 82 - Diagrama de execução de ações.....	92
Figura 83 - Interação multi-toque em superfícies verticais	95

LISTA DE TABELAS

Tabela 1 - Atributos calculados	79
Tabela 2 - Classes disponíveis e características	80
Tabela 3 - Assinatura do <i>delegate</i> utilizado	90
Tabela 4 - Valores dos fatores, em relação às classes e atributos calculados.	103

1. IRTAKTIKS

1.1. 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.

1.2. 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, 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.

1.2.1. 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.

¹ Divisão de pesquisas e treinamento avançado de Ciência da Computação e Matemática da Universidade de Nova Iorque, Estados Unidos da América.

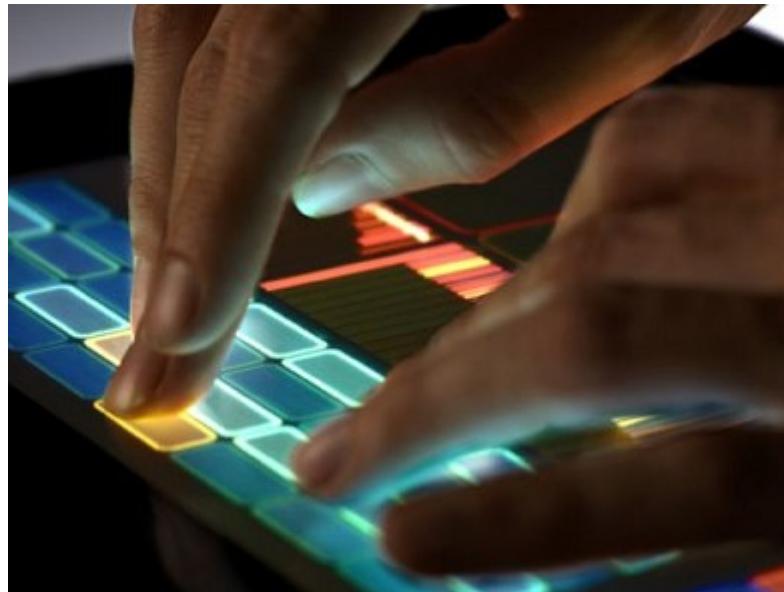


Figura 1 - Lemur Input Device

1.3. 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.



Figura 2 - 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 benefício 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 Jeffesson 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.

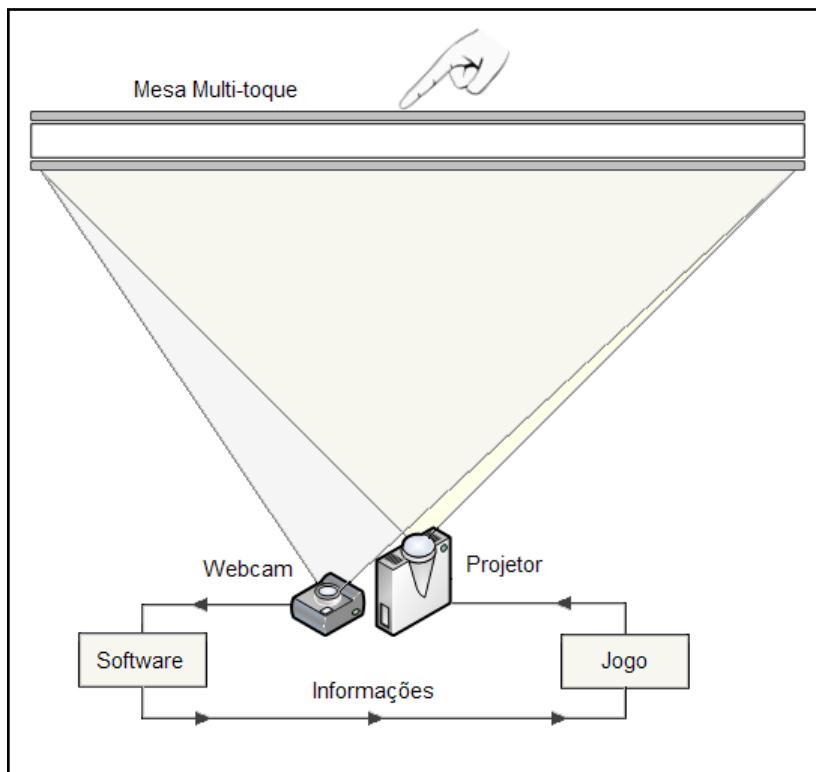


Figura 3 - Visão geral do sistema

Este trabalho está dividido em três capítulos principais. No capítulo a seguir, apresentaremos as bases teóricas utilizadas neste trabalho. Nele discutiremos sobre os dispositivos multi-toques atuais e suas principais características, os tipos de jogos relacionados a este projeto e sobre as ferramentas utilizadas em seu desenvolvimento. O terceiro capítulo descreve o projeto, sua concepção e desenvolvimento, além de problemas e soluções encontradas. O quarto capítulo conclui o trabalho, apresentando os resultados obtidos e possíveis trabalhos futuros.

2. BASES TEÓRICAS E TECNOLOGIAS EMPREGADAS

Neste capítulo serão apresentadas as bases teóricas e as tecnologias empregadas no desenvolvimento do projeto.

2.1. Dispositivos Multi-toques

Atualmente, a quantidade de dispositivos multi-toque não para de crescer. Novas tecnologias e pesquisas são notícias todos os dias, sendo cotada como a forma de interação do futuro. A seguir exibiremos os principais dispositivos multi-toques existentes e suas características, que serviram de inspiração para o desenvolvimento deste trabalho.

2.1.1. 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.



Figura 4 - Microsoft Surface

É constituída por uma superfície de acrílico de 30 polegadas disposta em forma de mesa. Suas dimensões são: 56 cm de altura, 53 cm de profundidade, 106 cm de largura. Em sua máxima configuração, embarca um computador com um processador *Intel Core Quad Xeon "WoodCrest"* de 2.66GHz, 4GB de memória RAM DD2 de 1066MHz e um HD de 1TB, de 7200rpm, sob o sistema operacional *Windows Vista*. Possui conectividade *Ethernet* 10/100/1000, *wireless* 802.11 b/g e *Bluetooth* 2.0, usando-os na comunicação com os aparelhos sobre sua superfície.

A visão computacional é formada por um canhão de leds infravermelhos direcionados sob a superfície de acrílico. A detecção é feita através do reconhecimento da diferença de iluminação entre o acrílico e os aparelhos sobre este. Quatro câmeras com resolução de 1920 x 960 obtêm as imagens e enviam ao computador embarcado, responsável por processar as informações e enviar o resultado ao projetor, que mostra a imagem sob a superfície de acrílico. Para que a imagem seja visível, o acrílico possui um material difusor colado à sua superfície.

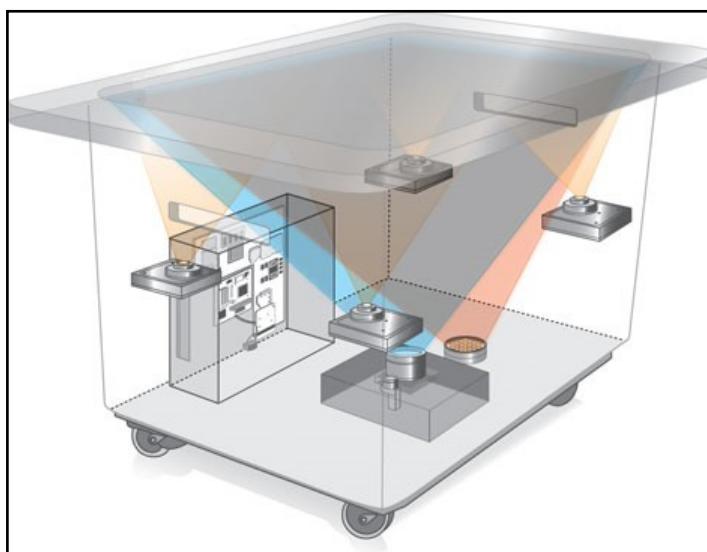


Figura 5 - Estrutura interna da Microsoft Surface

Em fase desenvolvimento pela *Microsoft*, existe um *SDK (Software Development Kit)* próprio para o desenvolvimento de aplicações multi-toque. Este *SDK* ainda é confidencial e de uso interno da equipe de desenvolvimento da *Microsoft Surface*.

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 fabricantes de *notebooks* passarão a oferecer produtos com esta funcionalidade.

2.1.2. 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 multiusuário 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 fiduciais² 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 os objetos sobre a mesa. Foi utilizado em shows e apresentações da cantora islandesa Björk, no *Coachella Festival*, em 2007 na Califórnia.



Figura 6 - *ReacTable*

Trata-se de uma mesa redonda e translúcida, utilizada em uma 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.

Os diversos tipos de objetos tangíveis representam diferentes módulos de um sintetizador analógico sendo que alguns podem interferir no funcionamento daqueles que estão à sua volta. Quando um objeto é posicionado sobre a mesa, em alguns

² Imagens únicas, geradas através de algoritmo usado na visão computacional para obter informações sobre o mundo real.

casos, aparecem controles que permitem a mudança, através do toque, do comprimento de onda, volume, tons e padrões musicais.

2.1.3. 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 de 3,5 polegadas.



Figura 7 - iPhone

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.

O dispositivo possui três sensores: um de proximidade, que previne que o rosto ou orelhas interajam com o dispositivo enquanto uma chamada é atendida; um de luz ambiente que altera automaticamente o brilho do display; e por último, um acelerômetro de três eixos, que de acordo com a orientação do aparelho, muda o posicionamento das informações exibidas em sua tela.

O *iPhone* permite o uso de comandos multi-toque utilizando uma camada de material capacitivo sob sua tela. Esse material capacitivo, responsável por armazenar energia, é organizado de acordo com as coordenadas do sistema no formato de uma grade.

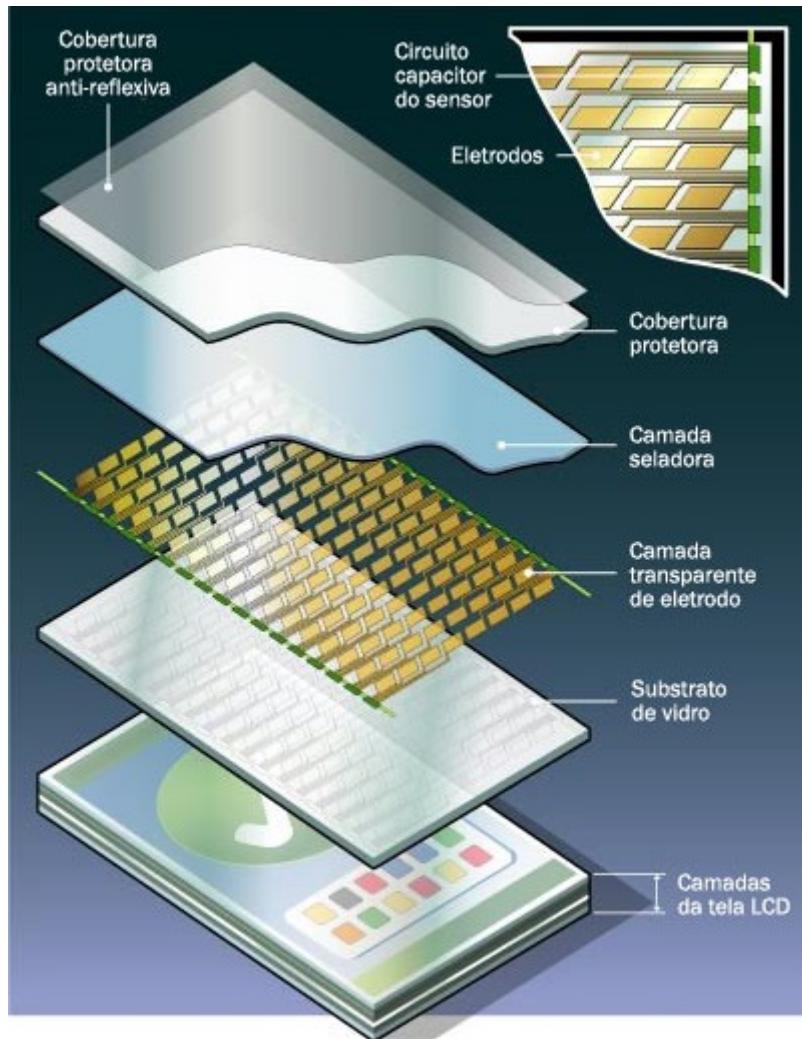


Figura 8 - Estrutura da tela multi-toque do iPhone

Quando o toque acontece, a energia magnética emitida pelo dedo humano é armazenada nos capacitores próximos à região do toque. A energia armazenada pelos é filtrada e analisada por software, determinando as propriedades de cada toque, como tamanho e posição na tela do iPhone.

Todas as ações multi-toque são reconhecidas através de *gestures* (gestos). Um exemplo de seu uso é o utilizado no redimensionamento de objetos. A ampliação é detectável através da separação de dois dedos, enquanto a redução através da aproximação de dois dedos.

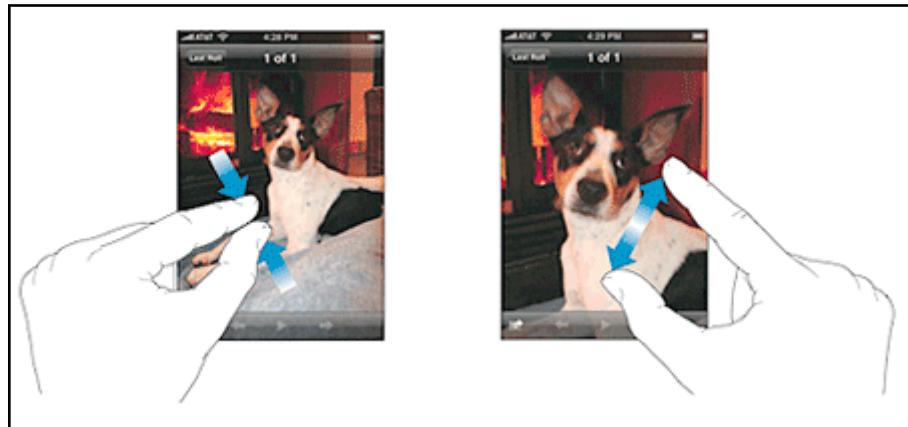


Figura 9 - Gestão de redimensionamento no iPhone

O iPhone permite o uso de diversos aplicativos, como planilhas, emails, fotos e tocadores de músicas, jogos, emuladores e sistemas GPS (*Global Positioning System*).



Figura 10 - Sistema GPS no iPhone

2.2. Jogos e Interatividade

Desde os primeiros jogos que lançados, novas formas de interação com jogador eram estudadas, de forma a transportar o jogador para dentro do universo do jogo e tornar a interação mais intuitiva e natural. Exemplos dessas pesquisas são os volantes para os jogos de corrida, pistolas para jogos de tiro em primeira pessoa e manches para jogos de aeronaves.

O *IR Taktiks*, por se tratar de um jogo de RPG tático, com características de estratégia, motivou o estudo mais profundo destes gêneros de jogos, suas principais características e vertentes.

2.2.1. Jogos de Estratégia

Do grego *stratègos*³, 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.

Quanto à classificação temporal, pode ser em tempo real, ou seja, 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. Ou baseados em turnos; onde cada jogador deve esperar seu momento de agir, aguardando a ação do adversário. Em alguns jogos, porém, podemos encontrar ambas as características. Como por exemplo, temos algumas ações podem ser em tempo real e o quando há combate entre jogadores, 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, como no caso dos tabuleiros de *wargames*; outros não possuem ligação com o mundo real, como por exemplo: Go, damas, xadrez, entre outros.

2.2.2. 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, em longas missões espaciais, utiliza o *RPG* como forma de

³ Stratos: Exército, Ago: Liderança.

⁴ Agência espacial norte-americana

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*.

2.2.2.1. Dungeons & Dragons

Dungeons & Dragons (abreviado como *D&D* ou *DnD*) foi o primeiro *RPG* comercial, publicado, como livro, em 1974 nos EUA. Desde então diversas publicações ajudam a retratar este universo de fantasia medieval. Neste jogo, cada jogador interpreta um personagem e este pretende a área de atuação dentro deste universo (bardo, monge, guerreiro, entre outros). Sempre atuam em grupo por um objetivo em comum e as partidas geralmente possuem várias sessões, que são chamadas de aventuras e esse conjunto de sessões é chamada campanha.

A história base da aventura é determinada pelo Mestre, e os resultados das escolhas do grupo variam de acordo com as regras e da interpretação dos personagens, normalmente uma ótima interpretação é generosamente recompensada. As extensas regras do jogo cobrem diversas áreas como interações sociais, usos de magia, combate e o efeito do ambiente nos personagens.

Alguns jogadores, para facilitar a estratégia de combate, utilizam maquetes para composição do cenário e miniaturas caracterizadas como representação dos personagens.



Figura 11 - Exemplo de campanha em andamento

2.2.3. Jogos de RPG 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 estudo⁵ 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*.

⁵ <http://www.newhorizons.org/strategies/literacy/kestrel.htm>

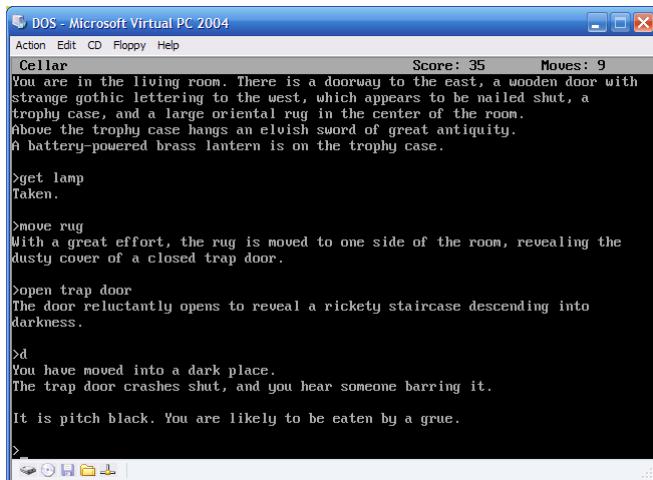


Figura 12 - Zork (1979)

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 13 - 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 possui uma continuação em filme.



