UNIVERSIDADE DE SÃO PAULO
ESCOLA DE COMUNICAÇÕES E ARTES

MARCOS FERNANDEZ CUZZIOL

Estados superpostos:
proposta de modelo matemático para games 3D

São Paulo
2012

MARCOS FERNANDEZ CUZZIOL

Estados superpostos:
proposta de modelo matemático para games 3D

Tese de doutorado produzida como exigência parcial para a obtenção do título de Doutor em Artes, apresentada à Escola de Comunicações e Artes da Universidade de São Paulo

Programa: Artes Visuais

Área de concentração: Poéticas Visuais

Orientador: Prof. Dr. Gilberto dos Santos Prado

São Paulo

2012

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, não comerciais, desde que citada a fonte.

Catalogação da Publicação

Serviço de Biblioteca e Documentação

Escola de Comunicações e Artes da Universidade de São Paulo

C993e

Cuzziol, Marcos Fernandez
Estados superpostos : proposta de modelo matemático para games 3D / Marcos Fernandez Cuzziol. – São Paulo, 2012.
192 p.

Orientador : Prof. Dr. Gilberto dos Santos Prado.
Tese (Doutorado) – Universidade de São Paulo, Escola de Comunicações e Artes.

Inclui referências.

1. Jogos eletrônicos. 2. Realidade virtual. 3. Modelos matemáticos. I.Cuzziol, Marcos Fernandez. II. Prado, Gilberto dos Santos. III. Título.

CDD 794.8

Marcos Fernandez Cuzziol

Estados superpostos: proposta de modelo matemático para games 3D

Tese de doutorado produzida como exigência parcial para a obtenção do título de Doutor em Artes, apresentada à Escola de Comunicações e Artes da Universidade de São Paulo

Programa: Artes Visuais

Área de concentração: Poéticas Visuais

Aprovado em:

Banca Examinadora:

_____	_____
(Nome)	(Assinatura)
_____	_____
(Nome)	(Assinatura)
_____	_____
(Nome)	(Assinatura)
_____	_____
(Nome)	(Assinatura)

Orientador:

Prof. Dr. Gilberto dos Santos Prado	_____
(Nome)	(Assinatura)

CUZZIOL, M. *Estados superpostos: proposta de modelo matemático para games 3D*. 2012. 192p. Tese (Doutorado) – Escola de Comunicações e Artes, Universidade de São Paulo, São Paulo, 2012.

RESUMO

O comportamento de personagens de games 3D em geral apresenta-se de modo repetitivo e previsível. No presente trabalho, avalia-se que o principal motivo para essa previsibilidade não está numa limitação intrínseca da programação desses comportamentos, mas, antes, no modelo matemático utilizado como base para a construção do programa. Através de estudos de caso, identifica-se o modelo mais comum empregado em games e mostra-se que esse modelo não precisa ser necessariamente respeitado. Exemplos de comportamentos de personagens de games e de outras criaturas virtuais são apresentados como demonstração de que é possível obter-se comportamentos não previsíveis e, ao mesmo tempo, não aleatórios, mas orientados a objetivos pré-estabelecidos, a partir de sequências de instruções programadas. Para avaliar um modelo matemático alternativo, apresenta-se primeiro uma breve descrição da mecânica quântica e das implicações que o desenvolvimento dessa teoria teve sobre a compreensão do real. O trabalho apresenta então propostas para a criação de um novo modelo matemático para games 3D, inspiradas na interpretação de Everett da mecânica quântica. Mostra-se a seguir como esse novo modelo poderia gerar games 3D que se adaptariam automaticamente às habilidades específicas do usuário, e como esse novo modelo poderia facilitar a criação de enredos realmente interativos.

PALAVRAS-CHAVE: GAMES, JOGOS ELETRÔNICOS, COMPORTAMENTO ARTIFICIAL, REALIDADE VIRTUAL, MECÂNICA QUÂNTICA, MODELOS MATEMÁTICOS

ABSTRACT

The behavior of 3D game's characters is usually repetitive and predictable. In this work, it is estimated that the main reason for this predictability is not due to an intrinsic limitation of the programming of these behaviors, but rather to the mathematical model applied for building the program. Through case studies, the most common model used in games is identified, and it's shown that this model does not necessarily have to be followed. Examples of behaviors of game characters and other virtual creatures are presented as a demonstration that it is possible to create unpredictable and, at the same time, not random, but rather oriented to pre-established goals, behaviors merely from programmed instruction sequences. To evaluate an alternative mathematical model, a brief description of quantum mechanics is presented, as well as the implications that the development of this theory had on our understanding of the real. The work then presents proposals for the creation of a new mathematical model for 3D games, inspired by the Everett interpretation of quantum mechanics. Following, it's shown how this new model could generate 3D games that adapt automatically to user-specific skills, and how this new model could facilitate the creation of truly interactive storylines.

KEYWORDS: GAMES, ARTIFICIAL BEHAVIOR, VIRTUAL REALITY, QUANTUM MECHANICS, MATHEMATICAL MODELS.

LISTA DE FIGURAS

Figura 1.1. Um cubo em ambiente 3D, representado por triângulos. 23
Figura 1.2. Trajetória parabólica, característica de movimento uniformemente acelerado pela gravidade. 31
Figura 1.3. *Crysis 2* (Electronic Arts, 2011): veículos, personagens e destroços descrevem trajetórias parabólicas quando arremessados por explosões (Electronic Arts, divulgação). 34
Figura 1.4. *Angry Birds Rio*: trajetórias explícitas. Para cada atualização de imagem, uma pequena marca de posição ajuda a traçar visualmente a trajetória realizada. 35
Figura 1.5. *Super Mario Brothers*, 1986. 36
Figura 1.6. A mecânica modificada de *Super Mario Brothers*: alterações no vetor de aceleração da gravidade. 37
Figura 1.7. *Micro Scooter Challenge*. 40
Figura 1.8. Adaptado de gráfico publicado em (CSIKSZENTMIHALYI, 1997, p. 31). 42
Figura 1.9. *Super Mini Racing*: versões de 2001 (à esquerda) e de 2006 (direita). 44
Figura 1.10. *IL-2 Sturmovik*. Imagem da versão Cliffs of Dover (divulgação). 45
Figura 1.11. Configuração de dificuldade – *IL-2 Sturmovik*. 48
Figura 1.12. Adolf Galland na cabine de seu Messerschmitt Bf-109E, em 1940. Note a luneta adaptada, que atravessa o para-brisa do avião (imagem extraída do livro *Luftwaffe Colours – Jagdwaffe – No. 06 – Battle of Britain Phase Two, August–September 1940*, p. 116). 49
Figura 1.13. *IL-2*: aviões, matrículas e distâncias identificados por textos em azul, no centro da imagem. 51
Figura 1.14. *IL-2*: na parte central e inferior da tela, um plano verde e um ícone 3D exibem os ângulos relativos entre o avião e o solo. Nos dois cantos inferiores da tela, instrumentos auxiliam a pilotagem. 52
Figura 1.15. *IL-2*: visão automática na direção do inimigo mais próximo (o pequeno ponto no centro exato da tela, identificado pelo texto em azul). Setas azuis e vermelhas indicam a localização e a distância aproximada de outros aviões. 53
Figura 1.16. *IL-2*: no lado direito, em tons de azul claro, a indicação de tempo expandido. 54
Figura 1.17. *Max Payne* (2001): *bullet-time* em ação. 55
Figura 1.18. *Max Payne 3*, com lançamento previsto para 2012, é ambientado em São Paulo, Brasil. Imagens de divulgação do game exploram o característico efeito de *bullet-time.* 56
Figura 1.19. Animação de personagem em *Incidente em Varginha 2*: o joelho dobrado momentaneamente ao contrário reforça a impressão de impacto do pé com o solo. 57
Figura 2.1. O plano complexo. 62
Figura 2.2. Limite para valores de z. 63
Figura 2.3. O conjunto de Mandelbrot. 64
Figura 2.4. Imagens da representação gráfica do conjunto de Mandelbrot, em aproximações sucessivas. 65
Figura 2.5. Quatro imagens que utilizam uma palete de cores para visualização do perímetro do conjunto de Mandelbrot. Da esquerda para a direita e de cima para baixo, as imagens mostram aproximações sucessivas. 66
Figura 2.6. Construtor universal de von Neumann. 68
Figura 2.7. Um "glider", elemento emergente do *Game of Life*, movimentando-se durante quatro gerações. 69

Figura 2.8. “Light weigh spaceship”..... 70
Figura 2.9. “Gosper’s glider gun”..... 70
Figura 2.10. Trecho da máquina universal de Turing criada por Paul Rendell no *Game of Life* (imagem Paul Rendell). 73
Figura 2.11. Um soldado de Incidente em Varginha, atacando próximo a um muro de proteção. 77
Figura 2.12. *Incidente em Varginha*: *Drakalis* atacando em grupo. 85
Figura 2.13. Frame do vídeo *Evolved Virtual Creatures*, de Karl Sims, 1994..... 87
Figura 2.14. Ecossistema virtual de *Eden* (adaptado sobre imagem de Jon McCormack). 91
Figura 2.15. *Eden* (imagem: Jon McCormack). 92
Figura 3.1. Padrão obtido ao se projetar a luz por uma única fenda. Se a luz fosse de natureza corpuscular, supunha-se que o resultado com duas fendas seria um padrão muito similar, formado pela sobreposição dos efeitos devidos a cada uma delas. 98
Figura 3.2. A luz que atravessa uma das fendas supostamente interfere com a que atravessa a outra, formando o padrão de franjas da figura. 99
Figura 3.3. Visão superior de ondas atravessando fendas. À direita, as ondas que atravessam uma fenda interferem com as que passam pela outra, criando “franjas”. 99
Figura 3.4. Elétrons, emitidos um a um através das duas fendas, também formam um padrão de interferência..... 101
Figura 3.5. A simples introdução de um sensor para determinar por qual das fendas cada elétron passa destrói o padrão de interferência do experimento. 102
Figura 3.6. Soma de Histórias de Feynman: cada trajetória possível contribui para o cálculo da probabilidade de *um* elétron sair de A e atingir B em determinado tempo. A experiência das duas fendas é ilustrada à direita. 111
Figura 3.7. Atualização do campo de possibilidades de posição de um personagem. 123
Figura 3.8. Campos de possibilidades de posição de cinco personagens. 125
Figura 3.9. O avatar do jogador, em laranja, o setor de contato visual, em amarelo, e o personagem localizado, em vermelho..... 126
Figura 3.10. Legenda de sete dos estados possíveis para um game de combate..... 129
Figura 3.11. Exemplo de diagrama. 129
Figura 3.12. Mapa de interação: possíveis mudanças de estados a partir de três estado iniciais superpostos. 131
Figura A1.1. Exemplo de sistema cibernético de segunda ordem. 157
Figura A1.2. Emergência de comportamentos com base em regras de interação. 158
Figura A1.3. Evolução de comportamentos emergentes em sistema cibernético de segunda ordem. 160

SUMÁRIO

INTRODUÇÃO 15
CAPÍTULO 1 23
1.1. Modelos e números 23
1.2. Simulação física 29
1.3. Leis de Newton 29
1.4. Variáveis de movimentação 32
1.5. Super Mario Brothers 35
1.6. Micro Scooter Challenge 38
1.7. Super Mini Racing 43
1.8. IL-2 Sturmovik 45
1.9. Max Payne 54
1.10. Incidente em Varginha 2 56
1.11. O limite do real 59
CAPÍTULO 2 61
2.1. Programação e imprevisibilidade 61
2.2. O conjunto de Mandelbrot 61
2.3. Game of Life 67
2.4. Comportamentos Emergentes em Games 3D 73
2.5. Programação evolutiva 80
2.6. Evoluindo Comportamentos 82
2.7. Evolving Virtual Creatures 85
2.8. Confrontando o programador 88
2.9. Eden 90
2.10. Criaturas deterministas 94
CAPÍTULO 3 97
3.1. A textura do real 97
3.1.1. Ondas e partículas 98
3.1.2. Natureza incerta 103
3.1.3. Uma maneira diferente de entender a realidade 106
3.1.4. Modelo determinista para realidade não determinista 108
3.1.5. Soma de histórias 110
3.1.6. Diagramas de Feynman 111
3.1.7. O gato de Schrödinger 115
3.1.8. A interpretação dos muitos mundos 116
3.2. Estados superpostos: proposta de modelo matemático de realidade para games 3D 118
3.2.1. Similaridades 120
3.2.2. Descritores de estados finitos 121
3.2.3. Descritores de estados: posição e velocidade 122
3.2.4. O observador 124
3.2.5. Consistência 126
3.2.6. Diagramas de personagens 127
3.2.7. Cálculo de probabilidades 131

3.2.8. Enredos interativos 133
3.2.9. Pensar diferente 134
CONCLUSÃO 137
REFERÊNCIAS 143
Publicações 143
Obras digitais 147
Sites 149
GLOSSÁRIO 151
APÊNDICE 1: AUTONOMIA CIBERNÉTICA 155
Autonomia em sistemas deterministas? 155
Interface Cibernética 156
Emergência! 157
Além da emergência 158
Autonomia cibernética 159
Notas sobre o desenvolvimento da ideia 161
APÊNDICE 2: DEFINIÇÃO COMPLETA DE PERSONAGEM 165
APÊNDICE 3: BREVE HISTÓRICO DA PRODUÇÃO BRASILEIRA DE GAMES 3D 185
ANEXO 1: GUIAS RÁPIDOS PARA OS GAMES 191

INTRODUÇÃO

Neste trabalho, apresento resultados de minha pesquisa sobre a construção de comportamentos artificiais para personagens de games 3D[1]. Uso a expressão *comportamento artificial* para designar todo o tipo de ação apresentada por objetos e personagens num game 3D. Prefiro essa designação à mais comum, *inteligência artificial*, por dois motivos. Primeiro, porque seria muito difícil defender que personagens programados tivessem algo que possa ser comparado com o que consideramos inteligência. Segundo, porque a ideia de comportamento é mais abrangente, englobando desde as simples ações de uma porta virtual que abre e fecha, por exemplo, até a simulação de reações táticas de grupos de personagens.

Minha experiência com games 3D iniciou-se em 1994, com a programação de um *engine*[2] próprio, mas foi em 1997, com a criação de comportamentos artificiais para os personagens de *Incidente em Varginha* (Perceptum, 1998), que me deparei com um aspecto aparentemente contraditório da programação: a possibilidade de gerar comportamentos imprevisíveis e adaptativos a partir de meras sequências de instruções.

De todas as técnicas e conhecimentos empregados no desenvolvimento de games 3D, a programação parece ser a menos compreendida e a mais sujeita a pré-concepções, mesmo entre boa parte dos desenvolvedores. É possível que isso decorra do fato de a programação ter duas atribuições distintas num game 3D, que podem ser ilustradas pelo seguinte comentário de Chris Crawford:

> Enquanto designers de games, nós somos deuses absolutos. Um tipo de deus diz, “O.K., agora esta folha cairá um pouquinho aqui e então esse vento vai soprar um pouco mais adiante”. O outro tipo de deus diz, “Aqui

[1] Games que são calculados em três dimensões, mesmo que visualizados numa tela comum, bidimensional.
[2] Programa responsável pela geração de imagens em games 3D. Pode incluir o gerenciamento de sons, animações e física (ver glossário).

> estão as leis da física. Virem-se". (CRAWFORD apud SCHIESEL, 2005, tradução nossa)[3].

Como o primeiro deus, o programador rege todo o rigoroso controle necessário à atualização do ambiente virtual. Num game 3D, objetos e personagens são construídos com polígonos. Cada vértice e cada face visível desses polígonos precisam ter seus parâmetros controlados com precisão e em tempo real. O mesmo vale para as texturas, as animações, os sons, a luz, enfim, para todos os elementos que compõem o ambiente virtual. Além disso, existe a simulação física: objetos têm suas posições calculadas com base nas forças virtuais que atuam sobre eles – e, para isso, usualmente calcula-se a aceleração decorrente da soma de forças sobre a massa do objeto, depois sua velocidade e, finalmente, sua nova posição. Para cumprir esses objetivos, a programação é rígida e previsível. O controle está totalmente nas mãos do programador. Mas estender esse tipo de controle ao comportamento de personagens seria um erro. De fato, até mesmo o uso descrito neste parágrafo, embora corriqueiro no desenvolvimento de games 3D, pode ser um erro se exagerado.

Como o segundo deus, o programador explora o fenômeno da emergência[4] ao dotar personagens de comportamento artificial baseado na interação constante entre diferentes instruções. Nesse segundo caso, a programação é rígida, mas seus resultados não são necessariamente previsíveis. Numa aparente contradição com a função do "primeiro deus", o controle do programador é significativamente reduzido e exercido de forma indireta quando se trabalha com sistemas emergentes. Falaremos sobre essa característica da programação no capítulo dois, mas é importante notar aqui que os dois tipos de programação são construídos exatamente da mesma maneira, como sequência de instruções. O que muda é o pensamento por trás do programa. E essas duas maneiras de pensar estão ilustradas de forma brilhante no comentário de Crawford, mesmo que a compreensão do que seriam as "leis da física" num game 3D seja crítica, como veremos no decorrer deste trabalho.

---

[3] As a game designer you are an absolute god. One kind of god says, "O.K., now this leaf will fall a little bit here, and then this wind will blow a bit over there." The other kind of god says, "Here are the laws of physics. Go for it". (CRAWFORD *apud* SCHIESEL, 2005)

[4] O fenômeno ocorre quando funções complexas e imprevisíveis emergem a partir da interação entre regras mais simples. Um exemplo clássico de emergência é o cérebro humano, formado por bilhões de células relativamente simples que se comunicam entre si e de onde emergem o raciocínio e a consciência.

A programação é a ferramenta utilizada para a criação de comportamentos artificiais de personagens, sejam eles emergentes ou não, mas o foco desta tese não está na programação. Está, antes, nesses diferentes modos de pensar e nos modelos que servem de base para a construção do programa.

Games são invariavelmente construídos sobre algum tipo de modelo matemático: um conjunto de regras, dados e fórmulas que define comportamentos de personagens e objetos no ambiente virtual. Por representarem ambientes tridimensionais populados por objetos com massa, sujeitos a forças e acelerações, modelos matemáticos são normalmente mais complexos em games 3D, e incluem fórmulas diretamente extraídas da mecânica clássica, de Isaac Newton. Essa estratégia permite simular, com relativa facilidade, comportamentos esperados de objetos reais, como trajetórias e colisões. Por esse motivo, modelos inspirados na mecânica clássica formam a base de praticamente qualquer simulação ou game em ambientes tridimensionais.

Um problema que enfrento desde que comecei a desenvolver comportamentos artificiais para personagens 3D, entretanto, é a aparente rigidez funcional dessas criaturas artificiais, ainda que elas sejam controladas por algoritmos que permitam certa imprevisibilidade. Pode-se jogar dezenas de vezes determinado nível de um game, mas, de modo geral, os personagens ainda estarão nos mesmos lugares e reagirão das mesmas maneiras. Até em superproduções recentes, como *Crysis* (Electronic Arts, 2008), personagens se comportam de modo essencialmente idêntico quando se deparam com o avatar do jogador, não importando quantas vezes a cena tenha se repetido ou o que o jogador tenha feito até chegar a eles. Essa repetição de comportamentos não parece ser problema para a indústria de games, e é difícil encontrar qualquer menção a essa característica na literatura da área. Quando muito, a repetibilidade de comportamentos parece ser tacitamente aceita como resultado da própria programação que rege esses personagens.

De fato, vários games baseiam-se na repetibilidade de comportamento de seus personagens. Existem aqueles que nem mesmo apresentam o que se poderia chamar de personagens – um exemplo é *Tetris* (Alexei Pajitnov, 1984), em que peças que caem devem ser encaixadas para formar fileiras contínuas. Em outros,

como *Pac-Man* (Namco, 1980), os personagens tem comportamentos propositalmente previsíveis. Outro exemplo, mais atual, é *Angry Birds* (Rovio Mobile, 2009), um game 2D de ação em que pequenos pássaros são atirados de um estilingue com a finalidade de destruir fortalezas e seus inimigos, os porcos. Mas os personagens de *Angry Birds* comportam-se como objetos – o game manteria o mesmo modelo matemático se eles fossem substituídos por pedras ou projéteis, por exemplo[5]. A peças de *Tetris* comportam-se deliberadamente como objetos, enquanto que a experiência proposta ao jogador por *Pac-Man* está baseada na previsibilidade de comportamentos de seus personagens inimigos, na capacidade de se prever o que eles farão para decidir se determinado caminho é viável ou não. Se os fantasmas de *Pac-Man* demonstrassem capacidades mais sofisticadas, mudando repentinamente de direção ou protegendo áreas importantes do tabuleiro, por exemplo, a experiência perderia muito do sentido original e o game seria completamente diferente.

Enfim, uma experiência de jogo pode perfeitamente ser desenvolvida com base em personagens previsíveis. Isso não é um problema. O problema seria limitar *qualquer* experiência de jogo a personagens previsíveis.

Num outro extremo estão games 3D atuais que apresentam cenas literalmente cinematográficas, em que personagens são animados previamente para atuar em trechos importantes do enredo. Conhecidas como *cutscenes*, essas sequências funcionam, mas às custas da interação: cabe ao jogador apenas assisti-las. Pré-programar sequências de atuação de personagens para o desenrolar de um enredo é uma solução comum em games 3D, mas é evidente que, nessas situações, o produto afasta-se das características de um game para se aproximar das de um filme. De certa maneira, *cutscenes* lembram as animações simples dos fantasmas inimigos de *Pac-Man* ou a animação de expressões dos pássaros de *Angry-Birds*. Elas estão lá para suprir algo que o modelo matemático não prevê. Personagens agem como objetos programados e previsíveis durante as partes interativas, e apresentam alguma capacidade dramática nas sequências lineares. Mas essas sequências são pré-determinadas e não podem se adaptar durante o desenrolar do

[5] Ainda assim, é inegável o efeito de empatia que esses pássaros provocam sobre os jogadores, através da animação de expressões cuidadosamente desenvolvidas. O game não teria o mesmo sucesso se eles fossem substituídos por pedras e projéteis.

game (pode-se apenas criar uma série delas, para selecionar a mais adequada em determinado momento). A limitação é essa: se não for possível desenvolver comportamentos que, de alguma forma, permitam a um personagem 3D “atuar” em tempo real, games estarão limitados a testes de habilidade interativos, talvez intercalados por sequências cinematográficas lineares e pré-definidas.

