



Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Deskworld: Software Simulador de Física para Superfícies com Tela Multi-Toque

Danilo Gaby Andersen Trindade
Victor Sampaio Zucca

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.^a Dr.^a Carla Denise Castanho

Brasília
2011

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Marcus Vinicius Lamar

Banca examinadora composta por:

Prof.^a Dr.^a Carla Denise Castanho (Orientadora) — CIC/UnB

Prof. Dr. Professor I — CIC/UnB

Prof. Dr. Professor II — CIC/UnB

CIP — Catalogação Internacional na Publicação

Trindade, Danilo Gaby Andersen.

Deskworld: Software Simulador de Física para Superfícies com Tela Multi-Toque / Danilo Gaby Andersen Trindade, Victor Sampaio Zucca. Brasília : UnB, 2011.

103 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2011.

1. multi-toque, 2. física, 3. jogo

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília-DF — Brasil



Universidade de Brasília

**Instituto de Ciências Exatas
Departamento de Ciência da Computação**

Deskworld: Software Simulador de Física para Superfícies com Tela Multi-Toque

Danilo Gaby Andersen Trindade
Victor Sampaio Zucca

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.^a Dr.^a Carla Denise Castanho (Orientadora)
CIC/UnB

Prof. Dr. Professor I Prof. Dr. Professor II
CIC/UnB CIC/UnB

Prof. Dr. Marcus Vinicius Lamar
Coordenador do Bacharelado em Ciência da Computação

Brasília, 20 de fevereiro de 2011

Dedicatória

Dedico as pessoas de bom coração.

Agradecimentos

Agradeço as pessoas de bom coração.

Resumo

Este trabalho foca na construção de um jogo interativo para vários jogadores em superfícies com telas sensíveis a vários toques simultâneos.

Palavras-chave: multi-toque, física, jogo

Abstract

This paper focus on the construction of an interactive multiplayer game for multi-touch surfaces.

Keywords: multi-touch, physics, game

Sumário

1	Introdução	1
2	Trabalhos Correlatos	8
2.1	Jogos simuladores de física	8
2.2	Superfícies sensíveis a toque	10
2.3	Jogos para mesas com telas multi-toque	12
3	Fundamentação Teórica	15
3.1	Processamento de Toques	15
3.1.1	Processamento das Imagens	16
3.1.2	Detecção de <i>Blobs</i>	18
3.1.3	Correção da Câmera	19
3.2	Aspectos de mesas multi-toque	20
3.2.1	Tamanho e Resolução	20
3.2.2	Interfaces	21
3.2.3	N Usuários	22
3.2.4	Detecção de Gestos	22
4	Construção da Mesa Multi-toque	26
4.1	Estrutura da Mesa	27
4.2	Superfície	27
4.3	Iluminação Infravermelha	27



4.4	Câmera	28
4.5	Projeto	28

5 *Game design* do *Deskworld* 29

5.1	Requisitos de interface	29
5.2	Conceito	29
5.3	Regras e Objetos	29
5.4	Interface	30

6 Arquitetura e Detalhes de Implementação 31



6.1	Arquitetura do jogo	31
6.1.1	Sistema de Componentes	31
6.1.2	Vantagens em relação ao sistema hierárquico	32
6.1.3	Comunicação entre componentes	33
6.2	<i>Design Patterns</i>	33
6.2.1	<i>Singleton</i>	34
6.2.2	<i>Observer</i>	34

7 Estrutura de implementação do simulador 36

7.1	<i>Game Engine</i>	36
7.2	<i>Bibliotecas de apoio</i>	36
7.2.1	SDL e <i>OpenGL</i>	37
7.2.2	Protocolo <i>TUIO</i>	37
7.2.3	<i>Touchlib</i>	38



Referências 39

A Configuração da *Touchlib* 42

Lista de Figuras

1.1	Jogo para <i>EDSAC OXO</i> [30]	2
1.2	Jogo para um osciloscópio <i>Tennis for two</i> [30]	2
1.3	Jogo para computadores antigos <i>Spacewars!</i> [30]	3
1.4	Console <i>Odyssey</i> da <i>Magnavox</i> [34]	3
1.5	Console <i>Atari 2600</i> da <i>Atari</i> [31]	4
1.6	Console <i>NES</i> da <i>Nintendo</i> [7]	4
1.7	Consoles de última geração. Da esquerda pra direita: <i>Wii</i> , <i>Playstation 3</i> e <i>Xbox 360</i> [13]	5
1.8	Controle para <i>Nintendo Wii</i> <i>Wiimote</i> [12]	6
1.9	Controle para <i>Playstation 3</i> <i>Playstation Move</i> [12]	6
1.10	Câmera para <i>Xbox 360 Kinect</i> [12]	6
2.1	Jogo para PC <i>Crayon Physics</i> (imagem retirada de [16])	8
2.2	Jogo para PC <i>Phun</i> [1]	9
2.3	Modo de criação do jogo para <i>PS3 Little Big Planet</i> [19]	10
2.4	Mesa com marcadores fiduciais <i>Reactable</i> [25]	11
2.5	Projeto de mesa multi-toque [5]	12
2.6	Esquema de detecção de toque FTIR [5]	13
2.7	Jogo para mesa multitoque: <i>Ecodefense</i> (imagem retirada de [4])	14
3.1	Diferentes imagens capturadas com iluminações diferentes [17].	16
3.2	Imagem capturada e suas filtragens [4].	17

3.3	Gráfico de P [4].	18
3.4	Gráfico de Q [4].	19
3.5	(a) Imagem da câmera com distorção. (b) Imagem da câmera corrigida. [22].	20
3.6	Matriz A [4].	20
3.7	Visão de um botão tradicional <i>OK</i> em duas visões diferentes [4].	22
3.8	Gesto de <i>Click</i> [33].	23
3.9	Gesto de <i>Drag</i> [33].	23
3.10	Reconhecimento de gesto no jogo <i>Black and White</i> [3].	23
3.11	Gesto de <i>Rotate</i> [33].	24
3.12	Gesto de <i>Scale</i> [33].	25
4.1	Projeto de Mesa multi-toque <i>Virttable</i> (imagem retirada de [17])	26
6.1	Exemplo de uso de componentes em entidades de jogo [4]	32
6.2	Exemplo de uso de hierarquia em entidades de jogo [4]	32
6.3	Padrão <i>Singleton</i> [4]	34
6.4	Padrão <i>Observer</i> [4]	35

Lista de Tabelas

7.1	Parâmetros de mensagens TUIO em superfícies interativas 2D	38
-----	--	----

Capítulo 1

Introdução

(Professora, neste capítulo começamos com uma grande enrolação sobre a história dos jogos e a evolução dos inputs, poderia por favor sugerir cortes? acredito que tenha ficado longo demais, passando do objetivo de introduzir o trabalho...)

A indústria do entretenimento é uma das que mais cresce, e o Brasil está entre as que apresentam maior crescimento nessa área [28]. Isso se dá pois tudo que é feito pode ser aplicado para essa indústria. Por exemplo, se dermos uma caixa de papelão a uma criança, ela irá fazer um forte ou uma espaçonave e se divertirá com aquilo. Isso faz com que essa indústria seja altamente lucrativa e muito buscada. Em [30], vemos a história dos primeiros *video games*, explicados a seguir. A indústria de *video games* começou somente com alguns programas demonstrativos para computadores desde 1952. Nessa época, foi criado um jogo que é simplesmente um jogo-da-velha eletrônico, chamado *OXO*, criado por *Alexander S. Douglas*, desenvolvido para um computador chamado *EDSAC*, um dos primeiros computadores digitais, que pode ser visto sendo emulado na figura [1.1]. O código do jogo era altamente eficiente, devido a baixa capacidade computacional do computador, mas como esse computador não foi difundido, esse jogo ficou somente no laboratório em que foi desenvolvido e poucos jogaram ele.

Em 1958, para demonstrar um aparelho osciloscópico em um dia de visita anual ao laboratório *Brookhaven National Laboratory* (Departamento de energia dos EUA), foi criado o primeiro jogo eletrônico, chamado *Tennis for two*, figura [1.2]. Apesar dele usar um computador analógico ao invés de um digital, ele inovou pelo fato de ser um jogo para mais de um jogador, e era bem fluído. Ele ainda é considerado como uma versão anterior ao jogo que fez grande sucesso chamado *PONG*, desenvolvido pela *Atari* em 1972. Porém, ele também nunca chegou a sair do seu laboratório.

Já em 1962, *Steve Russell* desenvolveu um jogo chamado *Spacewar!*, figura [1.3], inicialmente para um sistema chamado *PDP-1*. Ele consiste de um jogo de dois jogadores, cada um no controle de uma nave, em que o objetivo é atirar mísseis para destruir a nave do oponente. O jogo conseguiu tanta popularidade que em seguida foi adaptado para vários outros sistemas populares na época, e o jogo era um teste de sistema tão bom para o *PDP-1* que o sistema era vendido já com o jogo em memória. Isso fez com que o

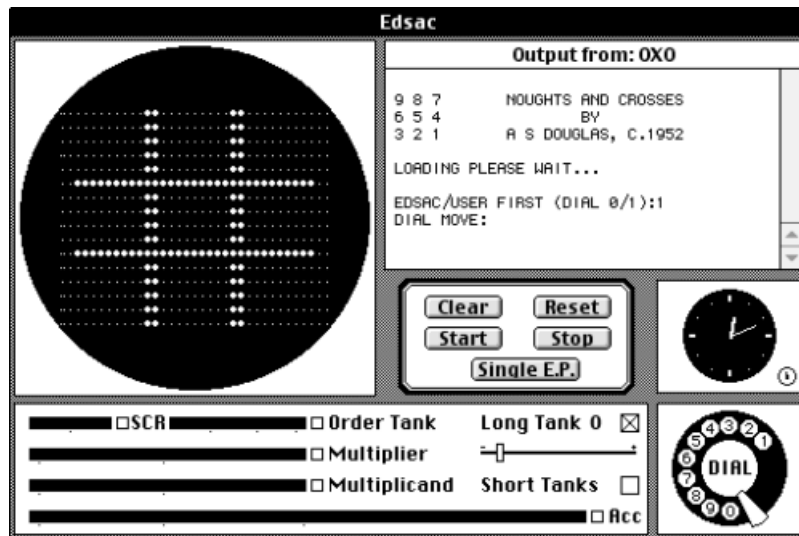


Figura 1.1: Jogo para *EDSAC OXO* [30]

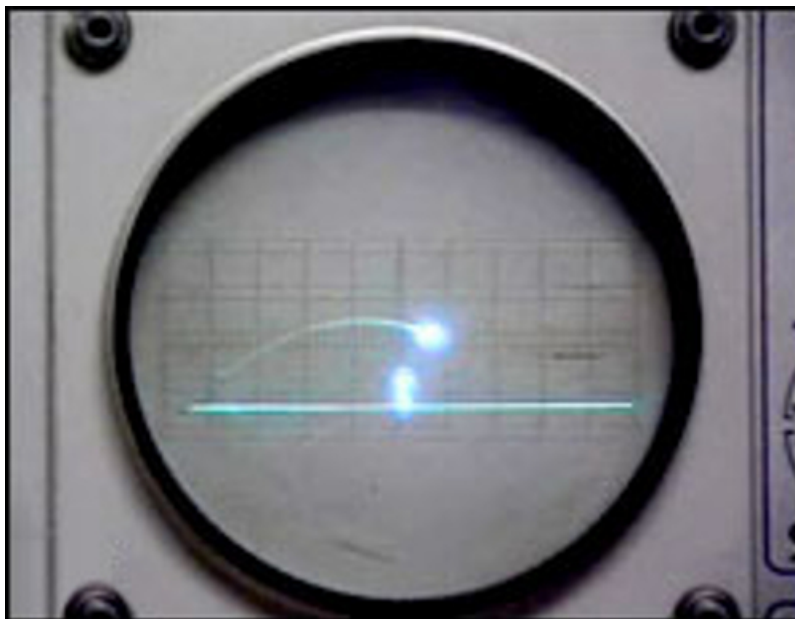


Figura 1.2: Jogo para um osciloscópio *Tennis for two* [30]

Spacewar! fosse o primeiro jogo de computador altamente difundido.