Figura 14 - Final Fantasy VII - Squaresoft (1997)

Juntamente com a popularização da internet, um novo gênero de *RPGs* foi criado: o MMORPG (*Massively Multiplayer Online Rolling 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 15 - World of Warcraft - Blizzard (2004)

2.2.3.1. Final Fantasy Tactics

O *Final Fantasy Tactics* é um *RPG* para o *Playstation* desenvolvido pela *Squaresoft*, em 1997, cuja principal inovação está a cargo de seu sistema de jogo, baseado em cenários tridimensionais isométricos.



Figura 16 - Cenário tridimensional isométrico

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.

Os personagens se movem por um cenário dividido em partes iguais, e o campo de ação de um personagem é determinado por suas características e profissão. Os inimigos somente podem ser atingidos se nele estiverem. Cada atributo do personagem e sua respectiva evolução são baseados em sua profissão. 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.



Figura 17 - Personagem e sua área de atuação

Existem diversos objetivos de batalha. A mais comum é derrotar todos os inimigos, porém em certas partes do jogo, o objetivo pode se alterar a resgatar algum personagem em outra parte do cenário, proteger um determinado personagem ou matar apenas um inimigo específico.

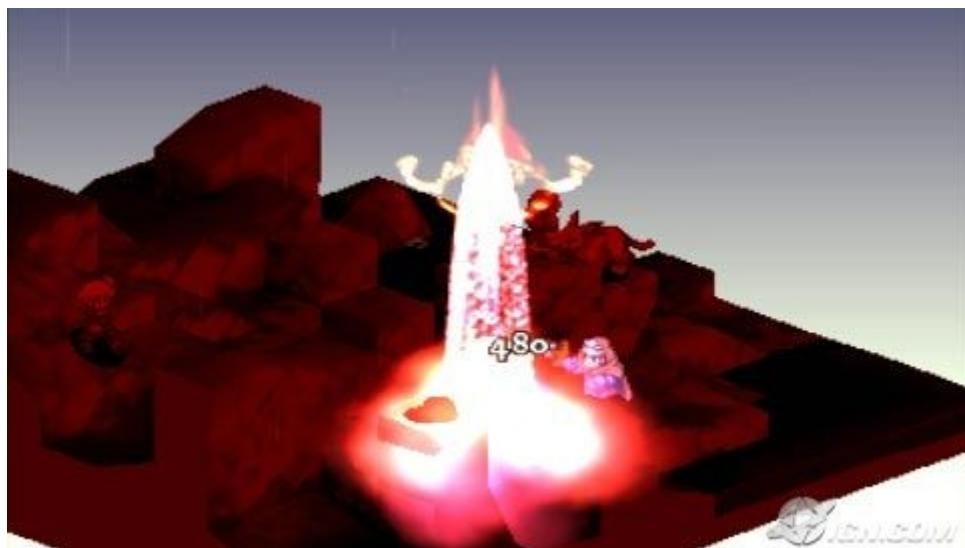


Figura 18 - Personagem efetuando um ataque

2.2.4. 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.

2.2.4.1. 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.

2.3. Implementações de Superfícies Multi-toque

Atualmente, existem diversas técnicas para a detecção de toques em superfícies, desde a análise de uma imagem pura, utilização de sensores medidores de pressão, utilização de um *grid* de filamentos eletrônicos, onde o toque simplesmente fecha faz com que os filamentos entrem em contato e propagem a corrente elétrica; até complexa utilização de circuitos capacitores que armazenam a energia elétrica emitida pelo corpo humano, quando o toque ocorre.

As mais simples e baratas técnicas ficam a cargo da utilização de iluminação infravermelha, sobre uma superfície de acrílico com um material difusor, de forma que a luz ambiente não influencie sua detecção. O material difusor é necessário para além de permitir a visualização da projeção, auxilia na difusão da iluminação infravermelha.

Esta técnica está sendo bastante utilizada no desenvolvimento de protótipos por todo o mundo. Há dois modos principais de utilização: Iluminação Difusa (*Diffused Illumination*) e Reflexão Total Interna Frustrada da Luz (*FTIR*).

A mesa multi-toque utilizada neste trabalho foi construída baseando-se nas técnicas de iluminação infravermelha, mais especificamente a *FTIR*.

2.3.1. Iluminação Difusa (*Diffused Illumination*)

Consiste na iluminação da superfície onde ocorrerá o toque por um canhão de *LEDs* infravermelhos. Quando a iluminação acontece sob a superfície é dado o nome de *Rear Illumination*. Já quando a iluminação acontece sobre a superfície é designado de *Front Illumination*.

Na *Rear Illumination*, quando um toque acontece sobre a superfície, a luz infravermelha emitida é refletida em todas as direções, com o auxílio do material difusor do acrílico. Esta luz refletida é lida por um software com o auxílio de uma *webcam*, que transforma essas reflexões em pontos brancos, determinando a posição onde o toque ocorreu.

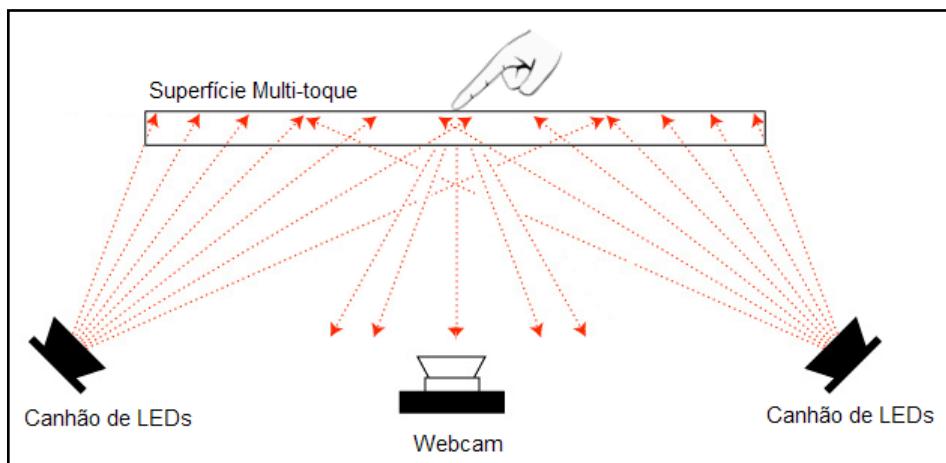


Figura 19 - *Rear Illumination*

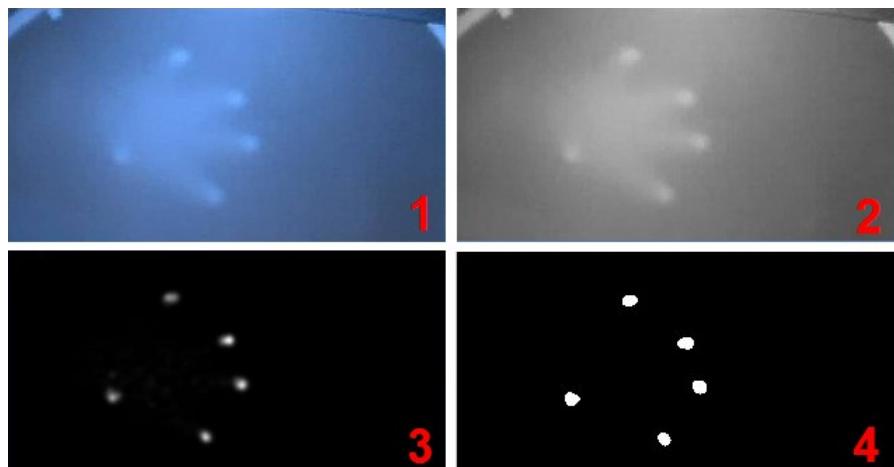


Figura 20 - Exemplo da detecção de toques utilizando *Rear Illumination*

Utilizando *Front Illumination*, quando o dedo encosta a superfície da mesa, este produz uma sombra na superfície difusora que está completamente iluminada. A imagem é obtida pela *webcam* e repassada para o software responsável por identificar os toques. Através de algoritmos, o software transforma a sombra em pontos brancos, determinando a posição dos toques.

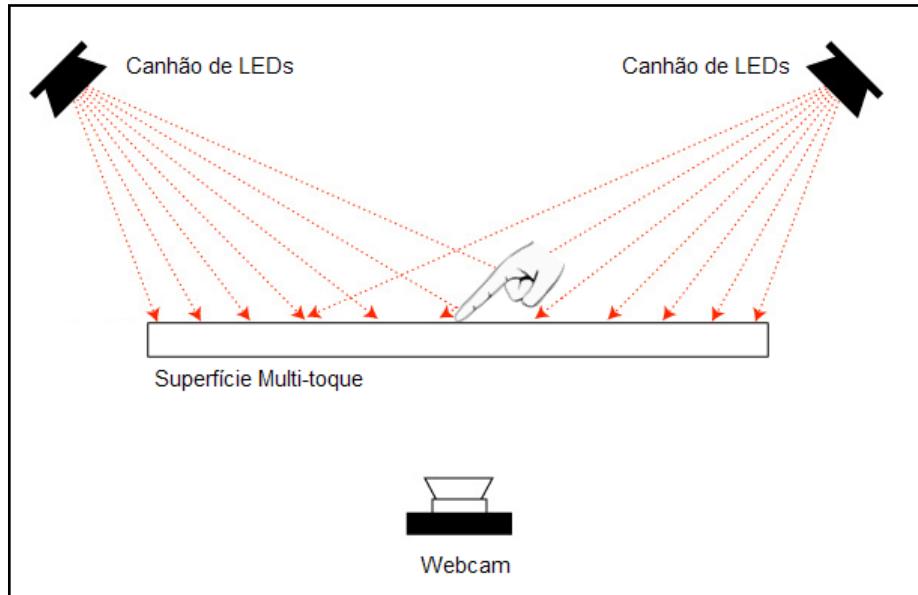


Figura 21 - Front Illumination

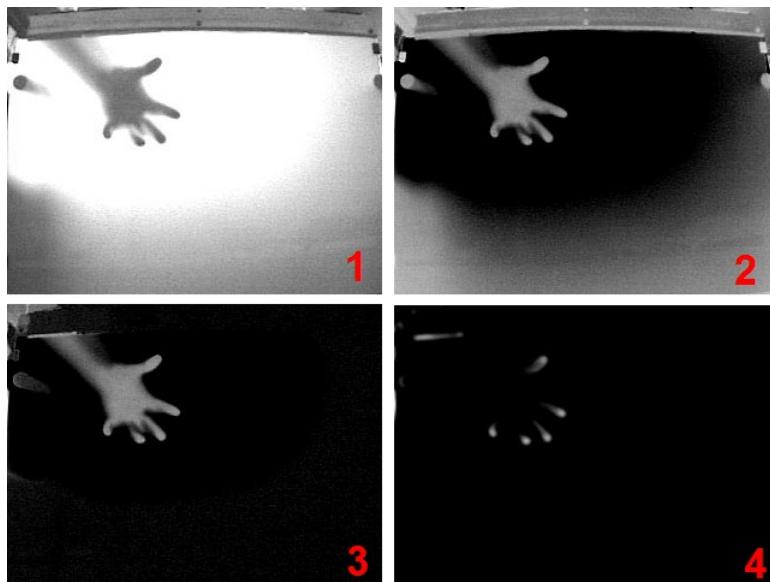


Figura 22 - Exemplo da detecção de toques utilizando Front Illumination

2.3.2. Reflexão Total Interna Frustrada da Luz (FTIR)

A reflexão da luz é o fenômeno físico em que um feixe de luz incide sobre uma determinada superfície e é refletida para o mesmo meio de propagação de origem. Quando a reflexão é total, todas as partículas do feixe de luz são redirecionadas ao meio de propagação de origem, ao contrário da parcial, onde algumas partículas atravessam a interface entre os meios de propagação, ocorrendo um desvio no ângulo de incidência do feixe de luz emitido, chamado de refração.

O fator que determina se uma reflexão será total ou parcial é o ângulo de incidência sobre a interface entre os meios de propagação que possuem um ângulo

crítico. Quando o feixe de luz incide com ângulo superior ao ângulo crítico (em relação à interface), ocorre reflexão parcial, e o feixe de luz é refratado. Quando o ângulo é menor, ocorre a reflexão total e o feixe de luz é refletido para o meio de propagação de origem.

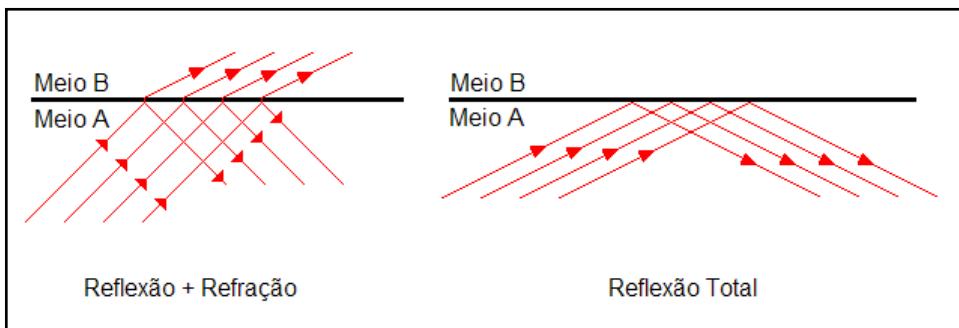


Figura 23 - Exemplos de reflexão com refração e reflexão total da luz

O pesquisador J. Y. Han, durante suas pesquisas sobre técnicas de interação multi-toque, percebeu que a pele é um material difusor, ou seja, quando um feixe de luz que seria refletido totalmente entra em contato com a pele, ele é difundido em todas as direções. A esse efeito de difundir a luz que seria totalmente refletida, se deu o nome de reflexão total interna frustrada da luz.

Transpondo a teoria para a aplicação em interfaces multi-toque, consiste em iluminar as laterais de uma superfície de acrílico com diversos *LEDs* infravermelhos. A luz emitida fica presa dentro do acrílico devido ao fenômeno da reflexão total da luz. Quando o dedo toca a superfície da mesa, a luz é difundida para baixo, onde uma *webcam* obtém imagens. Essa difusão é reconhecida como pontos brancos na imagem capturada e a posição dos toques é facilmente detectada.

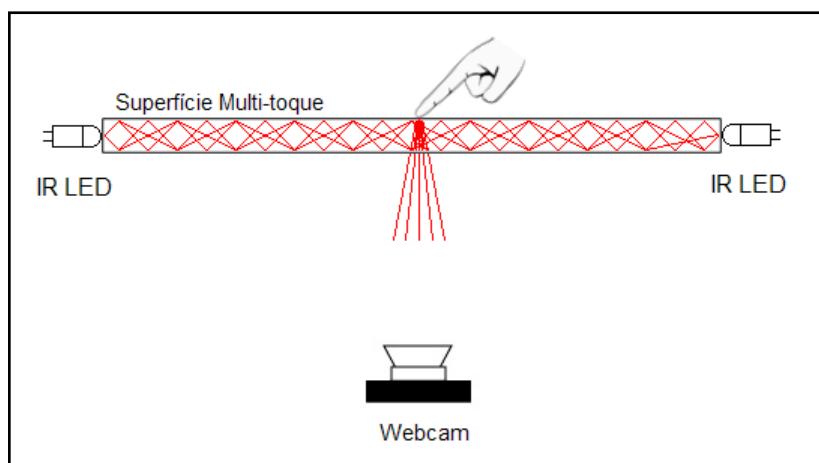


Figura 24 - Reflexão total interna frustrada da luz

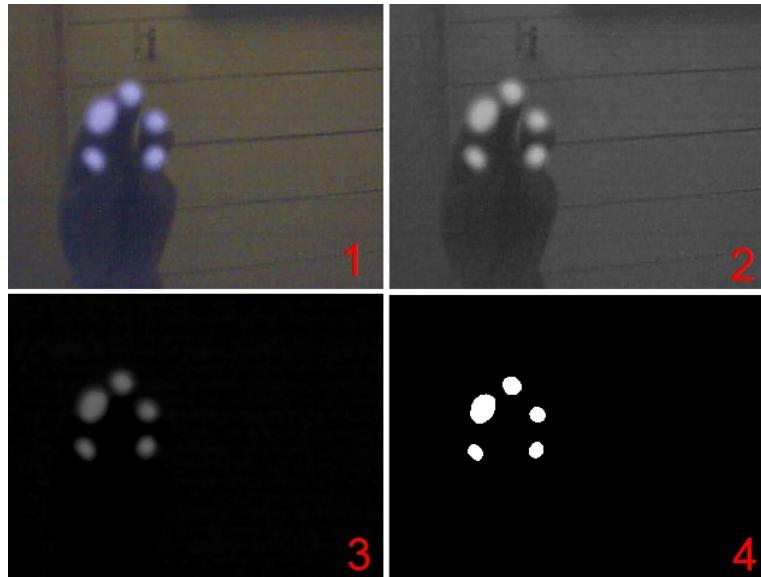


Figura 25 - Exemplo da detecção de toques utilizando FTIR

2.4. Tecnologias Utilizadas

Para o reconhecimento de toques sobre a superfície da mesa multi-toque, alguns softwares foram utilizados. Com a grande quantidade de superfícies multi-toque sendo desenvolvidas, um padrão para o armazenamento das informações e sua integração com outras aplicações foi sendo adotado pelos desenvolvedores de software.

Durante o projeto da *reacTable*, desenvolveu-se um protocolo capaz de armazenar informações sobre toques e objetos em qualquer superfície multi-toque. A esse protocolo deu-se o nome de *TUIO*. Porém o software responsável pela identificação de toques e fiduciais, o *reacTIVision*, necessitava conversar com o aplicativo que gerava os sons e efeitos. Para efetuar esta comunicação de mensagens do protocolo *TUIO*, a equipe da *reacTable* decidiu utilizar o protocolo *OSC* (*Open Sound Control*), que permitia ser meio de transporte para outros protocolos.

Após o desenvolvimento do *reacTIVision*, diversos softwares com o propósito de identificação de toques foram desenvolvidos. A grande maioria buscou seguir o mesmo padrão, ou seja, mensagens *TUIO* sob o protocolo *OSC*, tornando-os padrão no desenvolvimento de aplicações multi-toque.