Nesse sentido, a repetição de comportamentos descrita anteriormente é extremamente limitante em games que apresentam personagens mais complexos, dos quais se espera a capacidade de tomar decisões com base em propósitos próprios, ou mesmo em emoções simuladas. A limitação ocorre não apenas pela previsibilidade que os personagens possam apresentar, mas também pela dificuldade de se obter comportamentos que se adaptem a condições diferentes durante o game. Como consequência, os desafios que um game 3D baseado em personagens mais sofisticados propõe para seus jogadores apresentam-se também de forma relativamente rígida, e cabe quase que exclusivamente ao jogador adaptar-se para superá-los.

Durante meu trabalho com a criação de comportamentos programados, sempre me pareceu que a causa dessa limitação fosse anterior ao programa, que o modelo matemático da mecânica clássica fosse, pelo menos em parte, responsável por personagens mais previsíveis e menos interessantes. Quando a regra é desenvolver personagens sujeitos a forças de ação e reação, a acelerações, velocidades e trajetórias, a característica de previsibilidade torna-se apenas uma consequência natural do trabalho.

Como hipótese, assumiremos que o desenvolvimento de um modelo matemático diferente do tradicional poderia, por permitir a criação de personagens com comportamentos adaptáveis a mudanças de contexto durante o desenrolar do game, equilibrar a dificuldade de um game 3D às habilidades específicas do jogador, tornando a experiência virtual mais intensa e interessante para um público muito mais amplo. Além disso, assumiremos que o emprego de um novo modelo poderia indicar caminhos para a criação de enredos realmente interativos, em que a atuação dos personagens pudesse se adaptar à sequência de ações do jogador.

Antes de apresentar propostas para um novo modelo matemático, entretanto, será necessário mostrar que é possível contrariar o modelo da mecânica clássica em games 3D. Além disso, precisaremos demonstrar que a programação em si não limita o comportamento de personagens à previsibilidade ou à simples aleatoriedade. Faremos isso com a apresentação de exemplos e estudos de caso, extraídos tanto de games desenvolvidos por mim como de obras de terceiros. Desses exemplos, alguns já foram mencionados em minha dissertação de mestrado "Games 3D: Aspectos de Desenvolvimento", mas eles são retomados aqui em contexto diferente, sob o ponto de vista do modelo utilizado e não das técnicas de criação. O desenvolvimento deste trabalho ocorre na sequência resumida a seguir.

O primeiro capítulo discorre sobre a faceta da programação de games 3D que usualmente busca a previsibilidade. Veremos como todo o conteúdo de um game – desde texturas, sons, modelos 3D e até o próprio programa – resume-se a grupos de números, característica básica para a compreensão da importância dos modelos matemáticos empregados para integrar esses conteúdos num produto coerente. É aqui também que as equações de Newton são utilizadas como simulação física, modelando a parte mais mecânica dos comportamentos de personagens e objetos. A trajetória de destroços virtuais numa explosão e os saltos de um personagem são exemplos da aplicação de simulação física. Como objetivo principal desse capítulo, veremos exemplos de games que distorcem ou desrespeitam as fórmulas de Newton com resultados interessantes e mesmo paradoxais: da criação de um gameplay[6] mais agradável, passando pela adaptação automática da dificuldade de jogo às habilidades do jogador, até a criação de animações que parecem mais reais com base em movimentos que seriam impossíveis segundo as regras da mecânica clássica.

No segundo capítulo, analisaremos o aspecto imprevisível da programação de games 3D: a criação de comportamentos menos mecânicos, mais interessantes e adaptativos. Primeiro, entretanto, será necessário responder a algumas perguntas: Pode uma sequência fixa de instruções, um programa, gerar comportamentos imprevisíveis? Essa imprevisibilidade estaria limitada à aleatoriedade ou poderia ser

[6] Traduzido algumas vezes como "jogabilidade", *gameplay* é, na verdade, mais do que isso. Entendo *gameplay* como o conjunto de experiências interativas entre o jogador e o game, entre ser humano, hardware e software (ver glossário).

consistente com objetivos propostos e adaptativa? Veremos exemplos extraídos de fractais, de autônomos celulares, da programação de games e de obras de arte interativa. Finalizando o capítulo, tentaremos responder a uma pergunta proposta por Otto Rössler (1998): pode um programa forçar o programador a responder?

O terceiro capítulo é o mais complexo. Como possível alternativa para os games 3D, analisaremos um modelo de realidade radicalmente diferente do da mecânica clássica. Trata-se do modelo que suplantou o da mecânica de Newton nos primeiros anos do século XX, o da mecânica quântica. Algumas fórmulas serão apresentadas no decorrer do capítulo, com o simples intuito de ilustrar as ferramentas matemáticas utilizadas. Obviamente, não nos interessam as fórmulas em si, pois, ao tentar aplicá-las diretamente, estaríamos incorrendo em erro semelhante ao que criticamos quanto ao uso generalizado da mecânica clássica. Ao contrário, interessa-nos as possibilidades de novos conceitos de realidade que advêm da mecânica quântica. Essas novas maneiras de entender a realidade guiam, então, uma série de propostas para o desenvolvimento de um novo modelo matemático de realidade, especifico para games 3D. Veremos como as incertezas de posição e velocidade, aliadas ao cálculo de amplitudes de probabilidades da mecânica quântica, podem levar à criação de um modelo matemático que permita não apenas comportamentos interessantes e pouco previsíveis, mas que também facilite a adaptação de games 3D às habilidades específicas de diferentes jogadores humanos. Além disso, veremos como as características desse novo modelo podem facilitar o desenvolvimento de enredos interativos. Apresento essas propostas como linhas gerais, não como modelo completo e funcional, pois isso exigiria um trabalho bastante diverso, com foco na programação. Entretanto, acredito que as linhas gerais apresentadas possam ajudar significativamente na programação de um modelo funcional.

Após as considerações finais, apresento um glossário dos principais termos técnicos utilizados neste trabalho. No apêndice 1, uma breve descrição sobre o desenvolvimento da ideia de *autonomia cibernética*, que serviu de base conceitual para três exposições da bienal Emoção Art.ficial (em 2006, 2008 e 2010, no Instituto Itaú Cultural). Os dois apêndices seguintes foram publicados em minha dissertação de mestrado, mas podem contribuir para a contextualização e compreensão deste

trabalho: O apêndice 2 traz, a título de referência, a definição completa de um dos personagens de *Incidente em Varginha*, seus *sprites* e sua máquina de estados finitos, escrita em linguagem *C-Script*. No apêndice 3, um breve histórico da produção brasileira de games 3D é apresentado.

O anexo 1 apresenta guias rápidos para dois games: *Micro Scooter Challenge* (Perceptum, 2001) e *Incidente em Varginha* (Perceptum, 1998), dos quais extraio exemplos no primeiro e segundo capítulos. Cópias em CD-Roms desses dois games encerram este trabalho. Vale lembrar que, como esses games foram produzidos para Windows em 1998 e 2001, será necessário utilizar o modo de compatibilidade para que eles funcionem em versões mais atuais desse sistema operacional.

CAPÍTULO 1

1.1. Modelos e números

Objetos e personagens de games 3D são representados por modelos 3D. A palavra é a mesma que usamos em “modelos matemáticos”, na introdução – e não sem um bom motivo, pois o visual de personagens e objetos também é representado por modelos matemáticos. Modelos 3D, especificamente, são construídos por polígonos, sobre os quais se aplicam texturas, imagens que conferem o visual de suas superfícies. A unidade de construção de um modelo 3D é o polígono mais simples, o triângulo.

Quando pensamos em modelos 3D, é natural que os visualizemos como imagens gráficas, como o exemplo do cubo da figura abaixo. Programas de modelagem 3D apresentam modelos dessa forma e permitem que os criemos e editemos da mesma maneira, visualmente. Por isso é bastante comum que modelos sejam confundidos com as imagens que deles resultam.

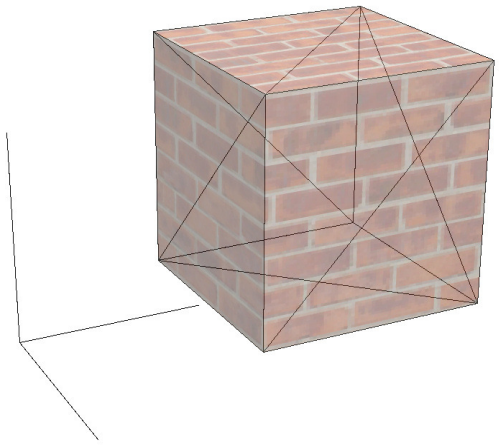


Figura 1.1. Um cubo em ambiente 3D, representado por triângulos.

Mas um modelo 3D não é o que vemos representado na tela. Trata-se de um conjunto de valores que representam coordenadas, índices de vértices que formam

triângulos, valores de propriedades de materiais, coordenadas de textura, entre outras atribuições, todas numéricas. Para propiciar uma ideia da complexidade dessa representação, convém analisar a definição de um objeto 3D em arquivo digital. O formato escolhido, Microsoft DirectX ASCII, apresenta valores numéricos que definem um modelo 3D em estrutura relativamente legível. Na listagem temos a definição do cubo da figura acima em formato DirectX. Notas explicativas foram incluídas após as barras duplas, no lado direito da listagem:

```
xof 0302txt 0064
Header {                                                // cabeçalho com número de objetos e materiais
        1;
        0;
        1;
}
Material CuboMaterial_0_0 {                             // definição de propriedades do material 0
        0.650980; 0.298039; 0.196078; 1.000000;;
        0.000000;
        0.100000; 0.100000; 0.100000;;
        0.000000; 0.000000; 0.000000;;
        TextureFilename {
                "tijolos02.jpg";                // arquivo de imagem da textura
        }
}
Mesh CuboMesh {                                         // definição do objeto
        24;                                             // número de vértices
        -1.000000;1.000000;-1.000000;,                  // coordenadas do vértice  0
        -1.000000;3.000000;-1.000000;,                  // coordenadas do vértice  1
        -3.000000;1.000000;-1.000000;,                  // coordenadas do vértice  2
        -3.000000;3.000000;-1.000000;,                  // coordenadas do vértice  3
        -1.000000;1.000000;-3.000000;,                  // coordenadas do vértice  4
        -3.000000;1.000000;-3.000000;,                  // coordenadas do vértice  5
        -3.000000;3.000000;-3.000000;,                  // coordenadas do vértice  6
        -1.000000;3.000000;-3.000000;,                  // coordenadas do vértice  7
        -1.000000;1.000000;-1.000000;,                  // coordenadas do vértice  8
        -3.000000;1.000000;-1.000000;,                  // coordenadas do vértice  9
        -3.000000;1.000000;-3.000000;,                  // coordenadas do vértice 10
        -3.000000;1.000000;-1.000000;,                  // coordenadas do vértice 11
        -3.000000;3.000000;-1.000000;,                  // coordenadas do vértice 12
        -3.000000;3.000000;-3.000000;,                  // coordenadas do vértice 13
        -3.000000;3.000000;-1.000000;,                  // coordenadas do vértice 14
        -1.000000;3.000000;-1.000000;,                  // coordenadas do vértice 15
        -1.000000;3.000000;-3.000000;,                  // coordenadas do vértice 16
        -1.000000;1.000000;-3.000000;,                  // coordenadas do vértice 17
        -3.000000;1.000000;-3.000000;,                  // coordenadas do vértice 18
        -3.000000;3.000000;-3.000000;,                  // coordenadas do vértice 19
        -1.000000;3.000000;-1.000000;,                  // coordenadas do vértice 20
        -1.000000;1.000000;-1.000000;,                  // coordenadas do vértice 21
        -1.000000;1.000000;-3.000000;,                  // coordenadas do vértice 22
        -1.000000;3.000000;-3.000000;;                  // coordenadas do vértice 23
        12;                                             // número de faces
        3;3,2,0;,                                       // três vértices da face 0
        3;3,0,1;,                                       // três vértices da face 1
        3;5,4,8;,                                       // três vértices da face 2
        3;5,8,9;,                                       // três vértices da face 3
        3;6,10,11;,                                     // três vértices da face 4
        3;6,11,12;,                                     // três vértices da face 5
        3;7,13,14;,                                     // três vértices da face 6
        3;7,14,15;,                                     // três vértices da face 7
        3;16,17,18;,                                    // três vértices da face 8
        3;16,18,19;,                                    // três vértices da face 9
        3;20,21,22;,                                    // três vértices da face 10
        3;20,22,23;;                                    // três vértices da face 11
        }
        MeshTextureCoords {                     // coordenadas de textura
                24;                             // número de vértices texturizados
                0.000000;-0.333333;,            // coordenadas de textura vértice 0
                0.000000;-0.666667;,            // coordenadas de textura vértice 1
```

```
                0.250000;-0.333333;,            // coordenadas de textura vértice 2
                0.250000;-0.666667;,            // coordenadas de textura vértice 3
                0.500000;0.000000;,             // coordenadas de textura vértice 4
                0.500000;-0.333333;,            // coordenadas de textura vértice 5
                0.500000;-0.666667;,            // coordenadas de textura vértice 6
                0.500000;-1.000000;,            // coordenadas de textura vértice 7
                0.250000;0.000000;,             // coordenadas de textura vértice 8
                0.250000;-0.333333;,            // coordenadas de textura vértice 9
                0.500000;-0.333333;,            // coordenadas de textura vértice 10
                0.250000;-0.333333;,            // coordenadas de textura vértice 11
                0.250000;-0.666667;,            // coordenadas de textura vértice 12
                0.500000;-0.666667;,            // coordenadas de textura vértice 13
                0.250000;-0.666667;,            // coordenadas de textura vértice 14
                0.250000;-1.000000;,            // coordenadas de textura vértice 15
                0.750000;-0.666667;,            // coordenadas de textura vértice 16
                0.750000;-0.333333;,            // coordenadas de textura vértice 17
                0.500000;-0.333333;,            // coordenadas de textura vértice 18
                0.500000;-0.666667;,            // coordenadas de textura vértice 19
                1.000000;-0.666667;,            // coordenadas de textura vértice 20
                1.000000;-0.333333;,            // coordenadas de textura vértice 21
                0.750000;-0.333333;,            // coordenadas de textura vértice 22
                0.750000;-0.666667;;            // coordenadas de textura vértice 23
        }
        MeshMaterialList {                      // lista de materiais do objeto
                1;                              // número de sub-objetos
                12;                             // número de faces
                0,                              // material da face 0
                0,                              // material da face 1
                0,                              // material da face 2
                0,                              // material da face 3
                0,                              // material da face 4
                0,                              // material da face 5
                0,                              // material da face 6
                0,                              // material da face 7
                0,                              // material da face 8
                0,                              // material da face 9
                0,                              // material da face 10
                0;;                             // material da face 11
                {CuboMaterial_0_0}
        }
}
```

O cubo é definido com 24 vértices, em vez dos 8 esperados, porque a cada triângulo podem estar associados material e mapeamento de textura próprios. São então necessários vértices diferentes para triângulos adjacentes, pois as coordenadas de textura podem ser diversas.

É com esse conjunto de valores que um programa deve lidar ao manipular um cubo no espaço virtual. Para ser exato, o conjunto de valores será significativamente maior. Não estão incluídas na definição do modelo as sequências numéricas que representam sua textura (no caso, armazenadas no arquivo externo “tijolos02.jpg”), os valores e vetores de iluminação, as texturas de sombreamento nem o mapeamento de reflexos ou de normais. Todos esses valores devem ser gerenciados pelo programa em tempo real. Cada coordenada deve passar por transformações matriciais e ser projetada segundo a posição atual da câmera. Cada pixel da textura deve ser recalculado de acordo com as posições das luzes

dinâmicas, do efeito de fog, dos mapeamentos de sombras e, caso aplicável, de vetores normais. Enfim, a imagem resultante deve ser recalculada dezenas de vezes por segundo para permitir a interação com os modelos digitais em ambiente virtual.

A imagem “tijolos02.jpg”, utilizada como textura no exemplo do cubo, também é formada exclusivamente por valores numéricos. O formato de arquivos Jpeg não é tão legível como o do exemplo do cubo, porque os valores que representam as cores dos pixels (sequências de três números inteiros para os componentes vermelho, verde e azul) estão condensados para otimização de espaço, evitando repetições desnecessárias. O início do arquivo é apresentado abaixo: do lado esquerdo, em valores hexadecimais[7] e à direita, os caracteres equivalentes em código ASC II[8]. Valores que estão fora da tabela ASC II são representados no lado direito por pontos. As letras “JFIF”, a partir da sétima posição numérica, fazem parte da identificação do formato Jpeg do arquivo.

```
3F  3F  3F  3F  00  10  4A  46  49  46     |   ? ? ? ? . . J F I F
00  01  01  00  00  01  00  01  00  00     |   . . . . . . . . . .
3F  3F  00  43  00  05  03  04  04  04     |   ? ? . C . . . . . .
03  05  04  04  04  05  05  05  06  07     |   . . . . . . . . . .
0C  08  07  07  07  07  0F  0B  0B  09     |   . . . . . . . . . .
0C  11  0F  12  12  11  0F  11  11  13     |   . . . . . . . . . .
16  1C  17  13  14  1A  15  11  11  18     |   . . . . . . . . . .
21  18  1A  1D  1D  1F  1F  1F  13  17     |   ! . . . . . . . . .
22  24  22  1E  24  1C  1E  1F  1E  3F     |   " $ " . $ . . . . ?
3F  00  43  01  05  05  05  07  06  07     |   ? . C . . . . . . .
0E  08  08  0E  1E  14  11  14  1E  1E     |   . . . . . . . . . .
1E  1E  1E  1E  1E  1E  1E  1E  1E  1E     |   . . . . . . . . . .
1E  1E  1E  1E  1E  1E  1E  1E  1E  1E     |   . . . . . . . . . .
1E  1E  1E  1E  1E  1E  1E  1E  1E  1E     |   . . . . . . . . . .
1E  1E  1E  1E  1E  1E  1E  1E  1E  1E     |   . . . . . . . . . .
1E  1E  1E  1E  1E  1E  1E  1E  3F  3F     |   . . . . . . . . ? ?
00  11  08  01  00  01  00  03  01  22     |   . . . . . . . . . "
00  02  11  01  03  11  01  3F  3F  00     |   . . . . . . . ? ? .
1F  00  00  01  05  01  01  01  01  01     |   . . . . . . . . . .
01  00  00  00  00  00  00  00  00  01     |   . . . . . . . . . .
02  03  04  05  06  07  08  09  0A  0B     |   . . . . . . . . . .
3F  3F  00  3F  10  00  02  01  03  03     |   ? ? . ? . . . . . .
02  04  03  05  05  04  04  00  00  01     |   . . . . . . . . . .
7D  01  02  03  00  04  11  05  12  21     |   } . . . . . . . . !
31  41  06  13  51  61  07  22  71  14     |   1 A . . Q a . " q .
32  3F  3F  3F  08  23  42  3F  3F  15     |   2 ? ? ? . # B ? ? .
52  3F  3F  24  33  62  72  3F  09  0A     |   R ? ? $ 3 b r ? . .
16  17  18  19  1A  25  26  27  28  29     |   . . . . . % & ' ( )
2A  34  35  36  37  38  39  3A  43  44     |   * 4 5 6 7 8 9 : C D
```

[7] Valores representados com base em 16 dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Úteis em programação porque cada 2 dígitos equivalem exatamente a um byte de 8 bits.
[8] Código que relaciona valores numéricos a caracteres.

```
45  46  47  48  49  4A  53  54  55  56      |    E F G H I J S T U V
                                (...)
```

O mesmo vale para um arquivo de som qualquer. No exemplo abaixo, note, ao lado direito, as letras “OggS” e, pouco adiante, “vorbis”, partes do cabeçalho de um arquivo de som do tipo Ogg. Essas “assinaturas” são importantes para que um programa possa diferenciar uma massa numérica de outra. De outro modo, seria difícil saber do que se trata o arquivo e, principalmente, como interpretá-lo.

```
4F  67  67  53  00  02  00  00  00  00      |    O g g S . . . . . .
00  00  00  00  6A  1E  00  00  00  00      |    . . . . j . . . . .
00  00  4B  3F  3F  3F  01  1E  01  76      |    . . K ? ? ? . . . v
6F  72  62  69  73  00  00  00  00  01      |    o r b i s . . . . .
44  3F  00  00  3F  3F  3F  3F  29  2D      |    D ? . . ? ? ? ? ) -
01  00  3F  3F  3F  3F  3F  01  4F  67      |    . . ? ? ? ? ? . O g
67  53  00  00  00  00  00  00  00  00      |    g S . . . . . . . .
00  00  6A  1E  00  00  01  00  00  00      |    . . j . . . . . . .
3F  3F  3F  3F  0E  49  3F  3F  3F  3F      |    ? ? ? ? . I ? ? ? ?
3F  3F  3F  3F  3F  3F  3F  3F  11  03      |    ? ? ? ? ? ? ? ? . .
76  6F  72  62  69  73  39  00  00  00      |    v o r b i s 9 . . .
42  53  3B  20  4C  61  6E  63  65  72      |    B S ;   L a n c e r
28  53  53  45  29  20  5B  32  30  30      |    ( S S E )   [ 2 0 0
36  31  31  31  30  5D  20  28  62  61      |    6 1 1 1 0 ]   ( b a
73  65  64  20  6F  6E  20  61  6F  54      |    s e d   o n   a o T
75  56  20  62  35  20  5B  32  30  30      |    u V   b 5   [ 2 0 0
36  31  30  32  34  5D  29  00  00  00      |    6 1 0 2 4 ] ) . . .
00  01  05  76  6F  72  62  69  73  22      |    . . . v o r b i s "
42  43  56  01  00  40  00  00  24  73      |    B C V . . @ . . $ s
18  2A  46  3F  73  16  3F  10  1A  42      |    . * F ? s . ? . . B
50  19  3F  1C  42  3F  6B  3F  19  42      |    P . ? . B ? k ? . B
4C  11  3F  1C  32  4C  5B  3F  25  73      |    L . ? . 2 L [ ? % s
3F  21  3F  3F  42  3F  5B  28  3F  3F      |    ? ! ? ? B ? [ ( ? ?
55  00  00  40  00  00  3F  41  78  14      |    U . . @ . . ? A x .
3F  3F  41  08  21  3F  25  3D  58  3F      |    ? ? A . ! ? % = X ?
3F  27  3D  08  21  3F  3F  39  78  14      |    ? ' = . ! ? ? 9 x .
3F  69  41  08  21  3F  10  42  08  21      |    ? i A . ! ? . B . !
                                (...)
```