O primeiro console doméstico lançado foi o *Magnavox Odyssey* [34], visto na figura [1.4], lançado em 1972. ele inaugurou uma idéia inovadora, de se criar um aparelho computacional independente de uma tela, para ser conectado a televisores, algo que até então não havia sido pensado, os computadores sempre vinham acoplados a suas telas. Além disso, ele também inaugurou outro conceito que é o de mídia removível para os consoles, que evolui para os cartuchos e, hoje em dia, aos DVDs e Blu-Rays. Porém, devido ao preço um pouco caro para a época e uma má estratégia de marketing, o *Odyssey* não foi muito

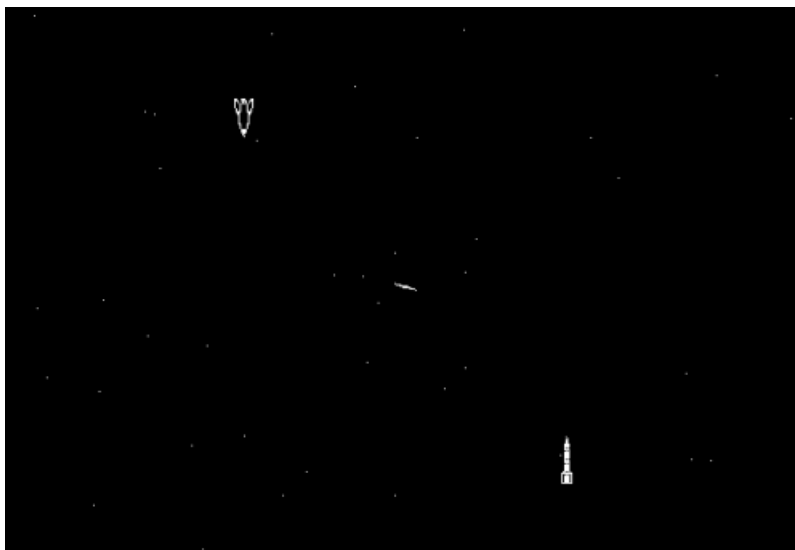


Figura 1.3: Jogo para computadores antigos *Spacewars!* [30]

popular. Uma outra companhia que inaugurou seu console em 1976, a *Atari*, teve uma melhor estratégia. Ela lançou seu nome no mercado devido ao seu jogo *Pong*, um simples jogo de tênis, que foi inicialmente lançado para uma máquina com uma tela que permitia o jogo através de um sistema de moedas em 1972 e foi adaptado para uma versão doméstica em 1973 [31]. Em 1976, utilizando um novo processador barato na época, o *MOS 6502*, a *Atari* lança seu primeiro console doméstico com possibilidade de trocar jogos, o *Atari 2600*, que pode ser visto na figura [1.5]. Esse console teve um enorme sucesso de acordo com [31].



Figura 1.4: Console *Odyssey* da *Magnavox* [34]

Após esse período, os *video games* passaram por um declínio, e somente por 1980 que a *Nintendo* resgatou esse mercado com o lançamento de um jogo conhecido como *Donkey Kong*, inicialmente desenvolvido em conjunto com a *Atari* mas uma divergência sobre direitos autorais do *Donkey Kong* fez as companhias terminarem a parceria [31]. Em



Figura 1.5: Console *Atari 2600* da *Atari* [31]

1985 a *Nintendo* lançou seu console, o *NES*, figura [1.6], e passou a dominar o mercado, com pouca competição de sua nova concorrente, *SEGA*, e seu console *Master System*.



Figura 1.6: Console *NES* da *Nintendo* [7]

Nesse momento a indústria do entretenimento eletrônico teve um grande crescimento, abrindo o mercado para cada vez mais competidores a medida que se foi evoluindo as gerações de consoles. A *Nintendo* se manteve forte por todo esse tempo, se tornando uma das três companhias que dominam esse mercado hoje em dia. Sua principal competidora, a *SEGA*, não conseguiu acompanhar, e seu último console, o *Dreamcast*, teve pouca venda. Atualmente a *SEGA* ainda é uma empresa grande, mas ela somente faz jogos para outros consoles.

Durante a evolução dos jogos, uma empresa fez parceria com a *Nintendo* para tentar lançar um novo sistema em que os cartuchos seriam trocados por CDs, diminuindo seu custo e aumenta a sua capacidade de armazenamento. Essa empresa é a *Sony*. Devido a desentendimentos, a parceria terminou, mas a *Sony* insistiu na idéia e em 1994 lançou o seu console, o *Playstation*. Esse console foi um sucesso de vendas enorme, mas o que realmente colocou a *Sony* no mapa dos consoles foi o seu próximo console, *Playstation 2*. Esse console foi uma grande revolução no seu tempo e saiu um ano antes dos consoles de

sua geração, o que ocasionou no maior índice de vendas de um console de *video game* de todos os tempos, vendendo mais de 140 milhões de unidades. Atualmente a *Sony* compete no mercado como uma das maiores empresas de diversão eletrônica.

Outra empresa que resolveu explorar esse mundo dos consoles é a *Microsoft*. Gigante no mundo da computação, ela viu a oportunidade de aplicar seus conhecimentos para criar um sistema barato, eficiente e inovador. Em 2001, ela lançou seu primeiro console, o *Xbox*. Esse console, que é da mesma geração que o *Playstation 2*, saiu um pouco tarde, afetando bastante as suas vendas. Ele foi feito baseado na arquitetura de um PC tradicional, mas por possuir uma produção muito cara, ele acabou dando altas perdas iniciais para a empresa. Porém, a *Microsoft* insistiu nesse mercado e hoje compete com as outras duas empresas por essa geração de consoles.

Atualmente, o mercado dos jogos eletrônicos conta com três consoles competindo pelo mercado, o *Nintendo Wii* da *Nintendo*, o *Xbox 360* da *Microsoft* e o *Playstation 3* da *Sony*, os três podem ser vistos na figura [1.7], além dos três estarem competindo com jogos desenvolvidos para os computadores pessoais, que hoje em dia existem em muitas residências. Nesse mercado de competição, para ser levado em consideração, deve-se ser inovador. Neste trabalho, procuramos renovar um estilo de jogo pouco explorado e fazê-lo divertido e para uma plataforma diferente.



Figura 1.7: Consoles de última geração. Da esquerda pra direita: *Wii*, *Playstation 3* e *Xbox 360* [13]

Durante a evolução natural da interação entre o homem e a máquina, busca-se cada vez mais a adaptação dos gestos naturais para o ambiente virtual. Ao fornecer ao usuário gestos fáceis de se usar, se diminui a curva de aprendizado para se utilizar algo, bem como mantém um ambiente mais familiar ao usuário. As grandes companhias competidoras já estão caminhando em direção a isso. O *Nintendo Wii* possui controles sensíveis ao movimento, permitindo o uso de gestos para o controle em seus jogos, o *Wiimote*, figura [1.8], assim como o *Playstation 3* possui seus controles sensíveis ao movimento e com noção de profundidade, o *Playstation Move*, figura [1.9], e o *Xbox 360* possui o *Kinect*, figura [1.10], uma câmera inteligente capaz de identificar gestos. Ao se utilizar do toque direto na tela, podemos habilitar interações como o arraste, a seleção pelo simples toque, a ampliação ou redução de tamanho por se segurar algo pelas pontas e expandir ou diminuir a distância entre elas. Por isso, decidimos desenvolver este trabalho em uma mesa sensível a multi-toques, uma interface até o momento pouco explorada e inovadora.



Figura 1.8: Controle para *Nintendo Wii Wiimote* [12]



Figura 1.9: Controle para *Playstation 3 Playstation Move* [12]



Figura 1.10: Câmera para *Xbox 360 Kinect* [12]

Essas mesas multi-toque já possuem alguns softwares disponíveis, porém não existe um bom software que demonstre inteiramente as capacidades de *input* da mesa. Dessa forma, este trabalho propõe criar um jogo simulador de física para uma mesa multi-toque, explorando assim as capacidades de *input* da mesa em um único jogo inovador e divertido.

Os jogos simuladores de física são essencialmente jogos em que se começa com um mundo em aberto, e o jogador pode desenhar nele os objetos que desejar, podendo atribuir a eles as propriedades que quiser, e eles irão se interagir de acordo com as leis da física, obedecendo gravidade, aceleração e empuxo. Assim, um jogo nesse estilo permitiria utilizar as capacidades da mesa demonstrando seus gestos disponíveis, como *scale* e *rotate*, bem como a detecção de gestos padrões como o desenho de um quadrado ou círculo.

Utilizando o jogo deste trabalho iremos mostrar as funcionalidades de mesas multi-toque, obtendo como produto final o jogo *Deskworld*, além da mesa que construímos para testar o jogo.

Como as mesas com tela multi-toque não são ainda disponíveis em massa para a residência dos usuários, sua maior taxa de uso está focada em exposições, onde os usuários possuem somente um encontro breve com a mesa. Assim, um jogo desenvolvido para uma mesa dessas necessita ser fácil de se utilizar e possuir suporte a muitos usuários ao mesmo

tempo, o que é feito com o jogo desenvolvido neste trabalho.

A proposta deste jogo é proporcionar interação entre mais de um jogador ao mesmo tempo inovando em um estilo livre de jogo, onde cada jogador é capaz de manufaturar seus objetos com auxílio de ferramentas implementadas para serem manuseadas sobre uma superfície multi-toque.



O **segundo** capítulo deste trabalho **irá** apresentar trabalhos correlatos ~~ao nosso~~, exibindo mesas parecidas a que ~~iremos~~ desenvolver e jogos similares ao proposto, ambos para mesas como para outros sistemas em geral, sensíveis ao toque ou não.

O **terceiro** capítulo **apresentará** o fundamento teórico por trás da mesa. Ele **irá** explicar os princípios de processamento de imagem necessários para se entender os toques na mesa, bem como os filtros utilizados e **irá** explicar os métodos de iluminação e como isso afetará nosso projeto.

No **quarto** capítulo, **temos** detalhes sobre a construção da mesa em si. Nele, estará exposto o projeto da mesa, explicando as decisões de *hardware* escolhidos de acordo com as dimensões da mesa e explicando como será feito fisicamente os passos necessários para a detecção de toque, com a utilização de LEDs e uma câmera infravermelha.

No **quinto** capítulo, **temos** o *game design* do jogo proposto, chamado *Deskworld*. **Iremos** explicar os detalhes do jogo, seu conceito, seus objetos e suas regras, tudo que será implementado dele.

Já no **sexto** capítulo, **veremos** a arquitetura do jogo e detalhes de implementação. Neste capítulo **mostraremos** como foi implementado todos os quesitos explicados no capítulo anterior em nível mais baixo, explicando as decisões do projeto e as escolhas feitas sobre a orientação a componentes.

Finalmente, no capítulo sete, **explicaremos** mais detalhes sobre as ferramentas utilizadas de suporte para a implementação do jogo proposto, como a Touchlib e a engine Box2D.



Capítulo 2

Trabalhos Correlatos



2.1 Jogos simuladores de física



No cenário de jogos simuladores de física em duas dimensões um muito popular é o *Crayon Physics* [16]. Neste jogo temos um giz de cera virtual e devemos com ele criar um cenário que permita que a bola alcance a estrela. O jogador pode desenhar a forma que quiser, e ela é adicionada ao mundo assim que ele acabar de desenhá-la. Uma vez no mundo, o objeto obedece a gravidade e adquire um momento, obtendo assim um movimento. Tudo isto é feito levando em conta leis da física como as três leis de *newton* aplicado a um mundo bidimensional.