O software de reconhecimento do toques *Touchlib*, foi o escolhido para o controle da mesa multi-toque devido à sua estabilidade, número de funcionalidades

e uso dos padrões propostos. Isso permite que o jogo funcione corretamente em qualquer superfície multi-toque que siga os padrões propostos, aumentando sua interoperabilidade com outros projetos.

2.4.1. OSC

O *Open Sound Control* (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.

2.4.1.1. 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*.

2.4.2. 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.

2.4.3. 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 26 - *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.

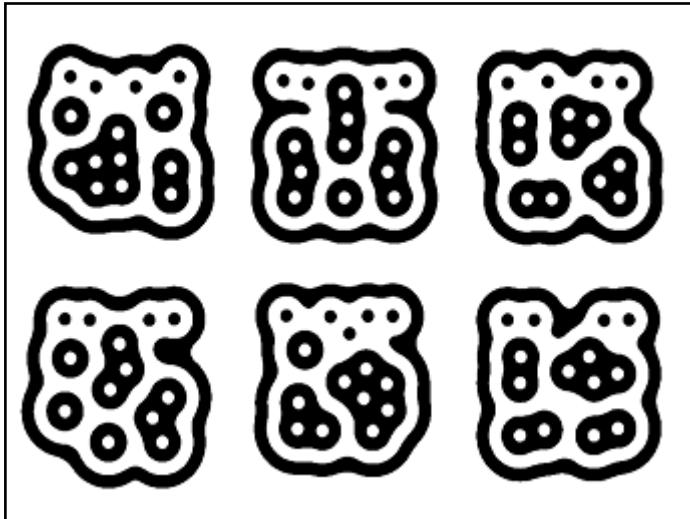


Figura 27 - Marcadores fiduciais

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*.

2.4.4. 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 em parceria com a *White Noise Audio*⁷; 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 com demais aplicativos através do protocolo *TUIO*, sob o protocolo *OSC*, utilizando a biblioteca *oscpack*.

O *Touchlib* aplica filtros nas imagens recebidas, a fim de melhorar a percepção de toques. Atualmente esta biblioteca trabalha apenas na plataforma *Windows*,

⁶ <http://nuigroup.com>

⁷ <http://whitenoiseaudio.com>

porém existem esforços sendo realizados para portá-la para outras plataformas, como *Mac* e *Linux*.

2.4.4.1. Configuração e Calibração

Para que o software funcione corretamente, é necessário que sua configuração seja feita, através de um aplicativo de calibração pronto enviado junto com o *Touchlib*. A configuração define a área aplicável do software sobre a superfície multi-toque, os filtros que serão utilizados e seus valores.

Ao executar o aplicativo de calibração, diversas janelas exibem cada um dos filtros disponíveis aplicados à imagem obtida pela *webcam*. Alterando-se os valores dos filtros é possível obter uma imagem nítida do toque. Após a configuração dos filtros, o aplicativo de calibração é iniciado.

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). O *Touchlib* limiariza e segmenta a imagem obtida pela *webcam* e a divide em 20 imagens menores. Quando um toque é detectado, sua posição é calculada através de uma interpolação linear apenas na respectiva fatia de imagem. Isso permite que a câmera não necessite estar perpendicular à superfície de projeção, pois a distorção provocada é compensada via software.

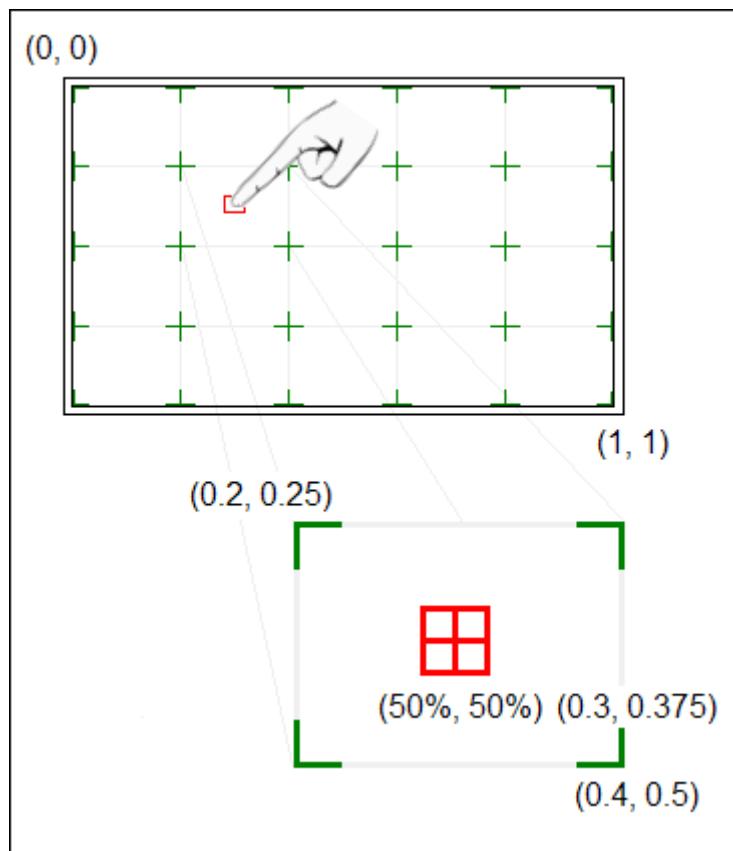


Figura 28 - Exemplo de interpolação no cálculo da posição do toque

Ao executar o aplicativo de calibração, uma representação dessas 20 subdivisões é exibida, através de cruzes verdes. Projetando-se essa representação sobre a superfície multi-toque que deverá ser controlada, é possível relacionar determinada posição da mesa com os diversos vértices destas subdivisões, parametrizando a interpolação realizada.



Figura 29 - Demonstração do software de calibração

O resultado final da configuração é um arquivo XML, com os valores de todos os vértices e os filtros e seus respectivos valores. Este arquivo deve ser movido para o mesmo diretório onde se encontra o *Touchlib*, de forma que este leia as novas configurações quando executado. Um exemplo deste arquivo encontra-se nos anexos.

2.4.5. Microsoft XNA

O *framework Microsoft XNA* é um conjunto de classes desenvolvidas com o foco no desenvolvimento de jogos eletrônicos para as plataformas PC, Xbox360 e Zune. Tendo sua primeira versão lançada em 2006, objetivou simplificar o desenvolvimento de jogos, reduzindo a quantidade de código necessária para se ter um jogo. Provê fácil acesso a recursos comuns em jogos, como gerenciamento de janelas, desenho, interação com usuário, carregamento de recursos externos, entre outros. Atualmente, encontra-se na versão 3.0 CTP (*Community Technical Preview*), e possui

O *framework Microsoft XNA* é integrado ao ambiente de desenvolvimento *Microsoft Visual Studio*, utilizando o C# como linguagem de programação básica. Projetado e construído sobre o *DirectX*, é dividido em quatro camadas:

2.4.5.1. Plataform

É a camada mais próxima do sistema operacional. Controla as *APIs*⁸ (*Application Programming Interface*) do *Direct3D*, *XACT*, *XInput* e *XContent*. Essas interfaces são utilizadas no controle do vídeo e sons, interação com o usuário através de periféricos e gerenciamento de conteúdos externos ao jogo.

2.4.5.2. Core Framework

Camada do *framework XNA* mais próxima do usuário. Provê acesso gerenciado e otimizado às interfaces da camada *Plataform*, executores de cálculos matemáticos e armazenamento de dados.

⁸ Conjunto de funções e estruturas de um software para utilização por programas aplicativos

2.4.5.3. Extended Framework

Essa camada auxilia na estruturação do jogo. Provê classes para o gerenciamento do *loop* principal do jogo, além de componentes e serviços. Possui as subcamadas *Application Model* e *Content Pipeline*. A primeira é responsável por auxiliar na estruturação da arquitetura do jogo, enquanto a segunda executa a transformação de recursos externos em estruturas conhecidas pelo *framework XNA*.

2.4.5.4. Games

Camada do *framework* que contém códigos prontos, *starter kits*⁹, *templates* e componentes.

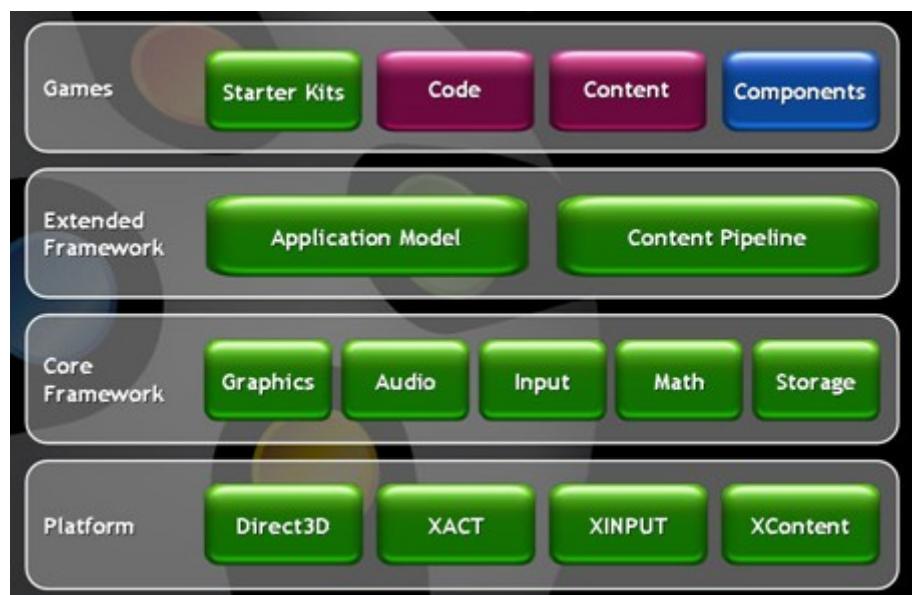


Figura 30 - Representação das camadas do framework XNA

⁹ Jogos simples, disponibilizados em caráter de referência para diminuir a curva de aprendizado das ferramentas e arquiteturas introduzidas pelo *framework XNA*

3. PROJETO

Este capítulo aborda questões sobre o desenvolvimento do projeto, sua concepção, arquitetura, dificuldades e soluções encontradas.

3.1. Concepção

O jogo desenvolvido é um *RPG* tático semelhante ao *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.

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 em conjunto com o cenário onde a batalha acontece.

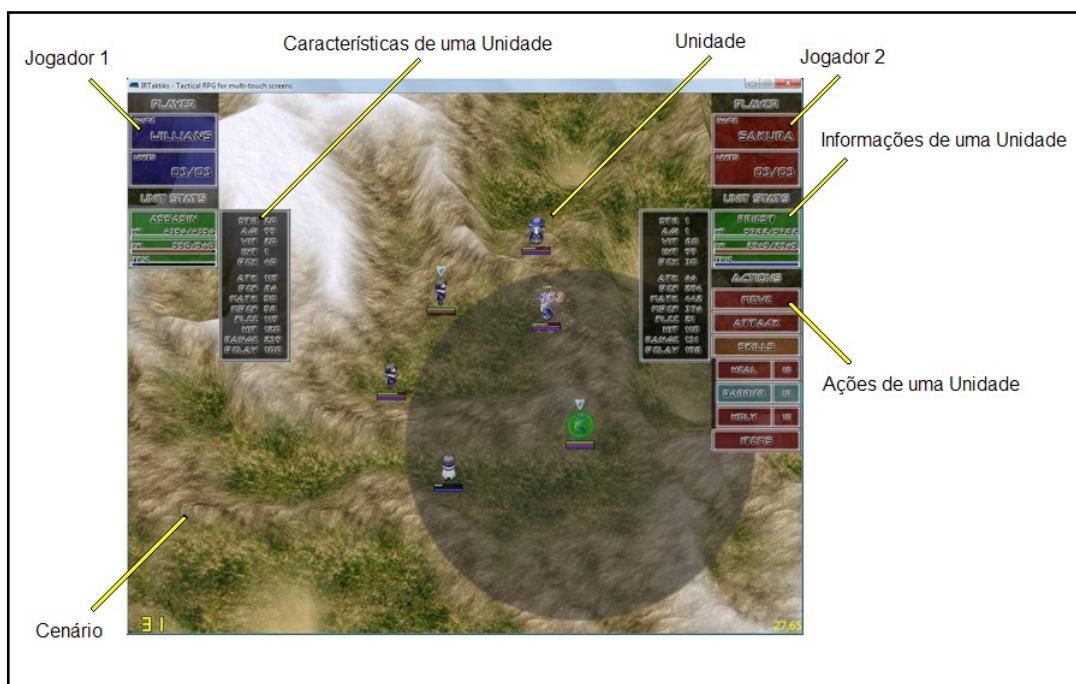


Figura 31 - Elementos do jogo

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. Para a detecção de objetos utilizáramos os fiduciais utilizados pela *reacTable*, porém devido a problemas em sua detecção, explicados neste capítulo, o controle dos personagens também ficou a cargo do toque. Já para o toque, utilizamos o software *Touchlib*.

Isso deu ao jogo portabilidade, uma vez que não é mais necessário utilizar-se de uma mesa para jogá-lo. Qualquer interface multi-toque em qualquer posição é suficiente para interagir com o jogo.

A mesa multi-toque é controlada pelo software de reconhecimento de toques, *Touchlib*. Como dito anteriormente, utiliza o protocolo *TUIO* juntamente com o protocolo OSC para se comunicar com aplicativos externos, no caso o jogo. O conjunto de bibliotecas *oscpack*, que implementa o protocolo OSC, será utilizado para efetuar a comunicação entre os softwares envolvidos (*Touchlib* e jogo).

O desenvolvimento do jogo se dará através do desenvolvimento de um protótipo inicial e uma versão final. Para agilizar o desenvolvimento do jogo, facilitar sua integração e deixá-lo mais robusto, confiável e de rápida manutenção, o *framework*, *Microsoft XNA 2.0* foi utilizado.

As ações que os usuários executarem sobre a mesa, como o toque de um ou mais dedos sobre sua superfície, será reconhecida pelo *Touchlib* através da análise das imagens enviadas por uma *webcam*. O *Touchlib* processa as informações e envia uma mensagem *TUIO* para cada dedo sobre a mesa, contendo as informações como posição, ângulo de movimentação, velocidades calculadas entre outras.

Estas mensagens *TUIO* são empacotadas dentro de envelopes OSC e enviadas através da biblioteca *oscpack*, para a aplicação cliente, no caso o jogo. O jogo decodifica o envelope OSC, utilizando também a biblioteca *oscpack*; e obtém a mensagem *TUIO* original, juntamente com as informações sobre os toques.

Para cada ação efetuada na mesa, um evento é disparado pelo módulo *Input*. Estes eventos sensibilizam os diversos componentes do jogo que se atualizam. Finalmente, o jogo é projetado com o auxílio de um projetor sob a superfície da

mesa. Com isso, o usuário possui a impressão de estar manipulando diretamente os objetos do jogo.

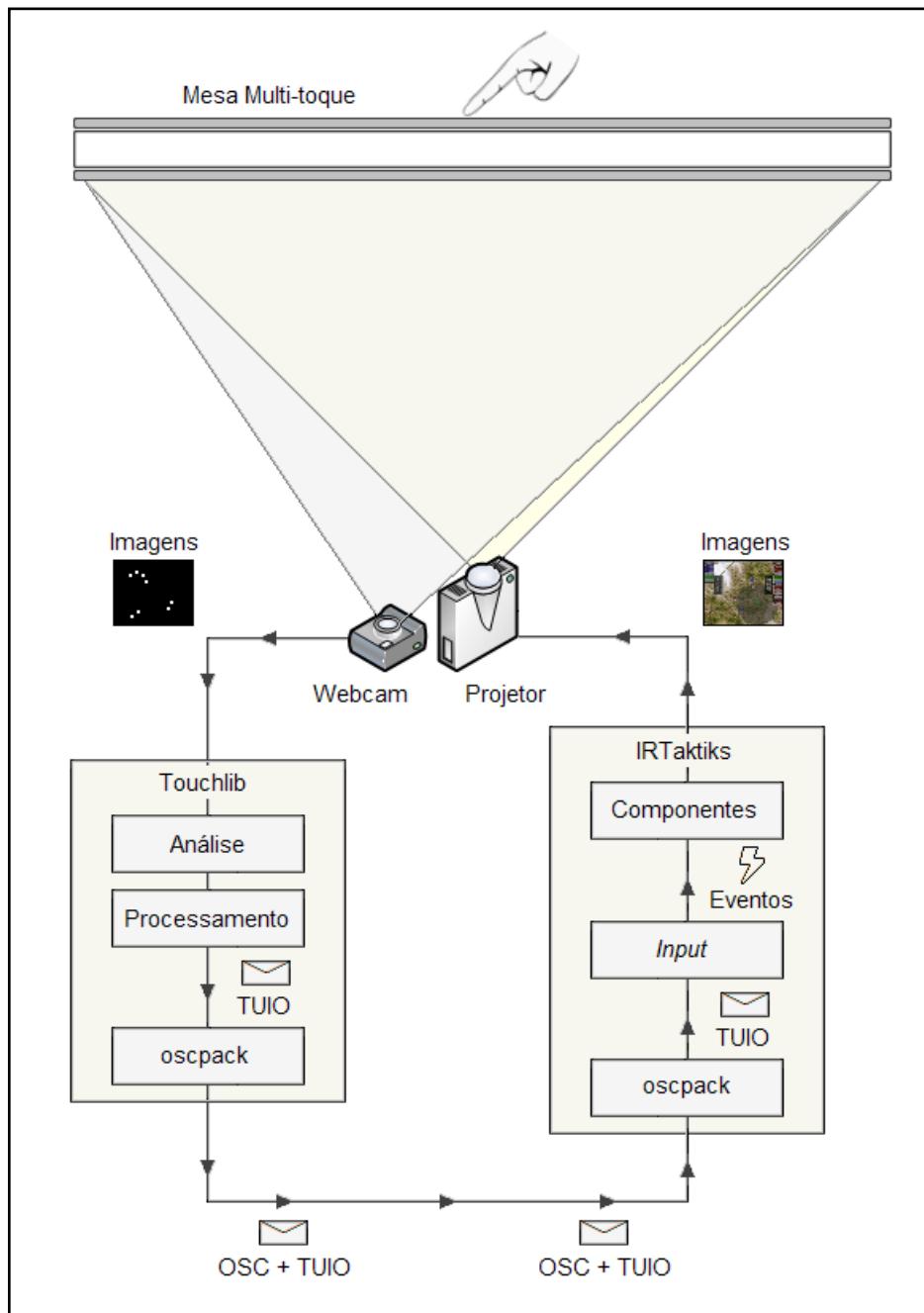


Figura 32 - Arquitetura do sistema

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 e melhorar o contraste entre as regiões detectáveis como toques. Também foram partes da reestruturação, a incorporação

de uma superfície difusora de projeção, necessária para a visualização da projeção do jogo; alteração da *webcam*, responsável pela captura de imagens para a visão computacional, para uma com maior campo de visão; e adição e configuração do software responsável pelo reconhecimento dos toques.