Qualquer arquivo de computador pode ser representado no formato acima. Não deve ser surpresa que isso valha para o próprio programa que executa um game 3D. No exemplo abaixo, o início da sequência numérica do arquivo principal de *Incidente em Varginha*:

```
47  41  4D  45  2E  57  44  4C  00  00      |    G A M E . W D L . .
00  00  00  00  00  3C  18  00  00  3F      |    . . . . . < . . . ?
5A  3F  50  41  54  48  20  22  2E  2E      |    Z ? P A T H   " . .
3F  5C  5C  72  65  73  6F  75  72  3F      |    ? \ \ r e s o u r ?
```

```
63  65  5C  5C  74  65  78  74  3F  75      |    c e \ \ t e x t ? u
3F  3F  22  3B  0D  0A  3F  3F  5C  7F      |    ? ? " ; . . ? ? \
5C  70  61  6C  65  74  74  08  0F  1E      |    \ p a l e t t . . .
3F  3F  66  6F  6E  74  29  0F  3F  3F      |    ? ? f o n t ) . ? ?
3F  3F  3F  6E  64  46  0F  3F  3F  61      |    ? ? ? n d F . ? ? a
63  74  6F  79  72  64  0F  3F  3F  68      |    c t o y r d . ? ? h
69  6E  67  3F  0F  3F  3F  3F  6F  76      |    i n g ? . ? ? ? o v
65  72  6C  61  79  5C  3F  0F  3F  3F      |    e r l a y \ ? . ? ?
57  64  6C  3F  0C  63  43  00  3F  61      |    W d l ? . c C . ? a
74  6F  3F  0D  67  69  6F  6E  3F  61      |    t o ? . g i o n ? a
6C  3F  0B  65  73  61  3F  0B  73  3F      |    l ? . e s a ? . s ?
65  3F  0B  6D  6A  73  68  69  70  3F      |    e ? . m j s h i p ?
18  1C  64  65  6E  74  72  3F  0C  73      |    . . d e n t r ? . s
3F  64  72  6F  6D  27  12  0D  0A  42      |    ? d r o m ' . . . B
3F  49  4E  44  20  3C  4D  6A  2E  0F      |    ? I N D   < M j . .
57  44  4C  3E  3F  00  6B  13  3F  04      |    W D L > ? . k . ? .
73  1B  5D  69  4D  10  6F  30  31  73      |    s . ] i M . o 0 1 s
1B  53  60  12  2E  73  1B  45  73  61      |    . S ` . . s . E s a
73  1B  52  00  14  3F  1F  3F  6F  30      |    s . R . . ? . ? o 0
32  73  1B  73  3F  1C  53  64  3F  3F      |    2 s . s ? . S d ? ?
                                (...)
```

É preciso dizer que arquivos numéricos são usados por qualquer mídia digital, não apenas por games. Mas existem diferenças básicas entre games 3D e mídias que usam arquivos similares aos dos exemplos acima de forma essencialmente linear. Nos games esses conteúdos são utilizados para criar um ambiente simulado, com o qual é possível interagir. Enquanto um personagem de filme digital aparece na tela como conjunto de pixels captados de um ator humano, ou como animação pré-produzida de um personagem tridimensional, num game 3D essas imagens são recriadas dezenas de vezes por segundo, em tempo real, com base em modelos. Isso é necessário para que seja possível interagir com o ambiente virtual.

É por esse motivo que os modelos matemáticos são tão importantes. Podemos dizer que esses modelos são o equivalente da realidade num ambiente virtual de game 3D. Por eles passam todos os conteúdos numéricos, e deles resultam a sequência de instruções que forma o programa executável. Enfim, tudo o que acontece num game 3D está modelado matematicamente de uma forma ou de outra, e até mesmo a interação com seres humanos depende deles. É estranho que essa modelagem matemática ainda seja tão mecânica, reativa e pouco explorada. E que ainda seja vista, basicamente, como mera imitação da realidade, ou melhor, daquilo que é interpretado como realidade.

1.2. Simulação física

Games 3D sempre apresentam algum tipo de simulação física. Um objeto móvel possui, no espaço tridimensional, variáveis numéricas que representam sua posição e sua velocidade em determinado momento. Quando existe a possibilidade de colisão com outros objetos, ou de alteração da velocidade, torna-se necessária outra variável numérica para representar a massa do corpo. Forças simuladas são usadas no cálculo de valores de aceleração aplicados aos objetos. A noção de trajetória, que define as posições do objeto ao longo do tempo, é fundamental em cálculos de atualização de ambientes virtuais 3D.

Via de regra, simulações físicas são implementadas em games através da aplicação de equações de movimento, que tiveram origem nas leis de Newton. Esse conjunto de equações e leis forma a base da *mecânica clássica*, parte da física que descreve o movimento de objetos macroscópicos. A aplicação em games vem de longa data: o primeiro game digital – *SpaceWar!*, programado originalmente por Steve Russel em 1962, no MIT – foi também o primeiro a utilizar a mecânica clássica como modelo de simulação[9].

1.3. Leis de Newton

Antes de continuar, convém apresentarmos brevemente as chamadas *leis do movimento,* de Isaac Newton, originalmente publicadas em seu livro *Philosophiæ Naturalis Principia Mathematica*, em 1687:

1. Qualquer corpo permanece em seu estado atual, seja em repouso ou em movimento retilíneo uniforme, até que uma força lhe seja aplicada.

2. A aceleração de um corpo é diretamente proporcional à força a ele aplicada e inversamente proporcional à sua massa (F=ma).

3. Para toda a ação existe sempre uma reação oposta de igual intensidade.

[9] *SpaceWar!* simulava, em 2D, duas naves que combatiam em órbita de um buraco negro.

Essas três leis fundamentam uma série de equações de movimento que também são utilizadas em games 3D. Talvez a mais comum seja a equação de movimento uniformemente acelerado:

$$y = y_0 + v_0 t + \alpha t^2 / 2$$

Onde *y* é a posição final, y_0 é a posição inicial, v_0 é a velocidade inicial, α é a aceleração e *t* o tempo decorrido entre o instante inicial e o final. Na forma descrita acima, a equação aplica-se a apenas um dos eixos do espaço tridimensional, o eixo *y* (as componentes de aceleração e velocidade são projetadas nesse mesmo eixo). Ao aplicarmos essa equação a personagens e objetos, nos três eixos do ambiente virtual, podemos obter uma série de trajetórias características em games 3D.

Prever o movimento de objetos reais é um objetivo das equações de movimento. Nos games 3D, porém, elas são usadas para definir novas posições, não para prevê-las. Desde que saibamos a massa, a posição e a velocidade inicial de um objeto ou personagem, bem como as forças a que ele está submetido, é possível definir onde ele estará nos instantes seguintes. Isso vale tanto para a trajetória de salto de um personagem como para os cálculos mais complexos do movimento de um avião 3D num simulador de voo – o que muda é a combinação de forças aplicadas, que podem gerar torque e rotação, exigindo outras equações específicas. O uso mais frequente é a atualização da posição de altura de um personagem. Por exemplo: enquanto o personagem está no solo, sua velocidade vertical é sempre zero, pois a força exercida pela gravidade é equilibrada pela reação contrária da superfície em que ele se apoia (terceira lei de Newton, acima). Entretanto, quando o personagem salta ele adquire uma velocidade vertical positiva e perde o contato com o solo. Através da equação de movimento uniformemente acelerado podemos calcular diversos pontos com alturas diferentes onde o personagem estará nos momentos seguintes. Normalmente, com o impulso do salto o personagem adquire também uma componente de velocidade horizontal e, submetido à aceleração da gravidade *g*, descreve uma curva como esta:

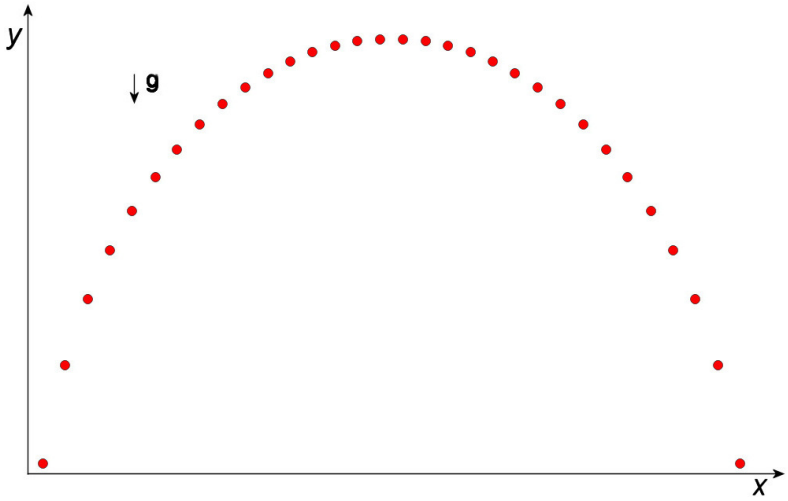


Figura 1.2. Trajetória parabólica, característica de movimento uniformemente acelerado pela gravidade.

Na figura acima, para cada trecho entre um ponto vermelho e outro, aplica-se a equação de movimento uniformemente acelerado nos eixos x e y (em três dimensões teríamos também o eixo z). As novas velocidades iniciais, para cada trecho, podem ser obtidas facilmente com a seguinte equação:

$$v = v_0 + \alpha t$$

Ou seja, a velocidade no final de um trecho pode ser calculada como a velocidade inicial somada à aceleração multiplicada pelo tempo decorrido. Essa velocidade alimenta novamente a equação de movimento uniformemente acelerado para o cálculo do trecho seguinte, e assim por diante. Na figura acima, podemos notar que a velocidade no eixo x não varia durante o salto (o espaço horizontal entre os pontos vermelhos é sempre o mesmo), pois ela não é afetada pela aceleração da gravidade. No eixo y, entretanto, a aceleração da gravidade atua diminuindo progressivamente a velocidade vertical, até invertê-la, no ápice da parábola. A partir de então, o personagem descreve uma curva descendente que o conduz novamente ao solo.

Um detalhe interessante no salto de personagens é que uma das leis de Newton pode ser facilmente burlada com o intuito de simplificar os cálculos. Para que o personagem altere seu estado atual e salte, a primeira lei de Newton exige a aplicação de uma força para cima, que o aceleraria até determinada velocidade quando ele deixasse o solo. Isso pode ser feito – e com certeza é feito em vários

games –, mas é muitas vezes mais simples aplicar diretamente ao personagem uma velocidade inicial, que tem resultado praticamente idêntico. Apenas a curta fase de aceleração é descartada, normalmente substituída de forma mais eficiente por uma animação do personagem iniciando o salto.

O exemplo do salto apenas ilustra o modo com que essas equações são empregadas em games 3D. Existem outras equações, evidentemente, mas o princípio é esse. De fato, a expressão “movimento uniformemente acelerado” refere-se a aceleração constante, imutável. Mas o modo como a equação é normalmente empregada em games, em períodos de tempo muito curtos (entre uma atualização da imagem e outra), permite que a aceleração seja modificada ao longo dos segmentos da trajetória, funcionando até mesmo para objetos que recebam forças variáveis, como carros de corrida ou aviões, por exemplo.

1.4. Variáveis de movimentação

A listagem abaixo mostra variáveis utilizadas para a movimentação de personagens no game *Robô Sucata* (44 Bico Largo, 2010), que teve participação minha e de Odair Gaspar em sua programação. A sequência de variáveis define forças de movimentação, energias gastas e necessárias, forças de resistência e velocidades limite para diferentes modos de movimento do personagem. Durante a execução do game, esses valores são utilizados em cálculos que definem a posição e a velocidade do personagem a cada momento, através de equações como a que vimos anteriormente.

```
// Correr
F32 runForce;               ///< Força usada para acelerar o personagem
F32 runEnergyDrain;         ///< Energia drenada por tick
F32 minRunEnergy;           ///< Energia mínima para correr
F32 maxForwardSpeed;        ///< Máxima velocidade para a frente ao correr
F32 maxBackwardSpeed;       ///< Máxima velocidade para trás ao correr
F32 maxSideSpeed;           ///< Máxima velocidade lateral ao correr

F32 horizMaxSpeed;          ///< Máxima velocidade na horizontal
F32 horizResistSpeed;       ///< Velocidade na qual a resistência do ar atua
F32 horizResistFactor;      ///< Fator de resistência para horizResistSpeed

F32 upMaxSpeed;             ///< Máxima velocidade vertical
F32 upResistSpeed;          ///< Velocidade na qual a resistência do ar atua
F32 upResistFactor;         ///< Fator de resistência para upResistSpeed

// Saltar
F32 jumpForce;              ///< Força exercida por salto
```

```
F32 jumpEnergyDrain;       ///< Energia drenada por salto
F32 minJumpEnergy;         ///< Energia mínima para saltar
F32 minJumpSpeed;          ///< Velocidade mínima para saltar
F32 maxJumpSpeed;          ///< Vel. máx para saltos mais longos

// Nadar
F32 swimForce;                   ///< Força de aceleração na água
F32 maxUnderwaterForwardSpeed;   ///< Velocidade máxima para a frente
F32 maxUnderwaterBackwardSpeed;  ///< Velocidade máxima para trás
F32 maxUnderwaterSideSpeed;      ///< Velocidade máxima lateral

// Rastejar
F32 crouchForce;                 ///< Força de aceleração enquanto rastejando
F32 maxCrouchForwardSpeed;       ///< Velocidade máxima para a frente
F32 maxCrouchBackwardSpeed;      ///< Velocidade máxima para trás
F32 maxCrouchSideSpeed;          ///< Velocidade máxima lateral

// Andar abaixado
F32 proneForce;                  ///< Força de aceleração enquanto abaixado
F32 maxProneForwardSpeed;        ///< Velocidade máxima para a frente
F32 maxProneBackwardSpeed;       ///< Velocidade máxima para trás
F32 maxProneSideSpeed;           ///< Velocidade máxima lateral
```

Quake (Id Software, 1996), também utiliza um conjunto de variáveis semelhantes, embora mais simples, para cálculo do movimento de personagens e objetos. A Id Software publicou o código fonte da versão original de *Quake*, em 1999, e é possível verificar em detalhes as estratégias adotadas para cálculo de movimento:

```
void SV_SetMoveVars(void)
{
       movevars.gravity            = sv_gravity.value;
       movevars.stopspeed          = sv_stopspeed.value;
       movevars.maxspeed           = sv_maxspeed.value;
       movevars.spectatormaxspeed  = sv_spectatormaxspeed.value;
       movevars.accelerate         = sv_accelerate.value;
       movevars.airaccelerate      = sv_airaccelerate.value;
       movevars.wateraccelerate    = sv_wateraccelerate.value;
       movevars.friction           = sv_friction.value;
       movevars.waterfriction      = sv_waterfriction.value;
       movevars.entgravity         = 1.0;
}
```

Durante o processamento do game, as funções utilizadas em *Quake* resultam nas familiares trajetórias da mecânica clássica.

```
(...)
SV_CheckVelocity (ent);
// add gravity
if (ent->v.movetype != MOVETYPE_FLY
   && ent->v.movetype != MOVETYPE_FLYMISSILE)
       SV_AddGravity (ent, 1.0);

// move angles
VectorMA (ent->v.angles, host_frametime, ent->v.avelocity, ent->v.angles);
(...)
```

Acessar o código fonte de games comerciais mais recentes não é tão simples, mas o emprego de equações de movimento pode ser facilmente comprovado através das trajetórias parabólicas descritas por objetos e personagens, principalmente como resultado de saltos e explosões (figura a seguir).

Figura 1.3. *Crysis 2* (Electronic Arts, 2011): veículos, personagens e destroços descrevem trajetórias parabólicas quando arremessados por explosões (Electronic Arts, divulgação).

Se avaliarmos a importância que as equações de movimento têm na programação de games 3D atuais, provavelmente concluiremos que, por alguma razão, o comportamento de objetos virtuais em ambiente 3D *precisa* ser comparável ao de objetos no mundo real. Entretanto, mesmo games 2D que não têm qualquer pretensão "realista" – como *Angry Birds Rio* (Rovio Mobile, 2011), na imagem seguinte – fazem uso frequente dessas equações, o que pode indicar que a mecânica clássica seja utilizada simplesmente por não se pensar em outras opções.

Figura 1.4. *Angry Birds Rio*: trajetórias explícitas. Para cada atualização de imagem, uma pequena marca de posição ajuda a traçar visualmente a trajetória realizada.

Mas existem exceções importantes. Elas ocorrem tanto em games casuais como naqueles considerados realistas. Pode parecer paradoxal que, ao subverter a mecânica clássica, designers de games sejam capazes de criar experiências mais interessantes para o jogador. Veremos alguns exemplos a seguir.

1.5. Super Mario Brothers

Criado por Shigeru Miyamoto e publicado em 1986 pela Nintendo, *Super Mario Brothers* foi um dos primeiros videogames a aplicar um nível relativamente complexo de simulação física a um personagem, não apenas a objetos. O personagem principal já havia aparecido em dois games de Miyamoto: *Donkey Kong*, de 1981, e *Mario Brothers*, de 1983. Nesses primeiros games, o modelo de física limitava-se a mapear, muitas vezes de forma direta, os botões do controle aos movimentos do personagem (por exemplo, ao pressionar-se o botão “esquerda”, o personagem movia-se a uma distância pré-definida naquela direção). Na versão de 1986, entretanto, um modelo mais elaborado de simulação física foi utilizado: Mario era submetido a aceleração quando comandado a se movimentar, desacelerava em virtude de fricção simulada quando o botão era solto, saltava em trajetórias características sob efeito de gravidade simulada e tinha sua velocidade limitada,

como que pelo atrito com o solo ou com o ar, tanto em movimentos horizontais como em saltos e quedas. Sempre achei que Super Mario Brothers diferia de outros games da época em aspectos importantes. O game era muito agradável de se jogar. Havia uma sensação fluida de controle, que ajudava a criar um forte vínculo com o personagem e seu ambiente, algo bem diferente dos comandos robotizados de outros games. Uma característica especial eram os saltos de Mario, o modo como se mantinha algum controle mesmo com o personagem no ar, a trajetória que podia ser modificada de acordo como o tempo em que se mantinha o botão de salto pressionado, detalhes que não se encontravam em tantos outros games similares que surgiram nos anos seguintes.

Figura 1.5. *Super Mario Brothers*, 1986.

Em seu livro *Game Feel – A Game Designer's Guide to Virtual Sensation*, Steve Swink analisa, define, apresenta exemplos e desenvolve métricas para um dos aspectos mais importantes da interatividade nos games: a mecânica que gera sensações reais a partir do virtual, característica que ele denomina *game feel*[10]. Particularmente interessante é a técnica por ele desenvolvida para medir tempos de respostas a comandos de games: A tela do game é filmada com uma câmera digital relativamente comum, em 60 quadros por segundo, com o controlador sendo

[10] Significando literalmente "sensação de jogo", *game feel* refere-se às sensações experimentadas pelo jogador durante sua interação com um game.

acionado na mesma imagem capturada pela câmera. Contando os quadros do vídeo entre um evento e outro é possível medir o tempo entre o acionamento de um comando e a resposta apresentada na tela com resolução de aproximadamente 17 milésimos de segundo.

O décimo terceiro capítulo do livro é dedicado a Super Mario Brothers. Swink analisa o game em detalhes, aplicando métricas desenvolvidas nos capítulos anteriores, e chega a resultados reveladores. Sobre a simulação de gravidade e os saltos característicos de Mario, por exemplo:

> Mas essa força da gravidade é variável. No momento em que o botão de salto é pressionado, certa velocidade para cima é instantaneamente transmitida a Mario, neutralizando a atração constante da gravidade e lançando-o num arco flutuante, gracioso. A velocidade ascensional é gradualmente reduzida enquanto a gravidade atua novamente. No ápice do salto, quando a velocidade atinge zero, a gravidade é elevada artificialmente por um fator de três, puxando Mario de volta para o chão com uma força muito maior do que aquela que ele superou anteriormente ao decolar (SWINK, 2009, p. 211, tradução nossa)[11].

Swink (2009) detectou um significativo aumento da aceleração de gravidade no ápice da curva, resultando na trajetória ilustrada abaixo.

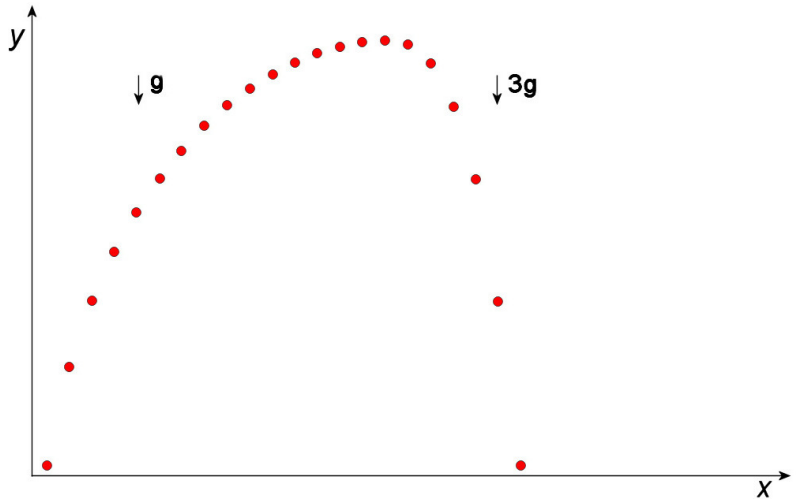


Figura 1.6. A mecânica modificada de *Super Mario Brothers*: alterações no vetor de aceleração da gravidade.