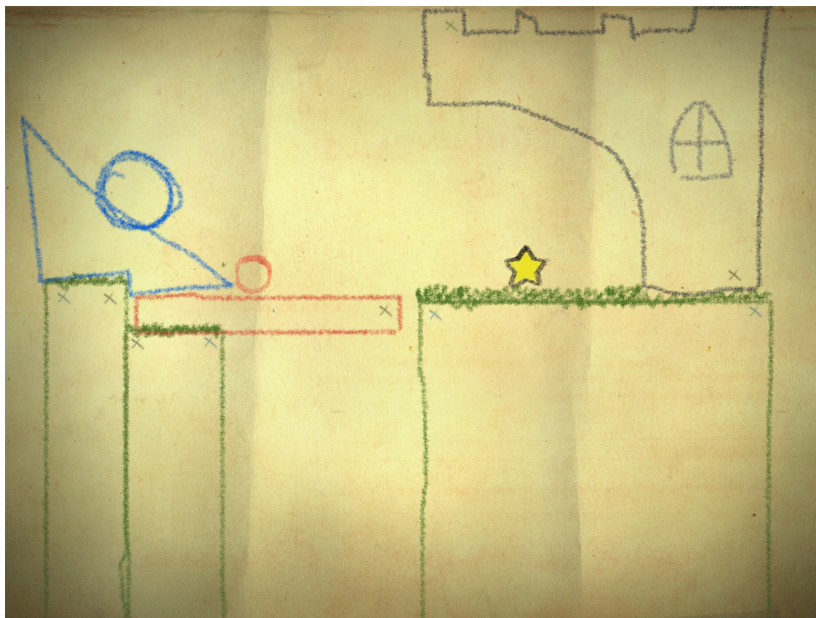



Figura 2.1: Jogo para PC *Crayon Physics* (~~imagem retirada de~~ [16])



Inspirado no *crayon physics* temos outro jogo desenvolvido em ~~código aberto~~ o *Numpty*

 *Physics* [8]. Temos o mesmo objetivo e ferramentas neste jogo, porém o jogo possui código aberto. Devido a isso, foi modificado e portado para funcionar em superfícies multi-toque através de um trabalho de graduação da Universidade Técnica de Viena  [23].


Temos também um jogo chamado *Phun* [1], figura [2.2] , que não possui objetivo. Ele é uma ferramenta simuladora com física que permite ao usuário construir o ambiente que desejar e fazer o que quiser com ele, podendo desenhar formas normais ou abstratas, mudar a taxa de gravidade e fricção dos objetos, fazer líquido e controlar densidade dos elementos, entre outros. O jogador é livre para montar o mundo que desejar e fazer o que quiser com ele.



Figura 2.2: Jogo para PC *Phun* [1]

Um jogo que abriu as portas para a popularização desse tipo de jogo se chama *Little Big Planet* [19]. Nesse jogo, inicialmente desenvolvido para *Playstation 3* [9] e depois para *Playstation Portable* [10], o jogador controla um boneco chamado *Sack Boy*, que ele pode mudar a aparência como quiser, e deve seguir por uma fase no estilo de jogo de plataforma até chegar ao seu final. A diferença está no estilo do jogo, que os criadores chamaram de *Play, Create, Share*, onde os jogadores são livres para criar suas próprias fases como quiserem e compartilharem elas com todos os outros jogadores do mundo. Neste modo, o *Sack Boy* pode colocar objetos em jogo para construir a fase, e os objetos se interagem de acordo com as leis da física, podendo possuir movimento e atrapalhar ou ajudar um ao outro, sendo perigosos ou não para o *Sack Boy*. Com esta ferramenta, os jogadores construíram inúmeras fases inovadoras ao redor do mundo, inventando coisas

que até os criadores nunca haviam imaginado. O sucesso foi tanto que os criadores do *Little Big Planet* já lançaram recentemente a sua continuação, *Little Big Planet 2* [20], onde a criação de fases vai além de somente jogos de plataforma, com o jogador podendo criar regras no jogo que permitem a criação de vários estilos de jogos diferentes, como por exemplo *Shooter* ou *RTS*.



Figura 2.3: Modo de criação do jogo para PS3 *Little Big Planet* [19]

Ainda no mesmo estilo de *Play, Create, Share*, a mesma empresa criadora do *Little Big Planet* lançou um jogo chamado *Modnation Racer* [21], que se trata de um jogo de corrida onde os jogadores podem criar as suas próprias pistas de corridas e compartilhar com todos. Lançado em Maio de 2010, já possui milhares de pistas criadas por usuários. Esses jogos demonstram a popularidade do estilo de se criar seu próprio jogo.

2.2 Superfícies sensíveis a toque

Existem vários trabalhos sobre superfícies sensíveis ao toque. Alguns mais conhecidos, como os *smartphones* e o *Ipad*, já estão sendo comercializados a um tempo. As mesas com tela multi-toque, por outro lado, são relativamente recentes no mercado, com muitas pesquisas ainda sendo desenvolvidas para habilitá-las ao público. Uma alternativa criativa foi proposta por [26] onde conseguimos montar uma pequena mesa multitoque menos de US\$50 dólares. Com materiais baratos como papelão, papel e vidro podemos construir uma mesa utilizando um PC e uma *webcam*.

Várias utilidades diversas são propostas para essas mesas. Temos por exemplo a *Reactable* [25], figura 2.4, disponível desde 2008, que, apesar de não exatamente tratar o toque do usuário, trata de uma mesa que utiliza marcadores fiduciais para habilitar a interação de seus usuários com a música, fornecendo um instrumento musical diferente.

Ela é utilizada em concertos e eventos. Podemos citar também a *Microsoft Surface* [18] que proporciona funcionalidades diversas como habilidade de compartilhamento de figuras e arquivos através da interação com objetos e ícones colocados sobre sua superfície e comandados via o toque do usuário.



Figura 2.4: Mesa com marcadores fiduciais *Reactable* [25]

~~Para os testes de desenvolvimento do jogo proposto neste trabalho, é necessário a utilização de uma mesa com tela multi-toque. Como foi decidida pela construção de uma mesa para isso, vários projetos tiveram que ser analisados. A idéia de todos é a mesma: Temos uma superfície onde é projetado a tela da mesa, e o usuário pode tocar a tela. A tela é iluminada com a utilização de leds e os toques são detectados por uma câmera que os interpreta em relação a posição na mesa.~~

Existem várias maneiras de se iluminar a mesa. A da figura acima se chama *Diffuse Illumination* (DI). Se trata de LEDs iluminando a mesa por baixo do acrílico. Esse método é utilizado na *Microsoft Surface* [18] e na mesa construída durante o desenvolvimento do jogo *Eco-Defense* [4] por exemplo.

Outro sistema de iluminação é o sistema FTIR (*Frustrated Total Internal Reflection*). Nesse sistema, os LEDs são imbutidos no acrílico da superfície da mesa. Assim, a iluminação sofre de um fenômeno denominado reflexão interna total frustrada. Isso significa que a iluminação fica retida dentro do acrílico sem conseguir sair. Ao momento que um dedo toca na superfície, a capacidade de refração da superfície muda, o que permite que a luz saia e ilumine o toque, e uma câmera infravermelho consegue captar essa luz, detectando o local do toque. Essa detecção é normalmente bem mais precisa que a detecção por DI, por possuir menos interferência do fundo que não está tocando a mesa, porém

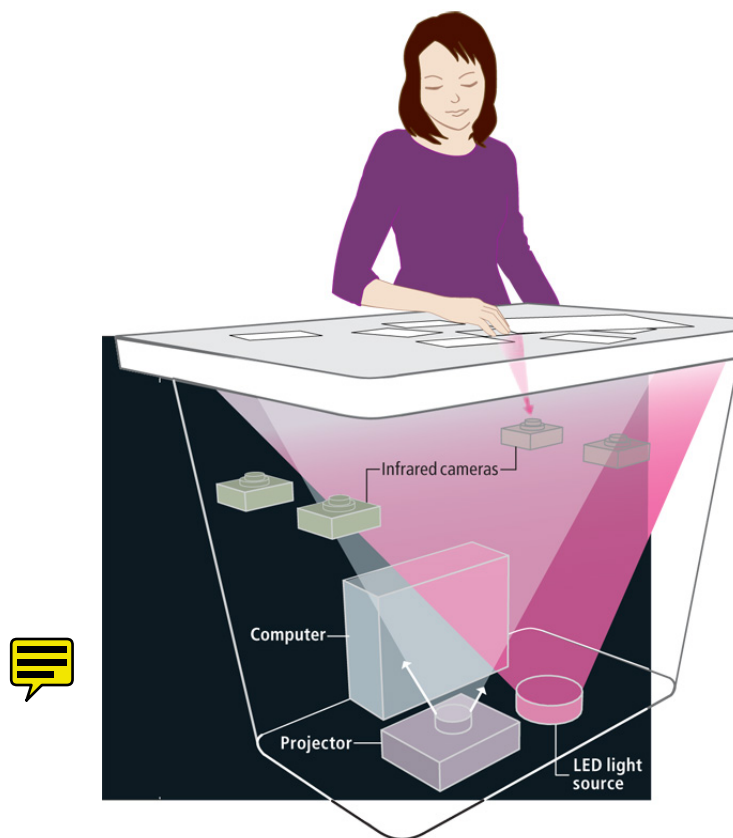


Figura 2.5: Projeto de mesa multi-toque [5]

ela não funciona muito bem para objetos que não são o corpo humano, sendo ruins para funcionamento com marcadores fiduciais. Um esquema desse modelo segue na figura 2.6.

A abordagem que a nossa mesa irá possuir trata de uma mesclagem das duas abordagens anteriores, como vemos no exemplo da *Virttable* [17], procurando proporcionar a melhor possível detecção de toque.

2.3 Jogos para mesas com telas multi-toque

Existem vários jogos produzidos para utilização em superfícies com tela multi-toque. Focando somente nos desenvolvidos especificamente para mesas como o nosso projeto utilizará, podemos citar muitos. Por exemplo, o *IRTaktiks* [27], um jogo de estratégia desenvolvido para uma mesa *FTIR* construída por eles. Este jogo é para dois jogadores, onde cada jogador tem suas unidades e deve derrotar as unidades inimigas. Porém, um problema dele é a identificação do jogador, pois ele não trata para saber de qual jogador pertence os toques feitos sobre a mesa, o que permite aos jogadores a até mesmo acidentalmente controlar as unidades do inimigo, e o jogo também limita o número de usuários para dois jogadores, algo que deve ser evitado em mesas multi-toque.

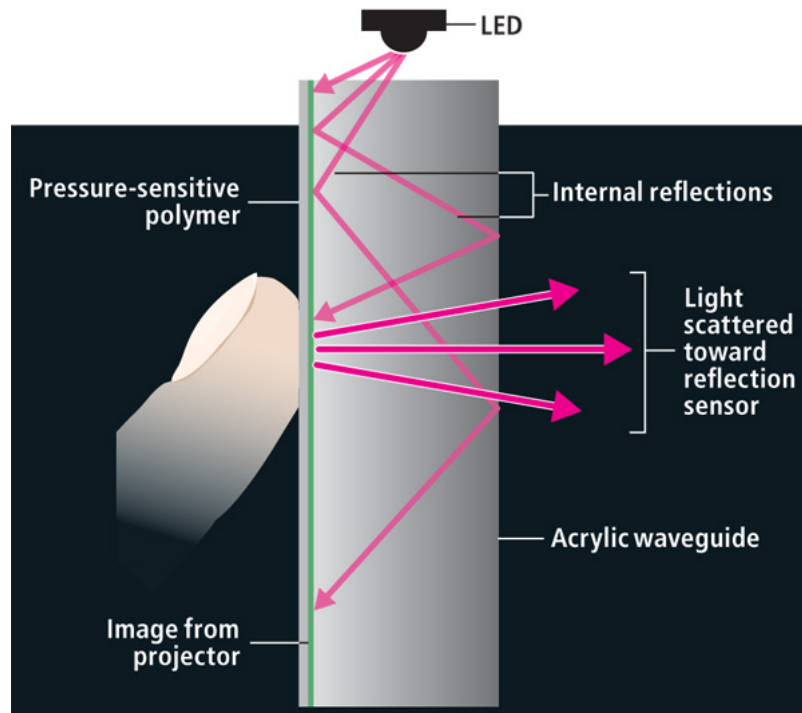


Figura 2.6: Esquema de detecção de toque FTIR [5]

Por se tratarem de superfícies normalmente sem limitações de número de toques, os jogos desenvolvidos para mesas devem focar em interatividade de vários jogadores. Jogos como *The Laundry Game* [15], que é um jogo de separação de roupas em categorias, e o *Game of Life* feito pelo *Verve Project* [29], que é o tradicional jogo da vida onde uma bactéria vive dependendo do número de bactérias ao seu redor, são simples e divertidos, e permitem que qualquer número de jogadores participe, mas eles param de jogar esses jogos depois de pouco tempo devido a falta de algo a se fazer no jogo.