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 atingidos.

3.2. Adequação da Mesa

3.2.1. 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, que facilita seu deslocamento. Possui 47 entradas para *LEDs* infravermelhos, de modo que a luz percorra o interior do acrílico capturada de acordo com o princípio da reflexão total interna frustrada da luz. Estas entradas são dispostas pelos quatro lados da mesa, intensificando a propagação da iluminação dos *LEDs*, em direção ao centro do acrílico.



Figura 33 - Mesa multi-toque utilizada no projeto

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 e formar um bom contraste nas áreas detectáveis como toque.

A mesa possuía 47 *LEDs* com tensão de barreira de potencial de 1,5V, alimentados por uma fonte de 9V. Para cada *LED*, existia um resistor de 150Ω ligado em série. Todos os ramos formados estavam ligados em paralelo, limitando a corrente elétrica em cada ramo em torno de 52mA.



Figura 34 - Contraste do toque na mesa antes da reestruturação

Para melhorar o contraste no centro da mesa e facilitar a manutenção, a parte elétrica foi completamente reestruturada. Inicialmente, tínhamos o objetivo de apenas substituir os componentes eletrônicos utilizados. Os novos resistores teriam menos resistência que os de 150Ω , a fonte teria de maior tensão e potência. Com isso, a corrente elétrica nos *LEDs* aumentaria e consequentemente a intensidade da iluminação.

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 evitar novas depredações, uma capa protetora de tecido foi providenciada pelo Centro Universitário Senac.

Para facilitar a manutenção, evitando dificuldades de reparos após eventuais novas depredações; decidimos tornar todas as ligações completamente modulares e de fácil substituição, pois não seria utilizada nenhuma solda ou cola na fixação dos componentes.

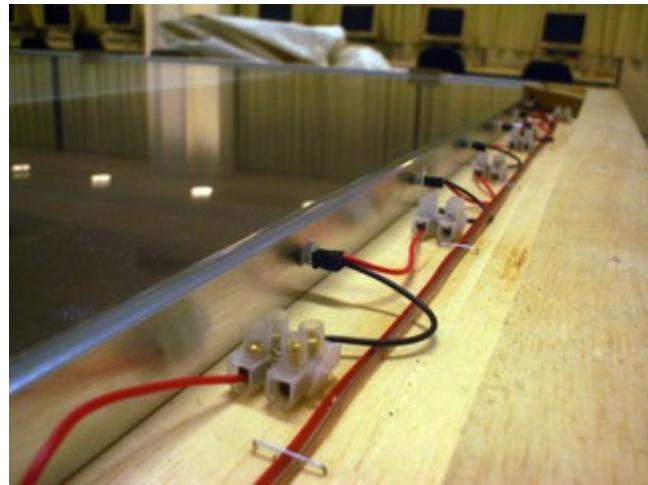


Figura 35 - Parte elétrica após a reestruturação

Após a reestruturação, 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 ramos. Cada ramo possui dois resistores: um de 56Ω e outro de $5,6\Omega$ ligados em série, limitando a corrente de 5 LEDs ligados em série. O último ramo por possuir apenas 2 LEDs, conta com dois resistores de 56Ω ligados em série.

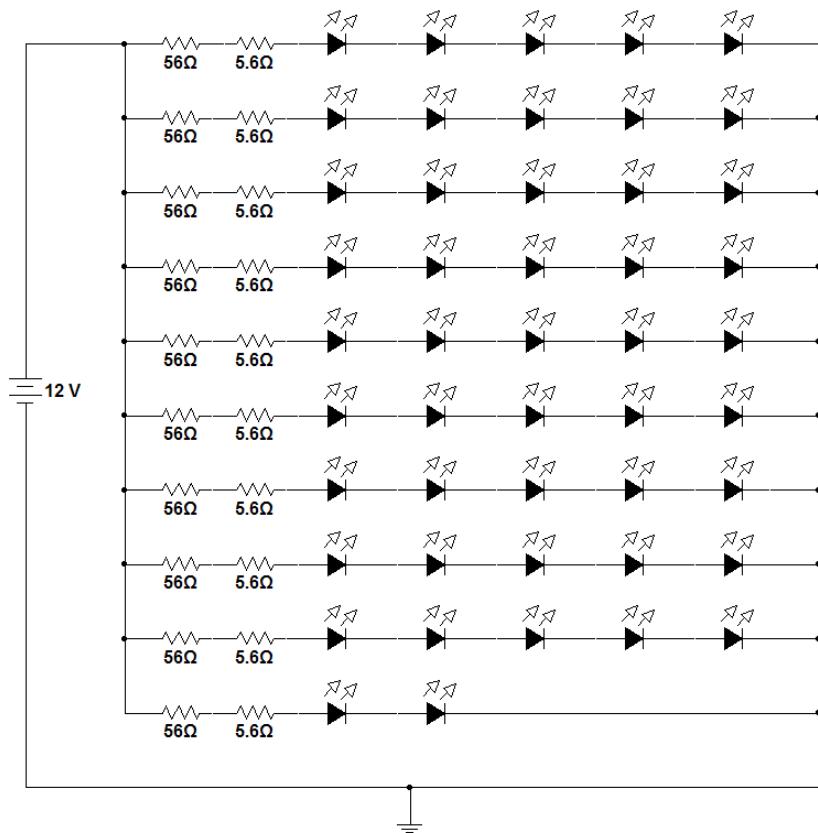


Figura 36 - Representação do circuito elétrico da mesa

No ramo que possui 5 *LEDs*, a corrente elétrica aumentou para 97mA, enquanto na de 2 *LEDs*, para 84mA. Como a intensidade de iluminação aumenta de forma linear com o aumento de sua corrente elétrica, podemos considerar que a iluminação da mesa teve um aumento de 53%.

$$V_T = V_F - V_0$$

$$9 - 1.2) \cdot 1 = 50 \cdot I$$

$$V_T = R \cdot I$$

$$I_A = 2mA$$

$$(V_F - V_0) \cdot qLed = R \cdot I$$

$$12 - 1.2) \cdot 5 = 1.6 \cdot I$$

V_T : Tensão Total

$$I_N = 7mA$$

V_F : Tensão Fonte

$\frac{I_A}{I_N} = 6$ do aumento da Intensidade do led

V_0 : Barreira de potencial do led

$$I_N$$

$qLed$: quantidade de leds

R : Valor da resistência

$$\frac{52 \cdot 10^-3}{97 \cdot 10^-3} = 0.53 = 53\%$$

I : Intensidade da corrente no circuito



Figura 37 - Contraste do toque na mesa após reestruturação

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

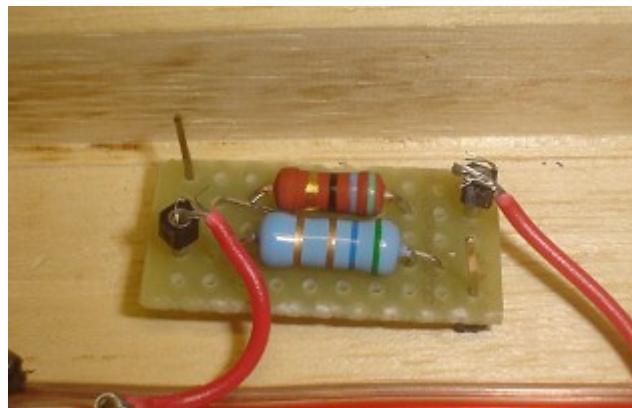


Figura 38 - Placa de circuito impresso com os resistores de 56Ω e $5,6\Omega$

Os LEDs foram acoplados à mesa utilizando conectores modulares. Com isso, quando houver alguma queima, é possível efetuar a troca sem maiores problemas e ferramentas, como um ferro de solda, por exemplo. Estes conectores são ligados à fiação elétrica e à placa de circuito impresso através de *plugs*.



Figura 39 - Conector com LED

3.2.2. Visão Computacional

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



Figura 40 - Microsoft LifeCam VX 6000

Esta *webcam* veio de fábrica com um filtro que inicia a passagem da luz infravermelha. Para a correta utilização neste projeto, este filtro teve que ser removido para que as imagens dos toques, que somente são visíveis ao espectro de luz infravermelha; pudessem ser passadas ao *Touchlib*.



Figura 41 - Filtro inibidor da luz infravermelha



Figura 42 - Toque com e sem o filtro inibidor da luz infravermelha

Para que pudéssemos ter uma imagem dos toques sem influências da iluminação externa, foi adicionado à câmera um filtro que somente permite a passagem do espectro de luz infravermelha. O filtro utilizado foi um filme fotográfico.



Figura 43 - Microsoft LifeCam VX 6000 desmontada e filtro inibidor da luz visível

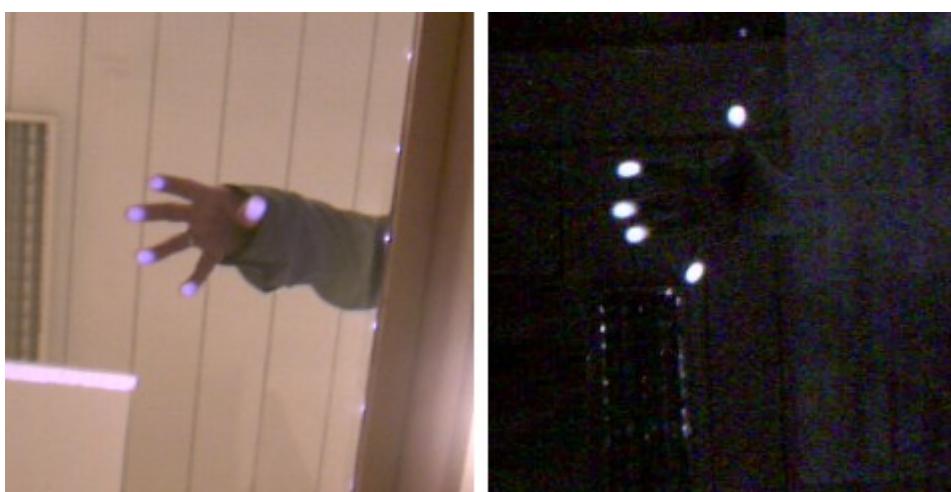


Figura 44 - Toque sem e com o filtro inibidor da luz visível

A webcam fica posicionada sob a mesa olhando para o acrílico, de modo a obter as imagens dos toques. Por possuir um ângulo de visão maior que às convencionais, pode ser colocada a uma distância maior em relação ao acrílico e mesmo assim cobrir uma área da mesa maior ou equivalente.

A projeção é feita com um projetor de resolução 800px por 600px e um espelho. A imagem é projetada no espelho que redireciona a imagem para 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.



Figura 45 - Sistema de projeção

O material ideal para este tipo de mesa é um polímero para projeções, fabricado pela *Rosco*¹⁰. Como este material não é encontrado facilmente no Brasil, sua utilização foi descartada. Para substituir o material difusor, foram realizados testes utilizando papel vegetal e sacolas plásticas brancas de polietileno.

Testes realizados, descritos a seguir, indicaram que os sacos plásticos forneceram maior nitidez na detecção dos toques, em relação ao papel vegetal. Este material não foi encontrado no tamanho necessário para cobrir uma área razoável da mesa e foi também descartado. Com isso, o papel vegetal foi adotado como anteparo de projeção.

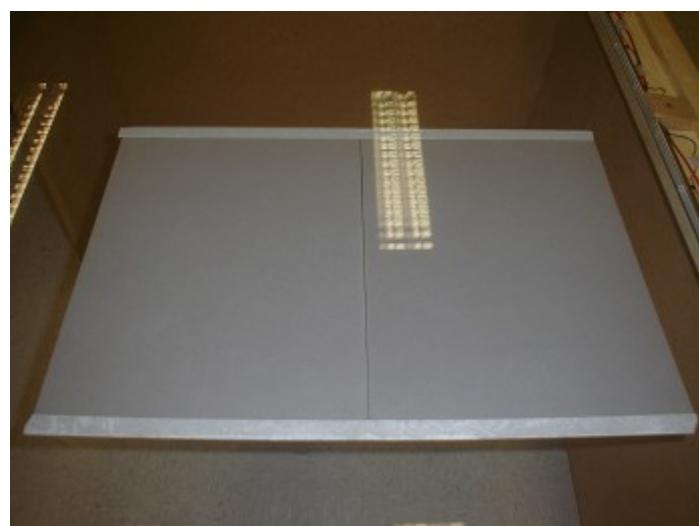


Figura 46 - Papel vegetal como anteparo de projeção

¹⁰ <http://www.rosco.com/>

3.2.3. Testes e Dificuldades Encontradas

A primeira dificuldade encontrada no desenvolvimento do projeto foi o baixo contraste entre as regiões compreendidas como toques na mesa desenvolvida pelos ex-alunos do Centro Universitário Senac. Optou-se por reestruturar toda a parte elétrica, trocando os seus componentes elétricos de modo a melhorar este contraste. Com um contraste maior, não é necessária força para interagir com a superfície, além de permitir uma leitura mais precisa por parte do software *Touchlib*.



Figura 47 - Comparativo do toque antes e depois da reestruturação

Uma das maiores dificuldades encontradas no desenvolvimento do projeto foi o reconhecimento de marcadores fiduciais do *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 um *LED* infravermelho, de maneira a deixar o marcador discernível na imagem obtida pela *webcam*, mesmo quando há incidência de luz do projetor sobre ele. Um circuito com o *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 48 - Copo e circuito com LED usado na iluminação do fiducial

Utilizando este circuito, os testes utilizando anteparos difusores de projeção que ficam sob a superfície do acrílico, de maneira a que os usuários possam ver as imagens projetadas do jogo puderam ser iniciados. Foram utilizados dois materiais nos testes: papel vegetal e sacos plásticos brancos de polietileno. Estes anteparos difusores são colocados sob a superfície de acrílico. Caso fossem colocados sobre a superfície, a frustração da luz refletida dentro do acrílico não aconteceria. Estes materiais foram escolhidos devido ao fato de serem semitransparentes, permitindo que a luz infravermelha o ultrapasse e que a projeção seja visível.

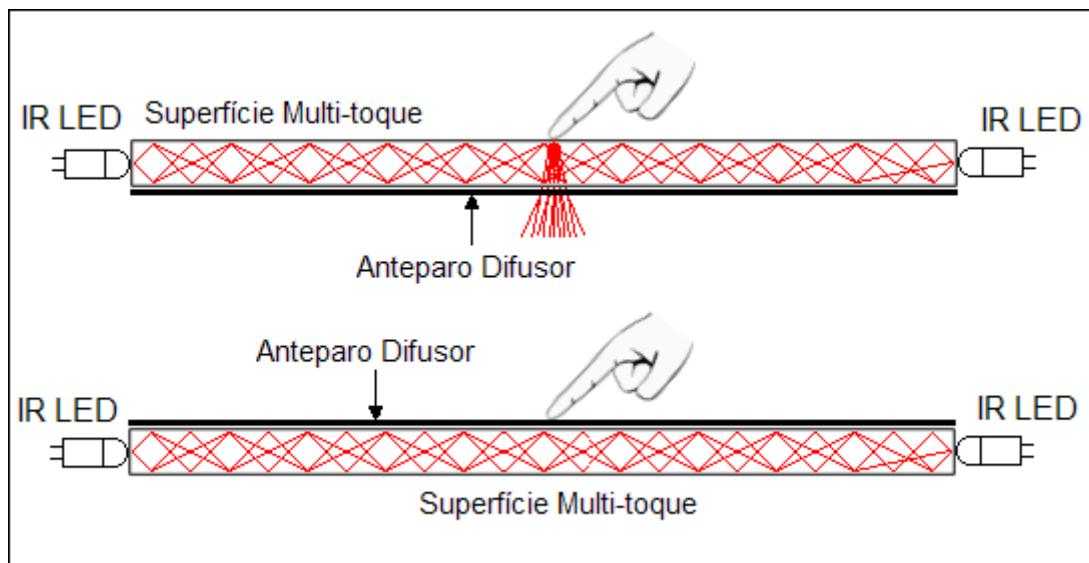


Figura 49 - FTIR utilizando anteparo difusor para projeção

A detecção do fiducial foi apenas possível nas imagens captadas pela webcam quando foi usado como material de anteparo os sacos plásticos brancos de

polietileno. Utilizando o papel vegetal, a imagem do fiducial era distorcida e borrada, inviabilizando a sua detecção. Outra limitação da utilização dos fiduciais foi a distância que a câmera deveria estar da superfície do acrílico para reconhecer o fiducial. Utilizando uma câmera convencional de resolução de 640px por 480px, com ângulo de visão de 45º, essa distância era de 25 cm.

A essa distância é inviável a execução de qualquer aplicação, pois a área útil da mesa ficaria em torno de 625cm² (25 cm x 25 cm); muito pouco quando comparado à área total do acrílico que é de 19200cm². Este teste não foi refeito utilizando a webcam VX 6000, pois esta apenas esteve disponível após grande parte do desenvolvimento do jogo baseando-se apenas em toques.

Com base nestes testes realizados e nas limitações encontradas, o uso de fiduciais foi descartado da concepção do projeto e decidimos apenas utilizar os toques como modo de interação.