[11] But this gravity force is variable. At the moment the jump button gets pressed, Mario is instantaneously imparted with a certain upward velocity, which counteracts the constant downward pull of gravity, launching him into a soaring, graceful arc. This upward velocity is gradually reduced as gravity takes hold again. At the apex of the jump, when velocity reaches 0, gravity is raised artificially by a factor of three, pulling Mario back to the ground with a much greater force than the one he overcame to get airborne in the first place (SWINK, 2009, p. 211).

Após atingir o ponto mais alto da trajetória, o personagem desce abruptamente. O resultado dessa manipulação da gravidade é um gameplay que prioriza o controle do jogador: como Mario pode ser levemente controlado durante o salto, na subida tem-se mais tempo para dirigi-lo ao seu objetivo, seja ele uma nova plataforma ou um bloco a ser quebrado. Com a descida abrupta, ganha-se novamente o controle mais efetivo, no solo, de forma muito mais rápida do que se respeitássemos a trajetória "correta" do salto. A descida acelerada facilita, também, o ataque a inimigos como os *Goombas*. Esse é apenas um exemplo de como Shigeru Miyamoto foi capaz de entender e explorar a mecânica de seus games, fator que em muito contribuiu para o estabelecimento de sua conhecida reputação como game designer.

> Miyamoto compreendeu *game feel* não em termos de simulação, mas de simplificação. Primeiro, ele considerava a sensação de um game artisticamente, como experiência estética combinada. Numa época em que o campo era dominado por engenheiros que, na tradição de Steve Russell, baseavam-se em complexas metáforas literais, como a atração gravitacional dos buracos negros ou a alunissagem de uma nave espacial, Miyamoto trouxe uma perspectiva revigorante e ingênua. Ele queria simplesmente criar games divertidos e coloridos, sobre personagens extravagantes, que as pessoas se sentissem bem ao jogar. Segundo, ele projetou jogos de forma holística, levando em consideração tanto o software como dispositivos de entrada. (até hoje, Miyamoto projeta controladores assim como games, uma raridade entre os designers, especialmente agora, com a morte do *arcade*). Finalmente, Miyamoto entendeu o poder da metáfora e como ela afeta a disposição dos jogadores para aprender e dominar um sistema complexo, bem como seu apego emocional a ele (SWINK, 2009, p. 202, tradução nossa)[12].

1.6. Micro Scooter Challenge

Desenvolvidos por mim e por Odair Gaspar, *Micro Scooter Challenge* (Perceptum, 2001), *Super Mini Racing* (Perceptum, 2001) e *Super Mini Racing Ice* (Perceptum, 2006) são games de corrida projetados para um público casual, um público que não tem, necessariamente, grande familiaridade com games. Entretanto, projetar a interação de um game casual traz desafios interessantes.

---

[12] Miyamoto understood game feel not in terms of simulation but of simplification. First, he regarded the feel of a game artistically, as a composite aesthetic experience. At a time when the field was dominated by engineers who, in the tradition of Steve Russell, drew on complex, literalistic metaphors like the gravitational pull of black holes or landing a spacecraft on the moon, Miyamoto brought a refreshing, naïve perspective. He simply wanted to make fun, colorful games about whimsical characters that felt good to play. Second, he designed games holistically, taking into consideration both software and input device. (To this day, Miyamoto designs controllers as well as games, a rarity among designers, especially now, with the death of the arcade.) Finally, Miyamoto understood the power of metaphor and how it affected players' willingness to learn and master a complex system and their emotional attachment to it (SWINK, 2009, p. 202).

Quando se desenvolve um game para um público mais restrito (como *Incidente em Varginha*, por exemplo), tem-se uma ideia bastante razoável do nível de habilidade do público-alvo. Isso ajuda muito a calibrar o nível de dificuldade do game, pois não é tão complicado avaliar certa habilidade média para o público-alvo e desenvolver o game com base nessa habilidade esperada.

Micro Scooter Challenge, entretanto, pretendia atingir um público mais amplo, com níveis de habilidades bastante diversos entre si. Se adotada a mesma estratégia utilizada num game não casual – escolher uma habilidade média necessária –, isso provavelmente deixaria a maior parte do público frustrada. Para alguns, o game seria difícil demais. Para outros, muito fácil.

O game, patrocinado pela fabricante suíça *Micro Mobility Systems AG*, é uma corrida de patinetes ambientada em várias cidades do mundo. Os cenários utilizam texturas fotográficas, mas os patinetes são pilotados por personagens com visual estilo "mangá". Essa estranha combinação permitiu alguns resultados interessantes: por exemplo, a velocidade dos patinetes na escala do cenário virtual excede 140 km/h. Em testes iniciais, com personagens de visual mais humano, tais velocidades não pareciam coerentes. Pior, com velocidades mais verossímeis (mais "realistas") o game todo não funcionava, pois os patinetes arrastavam-se com lentidão exasperante, algo que se esperaria de uma corrida real de patinetes. Ficou claro, desde o início do projeto, que copiar a realidade não traria os resultados esperados de um game de corrida como esse.

Figura 1.7. *Micro Scooter Challenge*.

Para que o game se adaptasse aos diversos níveis de destreza de um público mais amplo, minha primeira tentativa foi utilizar a solução padrão para esse problema: estabelecer níveis de dificuldade que poderiam ser alterados pelo jogador. Testes informais com vários jogadores logo provaram que essa estratégia não seria suficiente. Mesmo o nível mais fácil ainda se mostrava impossível para alguns, enquanto que o nível mais difícil parecia enfadonho para outros. Ampliar o número de níveis de dificuldade não me agradava, pois geraria uma escala absurda entre o extremamente fácil e difícil. Por exemplo, *Wolfenstein 3D* (Id, 1992), tinha dificuldades que iam de “don’t hurt me daddy” (“não me machuque, papai”, a mais fácil) até “I’m death incarnated” (“sou a morte encarnada”, a mais difícil). O rosto do herói do game era apresentado ao lado de cada nível, e lembro bem que “don’t hurt me daddy” era acompanhado pelo herói com touca e chupeta de bebê. Num game que exija destreza e habilidade, esse tipo de desafio aberto ao jogador pode até funcionar. Num game casual, não.

Era preciso encontrar outra solução, menos evidente. Pensando na época sobre os problemas que o visual menos realista dos personagens resolvia, ocorreu-me tratar a mecânica de jogo também de forma menos realista, para talvez criar uma

dificuldade de jogo automaticamente variável. A solução foi muito mais simples do que eu imaginava: programei o game para *alterar a segunda lei de Newton*, individualmente, para cada personagem controlado pelo computador. Assim, se o personagem estivesse muito atrás do jogador, a variável de força a ele aplicada tinha um efeito proporcionalmente maior na aceleração obtida. Se ele estivesse muito à frente, o efeito era contrário. Tecnicamente, alterei a famosa fórmula, que estabelece ser a força igual à massa multiplicada pela aceleração, para considerar também a distância entre o personagem e o jogador. Com isso, mesmo que o jogador controlasse mal seu personagem, os outros concorrentes nunca conseguiriam distanciar-se demais. Ao contrário, se o jogador fosse bastante hábil, os concorrentes também o seriam.

Nem tudo funcionou tão bem a princípio. Lembro-me de que, nos primeiros testes, os personagens concorrentes obtinham boas velocidades para alcançar um jogador hábil, mas não conseguiam fazer as curvas adequadamente, chocando-se contra obstáculos ao lado da pista e ficando ainda mais para trás. Foi necessário alterar também a variável de atrito com o solo nessas situações, permitindo que curvas mais fechadas pudessem ser feitas em alta velocidade. Outra falha evidente acontecia quando o jogador parava deliberadamente: os demais personagens ultrapassavam-no e também paravam a alguma distância à frente, esforçando-se em vão para impulsionar seus patinetes, como se houvesse cola na pista. Isso deixava o truque evidente e eliminava qualquer prazer que o game pudesse proporcionar, pois o jogador percebia que não precisava esforçar-se para vencer – bastava apenas correr bem nos metros finais. Para disfarçar esse efeito, programei limites de velocidade mínima para os personagens controlados pelo programa. Assim, quando o jogador parasse, eles continuariam correndo, mesmo que com menor velocidade.

Anos depois, tive a oportunidade de ler um livro de Mihaly Csikszentmihalyi intitulado *Finding Flow*, de 1997. *Flow* é um estado de imersão experimentado por pessoas que são hábeis em determinadas atividades, e que se sentem em hamonia quando as executam.

> O estado de *flow* tende a ocorrer quando as habilidades de uma pessoa são totalmente envolvidas em superar um desafio que está no limite de suas possibilidades. Experiências ideais geralmente envolvem um fino equilíbrio entre a habilidade de agir e as oportunidades disponíveis para a ação (...)

> Se os desafios são muito altos, fica-se frustrado, então preocupado, e, eventualmente, ansioso. Se os desafios são muito baixos em relação à habilidade, fica-se relaxado, depois entediado. Se tanto desafios como habilidades são percebidos como baixos, começa a sentir-se apático. Mas quando altos desafios são enfrentados com altas habilidades, então é provável que ocorra o profundo envolvimento que diferencia o estado de *flow* da vida comum (CSIKSZENTMIHALYI, 1997, p. 30, tradução nossa)[13].

O gráfico que representa essas relações entre dificuldade e habilidade é ilustrado na figura abaixo. É importante notar que Csikszentmihalyi menciona habilidades e desafios *percebidos*. Esse adjetivo faz bastante sentido num videogame, pois é tarefa comum para os desenvolvedores fazer com que desafios *pareçam* mais difíceis do que na realidade o são. Além disso, as habilidades de um jogador se aprimoram durante o desenrolar do game, como simples consequência do treinamento adquirido ao jogá-lo, e ele naturalmente achará desafios já vencidos muito mais fáceis se enfrentá-los novamente.

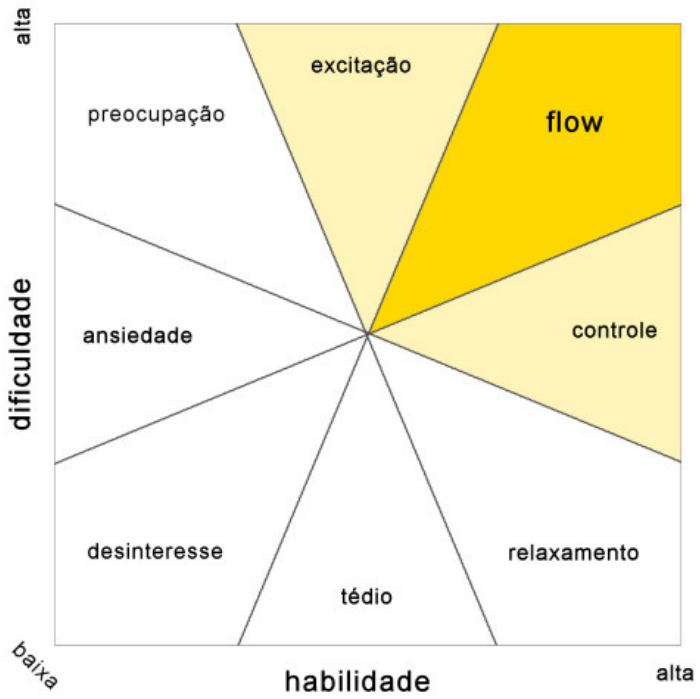


Figura 1.8. Adaptado de gráfico publicado em (CSIKSZENTMIHALYI, 1997, p. 31).

[13] Flow tends to occur when a person's skills are fully involved in overcoming a challenge that is just about manageable. Optimal experiences usually involve a fine balance between one's ability to act, and the available opportunities for action (...) If challenges are too high one gets frustrated, then worried, and eventually anxious. If challenges are too low relative to one's skills one gets relaxed, then bored. If both challenges and skills are perceived to be low, one gets to feel apathetic. But when high challenges are matched with high skills, then the deep involvement that sets flow apart from ordinary life is likely to occur (CSIKSZENTMIHALYI, 1997, p. 30)

O truque utilizado na simulação física de *Micro Scooter Challenge* faz com que a dificuldade do game esteja sempre próxima às habilidades do jogador. Com isso, a experiência tende a se encaixar na região do gráfico que representa o estado de *Flow* (e, para jogadores muito ou pouco hábeis, talvez nos de *Controle* e *Excitação*), de forma automática. Na prática, o equilíbrio entre desafio e habilidade só começa a ocorrer a partir de certa habilidade mínima, o que talvez evite a região de *Desinteresse* do gráfico, substituindo-a pela de *Ansiedade* para jogadores sem a habilidade mínima necessária. Como citado anteriormente, o truque funciona apenas enquanto não é percebido, mas, após as correções de velocidade mínima, a estratégia deixou de ficar evidente, pelo menos para a grande maioria dos jogadores.

Outro efeito positivo dessa alteração virtual da segunda lei de Newton foi o grande aumento do número de ultrapassagens – que são o ponto central de qualquer game de corrida. Como os personagens ficam progressivamente mais rápidos quando estão atrás e mais lentos quando estão à frente, é inevitável que o número de ultrapassagens aumente durante a corrida. Acredito que essas características tenham influenciado o sucesso relativo do game. Distribuído em 14 países por diferentes empresas, com versões em inglês e alemão, *Micro Scooter Challenge* ficou durante semanas entre os títulos mais vendidos na rede de lojas Saturn, na Alemanha, em 2001.

1.7. Super Mini Racing

O primeiro *Super Mini Racing* (Perceptum, 2001) foi desenvolvido em meados de 2001 com a mesma plataforma de *Micro Scooter Challenge*. A ideia de uma simulação de corrida com carros de brinquedo, em miniatura, serviu de base para esse game patrocinado por marcas bastante conhecidas, que teve uma tiragem de mais de 500 mil cópias distribuídas gratuitamente pelos jornais *O Estado de S. Paulo* e *Jornal da Tarde* nas edições de 12 de outubro de 2001.

Figura 1.9. *Super Mini Racing*: versões de 2001 (à esquerda) e de 2006 (direita).

Em 2006, *Super Mini Racing Ice* (Perceptum, 2006) foi publicado gratuitamente na web em parceria com o Canal Kids. Nessa versão, o visual foi atualizado para os padrões mais fotográficos da época, mas a manipulação da lei de Newton continuou favorecendo o equilíbrio entre as habilidades do jogador e a dificuldade do game.

O resultado dessas experiências indica que mecanismos de equilíbrio automático da dificuldade apresentada em relação às habilidades do jogador deveriam ser parte integrante do modelo matemático que rege um game 3D. Um primeiro problema seria generalizar tais mecanismos. Num game de corrida, os exemplos anteriores funcionam bem, mas o que fazer em games diferentes, que simulem combates ou outras aventuras? E como automatizar o nível de dificuldade sem que isso fique óbvio para o jogador?

Sobre a solução de *Micro Scooter Challenge*, uma possibilidade para generalizá-la é a seguinte: a equação modificada faz com que os personagens apresentem diversos comportamentos diferentes, com desempenhos que variam continuamente de, digamos, “inepto” a “muito hábil”. Em determinado instante do game, dessa ampla gama de comportamentos, o programa seleciona um, dependendo de uma medida de desempenho relativo entre o personagem e o jogador (no caso, a distância, positiva ou negativa, entre eles). Num game de simulação de combate, por exemplo, essa medida de desempenho poderia ser calculada com base na variável de saúde do jogador e nos danos que ele provocou a outros personagens inimigos. Mas como criar personagens com vasta gama de desempenhos, que pudessem ter seus comportamentos continuamente ajustados através dessa medida relativa? Voltaremos a essa importante questão no capítulo 3. Por hora, vejamos como alguns

games tidos como realistas podem adotar soluções que contrariam o real e a mecânica clássica.

1.8. IL-2 Sturmovik

Desenvolvido pela empresa russa *1C: Madox Games*, o simulador de combates aéreos *IL-2 Sturmovik* é um dos videogames de carreira mais longa, com atualizações publicadas em 2011, dez anos após seu lançamento, em 2001. Considero-o o simulador de voo mais consistente já desenvolvido em forma de videogame. Outros poderiam dizer "o mais realista", adjetivo que aparece dezenas de vezes nas avaliações do game feitas pela Gamespot.com e pela revista PC Gamer, para citar dois exemplos.

Mas o termo "realista" não é utilizado sem motivo. Para começar, o visual do game apresenta atenção impressionante a detalhes. Na recente versão *IL-2: Cliffs of Dover*, lançada em 2011, imagens geradas em tempo real são mais detalhadas do que se poderia esperar de fotografias de época (o game se passa em 1940).

Figura 1.10. *IL-2 Sturmovik*. Imagem da versão Cliffs of Dover (divulgação).

Como poderemos ver mais adiante, imagens da versão original de 2001 não ficam muito atrás. Mas o visual representa apenas uma pequena parte do extremo cuidado com detalhes presente durante o desenvolvimento do game. Modelos matemáticos

de simulação de comportamento dos aviões completam a experiência de *IL-2 Sturmovik*. Seu criador, Oleg Maddox, em entrevista ao site combatsim.com alguns meses antes do lançamento, explica que o modelo de voo utilizado no game inclui diversos submódulos, cada um com centenas de parâmetros:

- Modelo atmosférico com efeitos da altitude sobre a pressão, temperatura e densidade do ar.

- Equações de movimento não lineares, de seis graus de liberdade, utilizadas na indústria aeronáutica para análise de dinâmica de voo de aviões.

- Modelo aerodinâmico com submódulos para determinação de forças e movimentos produzidos por várias partes do avião. Esse modelo considera efeitos aerodinâmicos devido à compressibilidade do ar (número *Mach*), posição de todas as superfícies móveis, trem de pouso, cargas externas (bombas e armamentos externos), etc.

- Modelo de distribuição de massas, que computa, por exemplo, efeitos do consumo de combustível nos tanques e mudanças no centro de gravidade do avião.

- Modelo de propulsão, com curvas de torque e potência dos motores, eficiência dos radiadores de água e óleo dependendo da velocidade, efeitos do ângulo das pás da hélice, etc.

- Modelo de trem de pouso, que calcula forças exercidas durante pousos e decolagens.

Oleg Maddox compara a reação desses modelos matemáticos a dados coletados durante a Segunda Guerra Mundial a partir de testes em aviões reais:

> Tudo depende da aerodinâmica real, das leis da física de voo em **diferentes** altitudes e dos parâmetros do modelo do avião. Basta experimentá-lo, você vai fazê-lo reagir da mesma forma que é declarada nos **documentos reais de testes da NII VVS** (NII VVS era um instituto soviético de pesquisas de aviação militar, onde estavam em testes aviões russos da Segunda Guerra Mundial de todos os tipos, aviões alemães capturados – 98% de todos os

> tipos! – além de todos os modelos da Inglaterra e dos EUA cedidos em *lend lease* na Segunda Guerra Mundial) e segundo o relato de pilotos reais... Assim, **todos os aviões que aparecem em nosso simulador** serão modelados de acordo com estes documentos e consultas (Oleg Maddox em entrevista concedida a Bob "Grouxo" Marks, do site combatsim.com, em 2000, antes do lançamento do produto [http://www.combatsim.com/memb123/htm/2000/10/il2interview/page2.htm] acessada em agosto de /2011 – grifos do original, tradução nossa).[14]

Não há dúvida de que *IL-2 Sturmovik* seja um simulador altamente preciso. Na primeira vez que o joguei, em 2002, antes sequer de ler o manual, fundi o motor de meu avião logo após a decolagem. Tenho certeza de que fato similar ocorreria com quem tentasse decolar um avião real sem passar por um treinamento mínimo. Parte do apelo de um simulador de voo está em aprender a dominar esses aparelhos, em conhecer seus limites, e em fazer tudo isso sem arriscar a vida de ninguém. Para simplificar esse processo de aprendizagem e, também, para permitir que jogadores menos interessados na realidade da simulação pudessem apreciar o game, *IL-2* apresenta a possibilidade de ajuste de diversos parâmetros numa tela de configuração de dificuldade (figura abaixo). Na configuração "realistic", com todas as chaves ligadas, o game se aproxima muito de um simulador de voo comercial. Em "easy", torna-se um game para jogadores casuais.

[14] All depends of real aerodynamics and physics laws of the flight at DIFFERENT altitudes and parameters of the plane model itself. Just try it, you'll get it to react just as it is stated in REAL TRIAL DOCS OF NII VVS (NII VVS is a Soviet military aviation research institute, where were in trials Russian WWII planes of all types, all captured German WWII planes---98% of all types!---all WWII lend lease UK and USA planes) and like stated it real pilots... So, ALL PLANES, WHICH PRESENT IN OUR SIM, will be modeled according to these documents and consulting (Entrevista concedida a Bob "Grouxo" Marks, do site combatsim.com, em 2000, antes do lançamento do produto http://www.combatsim.com/memb123/htm/2000/10/il2interview/page2.htm acessada em agosto de /2011).

Figura 1.11. Configuração de dificuldade – *IL-2 Sturmovik*.

Mas quais seriam os limites dessa ilusão de realidade? E que soluções os desenvolvedores criaram para contorná-los? Veremos a seguir três relatos de pilotos da Segunda Guerra Mundial sobre situações comuns em combates aéreos, acompanhados por um paralelo dessas mesmas situações em *IL-2 Sturmovik*. Começaremos por duas passagens escritas pelo ás alemão Adolf Galland.

> A primeira regra de todo combate aéreo é ver o adversário primeiro. Como o caçador que persegue sua presa e movimenta-se despercebido até a posição mais favorável para o abate, o piloto de caça, na abertura de um combate aéreo, deve detectar o adversário o mais cedo possível, a fim de alcançar uma posição superior para o ataque. (GALLAND, 1986, p. 68, tradução nossa)[15]

Enxergar um avião inimigo a quilômetros de distância através de uma cabine de plástico transparente não é tarefa simples. Na tela de um computador, a primeira aparição de um avião inimigo invariavelmente ocupa apenas um pixel. Se a simulação levar o realismo ao pé da letra, a influência sobre a cor do pixel será, a princípio, mínima, pois o modelo do avião inimigo ocuparia uma área muito menor que a do próprio pixel, dificultando ainda mais a sua localização.