Outros jogos para a mesa são mais completos, possuindo dificuldade e sistema de pontuação. Um desses jogos seria o *Eco Defense* [4], onde se deve destruir nuvens de poluição antes que afetem a floresta ao redor, podendo se colocar torres de defesa para ajudar, e a dificuldade sobe quanto mais se joga, aumentando a velocidade das nuvens, até os jogadores perderem. Outro exemplo seria o *Station Defender* [11], onde os jogadores devem defender sua base construindo paredes ao redor dela. Ambos os jogos proporcionam durabilidade pela dificuldade que escala e focam na sobrevivência do jogador pelo maior tempo possível.

Diferente desses jogos, temos jogos livres de física, mencionados no capítulo 2.1, onde o jogador possui total liberdade para desenhar o que vier em mente para concluir o objetivo do jogo. Um dos jogos mencionados no capítulo 2.1 foi adaptado para utilização em superfícies com tela multi-toque, o *Numpty Physics* [23]. Essa adaptação foi feita permitindo aos jogadores completarem as fases tradicionais *single player* e adicionando novos estágios *multi player* para aumentar a interação dos jogadores.



Figura 2.7: Jogo para mesa multitoque: *Ecodefense* (imagem retirada de [4])

Outro jogo inspirado no *Crayon Physics* e adaptado para dispositivos multi-toque desenvolvida em Action Script é [2]. Este jogo não possui objetivo de alcançar a estrela, tal qual sua inspiração, é mais voltado a um *free form*.

Capítulo 3

Fundamentação Teórica

Tradicionalmente, **estamos** acostumados a desenvolver jogos focados em um único usuário, algumas vezes pensando em dois ou mais, cada um entrando com seu *input* via teclado ou mouse. Porém, para este trabalho, **temos** que focar em um paradigma diferente. Nele, **temos** que desenvolver para um número não determinado de usuários, entrando simultaneamente ou não seu *input* compartilhando uma única tela sensível ao toque. Por isso são utilizadas técnicas diferentes para reconhecimento de toque e processamento de imagem para tradução do *input*.

Neste capítulo, **iremos** ver como detectar toques, processando as imagens obtidas pela câmera, filtrando elas, detectando criação e movimentação de **blobs** nelas e como corrigir a imagem da câmera para eliminar distorção. Depois **seguiremos** para verificar aspectos gerais de mesas multi-toque, como o tamanho de uma mesa e a interface com o usuário, e **terminaremos** analisando como trabalhar para um número indeterminado de usuários e como detectar gestos sobre a mesa.

3.1 Processamento de Toques

Para que **seja reconhecido toques**, o computador deve ser dotado de visão computacional. Como é explicada em [4], a visão computacional **se trata** de computadores com capacidade **de se enxergar** o ambiente e processar **o que estão vendo**. Assim, **necessitamos** de um sistema capaz de **interpretar uma mesa sendo tocada** e reconhecer o ponto dos toques, extraíndo das imagens obtidas as informações de toque. Esse sistema é composto por um computador **tridimensional**, uma *webcam* sem filtro infravermelho e com um filtro para luz visível e um **projektor**. O projetor joga a imagem no acrílico da mesa, e a *webcam* **enxerga aonde** na imagem os toques estão sendo feitos e os interpreta no programa que estiver rodando. Detalhes de como a mesa é montada **veremos** mais a frente no capítulo 4, aqui **iremos** explicar como esses toques serão compreendidos pelo sistema.

3.1.1 Processamento das Imagens

As imagens terão de ser capturadas rapidamente em sequência, para se detectar movimento. Elas serão capturadas através de uma câmera sem filtro infravermelho e com um filtro para luz visível, como um negativo de filme, de forma a simplificar sua forma e apresentar imagens mais fáceis de serem tratadas, que não se confundem o toque com o fundo. Após serem capturadas, as imagens serão **filtradas por alguns outros filtros** de forma a destacar mais claramente os toques nela, e finalmente os conjuntos de píxeis agrupados com intensidade semelhantes são detectados nela e mapeados, são os chamados *blobs*, que marcam a posição exata onde o toque foi realizado.

Existem dois métodos principais de se iluminar uma mesa multi **touch screen**, **FTIR** e **DI**. **FTIR** (*Frustrated Total Internal Refraction*) trata de uma iluminação em que **LEDs** são postos em paralelo ao acrílico da mesa ou dentro do acrílico e a sua iluminação fica **'quicando'** dentro do acrílico devido ao fenômenos de reflexão total. Uma vez que um toque é feito no topo do acrílico, a iluminação **'escapa'** de dentro dele e ilumina o ponto de toque. A **DI** (*Diffuse Illumination*), por outro lado, ilumina por baixo do acrílico e normalmente enxerga tudo acima do acrílico, incluindo o fundo do ponto em que se toca. Como **vemos** em [17], a iluminação por **FTIR** é **melhor** para detecção de toques, enquanto que a iluminação por **DI** é **melhor** para se detectar a utilização de marcadores fiduciais. **Como esta mesa será usada em um outro projeto sendo desenvolvido paralelo** **que utiliza marcadores fiduciais, optamos por fazer uma única mesa que possua as duas** **iluminações ao mesmo tempo.** A imagem que uma câmera enxerga em cada tipo de iluminação pode ser vista na figura [3.1].

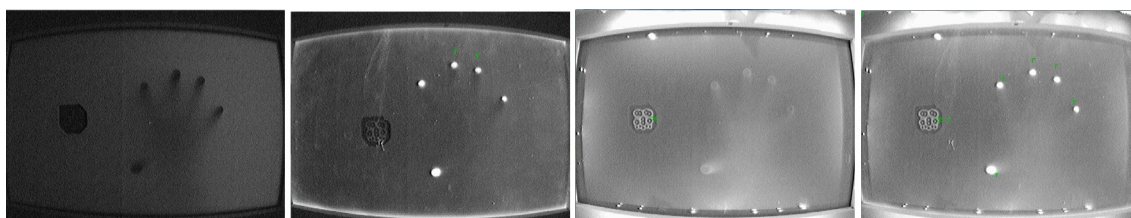
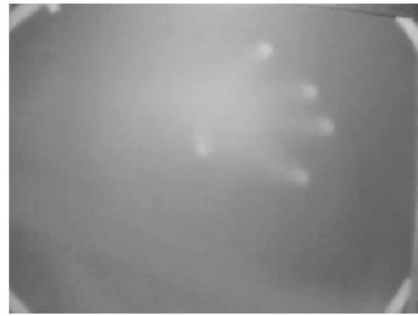


Figura 3.1: **Diferentes** imagens capturadas com iluminações **diferentes** [17].
~~Da esquerda para direita: Sem iluminação, somente FTIR, somente DI, híbrido.~~

Dependendo do **método** de iluminação, vários filtros diferentes devem ser aplicados, como explicados por [22]. Como a **nossa** iluminação será híbrida, teremos que aplicar todos os filtros correspondentes **as** duas iluminações para obter a melhor imagem possível. Assim, os filtros a serem aplicados **serão os vistos** na figura [3.2], ~~retirada de [4].~~ São eles:

1. Imagem capturada do dispositivo (Figura [3.2](a)). Devido a limitações da *Touch-lib*, utilizada neste trabalho, ela deve ser composta de um canal de 8-bits, sendo convertida se necessário.
2. Filtro de *Background*, que remove o fundo da figura, ~~visto em~~ (Figura [3.2](b)), ou seja aquilo que foi captado e que não é o toque do usuário ou marcador fiducial e



(a) Captura da câmera



(b) Filtro de *background*



(c) Filtro de *highpass*



(d) Filtro *scaler*



(e) Filtro *rectify*

Figura 3.2: Imagem capturada e suas filtragens [4].

está a mais na figura. Para tal, é armazenado um *frame* na mesa sem nada em cima, que ela usa para subtrair dos próximos *frames* esse *frame*, anulando tudo que não for modificado da imagem.

3. Filtro de *Highpass*, que aumenta o brilho de pontos a uma certa altura da mesa, ~~visto em~~ (Figura [3.2](c)). Esse valor é ajustado manualmente pois depende da mesa utilizada, com o propósito de atenuar os toques e marcadores fiduciais.
4. Filtro *Scaler*, ~~visto em~~ (Figura [3.2](d)), que aumenta o brilho da imagem total que restou, clareando mais os toques.
5. Filtro *Rectify*, ~~visto em~~ (Figura [3.2](e)), que reduz os 'ruídos' da imagem, deixando os *blobs* mais bem definidos e de fácil detecção.



3.1.2 Detecção de *Blobs*

Como dito na seção anterior, os chamados *blobs* são os resultados da detecção dos toques na superfície. Para se interpretar esses *blobs*, temos dois processos essenciais, chamados de *Blob-detection*, que interpreta o local em que o *blob* foi criado nas imagens filtradas anteriores, e *Blob-tracking*, que é a análise dos *blobs* em uma sequência de imagens. Utilizando essas duas análises, poderemos identificar a criação de *blobs*, verificar seus movimentos durante sua vida útil, bem como identificar a criação de novos *blobs* no sistema.

A detecção de *blobs* deve ser realizada a cada frame. Da imagem final obtida da figura [3.2], identificamos os contornos dos *blobs* que aparecem e analisamos para identificar se eles correspondem ao toque de dedos ou a marcadores fiduciais. Se for um toque, uma elipse é feita sobre ele, e com base nela, pode ser identificada posição, orientação e tamanho do *blob*. Esse *blob* então é adicionado a uma lista para ser processado.

Tendo essa lista, o *Blob-tracking* tem que ser feito para identificar a movimentação e criação de *blobs*. Usando o frame anterior de referência, verifica se o *blob* será criado, removido ou atualizado. Para se fazer essa análise, usa-se 2 conjuntos, P e Q . O conjunto P contém a lista de pontos que representam os *blobs* do frame anterior. Um exemplo seria um conjunto P com pontos (1,1) e (4,4), como visto no gráfico [3.3].

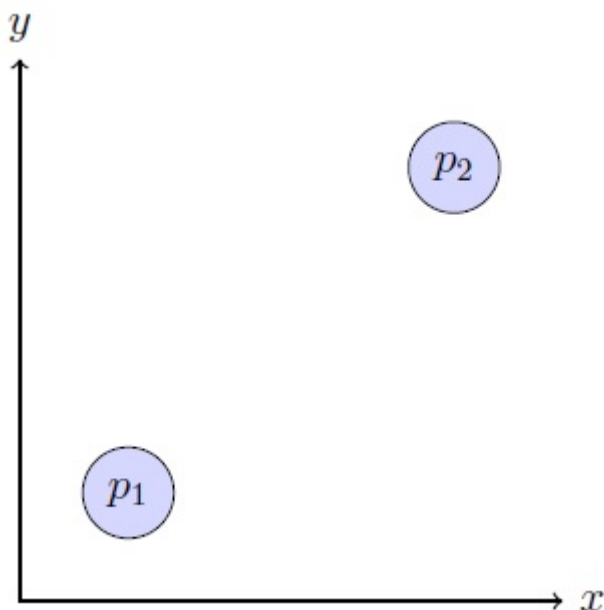


Figura 3.3: Gráfico de P [4].