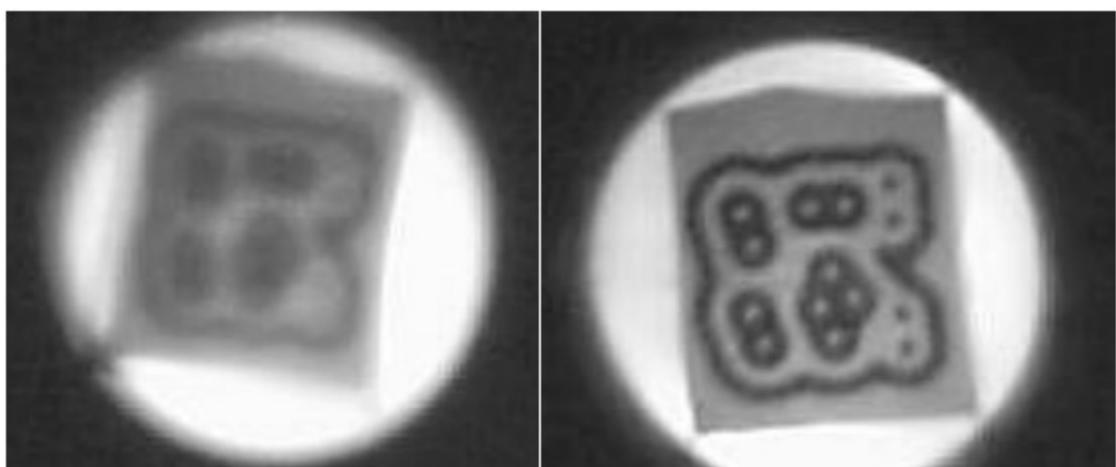


Figura 50 - Fiduciais sobre papel vegetal e saco plástico

Em relação ao toque não houve problemas de detecção utilizando anteparos difusores, pois em ambos os materiais (papel vegetal e saco plástico) foram detectados corretamente com a *webcam* convencional. Utilizando a VX 6000, a distância em relação à superfície do acrílico em que utilizamos a *webcam* foi de 85 cm. A esta distância a área útil da mesa ficou em torno de 5720cm² (65 cm x 88 cm).

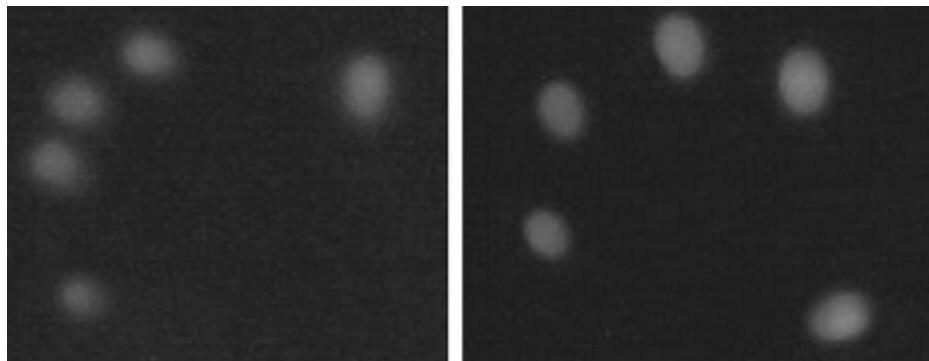


Figura 51 - Toque sobre papel vegetal e saco plástico

3.3. Jogo

Uma das primeiras etapas no desenvolvimento do jogo foi escolher qual seria o ambiente de desenvolvimento. 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 pelo fato deste utilizar uma arquitetura de comunicação baseada no protocolo *TUIO* juntamente com o *OSC*, através da biblioteca, *oscpack*; o jogo poderia ser desenvolvido em praticamente qualquer ambiente. Isso devido ao fato de existir diversas implementações do protocolo *OSC*, em diversas plataformas e linguagens de programação.

Dessa forma, a escolha foi baseada apenas em qual ambiente a produtividade seria maior e qual proveria mais recursos, como controle de versões, gerenciadores de conteúdo e linguagens com suporte a programação orientada a objetos. 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 *IDE* de desenvolvimento seria o *Microsoft Visual Studio* e linguagem de programação adotada seria *C#*, bastante conhecida dos integrantes do grupo.

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 o software que controla a detecção dos toques sobre a mesa (*Touchlib*) 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, sem alterá-lo.

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 informações contidas nas mensagens OSC+TUIO e dispara os eventos, é executado em uma *thread* apartada; aumentando o desempenho do módulo de comunicação.

3.3.1. Protótipo

O protótipo foi desenvolvido com o intuito de validar as tecnologias empregadas na concepção do projeto. Teve como foco de desenvolvimento a transmissão e reconhecimento das mensagens OSC+TUIO, geração de eventos no software em decorrência de ações realizadas na mesa pelos usuários; e criação de um simples protótipo de jogo utilizando o *Microsoft XNA 2.0*. Desta forma, após o desenvolvimento do protótipo, estaríamos seguros quanto à escolha dos softwares escolhidos para o reconhecimento dos toques, bem como do *framework* utilizado no desenvolvimento do jogo.

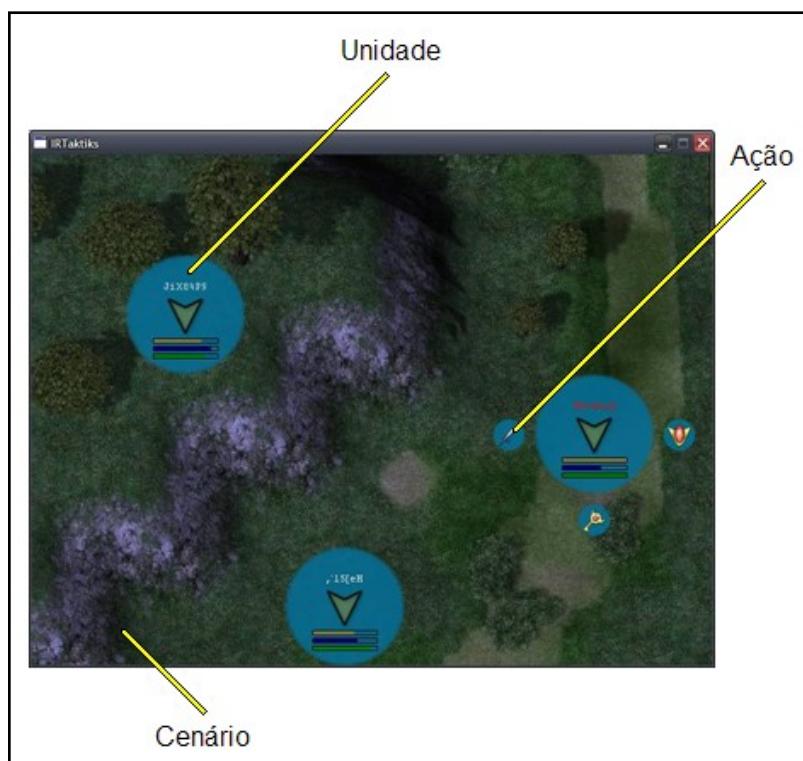


Figura 52 - Protótipo

O desenvolvimento deste protótipo foi de extrema importância, pois possibilitou uma visão geral do problema que seria desenvolver um jogo com inúmeras características, como por exemplo, atributos de personagens, classes, efeitos, animações, cenários, entre outros.

As funcionalidades do protótipo eram baseadas na interação entre as unidades. Cada unidade era representada por um fiducial do *ReacTIVision* que eram posicionadas sobre o acrílico da mesa, sem material difusor, de modo a permitir a detecção. A interação com as ações da unidade, como atacar e usar itens, eram através do toque. Os fiduciais eram detectados pelo *ReacTIVision* e os toques pelo *Touchlib*.

Ao desenvolver de maneira simples, a interação entre as unidades, verificou-se que a formação de times, adição de inúmeras características e ações diferentes para as unidades, além de um cenário tridimensional e animações seriam custosas em complexidade e tempo, caso a arquitetura do jogo não fosse bem planejada.

Os testes realizados em cima do protótipo se mostraram satisfatórios, como era de se esperar, devido à insignificante complexidade do mesmo. O tempo de resposta para o reconhecimento de uma ação foi o critério adotado para os testes. Utilizando cinco fiduciais, as ações sobre estes, como movimentação e rotacionamento foram instantâneos. O toque e a projeção sobre o anteparo difusor, devido a não reestruturação da mesa a tempo, não foram escopo do teste realizado.

A base da arquitetura do protótipo é similar à adotada na versão final e será explicada a seguir.

3.3.2. Versão Final

O desenvolvimento da versão final teve como foco principal sua arquitetura. Foi projetada de modo a deixar o jogo o mais leve e rápido possível, além de possibilitar a agregação de novas funcionalidades rapidamente e de maneira robusta. Utilizando reuso de módulos, processamento da placa de vídeo em conjunto com o do computador, cachê de texturas e imagens, e atualizações de componentes somente quando necessário; obteve-se velocidade de execução e manutenção, sem comprometer a qualidade dos 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 53 - 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.

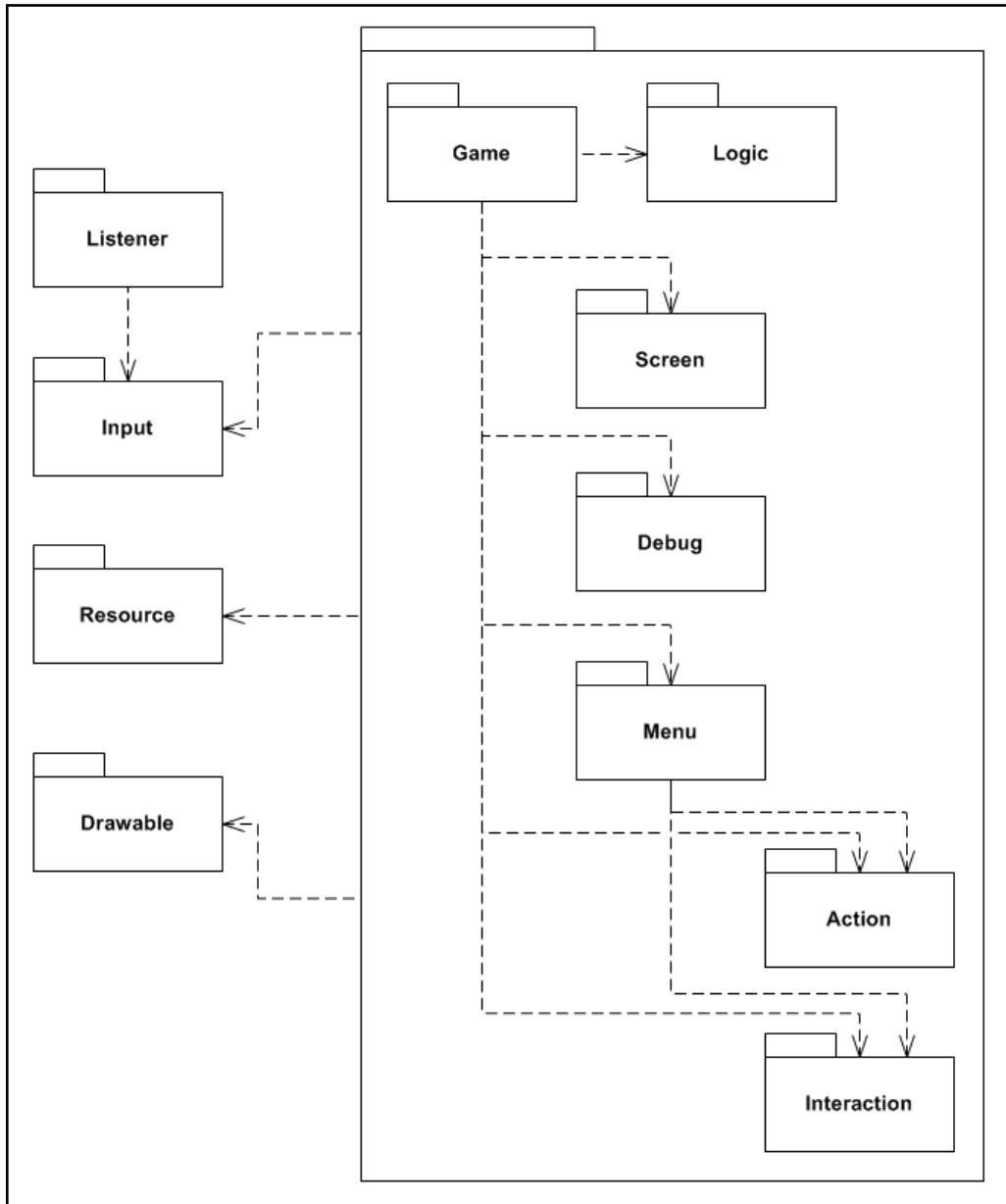


Figura 54 - Arquitetura da versão final

O módulo *Listener* é responsável pela decodificação das mensagens OSC+TUIO, enviando as informações para o módulo *Input*, que dispara os eventos de adição, movimentação e remoção de dedos sobre a mesa. O módulo *Resource* efetua o carregamento dos recursos gráficos que serão desenhados pelo módulo *Drawable*, como texturas, imagens, partículas, efeitos e fontes.

O módulo *Game* é a representação dos objetos do jogo, como os jogadores e suas unidades. Cada unidade possui características que são descritas pelo módulo *Logic*, ações que são implementadas no módulo *Action* e menus que são construídos pelo módulo *Menu*. A interação entre as ações e os menus é realizada pelo módulo *Interaction*.

O módulo *Screen* implementa as várias telas do jogo e suas transições, enquanto o módulo *Debug* é utilizado para auxiliar o desenvolvimento e testes de novas funcionalidades.

3.3.2.1. 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.

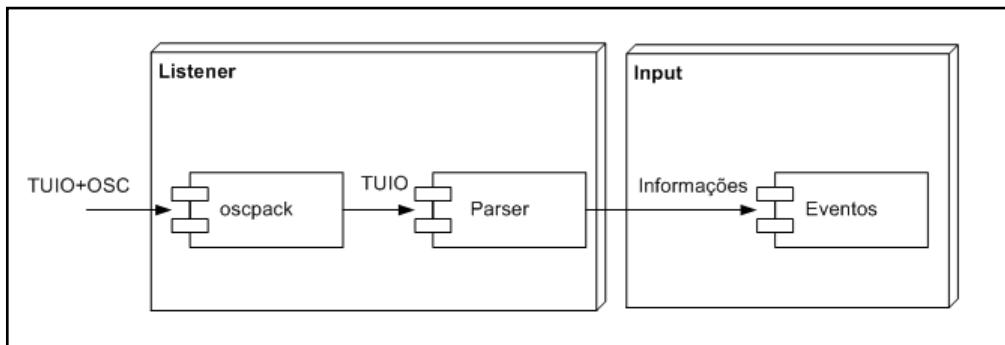


Figura 55 - Relação entre os módulos *Listener* e *Input*

As mensagens *TUIO* possuem informações sobre cada um dos toques e objetos que estão sobre a mesa. 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 do 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.

3.3.2.2. 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 consequentemente 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.

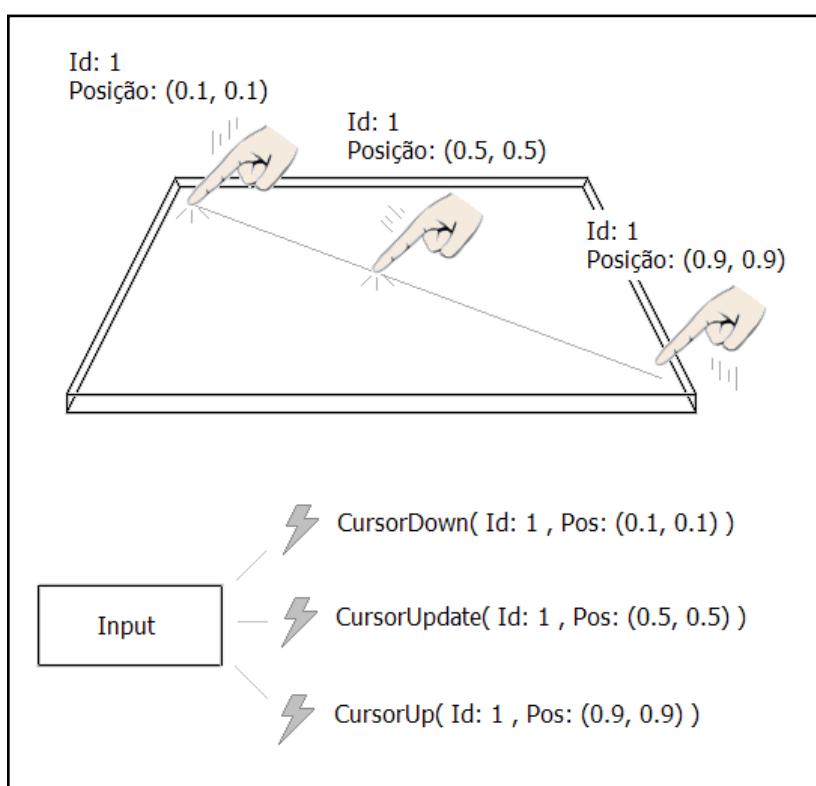


Figura 56 - Exemplo de eventos do módulo Input

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*.

3.3.2.3. 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 *PixelShader¹¹* 3.0 e *VertexShader¹²* 3.0. Placas de vídeo com suporte nativo ao *DirectX¹³* 9.0c, possuem esta característica.

¹¹ 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.

¹² 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.

¹³ Coleções de *Application Programming Interfaces* (API) que auxiliam na execução de tarefas multimídia, como jogos, áudio e vídeos; na plataforma *Microsoft*.

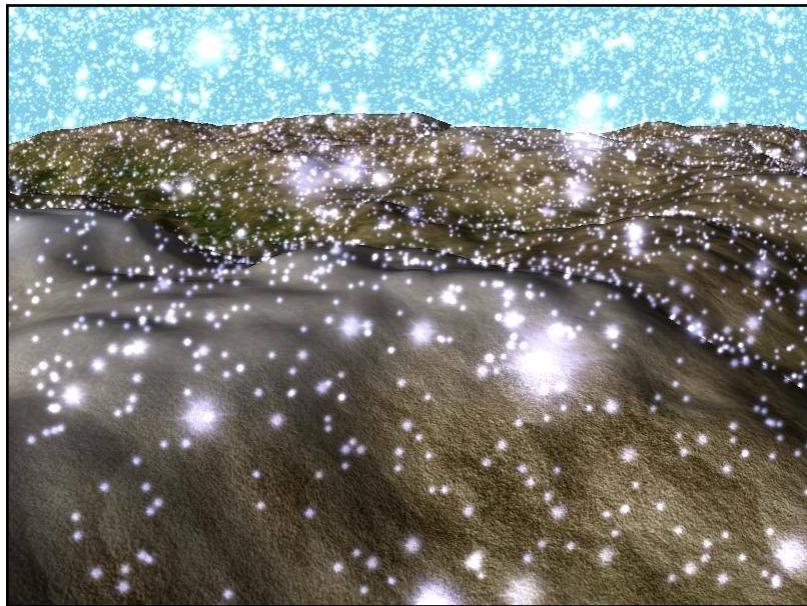


Figura 57 - Exemplo de utilização de efeitos hsl

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.