[15] The first rule of all air combat is to see the opponent first. Like the hunter who stalks his prey and manouevres himself unnoticed into the most favorable position for the kill, the fighter in the opening of a dog-fight must detect the opponent as early as possible in order to attain a superior position for the attack. (GALLAND, 1986, p. 68)

Mesmo em distâncias mais curtas, existe o problema de identificar o avião já visível como amigo ou inimigo. Durante a Segunda Guerra Mundial, antes de o uso de radares e de identificação digital ser amplamente adotado, a média distância o processo dependia da observação da silhueta e das cores do aparelho (insígnias de nacionalidade só seriam visíveis em distâncias mais curtas). Adolf Galland tinha uma luneta adaptada ao para-brisa de seu caça Messershmitt Bf-109E durante a batalha da Inglaterra, com a finalidade de identificar caças inimigos a distância (foto abaixo). O mesmo equipamento era padrão em aviões de combate da marinha americana à mesma época.

Figura 1.12. Adolf Galland na cabine de seu Messerschmitt Bf-109E, em 1940. Note a luneta adaptada, que atravessa o para-brisa do avião (imagem extraída do livro *Luftwaffe Colours – Jagdwaffe – No. 06 – Battle of Britain Phase Two, August–September 1940*, p. 116).

Galland cita ainda um incidente interessante ocorrido logo após a ocupação alemã da França, em 1940:

> Experimentei uma nova pintura de camuflagem cinza-esverdeada em meu avião, mas, infelizmente, às vezes era confundido com um dos Hurricanes[16] agora raramente vistos. Certa ocasião, quando o nosso ás Balthasar me selecionou como alvo para ilustrar seu virtuosismo, isso quase provou ser minha ruína. Por sorte, eu estava sintonizado na mesma frequência de rádio, ouvi-o mencionar um Hurricane ao seu companheiro, mas olhei ao redor e não podia vê-lo em lugar algum. Balthasar prosseguiu, de maneira muito interessante e lúcida, para explicar o início de seu ataque, e percebi

[16] Avião de caça de fabricação inglesa, utilizado na defesa da França em 1940.

> de repente que ele não se referia a ninguém mais além de mim. Eu o vi mergulhando sobre meu avião de uma posição ideal. Ele deve ter ficado atônito quando, pouco antes de pressionar o gatilho de suas armas, ouviu minha voz familiar em seus fones, dizendo em alto e bom som: "Já chega desse maldito absurdo!" (GALLAND, 1986, p. 51, tradução nossa)[17]

Wilhelm Balthasar foi um dos principais pilotos alemães, com experiência em combate aéreo iniciada na Guerra Civil Espanhola, onde derrubou 7 aparelhos Republicanos. Na época do incidente descrito por Galland, Balthasar tinha a seu crédito 30 aviões inimigos abatidos. Ainda assim, ele confundiu um colega que pilotava o mesmo tipo de aparelho (um Messerschmitt Bf-109E) com um avião inimigo, unicamente por causa da cor diferente de camuflagem. O que esperar então de alguém sem qualquer experiência real, jogando num simulador de voo?

IL-2 Sturmovik apresenta soluções simples e elegantes (mesmo que absolutamente não realistas) para o problema da identificação de aviões inimigos. Um pixel que represente um aparelho a grande distância não é mesclado com a cor de fundo, ficando assim mais evidente. Além disso, dependendo das configurações, o programa apresenta um pequeno texto com a distância em quilômetros, o tipo do avião e sua matrícula de identificação, logo acima do modelo 3D, conforme figura abaixo. A cor desse texto já indica se o avião é amigo ou inimigo, sem a necessidade de se ler o tipo do aparelho.

[17] I tried out a new grey-green camouflage paint on my plane, but unfortunately she was sometimes mistaken for one of the now rarely seen Hurricanes. On one occasion, when our ace pilot Balthasar picked on me as an object to illustrate his virtuosity, this nearly proved my undoing. As luck would have it, I was tuned in the same wavelength, and hearing him mention a Hurricane to his companion, looked around, but could not see one anywhere. Balthasar went on, in a most interesting and lucid manner, to explain the beginning of his attack, and I suddently realised that he meant no one else but me. I saw him diving on to me from an ideal position. He must have been astonished when, just before pressing his gun button, he heard my familiar voice in his earphones saying rather loudly: “That’s enough of this bloody nonsense!” (GALLAND, 1986, p. 51)

Figura 1.13. *IL-2*: aviões, matrículas e distâncias identificados por textos em azul, no centro da imagem.

Após o início do combate, outro problema acontecia com frequência: acompanhar visualmente os aviões que participavam da batalha, fossem eles amigos ou inimigos. Pierre Clostermann, piloto nascido no Brasil e naturalizado francês, descreve:

> Começo a me debater como um diabo, eu viro, volto, rodopio. Vejo tudo negro. Minha máscara de oxigênio, puxada pela força G, parece rasgar a pele do nariz. Desloco meu pescoço a vigiar todos os aviões que passam ao meu alcance em uma confusão terrível ... (CLOSTERMANN, 2001, p. 110, tradução nossa)[18]

Virar a cabeça para vigiar todos os lados não funciona, é óbvio, num videogame. A tela está em posição fixa e podemos apenas mudar o ângulo de visão da imagem que ela está apresentando. A solução mais comum é utilizar um controle direcional para alterar a visão do jogo, mas aí enfrentamos outro problema: a perda de referência de voo. Num avião real (ou mesmo num automóvel real), podemos olhar em volta enquanto nosso corpo permanece na mesma posição, sem que percamos a referência principal de direção do veículo. Ao girar a câmera de um simulador,

[18] Je commence à me débattre comme un diable, je tourne, volte, virevolte . Je vois noir. Mon masque à oxygène entraîné par les G doit m'arracher la peau du nez. Je me démanche le cou à surveiller tous les avions qui défilent à ma portée dans un méli-mélo effroyable... (CLOSTERMANN, 2001, p. 110)

entretanto, torna-se muito difícil controlar o aparelho, pois essa referência é perdida. Novamente, a solução encontrada pelos criadores de *IL-2 Sturmovik* é muito eficiente. Na visão externa, um plano de referência, representando o solo, e um ícone 3D, mostrando posição e ângulos de voo do avião, são apresentados na parte inferior central da interface (figura abaixo). Além disso, instrumentos simplificados permitem que se continue controlando o avião sem grandes inconvenientes.

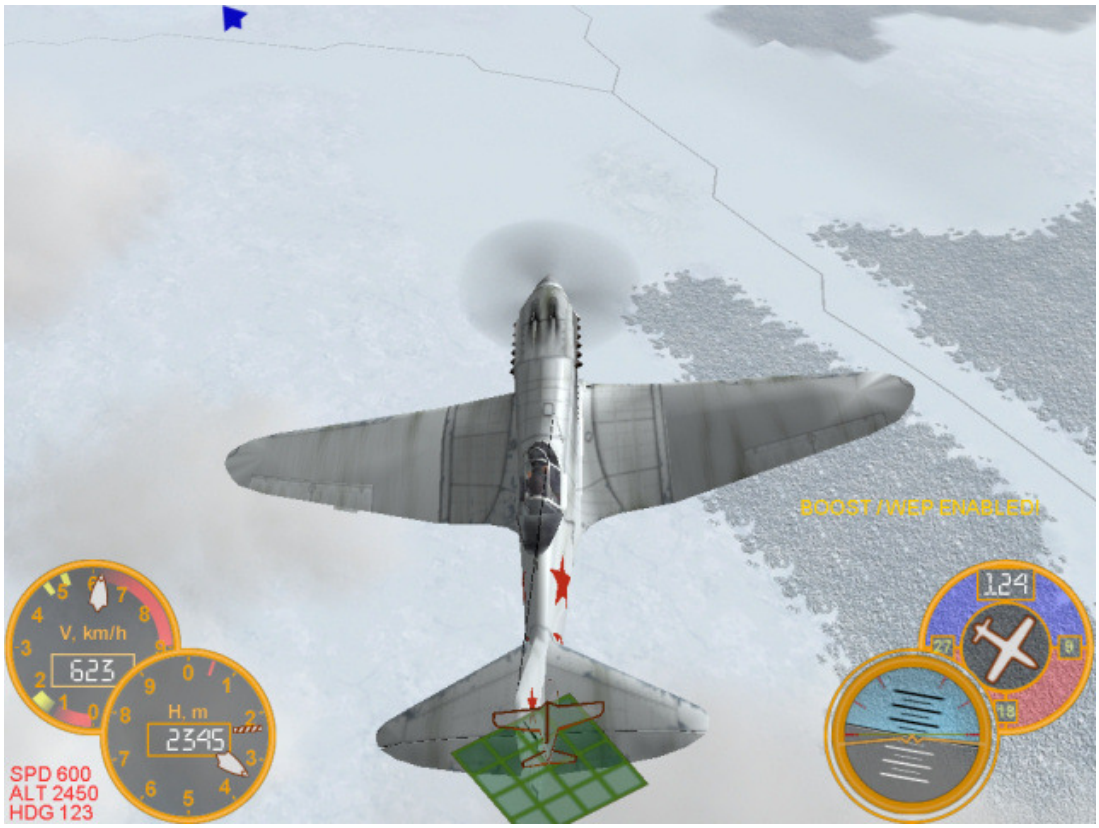


Figura 1.14. *IL-2*: na parte central e inferior da tela, um plano verde e um ícone 3D exibem os ângulos relativos entre o avião e o solo. Nos dois cantos inferiores da tela, instrumentos auxiliam a pilotagem.

Com isso, outra solução elegante torna-se possível para acompanhar aviões próximos. Um modo de câmera permite a visão de outros aviões alinhada com a do próprio aparelho controlado, em ângulo similar ao que se teria virando a cabeça para enxergá-lo (figura abaixo). Ainda outra solução que considero interessante são as setas azuis e vermelhas também visíveis na figura abaixo. Elas representam as direções onde se encontram aviões amigos e inimigos – quanto mais próximos, maior a seta.

Essas soluções de interface contornam problemas reais de modo brilhante, contribuindo em muito para a experiência do game. Entretanto, “realismo” não é um adjetivo que se lhes possa atribuir.

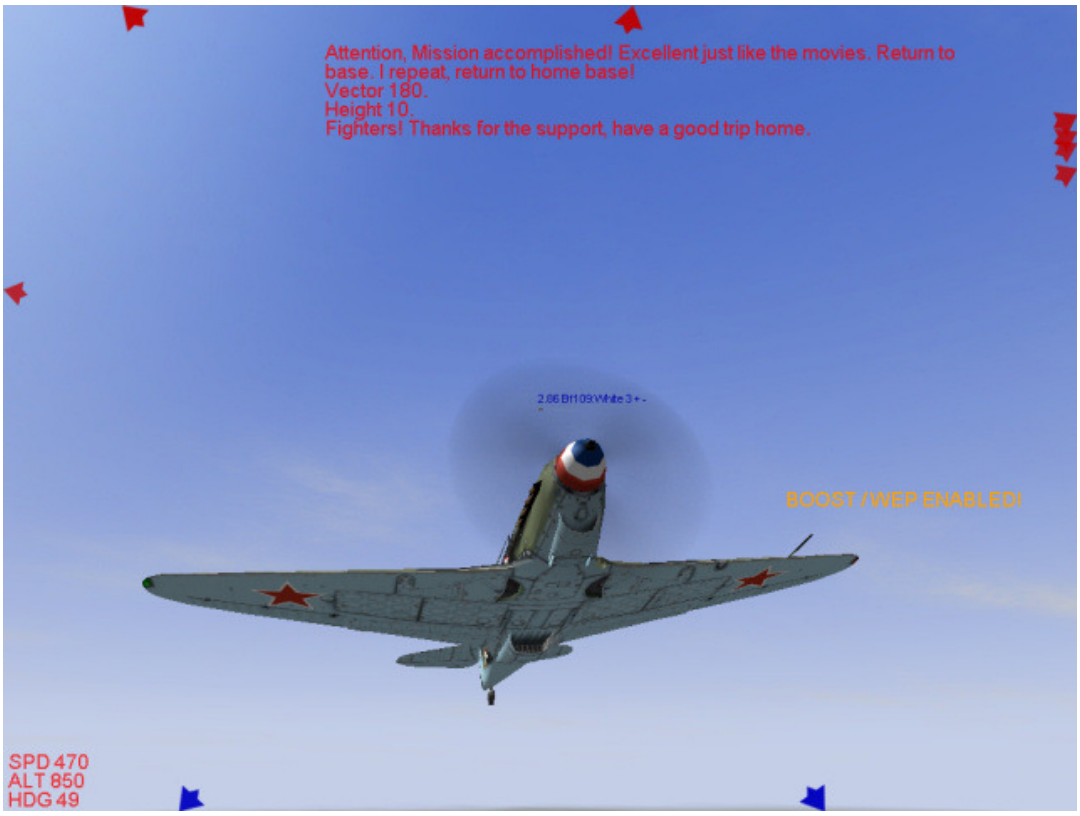


Figura 1.15. *IL-2*: visão automática na direção do inimigo mais próximo (o pequeno ponto no centro exato da tela, identificado pelo texto em azul). Setas azuis e vermelhas indicam a localização e a distância aproximada de outros aviões.

Algumas missões de guerra, notadamente as de escolta de bombardeiros, exigem voos longos, de várias horas. Mesmo missões mais curtas exigem um tempo razoável para se alcançar o alvo, combater e retornar à base. Pilotar um avião nessas condições é uma tarefa tediosa para a grande maioria dos jogadores, principalmente porque o voo em formação exige atenção constante. *IL-2* utiliza dois artifícios comuns nesses casos: piloto automático e aceleração do tempo. O piloto automático mantém o avião no curso e em formação, à distância correta dos outros integrantes do esquadrão, enquanto que a aceleração do tempo reduz longos e monótonos voos para até um oitavo do original, oferecendo um controle do tipo *fast-forward*.

Por outro lado, a aceleração do tempo também cria problemas. Como vimos, os modelos matemáticos do game são complexos e precisos. Com aviões voando oito

vezes mais rápido, entretanto, deve ser muito complicado atualizar todos esses cálculos em tempo real. Como resultado, colisões com os companheiros de esquadrão são frequentes, principalmente na versão original do game. Isso cria certa ansiedade ao se utilizar a aceleração do tempo: com ela, uma missão importante pode terminar em colisão injustificável, resultado de falta de poder de processamento para modelos matemáticos pretensamente realistas.

Figura 1.16. *IL-2*: no lado direito, em tons de azul claro, a indicação de tempo expandido.

Em *IL-2*, pode-se também *expandir* o tempo, em duas ou quatro vezes. Nessas condições, tudo se passa mais devagar e é literalmente possível desviar-se do caminho das balas, por exemplo.

1.9. Max Payne

Se em *IL-2* o tempo expandido é mera extrapolação do efeito principal de compressão para longos voos, em *Max Payne* ele é central para a experiência de jogo. Trata-se do *bullet-time*, tornado conhecido pelo filme *Matrix*, de 1999[19]. Tudo é

[19] O termo *bullet-time* é marca registrada da produtora de Matrix, a Warner Bros.

desacelerado e se passa como que em câmera lenta, sendo possível, por exemplo, acompanhar as balas em suas trajetórias. Mas, se num filme o *bullet-time* apenas aumenta nossa possibilidade de compreensão do que está acontecendo numa cena muito rápida, num videogame ele aumenta também a velocidade relativa da reação do jogador. Em *Max Payne*, desenvolvido pela empresa finlandesa *Remedy Entertainment* e publicado pela americana *Gathering of Developers* em julho de 2001, a capacidade de mira do jogador não é afetada pela dilatação do tempo, permitindo tiros extremamente precisos durante a câmera lenta.

Figura 1.17. *Max Payne* (2001): *bullet-time* em ação.

Figura 1.18. *Max Payne 3*, com lançamento previsto para 2012, é ambientado em São Paulo, Brasil. Imagens de divulgação do game exploram o característico efeito de *bullet-time.*

1.10. Incidente em Varginha 2

Ao animar um dos personagens de *Incidente em Varginha 2* (Perceptum, inédito), em 1999, utilizei pela primeira vez uma biblioteca de movimentos capturados. A animação de corrida, embora precisamente capturada de um ator real, parecia-me especialmente sem graça. Havia certo efeito de ausência de peso e o personagem parecia flutuar, movendo as pernas sem qualquer impacto aparente com o solo. Reforcei o movimento de subida e descida do corpo e isso ajudou, mas o contato dos pés com o solo ainda parecia extremamente suave. Editando o movimento das pernas de forma mais ou menos aleatória, notei repentinamente que a batida de um dos pés no solo estava muito mais convincente, causando nítida impressão de impacto. Não tardou para que o motivo fosse descoberto: esse era o efeito de um erro de animação. Durante a edição do movimento, dobrei acidentalmente um dos joelhos no sentido inverso, apenas um frame antes do toque do pé no solo (figura abaixo). Foi o suficiente para gerar um efeito curioso: a perna que continha o erro parecia realmente bater no chão, enquanto que a outra, com o movimento capturado original, ainda parecia flutuar.

Figura 1.19. Animação de personagem em *Incidente em Varginha 2*: o joelho dobrado momentaneamente ao contrário reforça a impressão de impacto do pé com o solo.

Como a distorção da articulação ocorria em um único frame da sequência animada, era impossível percebê-la como tal. Observando-se a animação em tempo real, não se via a dobra invertida do joelho, mas havia a forte impressão de que a barra da calça oscilava rapidamente com o impacto. Ora, para manter uma contagem de polígonos baixa, a barra da calça era modelada integralmente com o sapato e o resto da perna (como pode ser visto na figura acima): não havia como fazê-la oscilar separadamente. Ainda assim, a impressão de movimento independente da barra da calça era inegável. Creio que nosso cérebro reinterprete esse tipo de distorção – inverossímil, mas extremamente rápida – como algo conhecido: nesse caso, como a oscilação rápida da roupa durante o impacto do pé no solo.

Na verdade, meu erro resultou num encontro acidental com certa técnica que já vem sendo utilizada há décadas na animação. Em *The Animator's Survival Kit* (WILLIAMS, 2002), Richard Williams descreve o artifício da antecipação invisível. A antecipação tradicional é o movimento de preparação ao movimento principal. Por exemplo, antes de saltar para frente, o personagem deve inclinar-se para trás[20]. Já a antecipação invisível consiste, simplesmente, em fazer com que um movimento rápido seja precedido por outro, de preparação, porém rápido demais para ser percebido como antecipação tradicional. No exemplo da figura acima a antecipação invisível configura-se quando o joelho se dobra levemente ao contrário, em

[20] O movimento de antecipação tem função narrativa, pois indica ao observador o que está por acontecer.

preparação para o momento do impacto, no qual ele deverá dobrar-se no sentido correto e de forma mais violenta.

Seria de se supor que personagens “realistas” não tolerassem distorções desse tipo, que não respeitam limitações de inércia esperadas de um personagem real. Entretanto, tudo parece depender do momento e da velocidade com que elas ocorrem. Uma articulação dobrada ao contrário pode arruinar o efeito de uma animação se for percebida, mas pode contribuir enormemente com a impressão de realismo se for empregada na velocidade e no momento corretos. “Podemos tomar grandes liberdades com ações rápidas – mesmo com figuras realistas” (WILLIAMS, 2002)[21]. Tenho utilizado o artifício de antecipação invisível desde 1999, com bons resultados em movimentos de impacto, sempre trabalhando com distorções de, no máximo, 1/20 de segundo. Apesar de minha inclinação a compreender o efeito como reinterpretação que ocorre no cérebro, não há explicação teórica conhecida para o realismo resultante. Mas é certo que a antecipação invisível deixa os movimentos mais convincentes, tanto em personagens de desenhos animados como em personagens 3D supostamente realistas.

As sensações de massa e impacto, nesse exemplo, dependem unicamente da animação, não da simulação física. Swink (2009), comentando sobre o efeito conhecido como *squash and stretch,* muito utilizado por animadores para realçar artificialmente as propriedades elásticas de um objeto durante impactos, é preciso em sua explicação:

> *Ipso facto*, é a animação em si, no topo do objeto simulado, que transmite a sensação de peso e presença, as propriedades físicas do objeto. A visualização da animação torna-se o objeto na mente do jogador e, assim, quaisquer propriedades físicas exibidas pela animação são percebidas pelo jogador como parte do movimento do objeto, mesmo que elas não tenham nada a ver com a simulação. Quando os efeitos de animação se tornam indistinguíveis de resposta programada, a animação cumpre, de modo tão bom quanto, a tarefa de criar a sensação de peso, massa, presença e fisicalidade (SWINK, 2009, p. 157, tradução nossa)[22].

[21] “We can take great liberties with fast actions – even with realistic figures” (WILLIAMS, 2002).
[22] Ipso facto, it is the animation itself, sitting on top of the simulated object, which conveys the sense of weight and presence, the physical properties of the object. The animated visualization becomes the object in the mind of the player, and so any physical properties it displays as it animates are perceived by the player as part of the motion of the object, even though they have nothing to do with the simulation. When animated effects become indistinguishable from programmed response, animation does just as good a job of creating the sense of weight, mass, presence and physicality (SWINK, 2009, pg. 157).

1.11. O limite do real

Mesmo levando-se em conta exceções como as apresentadas, a simulação física de games 3D está intimamente ligada à mecânica clássica. A utilidade das equações de movimento para simular o comportamento de objetos sujeitos a forças virtuais é inegável. Entretanto, acredito não ser essa a maior influência da teoria de Newton sobre os desenvolvedores de games 3D. Durante o desenrolar de um game, por exemplo, objetos e personagens que não estão sequer visíveis continuam tendo seus movimentos atualizados dessa maneira. Mais do que a mera aplicação de um arcabouço matemático, a mecânica clássica formata nosso modo de interpretar aquilo que muitas vezes tratamos, de forma incorreta, como simulações da realidade.