O segundo conjunto, Q , conterá a lista de pontos que compõem os *blobs* do frame atual. Por exemplo, o conjunto Q com pontos (1,2), (1,4) e (4,3), visto no gráfico [3.4].

O número de elementos de Q é maior que o número de elementos em P , logo sabemos que pelo menos um *blob* novo foi criado. Para saber qual o *blob* criado e quais foram

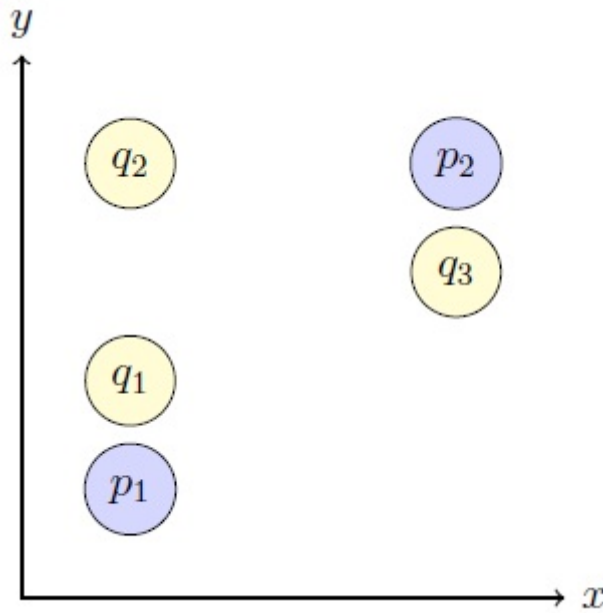


Figura 3.4: Gráfico de Q [4].

somente atualizados, **temos** que analisar as distâncias entre os *blobs*. Neste exemplo, **vemos** que o *blob* $p1$ se moveu para $q1$, o *blob* $p2$ se moveu para $q3$, e $q2$ é um novo *blob* criado neste frame. A cada frame, esse conjunto Q passa a ser o novo P , um novo conjunto Q é gerado, e o procedimento se repete.

3.1.3 Correção da Câmera

Quando uma câmera captura uma imagem sobre uma superfície plana, essa imagem possui uma distorção, onde os pontos mais distantes do centro da imagem ganham um pequeno ângulo em relação a imagem original pois são exibidos em forma reduzida. Para se detectar os *blobs*, é necessário executar alguns algoritmos **para** corrigir isso e colocar toda a imagem em um mesmo plano. Como dito por *Muller* em [22], várias bibliotecas de processamento de imagem e detecção de multi-toque já fazem isso. Na figura [3.5], **vemos** uma imagem antes e depois da correção.

Como explicado por *Muller*, essa correção é feita realizando um mapeamento de pontos na imagem em relação a um ponto do espaço. Para tal, é necessário considerar a distância focal, o centro da imagem, o tamanho de cada pixel e o coeficiente de distorção radial da lente, propriedades da própria câmera, bem como o vetor de translação e a matriz de rotação que analisam o espaço em que se encontra a câmera.

Assim, um ponto na câmera pode ser corrigido de acordo com a fórmula $m = A/Rt/M$, onde A é a matriz que possui os fatores da câmera, ~~vista em [3.6]~~, M é um ponto do espaço, R é a matriz de rotação da câmera e t é o vetor de translação da mesma.

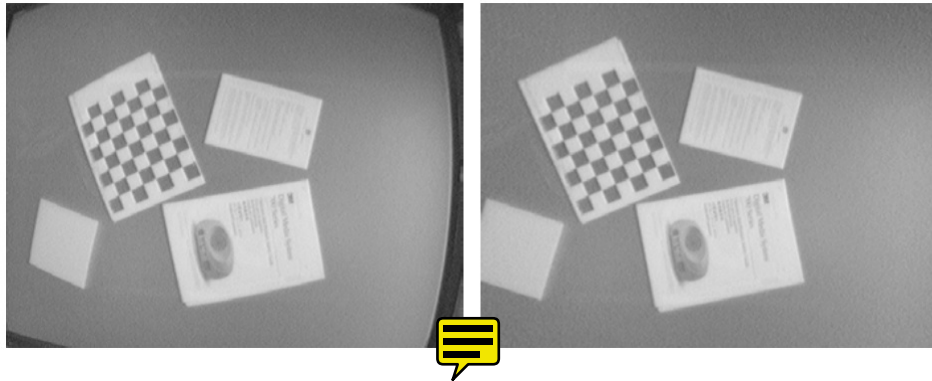


Figura 3.5: (a) Imagem da câmera com distorção. (b) Imagem da câmera corrigida. [22].

$$A = \begin{vmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{vmatrix}$$

Figura 3.6: Matriz A [4].


3.2 Aspectos de mesas multi-toque


As superfícies multi-toques mudam a forma em que o usuário interage com o computador. Devido a esse novo paradigma, é necessário se repensar nas capacidades oferecidas através delas ao usuário. Os softwares atuais levam em consideração uma pessoa sentada na frente de uma tela controlando um dispositivo apontador, o *mouse*. Uma mesa multi-toque, no entanto, convida N pessoas a partilhar uma utilização única dela com M toques diferentes. Assim, muitos softwares atuais não são adequados para a utilização na mesa.

Para que as mesas se tornem algo em comum no dia-a-dia das pessoas, elas devem possuir uma eficiência compatível com a realidade dos usuários. Nesta seção discutiremos o número de usuários, o número de toques, os tipos de toque e a resolução e tamanho da mesa.


3.2.1 Tamanho e Resolução


Mesas com objetivos diferentes possuem tamanhos diferentes. Se uma mesa for pequena, um número reduzido de pessoas poderá usar ela, e quanto maior mais pessoas poderão interagir. Porém, uma mesa muito grande impossibilita que um único usuário utilize ela totalmente, e se o *software* projetado não planejar para essa situação, pode ficar impossível de se utilizar ele na mesa.



 Existem algumas maneiras de se tratar isso sem modificar os *softwares*. A mesa pode ser projetada para se utilizar algo para apontar, permitindo a um usuário que alcance pontos mais distantes, ou então pode possuir uma função que mude o tamanho da tela e a traga mais para perto do usuário.

Em geral, as mesas possuem um tamanho maior que os monitores tradicionais, o  produz um problema sobre a resolução da imagem que será projetada. A maioria dos projetores possuem resolução padrão de 1024x768 pixels, o que, para mesas muito grandes, é muito pouco, e produzirá uma imagem bem distorcida. Para contornar isso, a solução é utilizar um projetor *short-throw* de alta resolução WXGA (*Wide Extended Graphics Array*), capazes de atingir resolução alta definição HD de 720p, alguns até chegando a 1080p. Essa é a abordagem que adotamos neste trabalho.


3.2.2 Interfaces

 Como citado por [4], interfaces sensíveis ao toque trazem ao usuário a possibilidade de interagir diretamente com os dados. Isso traz um grande apelo ao usuário, pois além de ser natural, é de fácil aprendizado. Por não possuir partes móveis, é interessante para ser usado, também, em equipamentos acessíveis ao público, tal como caixas automáticas e pontos de acesso.

Como dito antes, os *softwares* existentes são feitos para se utilizar com um *mouse*, que possui a área de apontamento bem pequena, logo, as interfaces possuem poucos píxeis para a seleção. Isso não se repete para interfaces de toque, pois o dedo  não tem menos precisão que a ponta de um ponteiro, devido ao tamanho do toque, e a detecção do toque pode sofrer de precisão. Como dito por [4], a câmera que captura a imagem a ser processada é o fator determinante da precisão da interface, ou seja, diversos aspectos como luz, distância focal e resolução alteram a usabilidade para o usuário.

Melhorando o sensor de *input* e aumentando a resolução  temos uma amostragem mais alta, o que ajuda no problema de precisão da câmera. Mas o problema da detecção do  tamanho do toque tem que ser considerado durante a construção da interface, pois o toque pode ser maior que o botão desejado e tem que se estimar onde realmente se deseja o toque. Em geral, o ponto central do toque é tomado como referência da posição do toque.

Além desses problemas, temos que considerar o ponto de visão do usuário. Quando um usuário utiliza um computador, ele está acostumado a possuir uma interface virada para ele. No caso da mesa, temos vários usuários e eles podem estar olhando a mesa de ângulos diferentes. Temos que cuidar para manter a interface agradável para todos os usuários que forem utilizar a mesa. A diferença de ângulo de visões é melhor exemplificado na figura [3.7].

Ao se planejar uma interface para mesa multi-toque então, deve-se prever a rotação da interface, podendo acomodar a aplicação não importando  aonde se encontra o usuário.

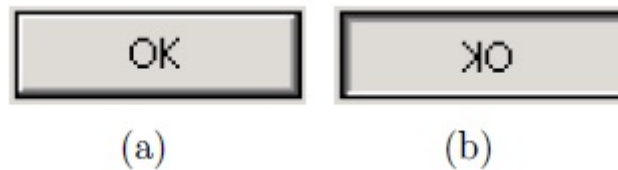







Figura 3.7: Visão de um botão tradicional *OK* em duas visões diferentes [4].



3.2.3 *N* Usuários




A maior dificuldade em se lidar com um número desconhecido (~~*N*~~) de usuários é identificar a quem pertence  qual toque e saber atribuir as propriedades necessárias a cada um deles. Podemos ~~pegar~~  o exemplo de [4], que cita o caso de uma aplicação *do desenho* em que um usuário deseja trocar a cor de seu pincel de amarelo para azul, *como trocar somente a cor para aquele único usuário?*

A solução nesse caso seria ~~ou de se~~ trocar a cor para todos os toques, ou então ~~a~~  de implementar a divisão de áreas de trabalho, onde cada usuário *possuiria a sua e não afetaria a de outro usuário*. Para os fins do jogo proposto, resolvemos inovar sobre essa segunda idéia, onde permitimos a divisão dos mundos para os usuários, assim cada  pode personalizar o seu mundo como quiser, mas ainda permitimos a interação entre os mundos, dando a sensação de que você edita parte de um mundo e a mesa seria um mundo todo.

3.2.4 Detecção de Gestos

A maioria dos *softwares*  leva em consideração gestos usando somente o *input* de um único *mouse*. Assim, são implementados normalmente gestos com um ponto de contato. Tradicionalmente, podemos citar:

- *Click* (figura [3.8]):  Gestos ~~mais~~ simples e tradicional. É detectado ao se estabelecer um ponto de contato *e retirado* logo em seguida.
- *Drag* (figura [3.9]):  Continuação do ~~último gesto~~, identificado ao se estabelecer um ponto de contato *e movendo* ele antes de se retirar o contato.

~~Alguns~~  poucos implementam alguns tipos de gestos usando somente um ponto de contato compostos de desenhos, como por exemplo no jogo *Black and White* em que o jogador pode  icionar um milagre diferente dependendo do movimento que faz com o mouse, como ~~visto~~  na figura [3.10].


Para se identificar *esse tipo*  de gesto, precisamos identificar a forma que o usuário está desenhando na tela. Isso é feito identificando a direção em que o jogador está desenhando



Figura 3.8: Gesto de *Click* [33].

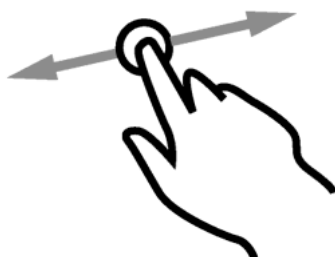


Figura 3.9: Gesto de *Drag* [33].



Figura 3.10: Reconhecimento de gesto no jogo *Black and White* [3].

e identificando o ângulo **em que** ele está seguindo. Como o ser humano não possui uma precisão exata para desenhar, esses valores tem que ser aproximados e comparados a um

banco de dados de gestos conhecidos, e aquele com que o padrão chegue o mais perto, dentro de um limite, será o gesto detectado. Caso não se aproxime de nenhum gesto conhecido, o gesto não é reconhecido e o usuário deve tentar **entrar** com outro gesto.