IRTaktiks - RPG Tatico para mesas multitoque
Centro Universitario Senac

Figura 58 - 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*¹⁴ do framework XNA.

¹⁴ Módulo do *framework* XNA responsável por carregar arquivos externos, como texturas, sons, modelos 3D, entre outros, para dentro do jogo.



Figura 59 - 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 60 - 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.

3.3.2.4. 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.

3.3.2.4.1. 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.

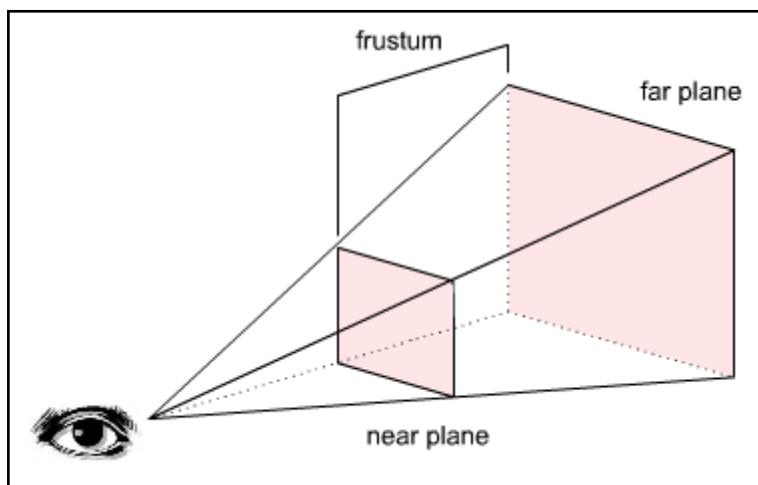


Figura 61 - Representação da área visível da cena

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.

3.3.2.4.2. 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 62 - Exemplo de sobreposição de sprites

3.3.2.4.3. 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.

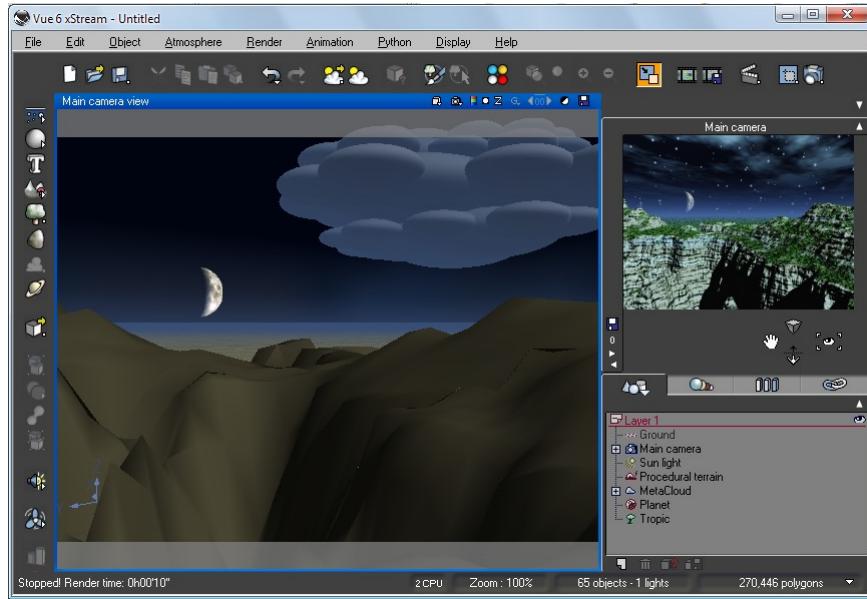


Figura 63 - Software *Vue xStream* 6

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, para evitar atrasos no andamento do projeto. 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). Utilizando arquivos de geometria menores, a qualidade do terreno seria comprometida, sendo somente recuperada com uso de efeitos e texturas, o que não foi conseguido.

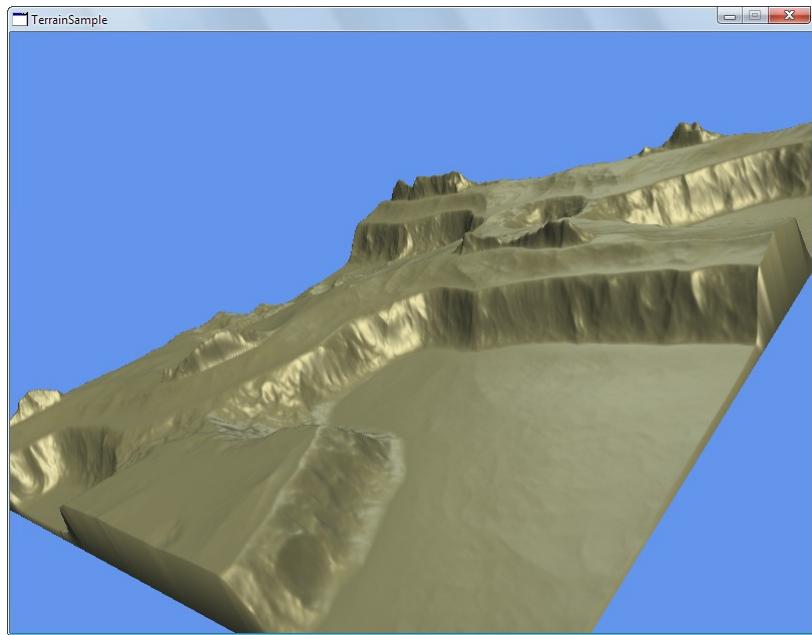


Figura 64 - Mapa utilizando arquivo de geometria (40Mb)

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*¹⁵. O tamanho do arquivo determina o tamanho do terreno que é gerado, enquanto cada pixel determina um vértice do terreno. A altura de cada um destes vértices é dada pela intensidade da cor branca do pixel em questão. O jogo lê o arquivo, criando os vértices para cada pixel 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*.

¹⁵ http://www.e-onsoftware.com/products/vue/vue_6_xstream/

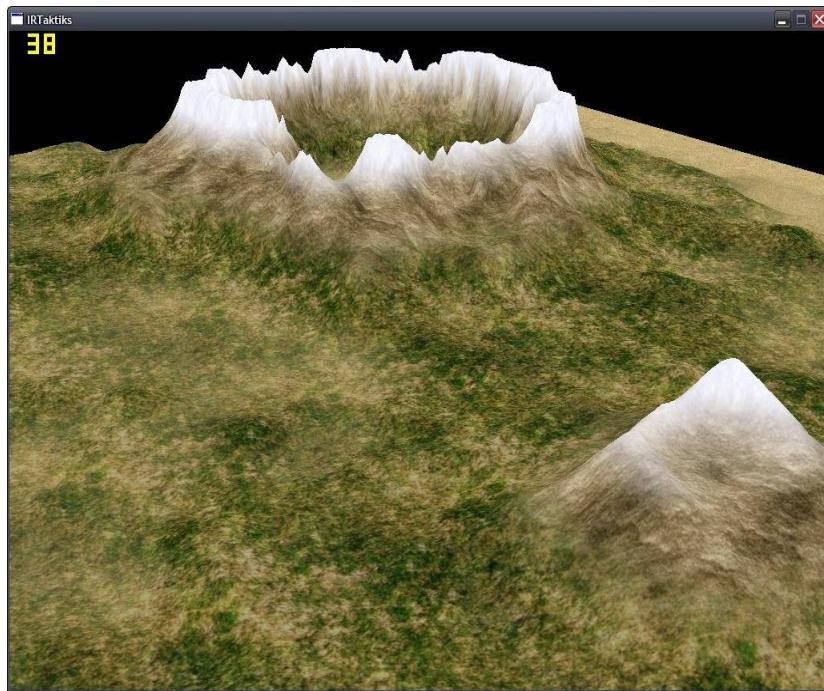


Figura 65 - Mapa usando heightmap e efeito hlsi de mesclagem

3.3.2.4.4. 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 66 - Exemplo de utilização de áreas

Da mesma maneira que o mapa, também utiliza um efeito *hsl* 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 67 - Áreas com e sem suavização

3.3.2.4.5. 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 se 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.

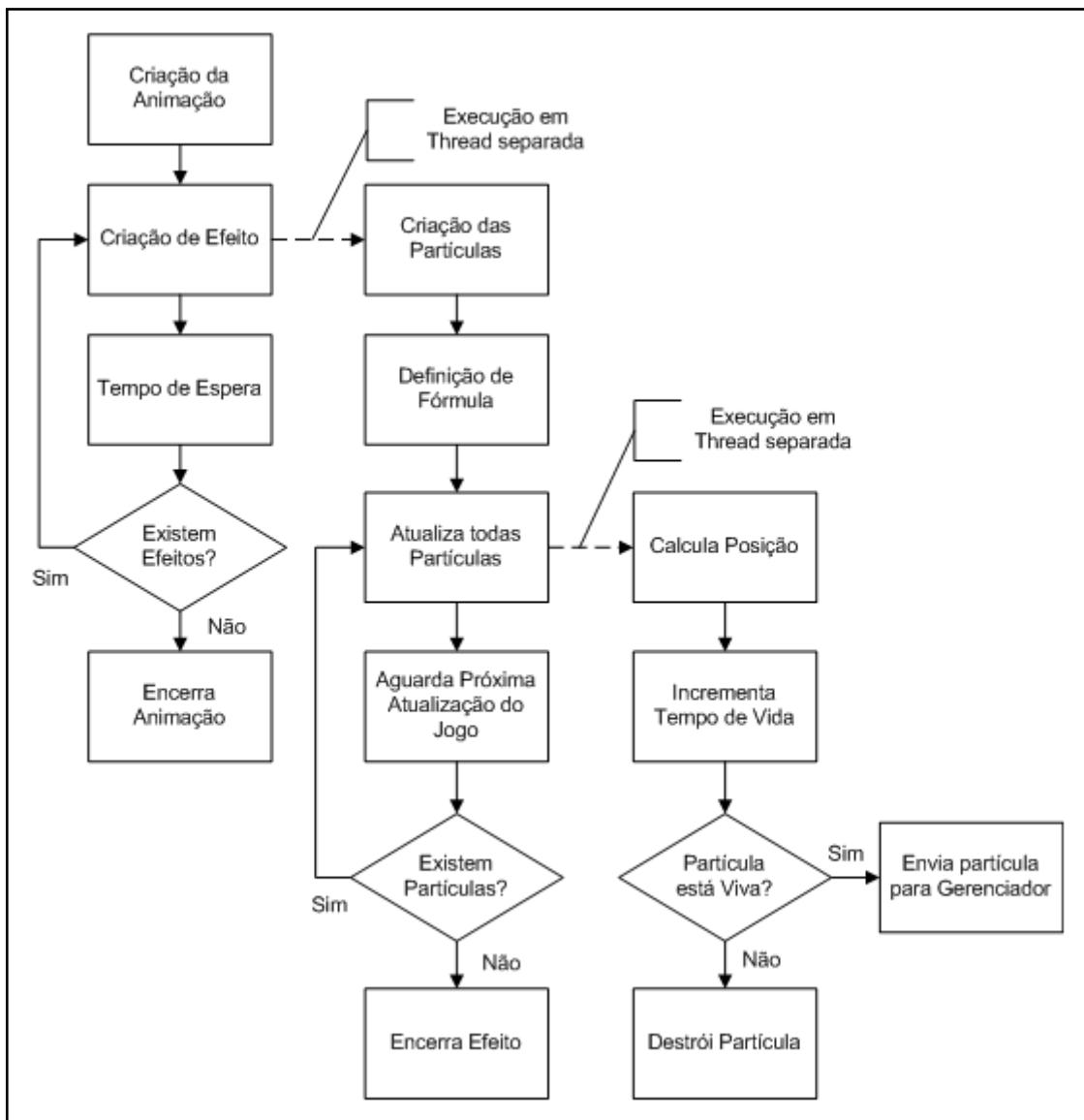


Figura 68 - Fluxo de execução de uma animação

O gerenciador de partículas utiliza ainda, um efeito *hls* 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.

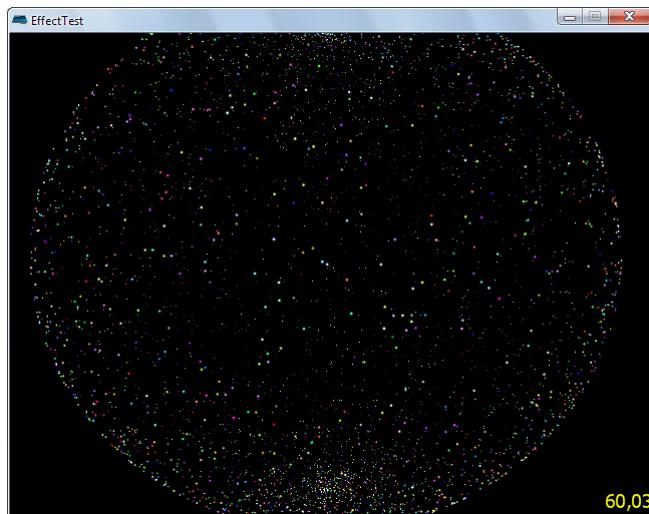


Figura 69 - Exemplo de efeito de partículas

3.3.2.4.6. 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 70 - Exemplo de exibição de informações

3.3.2.5. 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.

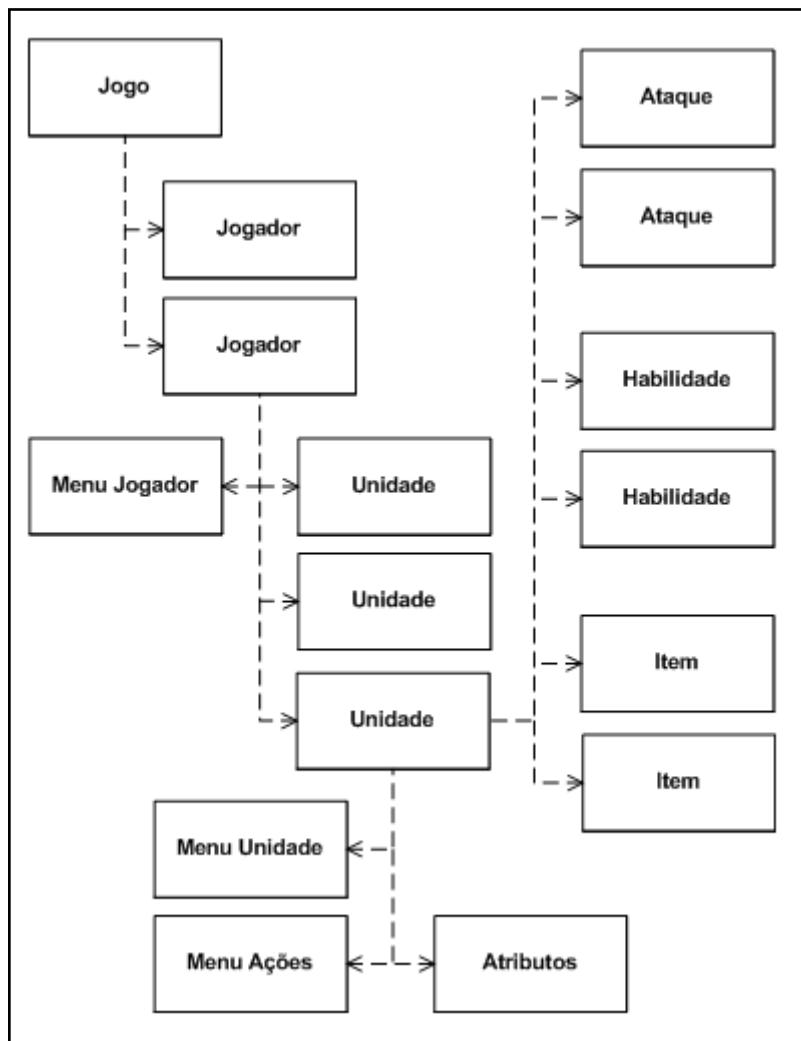


Figura 71 - Estrutura organizacional do jogo

Em resumo, o jogo possui dois jogadores. Cada um destes jogadores possui várias unidades e um menu responsável por exibir as informações de seu jogador. Cada unidade, por sua vez, possui características (atributos) e diversos ataques, habilidades, itens e dois menus. O primeiro responsável por exibir as informações e características da unidade e o segundo suas ações (ataques, habilidades e itens).

3.3.2.6. Módulo Logic

Responsável pela representação lógica das unidades, como por exemplo, 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 exibidos na tabela abaixo:

Atributos	Descrição
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

Tabela 1 - Atributos calculados

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, mostradas na tabela abaixo:

Classe	Habilidades	Bônus
Cavaleiro	Ataques Físicos	Ataques Físicos / Pontos de Vida
Paladino	Cura / Sacrifício	Pontos de Vida / Defesa Física e Mágica
Bruxo	Ataques Mágicos	Ataques Mágicos / Destreza / Pontos de Mana
Sacerdote	Proteção / Cura	Pontos de Mana / Defesa Física e Mágica
Assassino	Ataque Físico / Críticos	Ataques Físicos / Tempo de Espera
Monge	Ataques Físicos e Mágicos	Ataques Físicos e Mágicos

Tabela 2 - Classes disponíveis e características

3.3.2.7. 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.



Figura 72 - Tela de título

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.