Não há nada de real em games 3D. O que neles vivenciamos – objetos, personagens, física, comportamentos, sons, luzes, etc. – é modelado e construído através de valores numéricos. Não questiono a importância da sensação de realidade para que a experiência faça sentido sob o ponto de vista do jogador. Mas por que deveríamos, enquanto desenvolvedores, limitar a modelagem de mundos virtuais a uma pretensa realidade que sabemos não existir? Se o resultado visível ao jogador parecer coerente (ou "real"), conforme apresentado nos exemplos deste capítulo, qual a importância da procedência, real ou não, do modelo matemático utilizado?

Voltaremos ao tema no capítulo 3. Antes, faz-se necessário investigar possíveis limitações impostas pela programação nos games 3D.

CAPÍTULO 2

2.1. Programação e imprevisibilidade

De maneira ampla, podemos descrever um game 3D como uma série de agrupamentos numéricos gerenciados por um programa de computador. Os números representam modelos 3D, texturas, sons, animações, iluminação, textos, etc. Para que esses blocos numéricos façam algum sentido, eles devem ser tratados por um programa: *uma lista fixa de instruções que são executadas em ordem pré-determinada*.

Ora, se pretendemos obter reações interessantes, não previsíveis e coerentes com objetivos, de personagens de games 3D, precisamos responder às seguintes perguntas: Pode uma sequência de instruções fixas gerar resultados não previsíveis? Tais resultados estariam limitados à aleatoriedade, ou poderiam ser coerentes dentro de contexto e objetivos determinados?

Pode-se argumentar sobre essas perguntas apenas com base em trabalhos teóricos de diversos autores, alguns dos quais são citados abaixo. Devido a minha formação de programador, entretanto, sempre me interessei em pesquisar o porquê de forma mais direta, inclusive para poder compreender melhor o que esses trabalhos teóricos propunham. Nesse sentido, veremos a seguir descrições de alguns processos deterministas (sequências fixas de instruções) e de como seus resultados são gerados.

2.2. O conjunto de Mandelbrot

Imagens geradas a partir do conjunto de Mandelbrot estão entre os exemplos mais familiares da complexidade dos Fractais. Entretanto, explicações sobre o processo utilizado na criação dessas imagens tendem a exigir certo domínio de matemática, especialmente na área de números complexos, e isso pode afetar seriamente a compreensão de aspectos que são importantes para o desenvolvimento deste

trabalho. Detalharei a seguir esse processo, na intenção de extrair alguns pontos que considero importantes.

Tudo começa com números complexos. Um número complexo é formado por uma parte real e outra imaginária. A parte imaginária tem por unidade o valor *i* (definido como $\sqrt{-1}$, resultado que não é possível entre os números reais). Assim, *(3 + 2i)* e *(-2 - i)* são dois exemplos de números complexos ilustrados na figura abaixo.

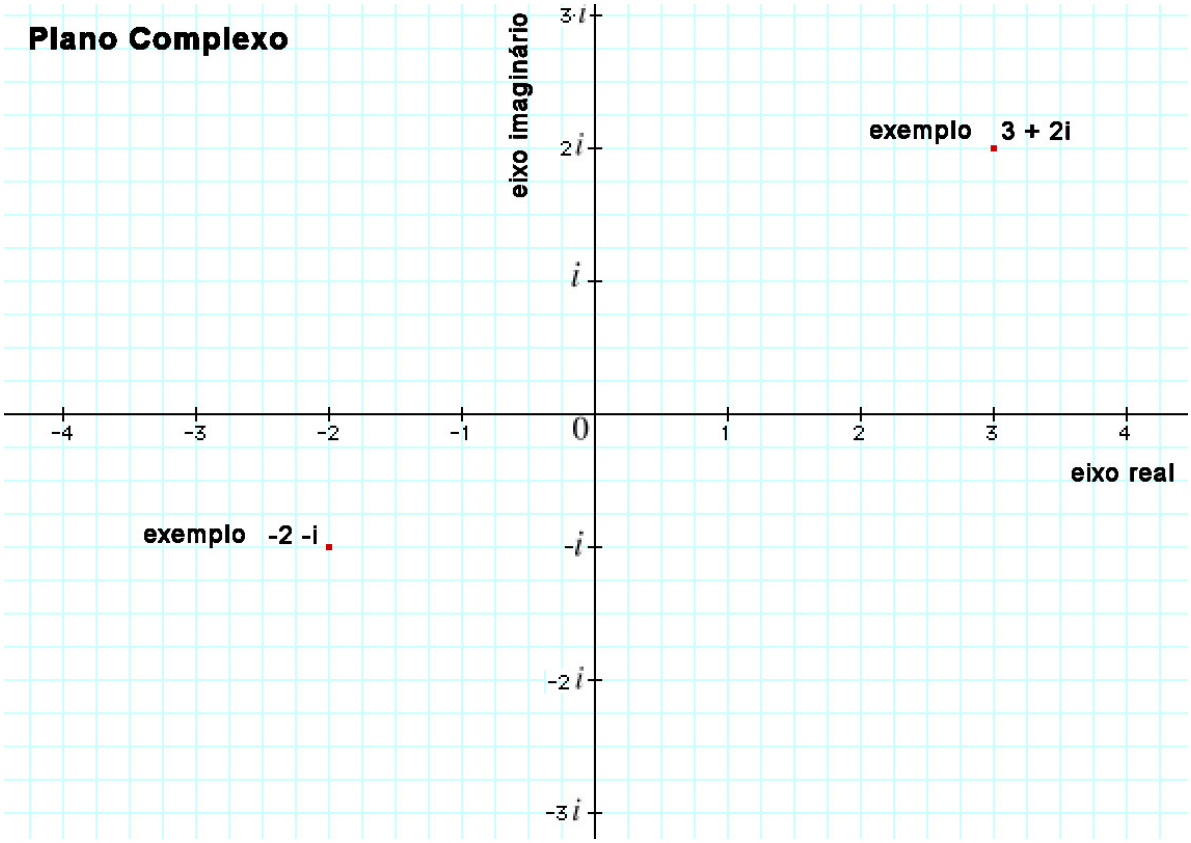


Figura 2.1. O plano complexo.

A figura também ilustra o plano onde as imagens fractais de Mandelbrot são geradas. Como cada número imaginário pode ser dividido em duas partes, podemos representá-los por dois eixos, um real e outro imaginário. O resultado é o *plano complexo*, espaço bidimensional infinito formado por todos os números complexos.

O conjunto do Mandelbrot é formado apenas pelos números complexos que obedecem a um critério específico, descrito a seguir. Chamemos de *c* um ponto qualquer do plano complexo. Tomemos outro número complexo (e, portanto, outro ponto do plano) que será denominado *z*. Mas o valor de *z* será definido numa

sequência de *feedback*, ou seja, *z* dependerá dele mesmo. Para simplificar, usaremos um índice numérico para identificar cada iteração[23] que gera um novo valor para *z*, ou seja: z_0, z_1, z_2, z_3, z_4, etc. Inicialmente, z_0 é igual a $0 + 0i$, ponto de origem do plano complexo. Para cada iteração, o novo valor de *z* será calculado de forma bem simples: multiplica-se o valor anterior por ele mesmo e soma-se o valor de *c*, que é o ponto do plano complexo que desejamos saber se pertence ou não ao conjunto de Mandelbrot. Assim, $z_1 = z_0 * z_0 + c$, então $z_2 = z_1 * z_1 + c$, depois $z_3 = z_2 * z_2 + c$, e ainda $z_4 = z_3 * z_3 + c$, e assim por diante. O ponto *c* pertencerá ao conjunto de Mandelbrot se – e somente se – nenhum dos valores de z ultrapassar a região circular ilustrada na figura abaixo[24].

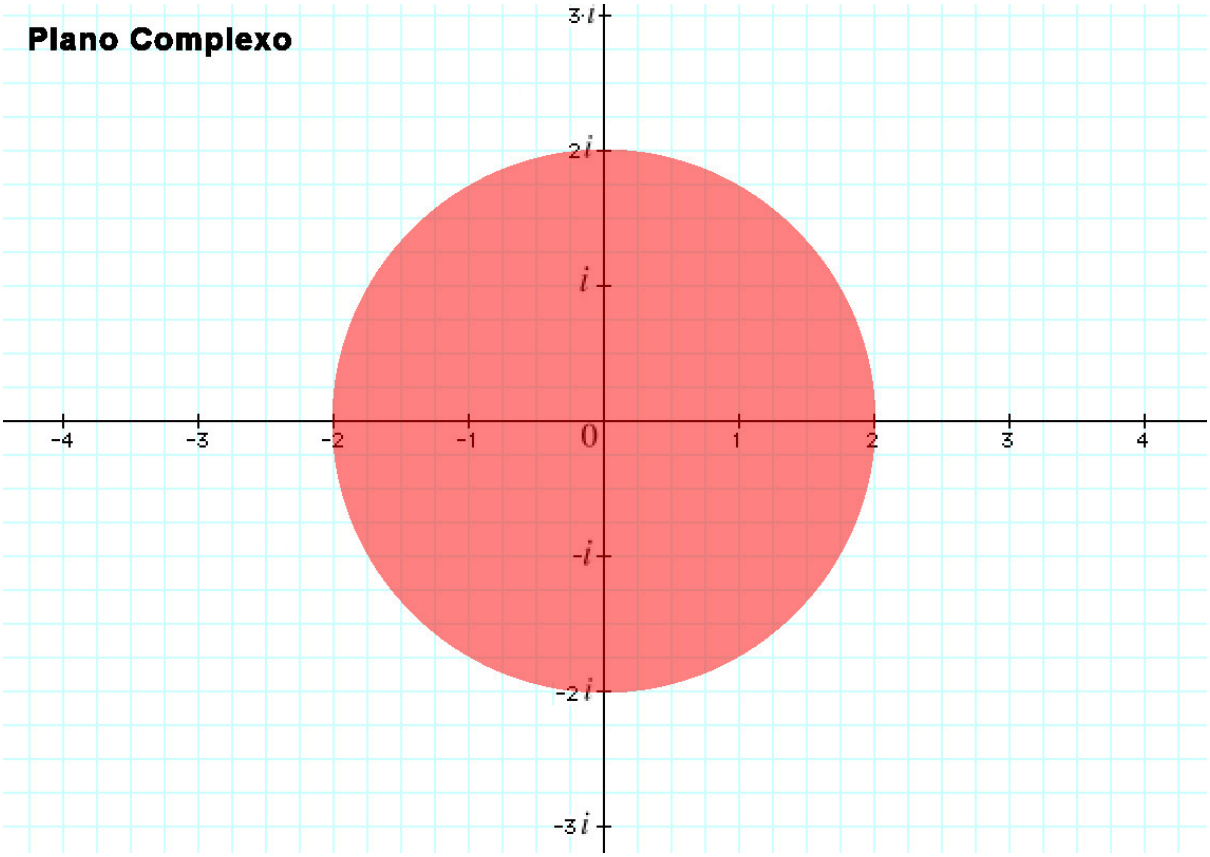


Figura 2.2. Limite para valores de z.

[23] Iteração é a repetição de um processo ou função, normalmente utilizando-se o resultado de uma etapa como dado de entrada para a próxima.

[24] Em outras palavras, *c* pertence ao conjunto de Mandelbrot quando o módulo de *z* nunca for maior que *2*.

Se testarmos essa condição para todos os pontos da região circular[25], atribuindo a cor preta aos pontos que pertencerem ao conjunto de Mandelbrot, obteremos o resultado abaixo[26]:

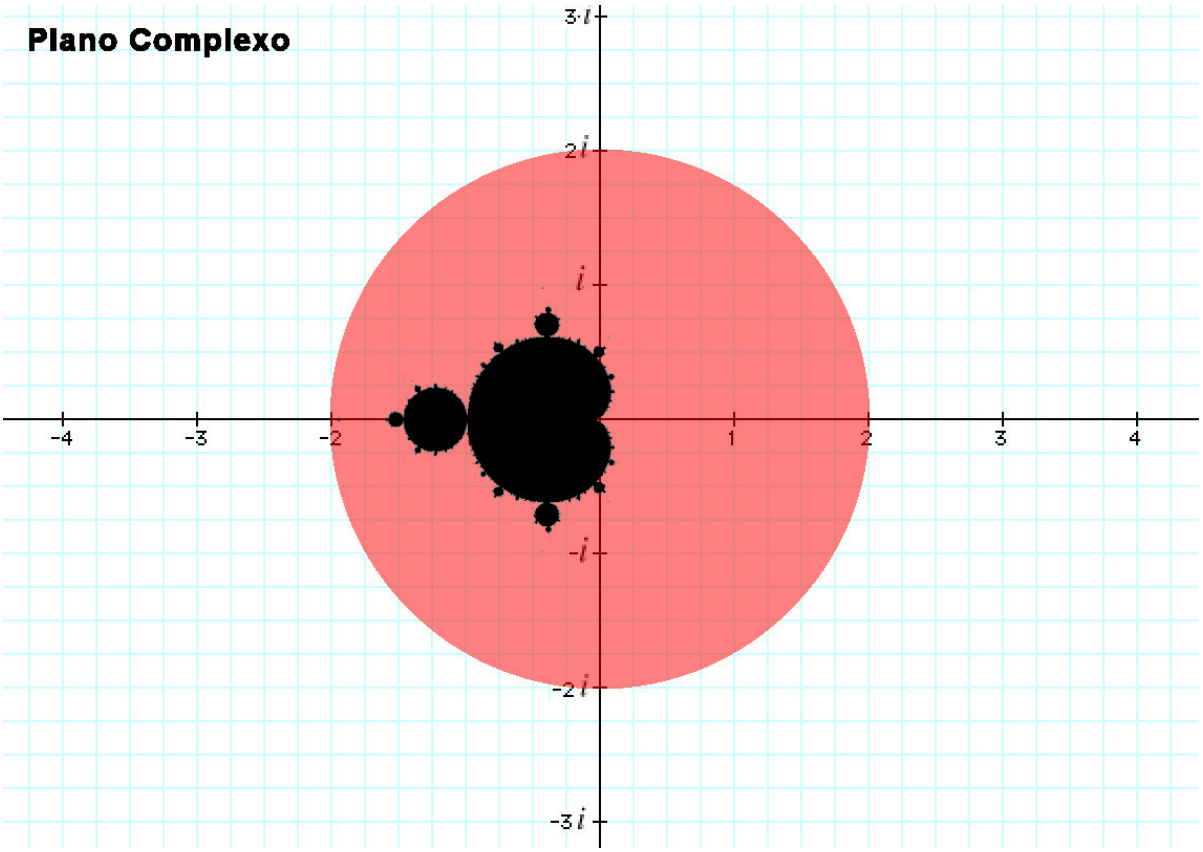


Figura 2.3. O conjunto de Mandelbrot.

O familiar cardioide é apenas a representação gráfica dos valores complexos que pertencem ao conjunto de Mandelbrot. Essa figura tem várias características notáveis. Por exemplo, ela pode ser ampliada indefinidamente e sempre aparecem novos cardioides, menores, que apresentam auto-similaridade, mas não existem dois cardioides idênticos. Já foi provado matematicamente que todas as áreas pretas estão conectadas[27], mas sua área total exata ainda é desconhecida. E, talvez o mais

---

[25] Se c estiver fora da região circular, z_1 também estará e portanto c não pertence ao conjunto.

[26] Todas as imagens do conjunto de Mandelbrot apresentadas nesta seção foram geradas especialmente para este trabalho através do software livre QuickMAN 1.10, originalmente escrito por Paul Gentieu (http://quickman.sourceforge.net).

[27] Nas imagens, os diversos cardioides ocasionalmente parecem isolados, mas apenas porque as conexões não são visíveis nessas escalas. Ao ampliarmos cada imagem, é possível enxergar essas conexões, bem como o surgimento de novos cardioides, menores.

intrigante, embora a área da representação gráfica do conjunto seja evidentemente finita, pois ela está contida num círculo de raio definido, seu perímetro é comprovadamente infinito. Quando dobramos a resolução da imagem, o perímetro é multiplicado por quatro, sempre (MANDELBROT, 1983).

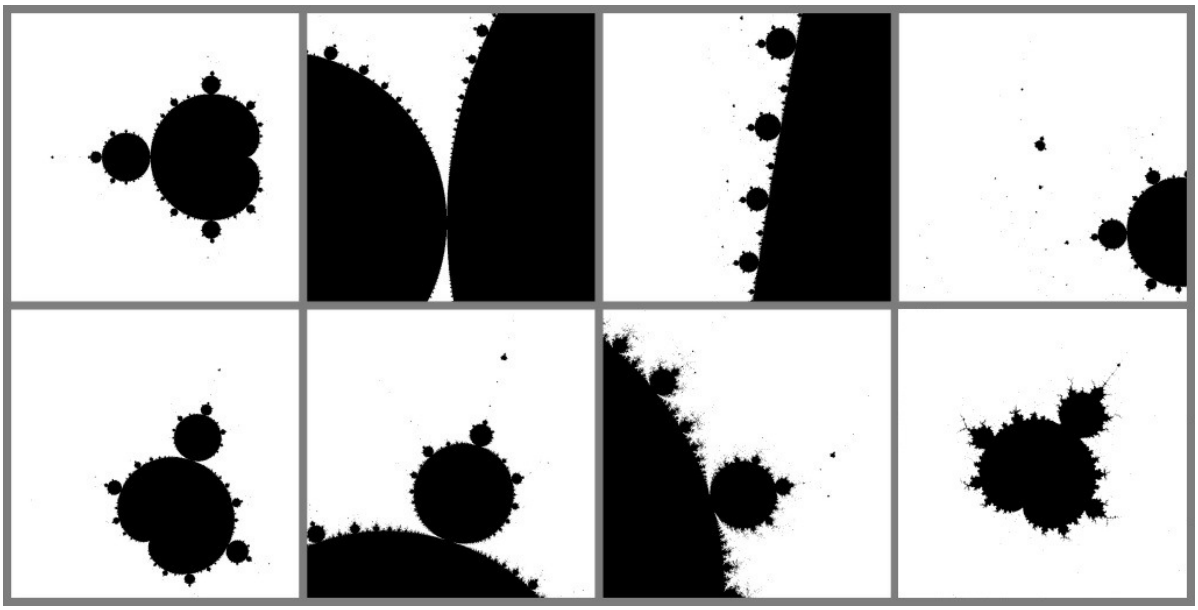


Figura 2.4. Imagens da representação gráfica do conjunto de Mandelbrot, em aproximações sucessivas.

A complexidade é ainda maior quando analisamos o perímetro infinito do conjunto. Nas imagens coloridas, isso é feito com uma palete de cores arbitrárias que são atribuídas aos pontos c que não fazem parte do conjunto. Se o valor de *z* escapa do círculo de raio 2 de forma muito lenta, o ponto *c* recebe a cor inicial da palete. Quanto mais rápido o crescimento de *z*, o ponto recebe sucessivamente as outra cores, até o limite da palete. Esse procedimento simples gera imagens coloridas altamente complexas, como pode ser visto na figura abaixo. Por mais intrincados que os padrões de cores pareçam, as cores nunca se cruzam. Isso acontece porque as áreas coloridas acompanham, em sucessão, a borda infinita do conjunto (MANDELBROT 1983).

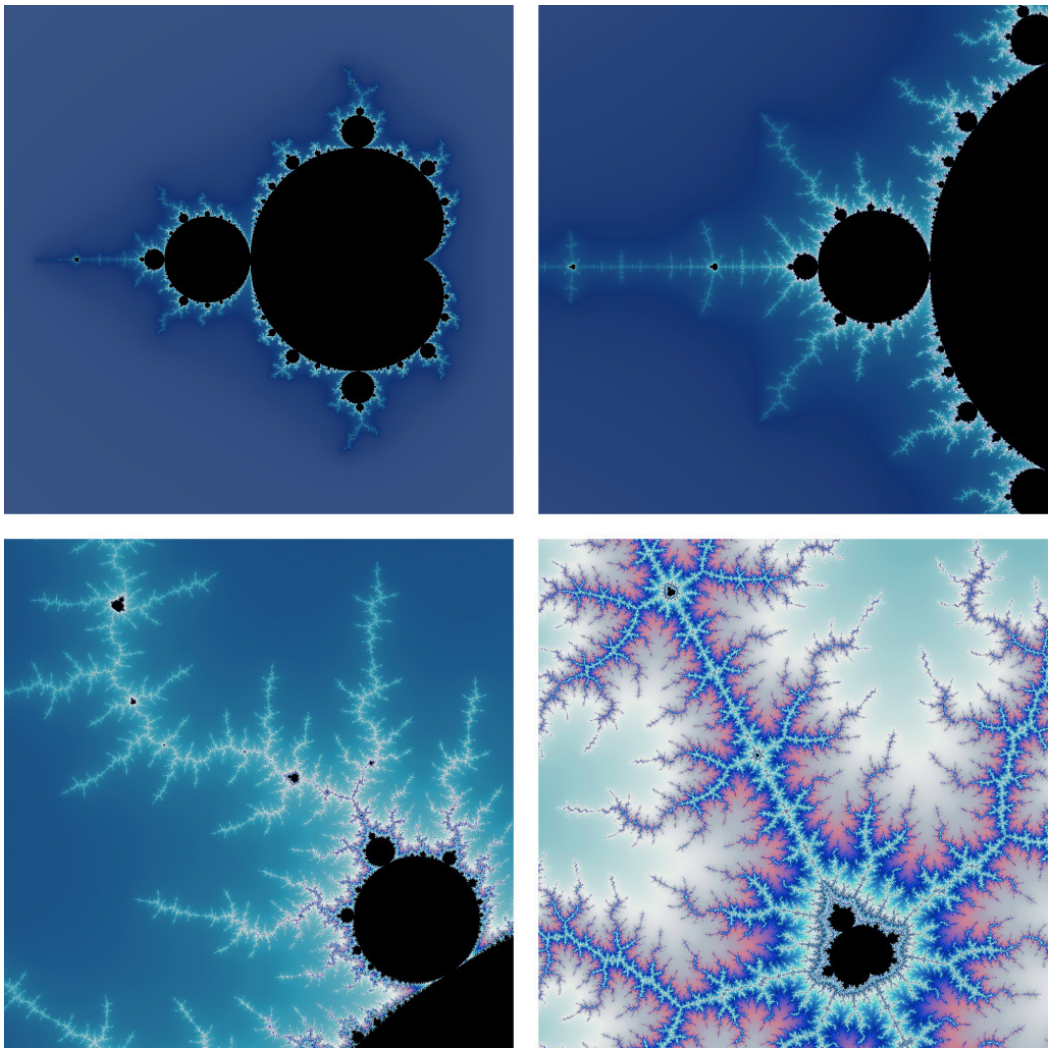


Figura 2.5. Quatro imagens que utilizam uma palete de cores para visualização do perímetro do conjunto de Mandelbrot. Da esquerda para a direita e de cima para baixo, as imagens mostram aproximações sucessivas.