No caso do jogo que foi desenvolvido neste projeto, a detecção de gestos será limitada a formas geométricas mais simples. Caso o usuário desenhe algo que o jogo reconhece como uma forma geométrica, ele tenta aproximar o máximo o desenho a forma desejada. Caso o programa não identifique o desenho, **a forma é reconhecido** como uma forma livre. O *Deskworld* consegue identificar círculos, retângulos, quadrados e triângulos.



Ao se desenvolver um *software* para uma mesa multi-toque, **possuímos vários pontos de contatos diferentes, habilitando a utilização de mais gestos novos além dos anteriores. Como exemplo, podemos citar:**

- *Rotate* (figura [3.11]: Dois pontos de contato são inseridos sobre a superfície e são movimentados em direções opostas enquanto mantêm a distância entre os dois dedos constantes. O ângulo entre os dois dedos é calculado e denomina a rotação do objeto.

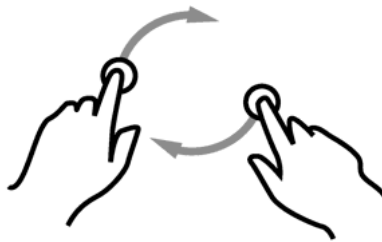


Figura 3.11: Gesto de *Rotate* [33].

- *Scale* (figura [3.12]: Análogo ao *Rotate*, utiliza de dois pontos de contato sobre uma superfície, porém neste caso **mantêmos** o ângulo constante e **variámos** a distância entre os dedos. Uma diminuição na distância significa uma diminuição na escala, enquanto um aumento de distância significa um aumento de escala.



Os gestos de *Scale* e *Rotate*, para o *software* de reconhecimento de gestos aplicado a nossa mesa, podem ser considerados os mesmos, pois ambos utilizam um mesmo gesto variando em fatores diferentes, ou seja, partem de dois pontos de contato sobre uma superfície e variam ou na distância entre os pontos de contato ou no ângulo entre eles, logo um gesto de rotacionar pode ocorrer ao mesmo tempo de um gesto de escalar e vice-versa. Analogamente, um gesto de *Drag* pode ser realizado durante esses dois gestos se ambos os pontos de contato forem movidos na mesma direção e ângulo.



Figura 3.12: Gesto de *Scale* [33].

Capítulo 4

Construção da Mesa Multi-toque

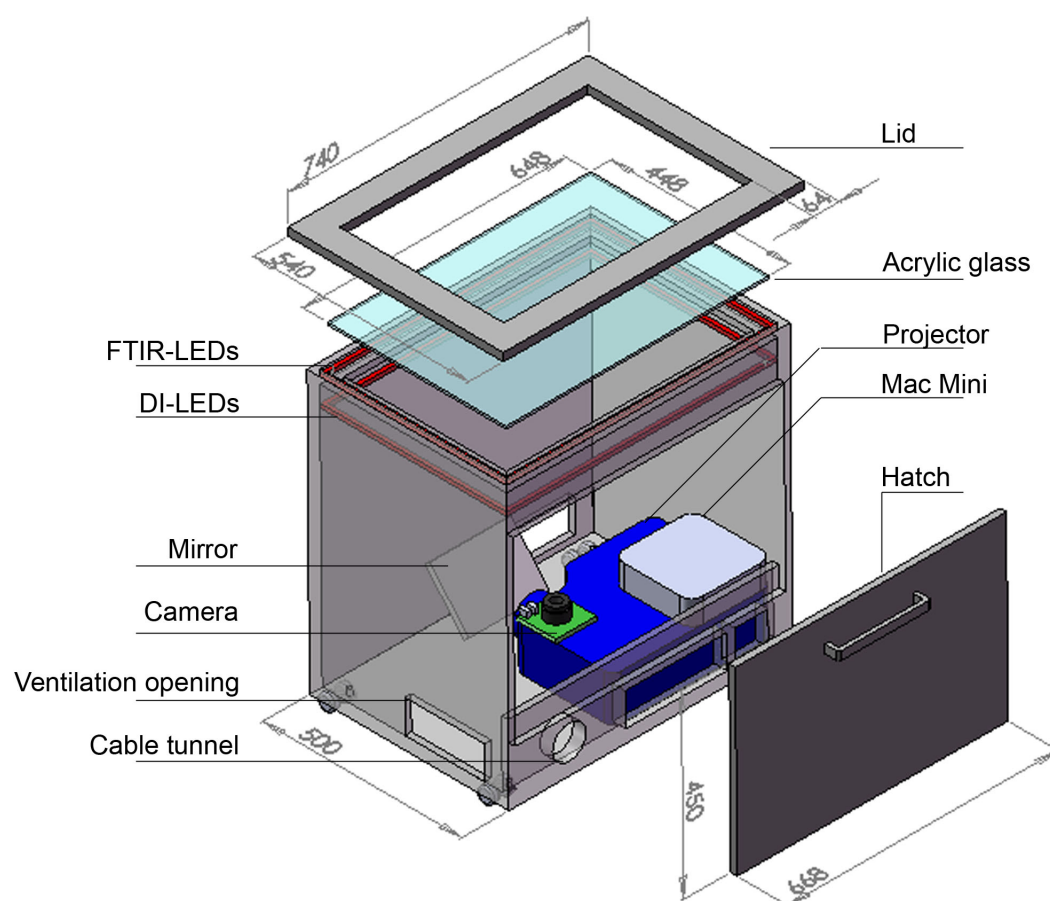


Figura 4.1: Projeto de Mesa multi-toque *Virttable* (~~imagem retirada de [17]~~)

Sendo uma importante parte do projeto, será apresentada neste capítulo o projeto e montagem de uma mesa multi-toque de médio custo ~~utilizada~~ para dar suporte ao jogo desenvolvido.

4.1 Estrutura da Mesa



A estrutura da mesa é feita de alumínio para dar **maior** estabilidade e leveza, permitindo **maior** mobilidade sem prejudicar sua eficiência. O material foi confeccionado sob medida, pois cada elemento influi na configuração da mesa. A mesa possui um metro de altura desde os pés até a superfície de acrílico.



Outra característica importante da mesa é a possibilidade de ser desmontada para facilitar **ainda mais** seu transporte. Apesar de possuir grandes medidas, ao se desmontar fica fácil de carregar e guardar. Portanto, retirando seus equipamentos, sua estrutura de alumínio pesa muito pouco. Além disso, mesmo montada, a mesa possui rodas com travas para poder ser movimentada com mais facilidade e podendo ser travada para o devido uso.

4.2 Superfície

O material da superfície é de acrílico, pois oferece o melhor custo/benefício. Isto porque, nenhum material é tão resistente, transparente com o seu custo. O **coeficiente de refração é ótimo** e permite a iluminação de toda superfície. A transparência permite a captação do movimentos com clareza pela câmera.

O acrílico tem dimensões de 648 x 448mm e 8mm de espessura. Para reter a projeção é **colado um papel vegetal na face do acrílico externa à mesa**. Esta película ajuda a destacar a iluminação dos dedos quando comparadas a da mão. Sem esta, a mão seria iluminada igualmente dificultando a detecção de toques feitos com os dedos. A sua superfície foi **encoberta** por silicone para obter-se maior precisão com os toques.


4.3 Iluminação Infravermelha

Para um bom reconhecimento de toques e de marcadores fiduciais, foi utilizado iluminação híbrida utilizando as técnicas de FTIR  e DI juntas. Os LEDs para a iluminação FTIR foram posicionados **em buracos no acrílico ao seu redor** apontado paralelamente a sua superfície. Aproveitando a resistência do acrílico furou-se a lateral do acrílico utilizando uma furadeira profissional para o posicionamento e fixação  dos LEDs. Os LEDs para a iluminação DI são distribuídos em **placas abaixo do acrílico** e apontando diretamente para ele.

4.4 Câmera




Para o processamento da imagem capturada com os toques relativo a mesa ~~será~~ utilizado uma câmera *PlayStation Eye*. A escolha foi baseada no baixo custo da câmera e sua alta taxa de quadros por segundo. Sua resolução pode ser configurada em 640 x 480 píxeis ou em 320 x 240 píxeis numa taxa de atualização respectiva de 60hz ou 120hz. (Até o momento nao achamos um driver que chegue nessa config, o máximo é 320x240 a 30 fps, o ps eye funciona a essa resolução no ps3 somente)

Para capturar os movimentos sobre a mesa com maior precisão, alteramos a câmera para filtrar o espectro de luz visível e só captar  infravermelha. Isto foi feito utilizando filme fotográfico sobrepondo a lente da câmera.

4.5 Projetor



Nesta mesa o projetor usado é um WXGA de resolução padrão 1080p a um  distância de 110cm da superfície de acrílico. O projetor é fixado ao fundo da mesa e ~~se~~ é utilizado um espelho que reflete a imagem dele para o topo.

Capítulo 5



Game *design* do *Deskworld*

5.1 Requisitos de interface



- Sua mecânica deve permitir a participação de diversos jogadores simultaneamente.
- Suportar e processar multi-toque simultaneamente.
- Capacidade de captar gestos.

5.2 Conceito



Deskworld tem como principal proposta deixar a encargo da criatividade do(s) jogador(res) a construções de cenários, objetos e seus objetivos no respectivo mundo. O principal objetivo do jogo é prover ferramentas aos jogadores para auxiliar-los na construção. Assim sendo, elementos encontrados em nosso planeta são representados em um plano bidimensional e podem ser regidos pelas leis da física. É possível desenhar um objeto e associá-lo a algum tipo de elemento de acordo com suas propriedades como densidade, massa, coeficiente de atrito e elasticidade. E a este objeto podemos aplicar uma força como a gravidade, fixá-lo no plano, entre outras opções. Oferecemos aos usuários a maior liberdade possível para que possa criar diferentes jogos, fazer desenhos ou a simulação de objetos.

5.3 Regras e Objetos

Basicamente, pode-se criar qualquer tipo de objeto bidimensional com ou sem auxílio das ferramentas fornecidas. A construção funciona como desenhar virtualmente, sendo o dedo o lápis e a superfície da mesa o papel.

Cada objeto pode ter as seguintes características:

- Densidade
- Coeficiente de Atrito
- Força Normal
- Massa
- Volume
- Velocidade

Além, dessas propriedades os objetos podem interagir entre si por meio de juntas, colisões ou como motores de outro objeto.

Neste jogo não existe uma regra pré-definida, ela pode ser definida pontualmente por um usuário ao criar um cenário e objetos.

5.4 Interface

Interface utilizada para testes é a mesa descrita no capítulo 4. Porém, o jogo é portátil para qualquer interface multi-toque.

Capítulo 6

Arquitetura e Detalhes de Implementação

6.1 Arquitetura do jogo

A evolução da tecnologia dos hardwares dos computadores pessoais proporcionou grande desenvolvimento à indústria de jogos eletrônicos levando-a a aprimorar seu produto ao longo do tempo. Este caminho foi marcado por grandes equipes e pela extensiva codificação. Os jogos eletrônicos mais modernos podem ultrapassar um milhão de linhas de código de acordo com *Rabin et al.* [24]. Portanto, foi necessário organizar estes extensivos códigos a fim de obter melhor performance no desenvolvimento de novos jogos. Esta organização pode ser realizada por meio da adoção de uma arquitetura bem definida.

Para este jogo escolhemos uma arquitetura de desenvolvimento baseada em um sistema de componentes.

6.1.1 Sistema de Componentes

Um sistema de componentes é caracterizado pela sua estrutura não-hierárquica. O jogo que se baseia em componentes terá somente uma classe pai que representará todas as entidades do jogo, enquanto quaisquer outras classes serão irmãs entre si, pois todas serão filhas da entidade do jogo. Cada classe filha da entidade é um componente, ou seja, a entidade do jogo é formada por uma composição de componentes.

É observado a independência de cada componente, onde podemos, por exemplo, ter a liberdade de transformar um elemento de uma classe jogador em veículo. Entretanto, mesmo com a mudança podemos manter a coerência lógica do jogo, pois estas classes terão a mesma estrutura que forma a entidade do jogo, como audio, corpo físico e aparência.