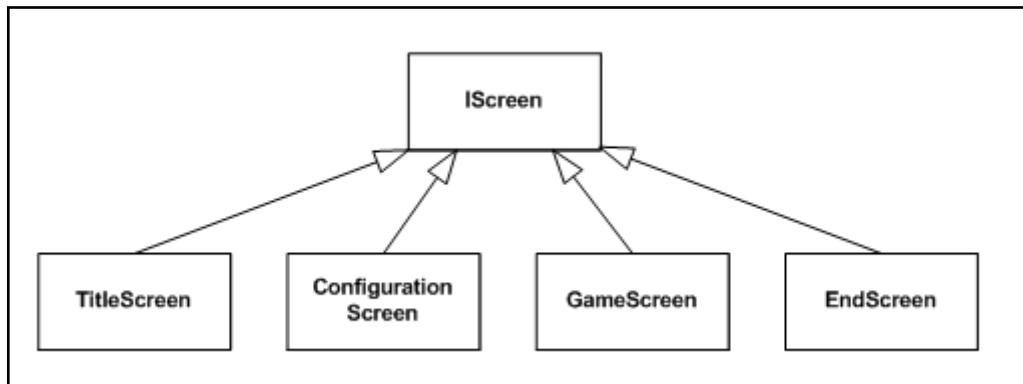


Figura 73 - Generalização de telas

3.3.2.8. 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.

3.3.2.9. 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 74 - 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 75 - 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.

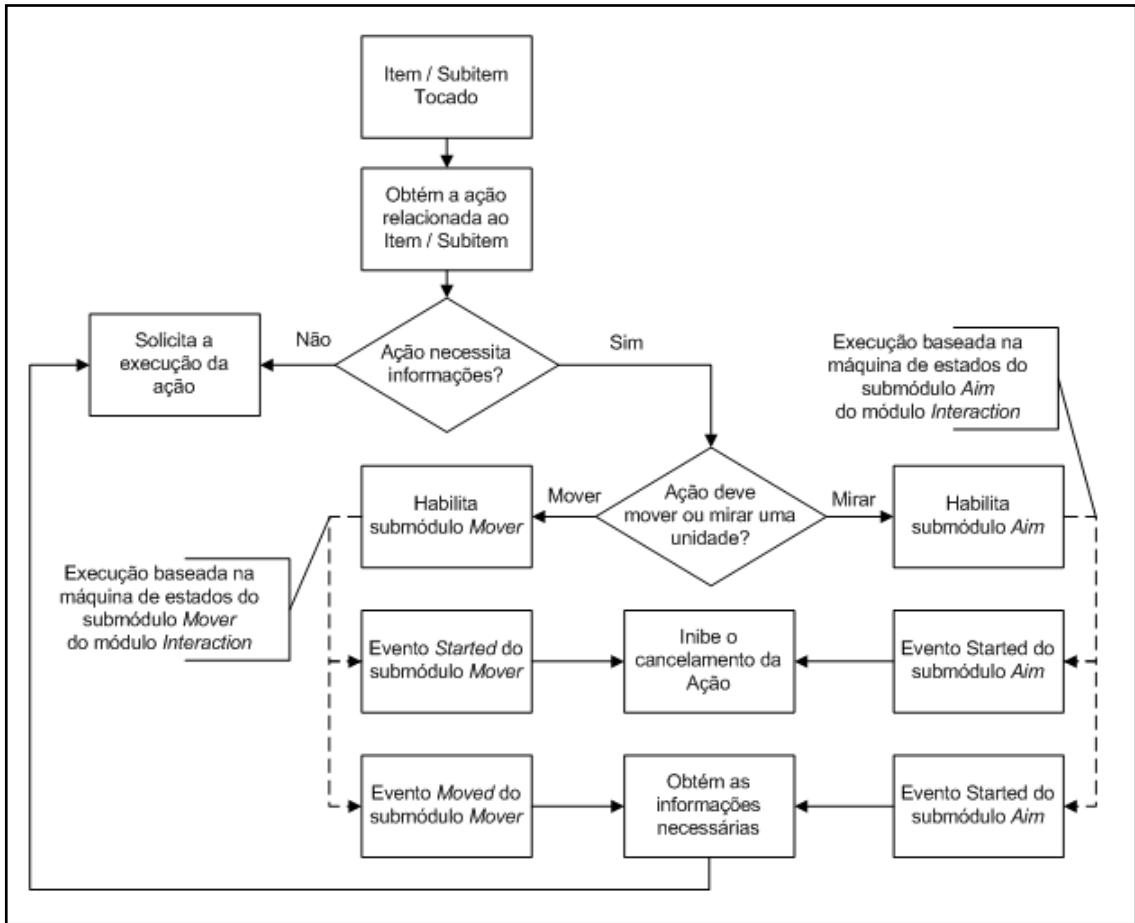


Figura 76 - Fluxo de execução de uma ação através do menu

3.3.2.10. 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 ítem 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.

3.3.2.10.1. 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 77 - 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.

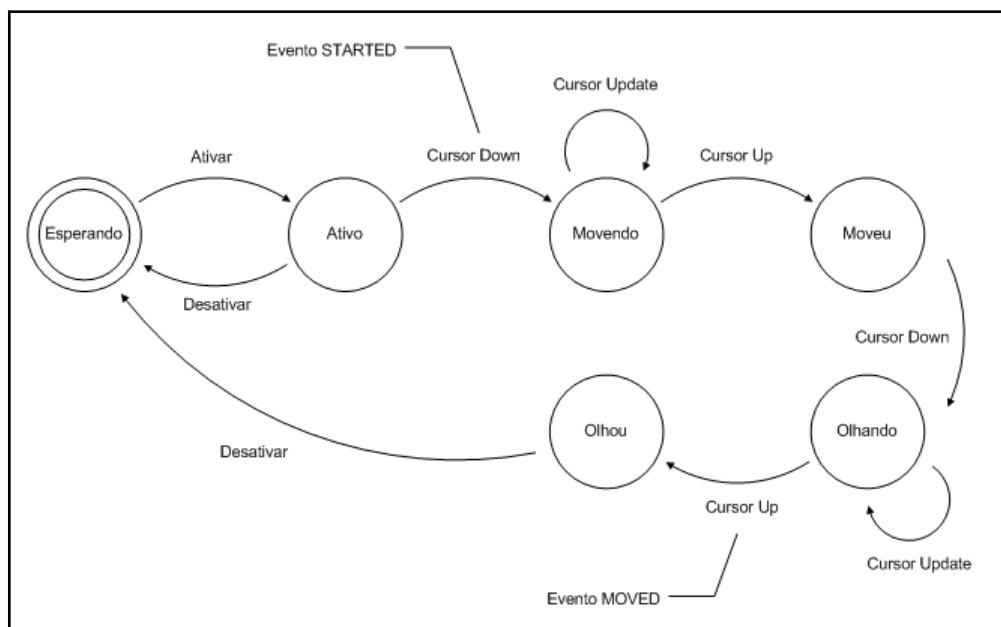


Figura 78 - Máquina de estados do submódulo Mover

3.3.2.10.2. 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.

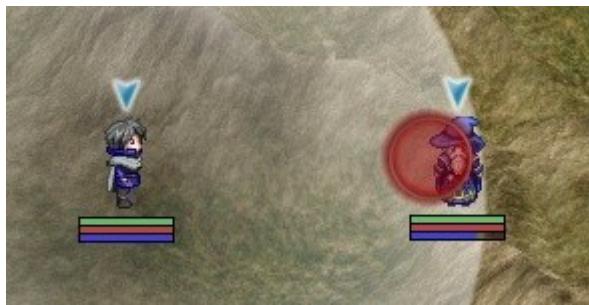


Figura 79 - Mira sobre uma unidade inimiga

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.

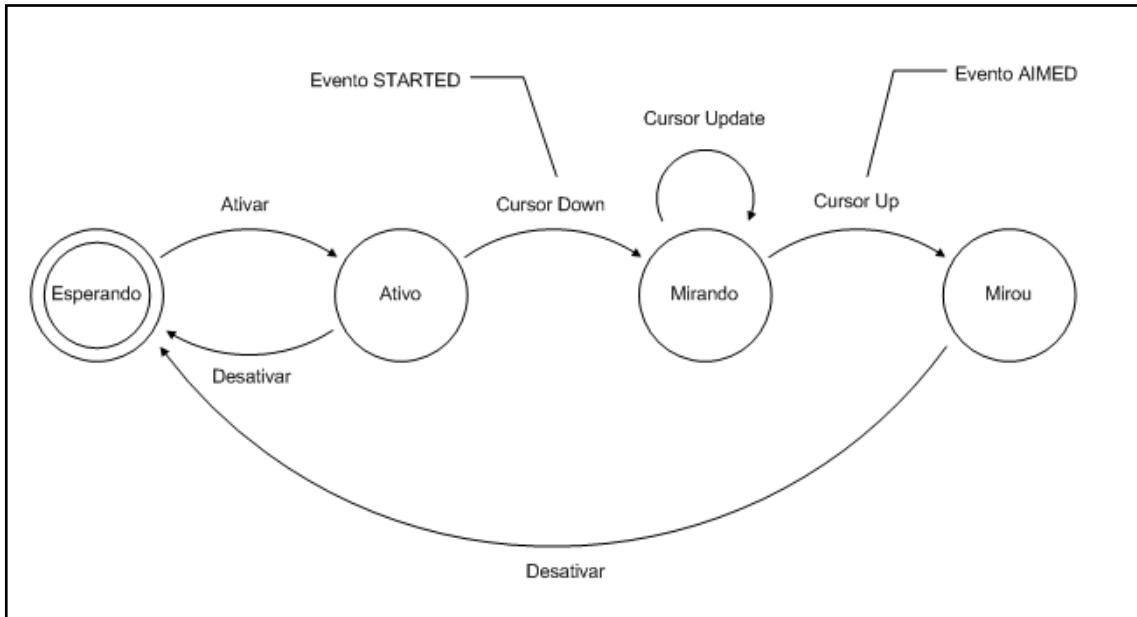


Figura 80 - Máquina de estados do submódulo Aim

3.3.2.11. 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 desta 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.

```
delegate void ExecuteD(Action action, Unit caster, Unit target, Vector2 position);
```

Tabela 3 - Assinatura do *delegate* utilizado

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.

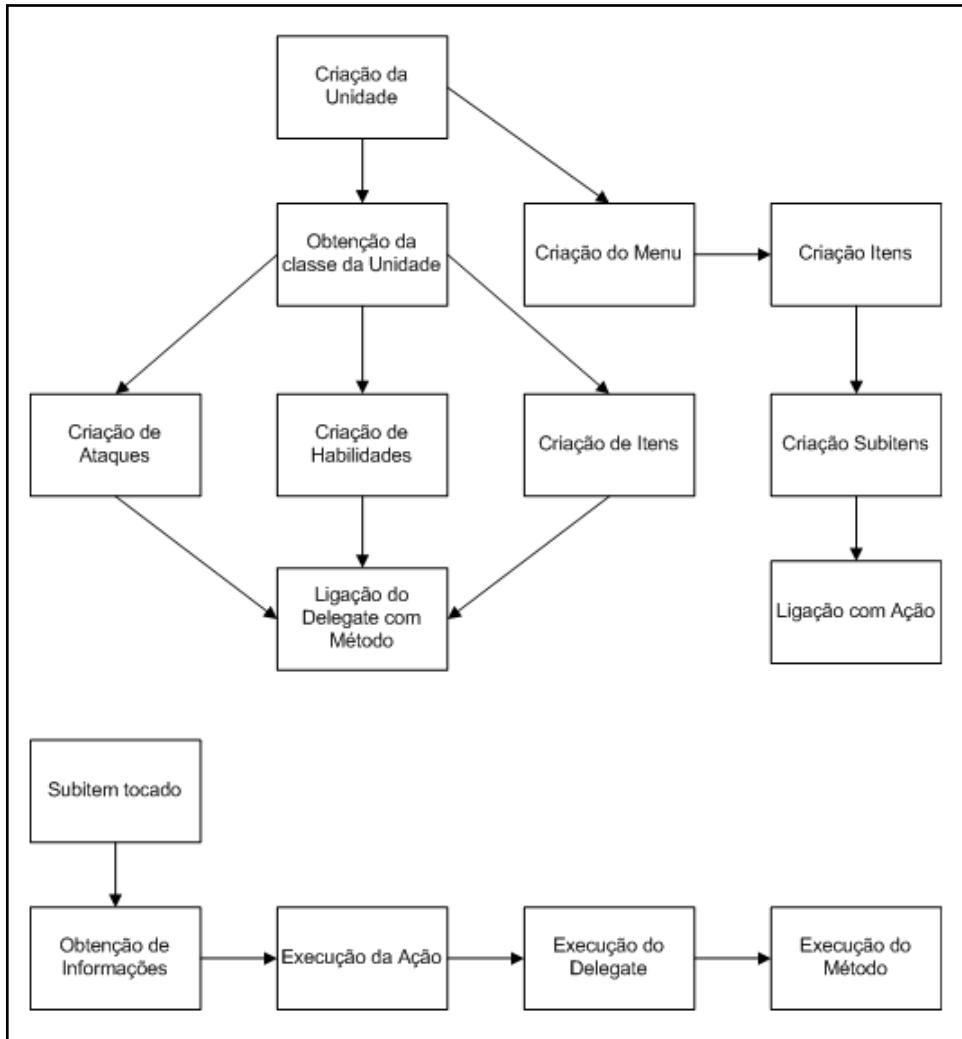


Figura 81 - Diagrama de criação e execução de ações

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 consequências 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 consequências 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.

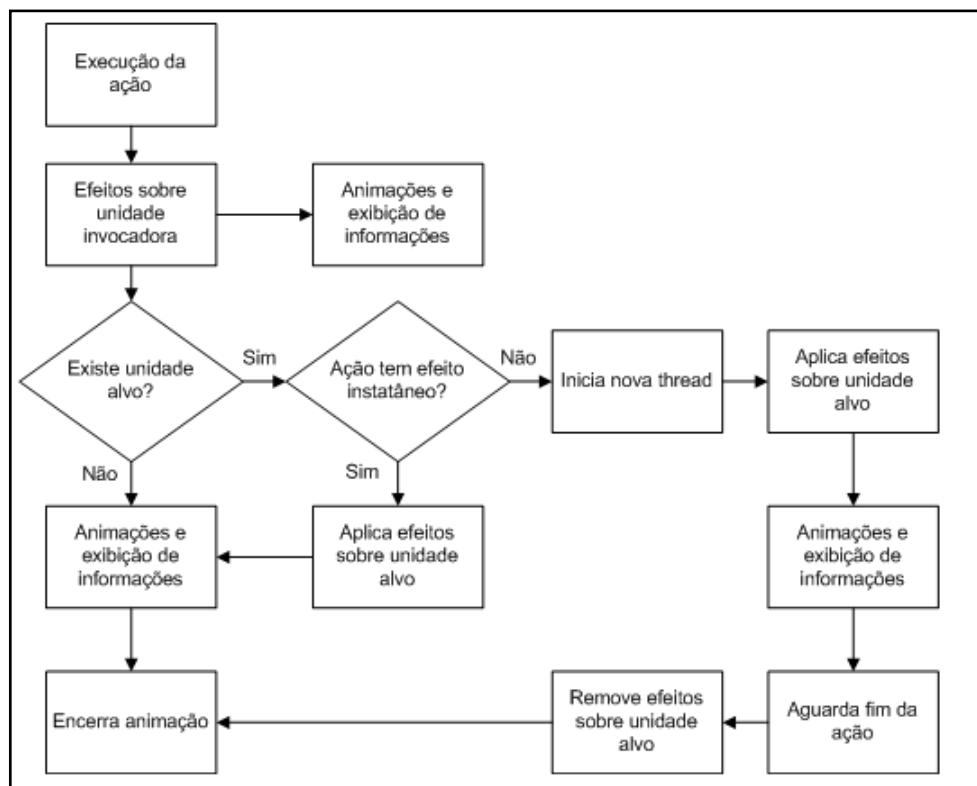


Figura 82 - Diagrama de execução de ações

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 a ela for tocado.

4. RESULTADOS

Neste projeto foi elaborado um protótipo para validação tecnologias de interação com superfícies, como a exploração de técnicas de captura de toques, reconhecimento de marcadores e comunicação entre visão computacional e jogo. Com base nos resultados e com a experiência adquirida na construção do protótipo foi desenvolvido a versão final do *IRTaktiks*.

O *IRTaktiks* é jogado por dois jogadores competindo entre si, que terão a disposição unidades de combates customizáveis, o controle destas unidades é feita por *gestures*, para mover basta tocar a unidade e arrastar o dedo pela superfície, com o toque também são feitas as seleções de menu, estas ações podem ser feitas paralelamente por ambos jogadores. Com a adequação da mesa, toques mais suaves podem ser reconhecidos, permitindo mais naturalidade do usuário.

O posicionamento dos personagens é um fator muito importante para a estratégia dos jogadores, afeta o campo de visão e o alcance de habilidades especiais e magias. Quando selecionada uma opção de ação, um círculo representando a efetividade desta é mostrado, quando o personagem se encontra em uma altitude elevada, o raio daquele maior do que se estivesse em menores altitudes.

Quando uma opção de ataque ou de auxilio selecionada, um círculo pequeno é visível sobre o personagem, a mira, este circulo deve ser arrastado sobre o alvo da ação, dentro da área de alcance como explicada anteriormente. Se a mira é arrastada sobre um aliado, a mira ficará verde, caso em um inimigo, ficará vermelha. Um ataque efetuado aparecerá um numero vermelho sobre o personagem, este numero representa a quantidade de pontos de vida que o alvo perde, no caso de auxilio aparecerá um numero verde, que é a quantidade de pontos de vida que o alvo ganha. Se houver a sigla MP após os números, serão afetados só pontos de mana do personagem e não os de vida. Quando acabar os pontos de vida de uma unidade ela desmaia, se um jogador tiver todas as suas unidades desmaiadas ele perde o jogo.

4.1. Trabalhos Futuros

Durante o desenvolvimento do trabalho, encontramos diversos enfoques para o trabalho, que poderiam ter sido exploradas, porém distanciando-nos do objetivo principal, que é desenvolver um jogo de *RPG* tático para interfaces multi-toque. Estes enfoques surgiram, principalmente, das dificuldades encontradas.

Destas idéias, as mais relevantes e que influenciarão projetos futuros, tanto na área acadêmica, quanto comercial são: Dinamismo em Visão e Projeção e Processos Interativos Multi-toque.

4.1.1. Dinamismo em Visão e Projeção

A idéia deste trabalho é estudar a interação multi-toque em diversas superfícies, com altura e largura variadas, ângulos de tangibilidade (paredes, mesas, tetos); além de diferentes materiais de projeção. A idéia deste trabalho surgiu durante os testes de projeção sob a superfície da mesa.