Como uma figura tão complexa, com área limitada mas perímetro infinito, pode resultar de uma sequência de instruções tão simples?

Imagens fractais do conjunto de Mandelbrot são resultado de um sistema determinista, pois cada pixel das imagens é calculado segundo um processo exato, que não deixa qualquer espaço para aleatoriedade. Entretanto, tais imagens apresentam características de *sistemas caóticos*, sistemas que são extremamente sensíveis à variação de suas condições iniciais[28] (KELLERT 1993). Num sistema convencional, não caótico, pequenas variações nas condições iniciais acarretam

[28] Como outro exemplo de sistema determinista caótico, podemos considerar os modelos matemáticos de previsão climática.

pequenas variações no desenrolar do processo. Por exemplo: se diminuirmos ligeiramente a velocidade de arremesso de uma bola, ela apenas cairá um pouco antes do que se mantivéssemos a velocidade original. Quando a alteração de velocidade for muito pequena, tendendo a zero, a diferença no tempo de queda também tenderá a zero, de modo previsível. Por outro lado, se modificarmos a posição de um ponto *c* no plano complexo, mesmo que essa mudança tenda a zero, a cor do pixel equivalente na imagem fractal pode ser alterada significativamente, de forma não previsível, devido à trama complexa e infinita que o conjunto de Mandelbrot ocupa sobre esse plano. As imagens fractais do conjunto de Mandelbrot são exemplos de que um sistema determinista, com resultados gerados a partir de uma sequência fixa de instruções, pode não ser previsível.

2.3. Game of Life

Na década de 40, sem o auxílio de computadores, o matemático John von Neumann projetou uma máquina virtual capaz de realizar cópias de si mesma, num ambiente denominado *autômato celular*. Como o próprio nome sugere, autômatos celulares são compostos por células que têm comportamento programado. As células formam normalmente uma matriz de pequenos quadrados que podem ser programadas para assumir estados diversos.

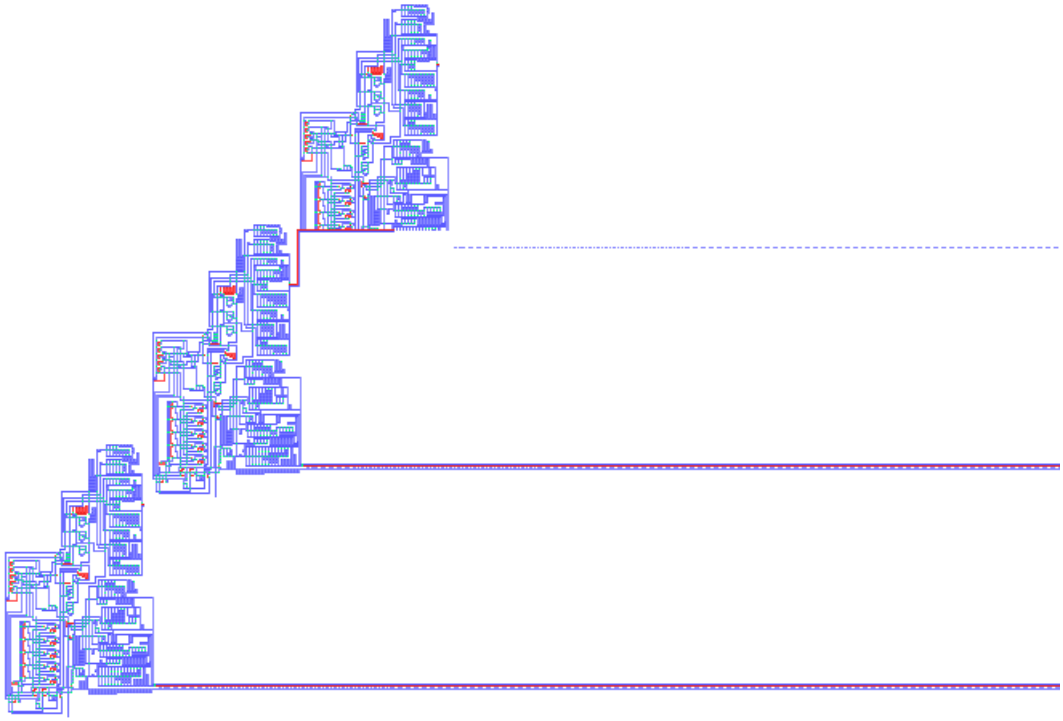


Figura 2.6. Construtor universal de von Neumann.

Detalhes sobre essa máquina, que ficaria conhecida como *construtor universal de von Neumann*, foram publicados apenas em 1966, no livro *Theory of Sef-Reproducing Automata*, completado por Arthur W. Burks após a morte de von Neumann. Para solucionar o problema da criação de autômatos auto-replicantes, von Neumann projetou regras relativamente complexas que definiam o estado de cada célula (representado por cores na figura acima), além de partir de padrões iniciais sobre os quais as regras eram aplicadas automaticamente.

Em 1970, o matemático inglês John Horton Conway trabalhou sobre problema similar, desenvolvendo um autômato celular que utilizava regras muito mais simples que as propostas por von Neumann. Trata-se de *Game of Life*, que foi apresentado originalmente na coluna *Mathematical Games*, de Martin Gardner, na edição de outubro de 1970 da revista *Scientific American*.

Game of Life desenvolve-se numa matriz potencialmente infinita de células quadradas. Cada célula pode estar “viva” (estado representado nas figuras abaixo pela cor preta) ou “morta” (em branco) e interage com suas oito vizinhas imediatas (duas horizontais, duas verticais e quatro diagonais). As interações ocorrem em

ciclos chamados de *gerações*, e consistem na aplicação, sobre cada célula, das quatro regras descritas abaixo:

1. qualquer célula viva com menos de duas vizinha vivas morre, como que por sub-população;

2. qualquer célula viva com duas ou três vizinhas vivas permanece viva na próxima geração;

3. qualquer célula viva com mais de três vizinhas vivas morre, como que por sobre-população;

4. qualquer célula morta com exatamente três vizinhas vivas passa a ser viva, como que por reprodução.

Aplicadas a um padrão inicial de células vivas, essas regras simples podem gerar complexidade. Sem o auxílio de computadores, Conway seguia a evolução desses padrões utilizando papéis quadriculados e tabuleiros do jogo GO, demonstrando uma série de padrões simples que denominou de “naturezas mortas” e “osciladores”. Ao analisar o padrão conhecido como “pentomino F”, Conway descobriu o primeiro elemento que se deslocava pela grade, batizando-o de *glider* (ilustrado na figura abaixo).

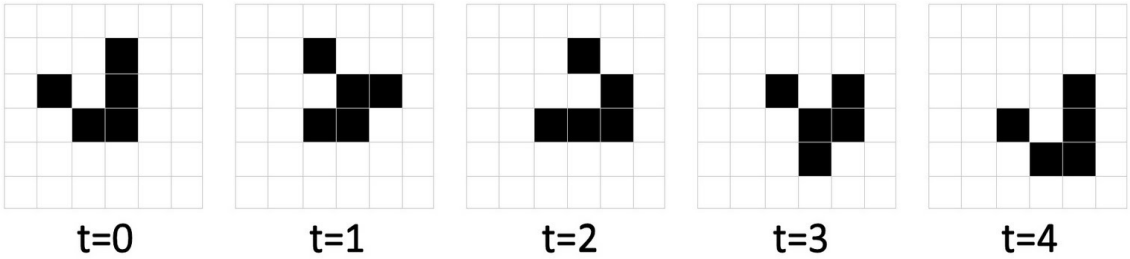


Figura 2.7. Um "glider", elemento emergente do *Game of Life*, movimentando-se durante quatro gerações

Vários elementos similares foram descobertos mais tarde, com auxílio de softwares. São hoje classificados como “espaçonaves”. Podemos ver um exemplo batizado de “lightweigh spaceship” na figura abaixo.

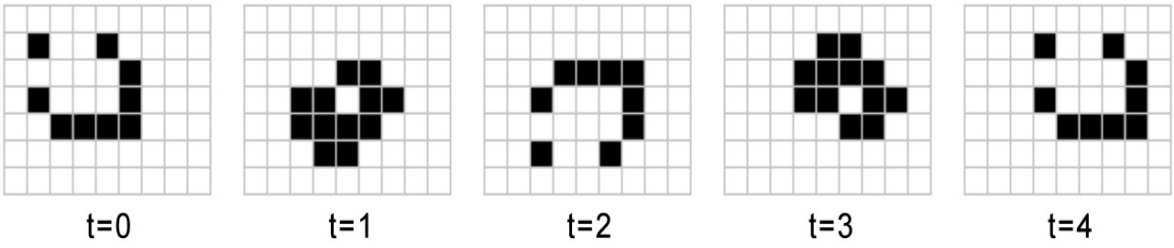


Figura 2.8. “Light weigh spaceship”.

Uma das suposições iniciais de Conway foi a de que nenhum padrão poderia crescer indefinidamente. No artigo da *Scientific American*, Conway propôs um prêmio de US$ 50 para a primeira pessoa que provasse que essa suposição estivesse errada até o final de 1970. O prêmio foi ganho em novembro do mesmo ano por uma equipe do MIT liderada por Bill Gosper. O padrão proposto, denominado “canhão de gliders de Gosper”, emitia gliders indefinidamente (figura abaixo)

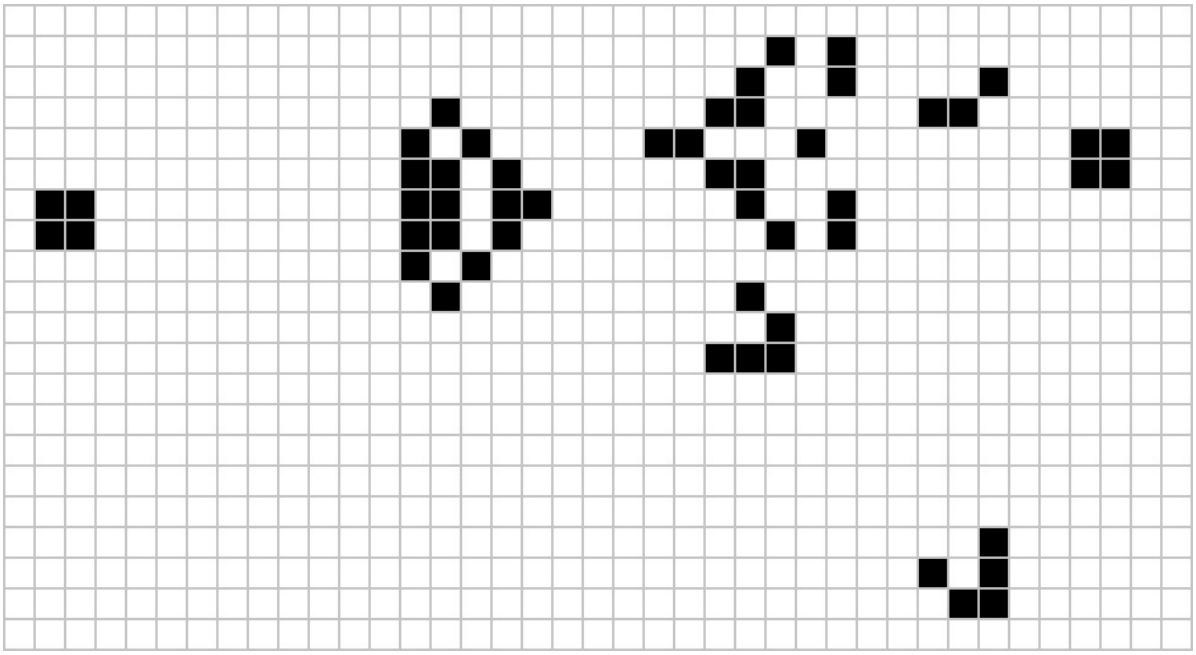


Figura 2.9. “Gosper’s glider gun”.

A característica de *Game of Life* que considero mais intrigante é o fato de que ele pode ser configurado como uma *máquina de Turing*. O matemático Alan Turing propôs tal máquina em 1936, como algo capaz de executar operações sobre números “computáveis”. Além disso, Turing mostrou que seria possível desenvolvê-la de modo a computar qualquer sequência numérica computável, o que a tornaria uma *máquina computadora universal* (TURING 1936). Na época em que foi feita, tal

definição não era trivial – na verdade, ela ainda pode parecer problemática para muitos nos dias de hoje. Intuitivamente, esperava-se (e, até certo ponto, espera-se) que problemas com dificuldade crescente só pudessem ser resolvidos por máquinas cada vez mais poderosas.

Mas Turing demonstrou que seria possível construir processos de complexidade ilimitada a partir de uma máquina de complexidade finita. Isso acontece porque processos complexos podem ser divididos em sequência de instruções mais simples. Por exemplo, entre as operações binárias mais básicas estão as combinações booleanas “E”, “OU” e “NÃO” (a primeira resulta num bit ligado se – e somente se – seus dois operandos estiverem ligados, a segunda tem o mesmo resultado quando qualquer um dos dois estiver ligado, enquanto que a terceira apenas inverte o estado do bit operando). É possível criar uma operação mais complexa, um operador de soma de dois números, como uma sequência dessas operações booleanas mais simples. Da mesma forma, usando-se o operador de soma determinado número de vezes, pode-se criar um operador de multiplicação – e assim por diante. Usando uma linha de raciocínio similar, Turing demonstrou que era possível realizar qualquer sequência computável com sua máquina universal. Sem usar o termo, em 1936, Turing propunha a noção de *algoritmo* como uma forma de se projetar processos gradativamente mais complexos a partir de instruções simples, sem outros limites além de tempo e capacidade de memória. O trabalho de Alan Turing influenciou fortemente o desenvolvimento dos computadores (batizados de acordo com sua máquina computadora universal).

Em abril de 2000, Paul Rendell construiu uma máquina de Turing no *Game of Life*, usando padrões como *gliders*, *glider-guns* e *light-weight spaceships* para processar operações (RENDELL, 2005). Em março de 2011, Rendell aperfeiçoou-a numa máquina universal de Turing completa. Isso equivale dizer que qualquer problema computável pode ser processado no Game of Life. Quando o programa (as regras e o modo de aplicá-las) do Game of Life foi definido por Conway, em 1970, não teria sido possível prever que ele tinha o potencial de se tornar um computador universal – sendo capaz, inclusive, de rodar seu próprio algoritmo.

Game of Life encaixa-se bem no conceito de máquinas não triviais, de Heiz von Foerster. Segundo essa proposta, tanto máquinas triviais como não triviais são deterministas, pois executam uma sequência de instruções. Entretanto, nas máquinas não triviais, esse determinismo pode gerar imprevisibilidade.

> A fim de compreender a diferença profunda entre esses dois tipos de máquinas pode ser útil visualizar "estados internos" nestas máquinas. Enquanto na máquina trivial apenas um estado interno participa sempre em seu funcionamento interno, na máquina não trivial é a mudança de um estado interno para outro que a torna tão elusiva (VON FOERSTER, 1972, p. 6, tradução nossa)[29].

Voltaremos aos “estados internos” quando analisarmos outro algoritmo, o da máquina de estados finitos. Mas é interessante notar que, em *Game of Life*, o estado de uma célula depende dos estados anteriores de células vizinhas. Portanto, o estado dessa máquina virtual como um todo, considerando-se todas as células, depende de (e é definido por) estados anteriores.

[29] In order to grasp the profound difference between these two kinds of machines it may be helpful to envision “internal states” in these machines. While in the trivial machine only one internal state participates always in its internal operation, in the non-trivial machine it is the shift from one internal state to another that makes it so elusive (VON FOERSTER, 1972, p. 6).

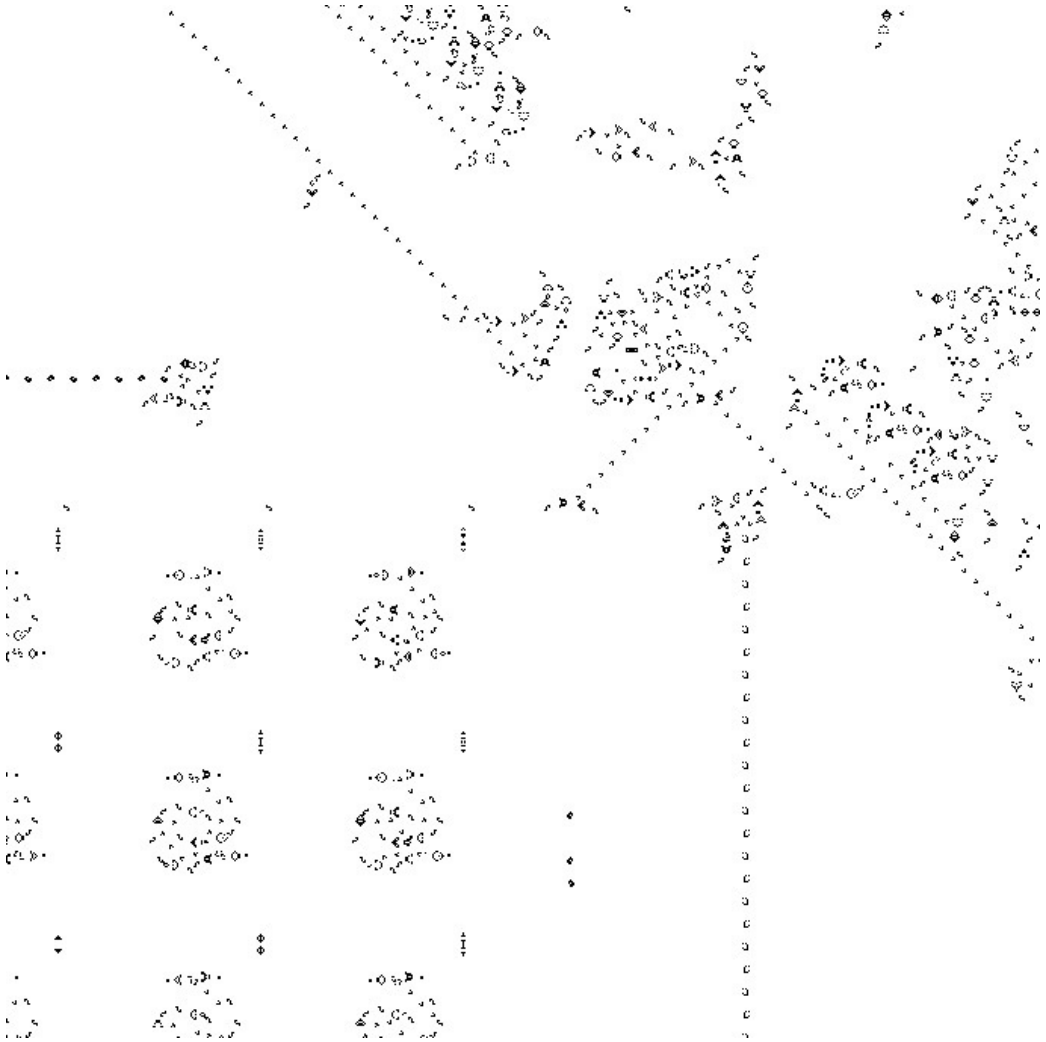


Figura 2.10. Trecho da máquina universal de Turing criada por Paul Rendell no *Game of Life* (imagem Paul Rendell).

2.4. Comportamentos Emergentes em Games 3D

Vimos que *Game of Life* pode gerar ordem e complexidade inesperadas a partir da repetição de regras básicas. Tal fenômeno é conhecido como emergência. Exemplos clássicos de sistemas emergentes são o formigueiro e o cérebro, ambos baseados na interação entre numerosos elementos relativamente limitados, formigas e neurônios.

Por sua construção igualmente baseada em regras, games são sistemas de software especialmente apropriados à emergência. Alguns exemplos já clássicos são as criações de Will Wright, *Sim City* (Maxis, desde 1989), *The Sims* (Maxis, desde 2000) e *Spore* (Maxis, 2008).

Games atuais são desenvolvidos com técnicas de orientação a objetos. Essas técnicas dividem o programa em subsistemas independentes, e é a interação entre esses objetos que constrói a funcionalidade do todo. Assim como ocorre nos games, alguns sistemas de software buscam exatamente a emergência para simular comportamentos complexos que não poderiam ser programados de forma explícita. Usando como exemplo o programa *StarLogo*, software modelador de comportamentos emergentes escrito por Mitch Resnick, a descrição de Steven Johnson (2001) para tais sistemas é precisa:

> Sistemas como *StarLogo* não são anarquias absolutas: eles obedecem a regras que definimos previamente, mas essas regras governam apenas os micro-motivos. O macro-comportamento é outra história. Não se controla isso diretamente. Tudo o que podemos fazer é configurar as condições que, cremos, tornarão possível tal comportamento. Então, apertamos 'play' e vemos o que acontece (JOHNSON, 2001, p. 169, tradução nossa)[30].

De fato, na programação de comportamento de personagens, o controle do programador limita-se às regras básicas de interação entre objetos, sejam eles personagens ou outros elementos do game. O único modo de saber ao certo qual o impacto dessas regras no comportamento macro é rodar o programa e assistir a seus desdobramentos. Macro-comportamentos exibidos por personagens são emergentes por não serem explicitamente programados ou previsíveis, mas é importante lembrar que tais comportamentos estão solidamente codificados nas regras originais, ainda que o programador responsável não os possa prever.