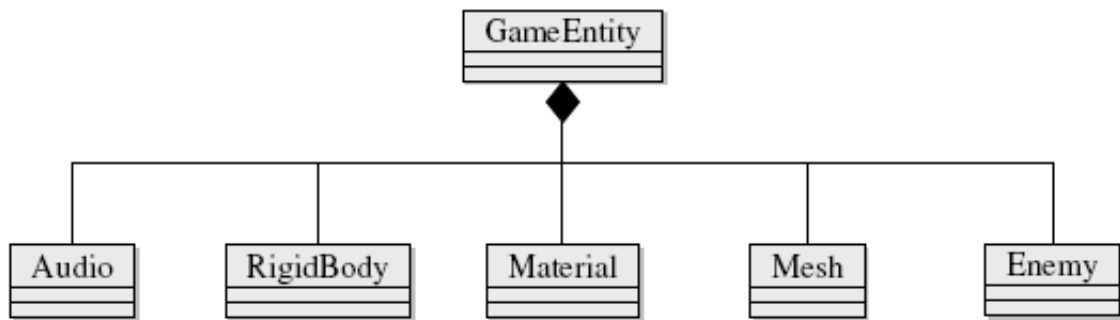


Figura 6.1: Exemplo de uso de componentes em entidades de jogo [4]

6.1.2 Vantagens em relação ao sistema hierárquico

Um sistema baseado em componentes é uma alternativa ao forte acoplamento provocado pela hierarquia de classes. Em linguagens orientadas à objeto a hierarquia de classes pode ser de grande ajuda como pode dificultar a codificação do software. Isto porque, em sistemas onde temos tipos de objetos pré-definidos, assim como as relações entre eles, a hierarquia pode ser de grande ajuda. Isto acontece pelo fato da fácil representação do jogo como um hierarquia. Por exemplo, em jogos de esporte, geralmente, temos características fortemente consolidadas. Um jogador de basquete, dificilmente, se transformará em uma bola. A forte acoplação entre as classes favorece a implementação de uma arquitetura hierarquizada.

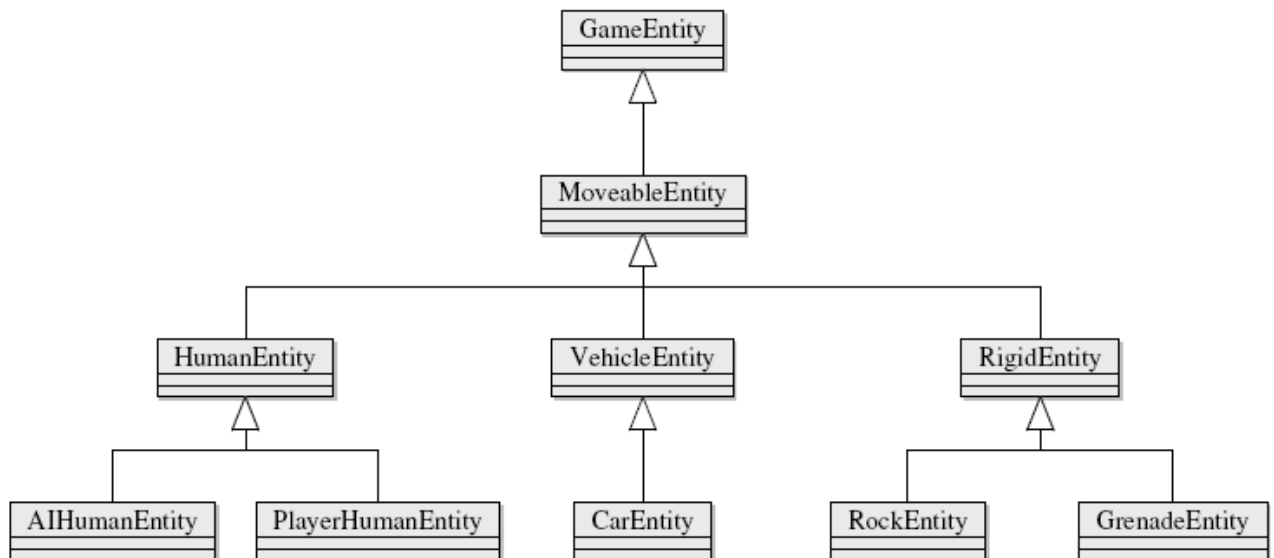


Figura 6.2: Exemplo de uso de hierarquia em entidades de jogo [4]

No entanto, no caso do jogo *DeskWorld* teremos problemas se utilizarmos este tipo

de arquitetura. Isto porque, de acordo com *Rabin et al.* [24], uma arquitetura majoritariamente baseada em hierarquia de classes possui significantes limitações. Uma de suas principais é a baixa flexibilidade devido ao forte acoplamento. O fato de termos classes hierárquicas gera a dependência *inter-classe*, já que, parte de suas características são herdadas de uma classe, sendo que, mudanças realizadas em características de uma classe refletiram sobre suas filhas.

Outro ponto apresentado em [24], é que, em linguagens bem difundidas no desenvolvimento de jogos, como *C++* e *Java*, a estrutura modelada em hierarquia é estática, não permitindo a alteração de classes em tempo de execução. Isto pode ser um grande entrave, pois certos jogos podem realizar diversas alterações drásticas no comportamento de entidades. Podemos exemplificar esse ponto tomando um jogo em que a morte de um certo inimigo possa se transformar em um item ou dinheiro.

6.1.3 Comunicação entre componentes

Para aproveitar a flexibilidade provida pelo sistema de componentes, precisamos realizar a comunicação entre componentes, visto que, será a responsável por engatilhar a transformação de uma entidade. É a mensagem que avisará ao objeto que seu comportamento irá ser alterado e associado a uma outra classe.

Esta comunicação pode ficar a cargo da entidade do jogo, onde, por exemplo, pode realizar a transferência de um som de uma classe de áudio para uma classe veículo. Num sistema simples, essa comunicação pode ser implementada por uma simples chamada de função onde ponteiros são passados referenciando suas respectivas características.

6.2 *Design Patterns*

Design Patterns são padrões de projeto de orientação a objetos que visam reconhecer padrões recorrentes de forma a aumentar a flexibilidade e permitir o reuso do código se usados de forma adequada. Isso se dá pois o seu uso aumenta a independência dos sistemas, o que também aumenta a capacidade de evolução do código em si.

Como foi descrito em [4], para cada padrão, são estabelecidos os seguintes aspectos:

- O **nome** usado para descrever, de maneira sucinta e precisa em no máximo duas palavras, o problema, a solução e as consequências inerentes do padrão.
- O **problema** e o contexto adequados para a aplicação do padrão.
- A **solução** descreve os elementos que compõem o padrão. Cada solução inclui relacionamentos entre classes e a colaboração entre estes casos existam.
- As **consequências** de se aplicar o padrão, ou seja, os resultados e as desvantagens.

Graças ao fato de jogos serem extremamente modulares, eles ganham um grande benefício por utilizar alguns padrões de projeto. Abaixo estão explicados os mais utilizados para este propósito.

6.2.1 *Singleton*

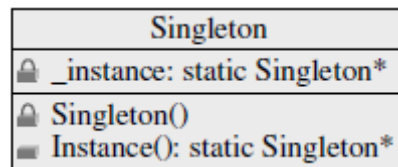


Figura 6.3: Padrão *Singleton* [4]

O padrão *Singleton* trata de um padrão aonde uma classe necessita necessariamente ser única durante todo o programa, mesmo ela podendo ser utilizada em várias situações.

Através desse padrão, a classe *Singleton* cuida para que ela seja instanciada somente uma vez. Podemos verificar a implementação desse padrão na figura [6.3], onde temos uma operação de classe, 'Instance()', que retorna a única instância, e é salvo como ponteiro estático dessa classe. Caso seja a primeira vez que é chamado, a classe é instaciada, caso contrário, retorna a instância já existente. O construtor é privado para que a classe não possa ser criada por fora dela.

Em jogos, o padrão *Singleton* é muito usado para controlar eventos e gerenciar mensagens, bem como controlar recursos. Em todos esses casos, apenas uma instância pode existir e ela deve permitir acesso global a ela.

6.2.2 *Observer*

O padrão *Observer*, visto na figura [6.4], define um padrão onde um objeto, chamado de *subject*, mantém uma lista de observadores e os notifica de mudanças de estados. Quando a informação é passada, os observadores consultam o *subject* para sincronizar os dados. O *subject* não precisa conhecer detalhes do observador.

Utilizando a classe *subject*, podemos adicionar ou remover observadores. O observador, no entanto, define uma interface para objetos que notificam sobre sua alteração para o *subject*. Essa é a única ligação entre o observador e o *subject*.

Este padrão também é reconhecido na indústria de jogo. Como a arquitetura de jogos é marcada pela interação entre entidades distintas, o *observer* deve ser usado. Assim as entidades se conhecem e aguardam receber atualizações de outras entidades que compõem o seu meio ambiente.

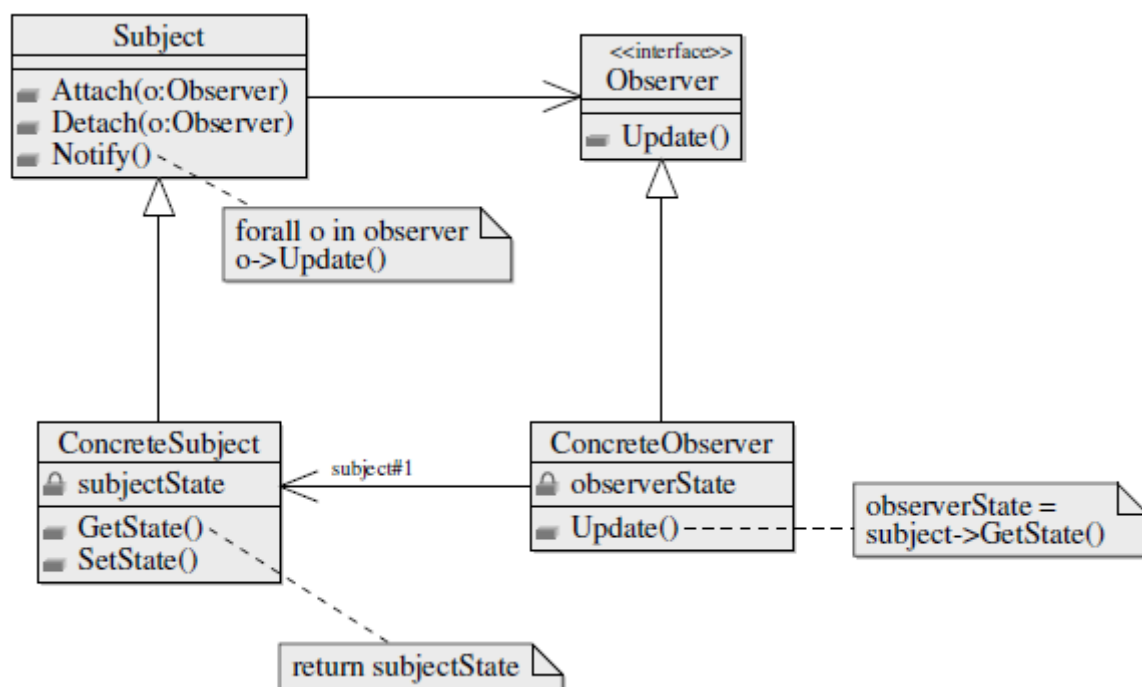


Figura 6.4: Padrão *Observer* [4]

Capítulo 7

~~Estrutura de~~ implementação do simulador

Neste capítulo, será descrito as ferramentas de suporte utilizadas na construção do simulador.