Através destes testes, buscávamos a melhor combinação entre: projetor, espelho e *webcam*, de forma a cobrir a maior parte possível da mesa; além de um anteparo difusor, de modo a obter a melhor visão possível (computacional ou não) do conjunto: contraste formado pelo toque, fiduciais e jogo.

O resultado deste trabalho deverá ser uma classificação das melhores práticas em visão computacional e projeção para interfaces multi-toque. Deverá abranger diferentes técnicas de construção de interfaces multi-toque, como: *Diffused Illumination*, *FTIR*, sensores de pressão e *grids*; além de diferentes cenários de interação através de mesas, chão, paredes, tetos e objetos não uniformes.

Cada classificação deverá resultar em um mini-protótipo aplicativo simples, a fim de demonstrar as boas práticas para cada cenário de interação.



Figura 83 - Interação multi-toque em superfícies verticais

4.1.2. Processos Interativos em Superfícies Interativas

Uma superfície multi-toque de um único usuário se diferencia de uma para múltiplos usuários, através da gerência das interações, ou seja, através do controle das interações dos diversos usuários sobre sua superfície. Esse controle deve garantir que a utilização de um recurso, por um usuário, não comprometa a utilização de outros, pelo restante dos usuários.

No *IR Taktiks*, por exemplo, um jogador não consegue atacar uma unidade adversária, se nesse mesmo instante, o outro jogador estiver com o dedo sobre a unidade alvo. Condições como essa são evitadas através da boa conduta dos jogadores, porém devem ser pensadas e levadas em consideração durante o desenvolvimento da aplicação.

O resultado dessa pesquisa será um estudo sobre práticas interativas e reconhecimento de gestos, de maneira a descrever quais os melhores tipos de interações para superfícies multi-toque. Exemplos dessas práticas são: utilização de dedos para a ampliação de imagens ou utilização da palma da mão para a seleção de área.

Deverá relacionar quais devem ser utilizadas em relação à quantidade de usuários, quantidade de dedos utilizados, de marcadores, ângulo da superfície de interação e quantidade de elementos interativos projetados. Estes cenários de utilização deverão ser demonstrados com mini-protótipos funcionais (aplicativos), demonstrando as práticas utilizadas.

4.2. Conclusão

Durante a realização deste trabalho, percebeu-se da importância de uma adequada gerência nas interações dos diversos usuários da interface multi-toque. Ignorar este quesito no desenvolvimento de uma aplicação para este fim pode transformar a interação, que deveria ser fácil e natural, em algo difícil e cansativo.

As principais características definidas durante a concepção do jogo foram atendidas pela versão final. Dois jogadores controlam suas respectivas unidades em batalhas, onde as características únicas de cada unidade, aliadas à tática são os fatores decisivos para derrotar o adversário.

A mesa multi-toque previamente desenvolvida foi completamente reestruturada, melhorando sua capacidade e qualidade na detecção de toques. A definição de softwares e protocolos para a comunicação com aplicativos a completaram, de modo a torná-la um módulo, passível de uso por qualquer aplicação multi-toque desenvolvida com base nos protocolos e padrões estabelecidos.

A utilização da área total da superfície da mesa para interação com o jogo não foi possível devido à falta de anteparos difusores que cobrisse sua extensão. A indisponibilidade do material ideal para esse tipo de superfície nos levou a alternativas que não proporcionaram a máxima qualidade na projeção e detecção de toques, limitando o uso total de sua superfície.

Graças ao tema escolhido e a abrangência deste novo modo de interação, este trabalho pretende contribuir com o desenvolvimento e popularização desta tecnologia; e através dela, permitir sua aplicação em diversos setores comuns da vida humana.

5. REFERÊNCIAS BIBLIOGRÁFICAS

HAN, J. Y. Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection. In: **Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology**, 2005.

BUXTON, Bill. **Multi-Touch Systems that I Have Known and Loved**. Disponível em: <<http://www.billbuxton.com/multitouchOverview.html>> Acesso em: 19 de junho de 2008.

WIKIPEDIA. **Microsoft Surface**. Disponível em: <http://en.wikipedia.org/wiki/Microsoft_Surface>. Acesso em: 19 junho 2008.

MICROSOFT. **MICROSOFT SURFACE**. Disponível em: <<http://www.microsoft.com/surface/index.html>> Acesso em: 19 junho 2008.

ZMOGINSKI, Felipei. **Plantão Info**: Microsoft demonstra multitoque do Windows 7. Disponível em: <<http://info.abril.com.br/aberto/infonews/052008/28052008-0.shl>> Acesso em: 19 junho 2008.

MICROSOFT. **Windows Vista Blog**: Microsoft demonstrates Multi-touch. Disponível em: <<http://windowsvistablog.com/blogs/windowsvista/archive/2008/05/27/microsoft-demonstrates-multi-touch.aspx>> Acesso em: 19 junho 2008.

MICROSOFT. **AT&T First to Introduce Microsoft Surface in Retail Stores to Enhance Mobile Shopping Experience**. Disponível em: <<http://www.microsoft.com/presspass/press/2008/apr08/04-01SurfaceRetailPR.mspx>> Acesso em: 19 junho 2008.

WIRED. **ReacTable Tactile Synth Catches Björk's Eye -- and Ear**. Disponível em: <http://www.wired.com/entertainment/music/news/2007/08/bjork_reacTable> Acesso em: 19 de junho de 2008.

COOLEST GADGETS. **Björk Showcases reacTable on her New World Tour**. Disponível em: <<http://www.coolest-gadgets.com/20070509/bjork-showcases-reactable-on-her-new-world-tour/>> Acesso em: 19 de junho de 2008.

MUSIC TECHNOLOGY GROUP, Pompeu Fabra University. **Reactable**. Disponível em: <<http://reactable.iua.upf.edu/>> Acesso em: 19 de junho de 2008.

WIKIPEDIA. **ReacTable**. Disponível em: <<http://en.wikipedia.org/wiki/ReacTable>>. Acesso em: 19 junho 2008.

FILE. **Interactive Sonic Systems Team**. Disponível em: <http://www.filefestival.org/site_2007/pop_trabalho.asp?id_trabalho=1839&cd_idioma=1&acao=visualizar>. Acesso em: 19 de junho de 2008.

MACWORLD SÃO FRANCISCO. **Apple Reinventa Telefone com o iPhone.** Disponível em: <<http://latam.apple.com/pr/articulo/?id=1361&r=br>> Acesso em: 19 de junho de 2008.

APPLE. **iPhone: Using the Multi-touch Display.** Disponível em: <<http://support.apple.com/kb/HT1636>> Acesso em: 19 de junho de 2008.

COMO TUDO FUNCIONA. **Como funciona o iPhone.** Disponível em: <<http://eletronicos.hsw.uol.com.br/iphone.htm>> Acesso em: 19 de junho de 2008.

WIZARDS. **Dungeons & Dragons.** Disponível em: <<http://www.wizards.com/default.asp?x=dnd/dragon>> Acesso em: 19 de junho de 2008.

WIKIPEDIA. **Dungeons & Dragons.** Disponível em: <http://en.wikipedia.org/wiki/Dungeons_&_Dragons>. Acesso em: 19 junho 2008.

KESTREL, Gwendolyn F.M. **Working Hard at Play.** Disponível em: <<http://www.newhorizons.org/strategies/literacy/kestrel.htm>> Acesso em: 10 de junho de 2008.

WIKIPEDIA. **Zork.** Disponível em: <<http://en.wikipedia.org/wiki/Zork>>. Acesso em: 19 junho 2008.

WIKIPEDIA. **Role Playing Game.** Disponível em: <http://en.wikipedia.org/wiki/Role-playing_game>. Acesso em: 19 junho 2008.

WIKIPEDIA. **Role Playing Game (Video Game).** Disponível em: <[http://en.wikipedia.org/wiki/Role-playing_game_\(video_games\)](http://en.wikipedia.org/wiki/Role-playing_game_(video_games))>. Acesso em: 19 junho 2008.

WIKIPEDIA. **Massively multiplayer online game.** Disponível em: <http://en.wikipedia.org/wiki/Massively_multiplayer_online_role-playing_game>. Acesso em: 19 junho 2008.

WIKIPEDIA. **Final Fantasy Tactics.** Disponível em: <http://en.wikipedia.org/wiki/Final_fantasy_tactics>. Acesso em: 19 junho 2008.

GAME SPOT. **Final Fantasy Tactics Review.** Disponível em: <<http://www.gamespot.com/ps/strategy/finalfantasytactics/review.html>>. Acesso em: 19 de junho de 2008.

IGN. **Final Fantasy Tactics Returns.** Disponível em: <<http://psp.ign.com/articles/750/750839p1.html>>. Acesso em: 19 de junho de 2008.

BIMBER, Oliver; RASKAR, Ramesh. **Spatial Augmented 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, Ronald T. A survey of Augmented Reality. In: **Presence: Teleoperators and Virtual Environments**. v. 6. p 355-385. ago 1997.

MULTIGESTURE.NET. **A Multi-Touch and Multi-Gesture research blog**. Disponível em: <<http://www.multigesture.net/>> Acesso em: 19 de junho de 2008.

WHITE NOISE AUDIO. **White Noise Audio Software**. Disponível em:
<<http://www.whitenoiseaudio.com/>> Acesso em: 19 de junho de 2008.

NUI GROUP. **NUI Group**. Disponível em: <<http://nuigroup.com/>> Acesso em: 19 de junho de 2008.

HAN, J. Y.. **NYU Courant Institute of Mathematical Sciences**. Disponível em:
<<http://www.cs.nyu.edu/~jhan/>> Acesso em: 19 de junho de 2008.

CNMAT, UC Berkeley. **Open Sound Control**. Disponível em:
<<http://opensoundcontrol.org/>> Acesso em: 19 de junho de 2008.

WIKIPEDIA. **Open Sound Control**. Disponível em:
<http://en.wikipedia.org/wiki/OpenSound_Control>. Acesso em: 19 junho 2008.

BENCINA, Ross. **OSCPack**. Disponível em:
<<http://www.audiomulch.com/~rossb/code/oscpack/>>. Acesso em: 19 de junho de 2008.

TUIO. **TUIO: A Protocol for Tangible User Interfaces**. Disponível em:
<<http://tuio.lfsaw.de>>. Acesso em: 19 de junho de 2008.

KALTENBRUNNER, Martin, et All. **TUIO: A Protocol for Table-Top Tangible User Interfaces**.

KALTENBRUNNER, Martin, BENCINA. **reacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction**.

NUIGROUP. **Touchlib: A multitouch Development Kit**. Disponível em:
<<http://nuigroup.com/touchlib>>. Acesso em: 19 de junho de 2008.

BLOG TEAM XNA. **What is the XNA Framework**. Disponível em:
<<http://blogs.msdn.com/xna/archive/2006/08/25/724607.aspx>>. Acesso em: 19 de junho de 2008.

MICROSOFT. **MSDN Library**. Disponível em: <<http://msdn.microsoft.com>> Acesso em: 10 junho 2008.

FEGGELEN, Eric van. **Microsoft XNA Framework; Drawing a complex mesh.** Disponível em:<<http://www.fegelein.com/?p=21>> Acesso em 10 junho de 2008.

MICROSOFT. **XNA Creators Club Online.** Disponível em:
<<http://creators.xna.com/>> Acesso em: 10 junho 2008.

XNA UK USER GROUP. Generic XNA. In: **RandomChaos.** Disponível em:
<<http://xna-uk.net/blogs/randomchaos/>>. Acesso em: 10 junho 2008.

POPULAR MECHANICS. **Microsoft Surface: Behind-the-Scenes First Look.** Disponível em:
<<http://www.popularmechanics.com/technology/industry/4217348.html?page=1>>
Acesso em: 19 de junho de 2008.

6. APÊNDICES

6.1. 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:

lvl: Valor do atributo relacionado ao nível.

str: Valor do atributo relacionado à força.

agi: Valor do atributo relacionado à agilidade.

vit: Valor do atributo relacionado à vitalidade.

int: Valor do atributo relacionado à inteligência.

dex: 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*)

$$maxlife = \left(1 + \left(\frac{vit}{80}\right)\right) \cdot \left(20 + (5 \cdot lvl) + \left(@ \cdot \left((1 + lvl) \cdot \left(\frac{lvl}{2.5}\right)\right)\right)\right)$$

- Pontos de mana total - (*maxmana*)

$$maxmana = \left(1 + \left(\frac{int}{50}\right)\right) \cdot (5 + (@ \cdot lvl \cdot 10))$$

- Ataque - (*atk*)

$$atk = \left(\left(str + \frac{\left(\frac{str}{10}\right)^3 + (2 \cdot lvl)}{2} \right) \cdot @ \right) + \left(\frac{dex}{5} \right)$$

- Defesa - (*def*)

$$def = \left(2.5 \cdot vit + \left(\frac{vit}{3} \right) + \left(\frac{int}{5} \right) + lvl \right) \cdot @$$

- Ataque mágico - (*matk*)

$$matk = \left(\left(\frac{int}{5} \right)^2 + lvl \right) \cdot @$$

- Defesa mágica - (*mdef*)

$$mdef = \left(2.5 \cdot int + \left(\frac{int}{3} \right) + \left(\frac{vit}{5} \right) + lvl \right) \cdot @$$

- Taxa de desvio - (*flee*)

$$flee = 20 + agi$$

- Taxa de acerto - (*hit*)

$$hit = 80 + dex$$

- Alcance de Ataque - (*arange*)

$$arange = 100 + (2 \cdot dex)$$

- Alcance de Habilidade - (*hrange*)

$$hrange = 100 + (1.5 \cdot int) + (1.5 \cdot dex)$$

- Alcance de Movimento - (*mrange*)

$$mrange = 100 + dex + agi$$

- Tempo de espera - (*time*)

$$time = \frac{1}{1000} + \frac{agi}{99000}$$

6.2. 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 4 - Valores dos fatores, em relação às classes e atributos calculados.

6.3. 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 segue¹⁶:

$$\% = c.\text{hit} - t.\text{flee}$$

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.

$$\text{resultado} = \text{resultado} \pm 0.1 \cdot \text{resultado}$$

6.3.1. 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

¹⁶ Para a representação das unidades invocadoras (*caster*) e das unidades alvo (*target*) nas fórmulas, serão utilizados os prefixos “c” e “t”, respectivamente.

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*)

$$short = c.atk - (0.3 \cdot t.def)$$

- Ataque Longo - (*long*)

$$long = (1.5 \cdot c.hit) - (0.3 \cdot t.def)$$

6.3.2. 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.

6.3.3. 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.

6.3.3.1. 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.

6.3.3.2. 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.

6.3.3.3. 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.

6.3.3.4. 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.

6.3.3.5. 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.

6.3.3.6. Revenge

Inflige dano equivalente a 50% de um ataque curto a uma unidade alvo. Porém os alvos 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.

6.3.3.7. 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.

6.3.3.8. 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.

$$dano = c.mathk - (0.3 \cdot t.mdef)$$

6.3.3.9. 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.

$$dano = \left(\left(15 \cdot \left(\frac{c.level}{100} \right) \cdot (c.atk + c.mathk) \right) - \left(\frac{t.def + t.mdef}{2} \right) \right) \cdot \frac{c.mana}{c.maxmana}$$

6.3.3.10. 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.

6.3.3.11. 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.

$$pontos = \frac{c.level + c.int}{8} \cdot 44$$

6.3.3.12. 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.

6.3.3.13. 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.

$$t.\text{def} = 1.5 \cdot t.\text{def}$$
$$t.\text{mdef} = 1.5 \cdot t.\text{mdef}$$

6.3.3.14. 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.

$$\text{dano} = \left(\frac{c.\text{lvl} + c.\text{int}}{6} \cdot 50 \right) - (2 \cdot t.\text{mdef})$$

6.3.3.15. 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.

$$\text{dano} = \left(\frac{t.\text{int}}{c.\text{int}} \cdot \frac{5 \cdot t.\text{maxmana}}{100} \right) + \frac{5 \cdot t.\text{maxmana}}{100}$$

6.3.3.16. 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.

$$\text{dano} = \left(\frac{c.\text{lvl} + c.\text{int}}{6} \cdot 50 \right) - (2 \cdot t.\text{mdef})$$

6.3.3.17. 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.

$$dano = \left(\frac{c.\textit{lvl} + c.\textit{int}}{6} \cdot 7\right) - (2 \cdot t.\textit{mdef})$$

7. ANEXOS

7.1. Arquivo de Configuração Touchlib

```
<?xml version="1.0" ?>
<blobconfig   distanceThreshold="250"
               minDimension="2"
               maxDimension="250"
               ghostFrames="0"
               minDisplacementThreshold="2.000000" />
<bbox   ulX="0.000000"
        ulY="0.000000"
        lrX="1.000000"
        lrY="1.000000" />
<screen>
    <point X="0.000000" Y="0.000000" />
    <point X="160.000000" Y="0.000000" />
    <point X="320.000000" Y="0.000000" />
    <point X="480.000000" Y="0.000000" />
    <point X="640.000000" Y="0.000000" />
    <point X="0.000000" Y="160.000000" />
    <point X="160.000000" Y="160.000000" />
    <point X="320.000000" Y="160.000000" />
    <point X="480.000000" Y="160.000000" />
    <point X="640.000000" Y="160.000000" />
    <point X="0.000000" Y="320.000000" />
    <point X="160.000000" Y="320.000000" />
    <point X="320.000000" Y="320.000000" />
    <point X="480.000000" Y="320.000000" />
    <point X="640.000000" Y="320.000000" />
    <point X="0.000000" Y="480.000000" />
    <point X="160.000000" Y="480.000000" />
    <point X="320.000000" Y="480.000000" />
    <point X="480.000000" Y="480.000000" />
    <point X="640.000000" Y="480.000000" />
</screen>
<filtergraph>
    <dsvlcapture label="dsvlcapture0" />
    <mono label="mono1" />
    <smooth label="smooth2" />
    <backgroundremove label="backgroundremove3">
        <threshold value="50" />
    </backgroundremove>
    <brightnesscontrast label="brightnesscontrast4">
        <brightness value="0.0980392" />
        <contrast value="0.403922" />
    </brightnesscontrast>
    <rectify label="rectify5">
        <level value="25" />
    </rectify>
</filtergraph>
```