7.1 *Game Engine*

A *engine* utilizada neste jogo foi a *Box2D*. Seu manual pode ser encontrado em [6]. A principal funcionalidade da *Box2D* neste jogo foi auxiliar na criação dos objetos e no tratamento de eventos relacionados a simulação das leis da física. Estes eventos estão apresentados abaixo:

- Gravidade
- Colisão
- Força de ação e reação
- Força de atrito
- Força de fricção

Além dessas funcionalidades, a *Box2D* é responsável pela gerência de memória, junção de objetos e/ou fixação deles.

7.2 *Bibliotecas de apoio*

Para facilitar a implementação de foram utilizadas diversas bibliotecas. Uma das bibliotecas padrão do C++ a STL (*Standart Template Library*) possui algumas estruturas

complexas de dados já implementadas, sendo amplamente utilizada neste simulador. Para a parte gráfica do jogo foi utilizada a biblioteca SDL (*Simple Direct Layer*) conjuntamente com a *OpenGL* (*Open Graphics Library*). Com relação ao áudio, utilizou-se a *SDL_mixer*. Para tratamento de *input* foi utilizada a *Touchlib*.

7.2.1 SDL e *OpenGL*

Por oferecer uma boa abstração de *hardware*, ser bem difundida no mercado de jogos e possuir ampla documentação, utilizou-se a SDL para disponibilizar o acesso ao ambiente *OpenGL*. Conjuntamente com a SDL, na parte gráfica, foi utilizado a *OpenGL* (*Open Graphics Library*) que realiza a renderização das imagens.

A *SDL_mixer* é utilizada para permitir a reprodução de diversas faixas de audio simultaneamente.

7.2.2 Protocolo *TUIO*

Comunicação entre o usuário da mesa com o aplicativo é realizada de acordo com o protocolo *TUIO* [14] (*Tangible User Interface System*). Este protocolo especificado para atender as necessidades da comunicação das interfaces tangíveis. Interfaces tangíveis são interfaces sensíveis a toque, capazes de serem controladas por movimentos corporais e gestos. A implementação é simples e visa melhorar a performance na comunicação. Para isso, ele opera sobre a camada UDP (*User Datagram Protocol*) de transporte utilizando três tipos de mensagens: *set*, *alive* e *fseq*. Mensagens *set* são utilizadas para informar o estado de um objeto. Mensagens *alive* indicam o conjunto de objetos presentes na interface através de uma identificação única atribuída a cada novo elemento reconhecido. Mensagens *fseq* são transmitidas antes da etapa de atualização de cada quadro para marcar-lo unicamente, associando-o a cada mensagem *set* e *alive* dele. Resumindo o funcionamento do protocolo:

- Parâmetros do objeto são enviados após mudança de estado através da mensagem *set*
- Objetos removidos da interface são comunicados através de mensagens *alive*
- Cliente deduz a lista de objetos adicionados e removido por meio das mensagens *set* e *alive*
- Mensagens *fseq* associam um ID a um conjunto de mensagens *set* e *alive* do quadro

Apesar da camada UDP de transporte não oferecer garantia de entrega dos dados, o *TUIO* implementa redundância de informação para se precaver contra a perda de dados na transmissão. Além disso, o estado de um objeto, mesmo inalterado, é enviado periodicamente em uma mensagem *set*. Portanto, o protocolo é ideal para ser utilizado em

aplicativos controlados interfaces multitoques otimizando a iteração e confiabilidade da comunicação.

Parâmetros das mensagens TUIO

Parametros	Significado dos parametros	Tipo
s	sessionID (ID temporário do objeto)	int32
x, y	posição	float32
X, Y	vetor de movimento (velocidade de movimento e direção)	float32
m	aceleração de movimento	float32
w, h	largura e altura do <i>blob</i>	float32

Tabela 7.1: Parâmetros de mensagens TUIO em superfícies interativas 2D

7.2.3 *Touchlib*

Touchlib [32] é uma biblioteca para auxiliar o processamento de imagens capturadas pela webcam em iterações com superfícies multitoque. Ela lida com o acompanhamento de *blobs* de luz infravermelha, e envia para seus programas esses eventos multitoques, como o encostar do dedo, deslocamento do dedo e o retirar do dedo. Inclui um aplicativo de configuração, algumas demos para exemplificar seu uso. Interage com a maioria dos tipos de webcams e os dispositivos de captura de vídeo sendo muito útil para um projeto de *software* para mesa multitoque. Em contrapartida só tem suporte para para sistema operacional *Windows*.

Seu funcionamento consiste na aplicação de filtros de forma customizável como os mostrados nesta figura [3.2]. A execução destes filtros são realizadas pela *Touchlib* com o uso da *OpenCV* (*Open Source Computer Vision Library*), uma biblioteca para o desenvolvimento de aplicativos de processamento de imagens. A execução da *Touchlib* funciona de acordo com o paradigma cliente-servidor, sendo a mesma atuando como servidor e a aplicação como cliente. Esta comunicação cliente-servidor é feita utilizando o protocolo TUIO [14] (*Tangible User Interface System*), permitindo uma arquitetura distribuída, onde o aplicativo é executado separadamente da aplicação do processamento de imagens.

Referências

- [1] Algoryx. Phun. <http://www.phunland.com/wiki/Home>, acessado em 26/06/2010. vii, 9
- [2] M. Barcelona. Multitouch crayon physics. <http://blog.multitouch-barcelona.com/2008/05/multitouch-crayon-physics-is-available.html>, acessado em 26/06/2010. 14
- [3] B. Bors. The role of the mouse in audio games. <http://www.game-accessibility.com/index.php?pagefile=roleMouseAudioGames>, acessado em 26/06/2010. viii, 23
- [4] P. G. Brandão and S. C. R. Braga. Construção de um jogo eletrônico multiusuário em uma superfície de projeção multitoque. Trabalho de graduação, Universidade de Brasília, Departamento de Ciência da Computação, Brasília, July 2009. vii, viii, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 32, 33, 34, 35
- [5] S. F. Brown. How it works: Multi-touch surfaces explained. *Scientific American*, June 2008. Supplement to the feature "Hands On Computing: How Multi-touch Screens Could Change The Way We Interact With Computers and Each Other", printed in the July 2008 issue of Scientific American. <http://www.scientificamerican.com/article.cfm?id=how-it-works-touch-surfaces-explained>, acessado em 26/06/2010. vii, 12, 13
- [6] E. Catto. *Box2D v2.1.0 User Manual*, 2007-2010. Disponível em <http://www.box2d.org/manual.html>. 36
- [7] T. D. Different. Ten great rivalries. <http://www.becks.com/2009/02/06/ten-great-rivalries/>, acessado em 26/06/2010. vii, 4
- [8] T. Edmonds. Numpty physics. <http://numptyphysics.garage.maemo.org/>, acessado em 26/06/2010. 9
- [9] S. C. Entertainment. Playstation 3. <http://us.playstation.com/ps3/>, acessado em 26/06/2010. 9
- [10] S. C. Entertainment. Playstation portable. <http://us.playstation.com/psp/index.htm>, acessado em 26/06/2010. 9
- [11] H. Hartman. Multiplayer games and physics on multi-touch screen devices. Bachelor's thesis, Luleå University of Technology, Departament of Skellefteå Campus, 2008. 13

- [12] P. J. Hruschak. Wiimote vs. move vs. kinect: Comparing control schemes in the three-way battle for motion control. <http://www.gamertell.com/gaming/comment/wiimote-playstation-move-kinect-motion-controller-comparison/>, acessado em 26/06/2010. vii, 6
- [13] Izanagi. Game locker: Ps3 vs xbox 360 vs wii. <http://techknowbabel.com/2010/02/27/game-locker-ps3-vs-xbox-360-vs-wii/>, acessado em 26/06/2010. vii, 5
- [14] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Constanza. Tuio: A protocol for table-top tangible user interfaces. In *6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, 2005. 37, 38
- [15] R. Khaled, P. Barr, H. Johnston, and R. Biddle. Let's clean up this mess: exploring multi-touch collaborative play. In *CHI '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, pages 4441–4446, New York, NY, USA, 2009. ACM. 13
- [16] Kloonigames. Crayon physics. <http://www.crayonphysics.com/>, acessado em 26/06/2010. vii, 8
- [17] J. Luderschmidt. The multi-touch virttable. <http://johannesluderschmidt.de/lang/en-us/the-multi-touch-table-virttable/153/>, acessado em 26/06/2010. vii, viii, 12, 16, 26
- [18] Microsoft. Microsoft surface. <http://www.microsoft.com/surface>, acessado em 26/06/2010. 11
- [19] M. Molecule. Little big planet. <http://www.littlebigplanet.com/>, acessado em 26/06/2010, October 2008. vii, 9, 10
- [20] M. Molecule. Little big planet 2. <http://www.littlebigplanet.com/en-us/2/>, acessado em 26/06/2010, November 2010. 10
- [21] M. Molecule. Modnation racer. <http://www.modnation.com/index/>, acessado em 26/06/2010, May 2010. 10
- [22] L. Y. L. Muller. Multi-touch displays: design, applications and performance evaluation. viii, 16, 19, 20
- [23] T. Perl and S. Kögl. Adaptation and evaluation of numpty physics for multi-touch multi-player interaction. Bachelor's thesis, Vienna University of Technology, Institute of Computerized Automation, Vienna, August 2009. 9, 13
- [24] S. Rabin, editor. *Introduction to game development*. Charles River Media, 2005. 31, 33
- [25] Reactable. <http://www.reactable.com/>, acessado em 26/06/2010. vii, 10, 11
- [26] S. Sandler. Mtmini - diy multitouch mini pad. <http://sethsandler.com/multitouch/mtmini/>, acessado em 26/06/2010, 2008. 10

- [27] W. S. Schneider, N. C. Dias F., L. H. M. Mauruto, and F. R. Miranda. Irtaktiks: Jogo de estratégia para mesa multitoque. *SBGames 2008, Belo Horizonte*, November 2008. 12
- [28] SEBRAE. Brasil e china puxarão crescimento na área de entretenimento até 2014. http://www.sebrae-sc.com.br/novos_destaquos/oportunidade/default.asp?materia=18891, acessado em 26/06/2010. 1
- [29] T. Streeter. The verve project. <http://verveproject.blogspot.com/2008/01/multitouch-game-of-life.html>, acessado em 26/06/2010, January 2008. 13
- [30] T. Teabag. Retro computing corner: The world's first videogames. <http://www.teamteabag.com/2008/05/17/retro-computing-corner-the-worlds-first-videogames/>, acessado em 26/06/2010. vii, 1, 2, 3
- [31] Terra. Conheça a história da atari. <http://games.terra.com.br/interna/0,0I1966183-EI1702,00.html>, acessado em 26/06/2010. vii, 3, 4
- [32] Touchlib: a library for creating multi-touch interaction surfaces. <http://nuigroup.com/touchlib/>, acessado em novembro/2010. 38
- [33] G. Works. Established multitouch gesture support. <http://gestureworks.com/about/supported-gestures/>, acessado em 26/06/2010. viii, 23, 24, 25
- [34] O. T. Zmovirzynski. Odyssey - o primeiro console de videogame da história. <http://midiaville.com.br/blog/momento-de-distrair/odyssey-o-primeiro-console-de-videogame-da-historia/>, acessado em 26/06/2010. vii, 2, 3

Apêndice A

Configuração da Touchlib

A configuração do Touchlib é ~~toda~~ armazenada em um arquivo XML, config.xml.

```
cmtt
<?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>
    <!-- Coordenadas de calibração da superfície -->
</screen>
<filtergraph>
    <dsvlcapture label="dsvlcapture0" />
    <mono label="mono1" />
    <backgroundremove label="backgroundremove2">
        <threshold value="0" />
    </backgroundremove>
    <highpass label="highpass3">
        <filter value="12" />
        <scale value="11" />
    </highpass>
    <scaler label="scaler4">
        <level value="3" />
    </scaler>
    <brightnesscontrast label="brightnesscontrast5">
        <brightness value="0.0980392" />
        <contrast value="0.788235" />
    </brightnesscontrast>
    <rectify label="rectify6">
        <level value="25" />
    </rectify>
</filtergraph>
```