O algoritmo mais comum para criação de comportamento artificial de personagens é a máquina de estados finitos. Nele, uma série de estados lineares simples (por exemplo: andar, esperar, correr, atirar, morrer, etc.) são chaveados por regras de transição. Assim, por exemplo, um personagem que está no estado "esperar" pode mudar para os estados "andar" ou "correr" desde que um objetivo válido apareça em seu campo de visão. Comportamentos simples, como perseguir outros personagens ou fugir deles, são derivados dos estados disponíveis e das regras de transição. É importante notar também que as máquinas de estados finitos não se limitam a personagens. Uma porta, por exemplo, pode ser representada por quatro estados

---

[30] Systems like StarLogo are not utter anarchies: they obey rules that we define in advance, but those rules only govern the micromotives. The macrobehaviour is another matter. You don't control that directly. All you can do is set up the conditions that you think make that behavior possible. Then you press play and see what happens. (JOHNSON, 2001, p. 169)

(fechada, abrindo, aberta, fechando), que são disparados de acordo com os comandos do interator e com o estado atual da porta. Entretanto, a interação entre personagens normalmente resulta em comportamentos mais complexos, muitas vezes indesejáveis.

Para ilustrar a aplicação prática desses algoritmos, bem como a emergência de novos comportamentos a partir de estados e regras, segue uma descrição parcial de dois estados e suas transições, utilizados em personagens de *Incidente em Varginha* (Perceptum, 1998):

Estado 1: Sair de linha visual quando sob ataque

> Quando atacado, o personagem deve mover-se até que não exista linha visual entre ele e seu oponente. Esse estado simula a procura por abrigo e, ao mesmo tempo, o movimento torna o personagem um alvo mais difícil, mesmo enquanto ele permanece em linha visual com o oponente.

Estado 2: Procurar linha visual quando atacando

> Quando atacando, o personagem deve mover-se até que exista linha visual entre ele e seu oponente. Esse estado simula a busca por posições de ataque viáveis, evitando, por exemplo, que o personagem procure inutilmente atingir seu oponente atirando contra uma parede ou coluna.

Separadamente, os dois estados podem ser considerados como programação explícita, não emergente. Com frequência, entretanto, esses estados sobrepõem-se: o personagem ataca enquanto é atacado. As transições entre estados podem criar comportamentos diversos, não explícitos na programação, que emergem dos estados básicos, das transições entre estados e das condições de interação. Por exemplo:

- Prioridade para o estado 1 com período de execução de 0.1 segundos, resulta em comportamento “covarde”. O personagem esconde-se ao primeiro sinal de ataque. Imediatamente após sair da linha visual, o estado 2 leva-o a deixar o esconderijo para onde retorna um décimo de segundo após. E assim

sucessivamente, enquanto o personagem estiver simultaneamente atacando e sob ataque. Na prática, o personagem oscila rapidamente à beira de um ponto de abrigo, como uma esquina ou uma coluna, de modo inverossímil. Cabe comentar que a inverossimilhança refere-se apenas à imitação de um personagem real, uma vez que o comportamento é adequado às (bem como consequência das) regras estabelecidas na maquina de estados finitos.

- Prioridade para o estado 1 com período de execução de 4 segundos resulta em comportamento prudente, mas verossímil. O personagem ataca por alguns segundos e busca abrigo. Volta a atacar e a buscar abrigo, num ciclo contínuo enquanto a superposição de estados perdurar.

- Prioridade para o estado 2 com período de execução de 0.1 segundos: comportamento agressivo. O personagem simplesmente ataca, não procurando abrigo. Caso seu oponente fuja, saindo da linha visual, procura imediatamente segui-lo, de acordo com o estado 2.

Levando-se em conta ainda que as próprias transições podem ser modificadas durante o desenrolar do game, esses dois estados simples podem criar comportamentos complexos e interessantes. Soldados que atacam de posições protegidas, nunca totalmente expostos, que podem se acovardar repentinamente, ou tornarem-se ainda mais agressivos quando atingidos. Esses comportamentos decorrem dos dois estados e das transições escolhidas, mas dependem também da ação do oponente. Por exemplo, enquanto não for atacado, o personagem não buscará abrigo, independentemente das transições escolhidas. Todos esses fatores tornam os comportamentos emergentes difíceis de prever e de projetar.

Figura 2.11. Um soldado de Incidente em Varginha, atacando próximo a um muro de proteção.

Dependendo do número de estados (que totalizam várias dezenas em games comerciais) e das variações de transição, a complexidade dos comportamentos aumenta tanto que é praticamente impossível prever precisamente a reação de um personagem. Mudanças mínimas nas regras de transição podem gerar alterações significativas no comportamento geral. A solução seria a simulação exaustiva das reações dos personagens, para cada pequena alteração nos algoritmos de controle. Isso não é viável para personagens com grande número de estados, pois as combinações possíveis são muito numerosas e, normalmente, difíceis de reproduzir sob ambiente de testes. Resta ao desenvolvedor aplicar os testes viáveis e esperar pelo melhor.

Outros fatores do ambiente virtual também influenciam o comportamento de personagens. Como exemplo, consideremos a própria linha de visão citada acima. De forma geral, para saber se um personagem enxerga seu oponente, o programa deve traçar uma linha matemática entre os olhos do personagem e a cintura do oponente, seu ponto central. Se essa linha for interrompida por qualquer objeto sólido (parede, coluna, outro personagem, etc.), assume-se que o oponente não está visível; caso contrário, o oponente estará visível. Esse algoritmo funciona muito bem na grande maioria das situações, mas há exceções. Imaginemos, por exemplo, que exista um corrimão exatamente à frente da cintura do oponente. Nesse caso, ainda que o oponente esteja em grande parte exposto, a linha de visão do personagem será interrompida e este não o enxergará. Por outro lado, se o

oponente também for um personagem controlado pelo computador, ele enxergará seu antagonista normalmente e o atacará, pois a linha matemática que define sua visão parte da altura de seus olhos, que estão desimpedidos. Mesmo atacado, o primeiro personagem não buscará abrigo, pois sua linha de visão continua interrompida, condição estabelecida pelo programador como indicador de que se está em lugar seguro. Ao contrário, guiado pelo estado 2, o personagem procurará uma boa posição de tiro, andando de um lado para o outro na tentativa de desobstruir a linha de visão. Situações similares são observáveis mesmo em superproduções como *Half-Life* (Valve, 1998) e *Medal of Honor* (EA, 2002).

Existem soluções para o problema específico apresentado acima. Ele é gerado não pelas transições ou estados, mas pelo sistema simplificado de simulação de visão. Se a linha matemática de sondagem fosse dirigida aos olhos do oponente (e não à sua cintura), ela não seria obstruída nesse caso específico. Mas o seria em outros, potencialmente mais críticos. Por exemplo, uma pequena tabuleta entre os personagens e à altura de seus olhos impediria que qualquer deles enxergasse o outro. Outra solução seria aumentar a resolução da visão simulada, acrescentando uma segunda linha matemática de sondagem que ligasse os olhos do personagem aos de seu oponente. A combinação de duas linhas em ângulos diferentes evitaria grande parte dos problemas de obstrução, mas dobraria o custo de processamento do sistema de visão. Ainda assim, ocorreriam problemas: o oponente estaria abrigado quando exatamente atrás de um poste de pequeno diâmetro que, mesmo deixando-o exposto em grande parte, obstruiria as duas linhas de visão. Talvez pudéssemos utilizar quatro linhas, acrescentando uma para cada ombro. Ou seis, com mais duas para os pés. Mas e se o oponente estivesse atrás do poste e de lado? Acrescentaríamos mais duas linhas, para os extremos do peito e das costas? E se apenas os braços estivessem visíveis?

Aumentar a resolução da simulação é possível. Mas o custo de processamento aumenta rapidamente e pode comprometer todo o sistema. Pode-se argumentar que o poder de processamento cresce constantemente e que, eventualmente, simulações precisas de processos físicos ou orgânicos serão possíveis. Ainda que isso aconteça, como vimos no capítulo anterior, bons games não se baseiam na simulação precisa, mas na consistência da realidade virtual apresentada. Criar

algoritmos na base da força bruta, tentando prever cada situação possível, dificulta a aplicação de conceitos de orientação a objetos e de máquinas de estados finitos. Por outro lado, algoritmos elegantes, com relativamente poucos estados e regras de transição mais simples, tendem a gerar resultados mais consistentes e emergentes. Cabe ao desenvolvedor conhecer as características do meio digital, utilizando-as a seu favor ou subvertendo-as.

Em 1996, durante o desenvolvimento do game *Incidente em Varginha* (Perceptum, 1998), pude testemunhar alguns exemplos de comportamentos emergentes indesejáveis, embora fascinantes. Um dos mais complexos dizia respeito aos soldados inimigos do game. Tentando reforçar o comportamento de grupo, incluí na máquina de estados finitos desses personagens a seguinte regra: se o soldado estivesse dentro de uma faixa de distância determinada de outro soldado, ele deveria primeiro aproximar-se do companheiro para apenas depois atacar o jogador. A intenção era simplesmente agrupar os personagens antes de atacar, mas não era exatamente isso o que ocorria. Um dos soldados voltava as costas ao inimigo para se aproximar do companheiro. O outro soldado, obedecendo à mesma regra, vinha na direção oposta. Quando ambos atingiam a distância mínima, o estado de ataque deveria ser disparado. Entretanto, com a inércia do movimento, os soldados ultrapassavam novamente a distância mínima estabelecida e, basicamente, trocavam de posições. O segundo soldado voltava então as costas ao jogador para obedecer a regra acima antes de atacar. E todo o processo repetia-se indefinidamente.

Resultavam daí estranhas coreografias entre esses personagens, algo que seria extremamente difícil de programar intencionalmente. Com poucos soldados, o comportamento beirava o ridículo, com personagens girando uns em torno dos outros em pequenos círculos. Porém, quando algumas dezenas de soldados estavam em ação, situação comum no game, a complexidade dos movimentos era desconcertante. Eles giravam lentamente em várias figuras circulares – de dois, quatro ou oito personagens – que interagiam entre si formando padrões ainda mais complexos. A sincronização emergente de movimentos levava vários personagens a emitir a mesma fala ao mesmo tempo, criando coros de vozes que proferiam as

limitadas frases dos soldados de forma assustadora, como se participassem de alguma dança ritual.

Por mais fascinante que fosse, é óbvio que esse comportamento de grupo não era desejável para tais personagens. Após semanas de experimentações – tanto para explorar as possibilidades do comportamento como para entendê-lo parcialmente –, uma pequena alteração na regra foi suficiente para que tudo voltasse ao planejado. Mas não foi possível evitar uma nítida sensação de perda ao efetuar a correção.

Dado o caráter imprevisível dos comportamentos emergentes, não é possível planejá-los a partir de diretrizes básicas. Tendo unicamente as regras locais como interface, ao programador caberia apenas recorrer ao método da tentativa e erro, conforme ilustrado acima. Mas essa limitação pode ser eliminada com a inclusão de um laço de realimentação (feedback) entre comportamentos e regras. É surpreendente constatar que tal objetivo pode ser alcançado através de sequências fixas de instruções, ou seja, de forma determinista.

2.5. Programação evolutiva

Difundidos pela pesquisa de John Holland nas décadas de 60 e 70, os algoritmos genéticos são especialmente aptos a esse tipo de solução. A pesquisa de Holland tinha dois objetivos principais: contribuir para a compreensão dos processos de adaptação natural e projetar sistemas artificiais que apresentassem propriedades similares às de sistemas naturais.

Por exemplo, no lugar de planejar e escrever um programa, imagine que simplesmente sorteássemos instruções ao acaso para formá-lo. Cada número sorteado definiria uma instrução específica (o que é muito conveniente, pois as instruções de um programa não passam de valores numéricos interpretados por um processador, como vimos no primeiro capítulo). Definimos dessa maneira um genoma muito simples para o programa: uma sequência de variáveis que, ao assumir valores específicos (após o sorteio), representa um genótipo, um dos muitos programas possíveis dentro do genoma proposto.

Dificilmente um programa escrito assim faria qualquer sentido. Ainda menos provável seria que ele funcionasse como solução para um problema específico. Potencialmente, entretanto, existem sequências de instruções permitidas pelo genoma que solucionariam diversos problemas diferentes. Convencionalmente, encontrar a sequência correta de instruções para solucionar tais problemas seria função de um programador. É justamente aqui que um novo paradigma se faz presente:

> O novo paradigma baseia-se fortemente nas regras da seleção natural, procriando novos programas a partir de uma variada reserva de genes. As primeiras décadas do software foram essencialmente criacionistas em sua filosofia – uma vontade todo-poderosa conclamava o programa à existência. Mas a próxima geração é profundamente darwiniana. (JOHNSON, 2001, p. 169, tradução nossa)[31].

Sequências de instruções inicialmente aleatórias podem evoluir até uma solução adequada. Para tanto, é necessário um laço de realimentação entre os resultados dos programas e as sequências de valores que os compõem. Essa realimentação é executada pelo algoritmo genético, que nesse caso pode ser descrito de modo bastante simplificado como:

1. Gerar uma população de programas aleatórios.

2. Avaliar o resultado de cada programa (função de avaliação).

3. Caso exista resultado satisfatório, parar execução e apresentar solução.

4. Selecionar os programas mais eficientes e eliminar os restantes.

5. Gerar nova população de programas através de hibridização, mutação ou clonagem dos genótipos dos programas selecionados.

6. Retornar ao passo 2.

---

[31] The new paradigm borrows heavily from the playbook of natural selection, breeding new programs out of a varied gene pool. The first few decades of software were essentially creationist in philosophy – an almighty power will the program into being. But the next generation is profoundly Darwinian. (JOHNSON, 2001, p. 169)

É interessante notar que não há nada de essencialmente diferente no algoritmo genético descrito acima: ele é apenas uma sequência de instruções, como qualquer outro programa. Mas sua aplicação é relativamente aberta. Como os resultados dos programas são selecionados segundo uma função de avaliação, é possível evoluir programas com finalidades completamente diferentes por meio de uma simples troca da função de avaliação no mesmo algoritmo genético.

Funções de avaliação de comportamentos emergentes seriam, então, suficientes para refinar sucessivamente as regras básicas que os definem. As regras não seriam programadas, mas, antes, “cultivadas” por evolução artificial. Ao programador do sistema caberia apenas definir uma função de avaliação apropriada para o comportamento projetado. Ele não precisaria sequer compreender como as soluções decorrentes funcionariam.

2.6. Evoluindo Comportamentos

Em 1997, após a constatação das possibilidades emergentes das máquinas de estados finitos, trabalhei para que o comportamento artificial dos personagens de *Incidente em Varginha* fosse, ainda que parcialmente, evoluído através de algoritmos genéticos simplificados. Um programa próprio simulava, em 2D, diversos ambientes do game. Nessa simulação, apelidada de “Tanque”, os personagens eram representados por meros ícones, mas o comportamento geral era regido pelas mesmas regras de suas respectivas máquinas de estados finitos. Comportamentos como perseguição, fuga, ataque, etc., eram simulados em tempo real sobre um mapa 2D. Um personagem específico, também controlado pelo computador, representava o jogador. No ecossistema virtual do “Tanque”, o jogador exercia o papel de predador, perseguindo e atacando constantemente a população de inimigos, definindo assim a seleção dos mais aptos.

Os diversos estados permaneciam fixos, mas as variáveis e as regras de transição sofriam pequenas mutações aleatórias de geração para geração. Por exemplo, a

função TropEscape[32], que executa o estado de fuga para os soldados, é definida pelo seguinte script:

```
ACTION TropEscape {
      SET MY.IF_NEAR,         NULL;
      SET MY.IF_FAR,          TropHide;
      SET MY.EACH_CYCLE,      CycleTropHide;
      SET MY.IF_HIT,          TropHit;
      SET MY.CAREFULLY,       1;
      SET MY.BERKELEY,        0;
      SET MY.SKILL4,          4;    // skill4 = state
      SET MY.TEXTURE,         Trop1Tex;
      SET MY.DIST,            30;
      SET MY.SPEED,           0.4;
      SET MY.TARGET,          REPEL;
}
```

No “Tanque”, o valor da variável my.speed, por exemplo, podia sofrer pequenas alterações aleatórias de uma geração para outra, aumentando ou diminuindo a velocidade do personagem no estado de fuga. O mesmo ocorria com outras variáveis e regras de transição.

Após um tempo de simulação pré-estabelecido, os personagens sobreviventes eram selecionados e serviam de base para uma nova geração. Não se tratava estritamente de um algoritmo genético, pois o próprio genoma era muito simplificado, representando apenas valores de um grupo parcial de variáveis, sem interferir diretamente no algoritmo do programa propriamente dito. Ainda assim, resultados extremamente interessantes foram obtidos.

A intenção era explorar comportamentos emergentes inesperados. Esse objetivo certamente foi atingido, mas poucos resultados foram práticos, pelos motivos que estão expostos a seguir. No geral, a experiência foi bem sucedida, servindo para enfatizar, na prática, a importância da função de avaliação nesse tipo de processo.

A primeira função de avaliação empregada foi simplesmente o tempo de sobrevivência dos personagens. Conforme descrito acima, os personagens que

[32] O código completo da máquina de estados finitos dos soldados de Incidente em Varginha é apresentado no apêndice 2.

sobreviviam eram selecionados como base para a próxima geração. Isso resultou em comportamento extremamente adaptado, mas inútil: os personagens evitavam o confronto a qualquer custo. Tão logo entravam na linha de visão do jogador, os inimigos corriam em velocidade extrema até que ficassem protegidos por algum obstáculo, como uma coluna ou parede, e então paravam novamente. Ao serem encurralados, eles corriam em ziguezague, colidindo contra paredes e, eventualmente, escapavam ao jogador. A velocidade desses movimentos era tão elevada que tornava praticamente impossível qualquer tentativa de combater esses personagens que agiam como baratas velocíssimas. O comportamento resultante, embora inapropriado, satisfazia plenamente o quesito sobrevivência. Era necessária uma nova função de avaliação.

Uma segunda tentativa foi feita com a seguinte função de avaliação: maximizar tanto a sobrevivência como os danos infringidos ao jogador. Após várias dezenas de gerações, resultaram personagens que se escondiam quase como os anteriores, mas com uma diferença fundamental: enquanto corriam para um abrigo, viravam-se repentinamente e acertavam o jogador de modo quase infalível. A variável que controlava a precisão do tiro foi alterada no processo, e isso resultava em tiros certeiros, mesmo em movimento. Novamente, a função de avaliação estava completamente atendida, mas o resultado não era o esperado.

Funções mais complexas foram então desenvolvidas, enquanto determinados parâmetros, como velocidade máxima e precisão, foram retirados do processo evolutivo. Uma função que apresentou resultados interessantes foi a maximização, não do período de sobrevivência de um único personagem, mas da sobrevivência de um grupo de personagens. Daí resultaram comportamentos de grupo relativamente sofisticados, especialmente perceptíveis nos personagens *Drakalis*. Dependendo apenas de dentes e garras, sem armas para agir a distância, os *Drakalis* desenvolveram surpreendentes táticas de caça em grupo. Frequentemente, um ou dois elementos pareciam fugir timidamente do jogador, enquanto conduziam-no para o restante do grupo, que permanecia escondido. O jogador via-se então cercado, sem grandes possibilidades de defesa.

Figura 2.12. *Incidente em Varginha*: *Drakalis* atacando em grupo.

Apesar dos sucessos ocasionais, o “Tanque” apresentava um problema sério: ele evoluía personagens quase invencíveis. Obter personagens extremamente adaptados foi um excelente resultado de pesquisa, mas inimigos assim tornariam o game desinteressante. Gerar personagens mais verossímeis, por outro lado, exigiria funções de avaliação relativamente complexas e, possivelmente, menos objetivas. A solução adotada em 1997 foi, simplesmente, parar o processo evolutivo antes que a adaptação à função de avaliação fosse excessiva, limitando o número de gerações de personagens. As variáveis geradas eram, então, arredondadas para valores com um dígito significativo (por exemplo: 0.372 tornava-se 0.4), o que também reduzia a eficiência da adaptação.

2.7. Evolving Virtual Creatures

A concepção do “Tanque” baseou-se no trabalho *Evolving Virtual Creatures*, de Karl Sims (SIMS, 1994). Nele, criaturas artificiais apresentam comportamentos emergentes em ambiente simulado. Tanto a morfologia das criaturas como suas redes neurais, responsáveis pelo acionamento da musculatura, são geradas automaticamente por algoritmos genéticos. O corpo das criaturas é formado por blocos unidos por articulações, sobre as quais atuam músculos virtuais, e sensores. O número de blocos, suas posições relativas e dimensões, as contrações da musculatura e suas reações aos dados coletados pelos sensores são evoluídos por

algoritmos genéticos. Diferentes funções de avaliação direcionam a evolução das criaturas a comportamentos específicos, como nadar, andar, saltar ou perseguir um determinado ponto no espaço 3D. Por exemplo, a função de avaliação para o comportamento “nadar” examina a velocidade da criatura na simulação de líquido. As criaturas que mais se deslocam em determinado período de tempo são selecionadas e seus códigos genéticos servem de base para uma nova geração de criaturas, através de hibridização, mutação ou clonagem. Esse processo é então repetido por várias gerações, resultando em criaturas que se movimentam com bastante eficiência.

Karl Sims implementou seu programa num computador Connection Machine CM-5. O resultado final, apresentado em vídeo[33] de 1994, mostra criaturas que se movimentam de forma orgânica: algumas delas nadam agitando uma cauda, outras fazem uso de diversas nadadeiras, outras ainda oscilam o corpo em movimento senoidal, como cobras. As funções de avaliação para os outros comportamentos também geraram resultados interessantes. Determinada criatura anda sem pernas, apenas agitando um apêndice à sua frente que, por inércia, desloca o corpo principal em movimentos oscilatórios, numa estratégia similar à utilizada para deslocar móveis pesados sem a ajuda de rodas.

[33] Disponível em http://www.archive.org/details/sims_evolved_virtual_creatures_1994, em 14/09/2011.

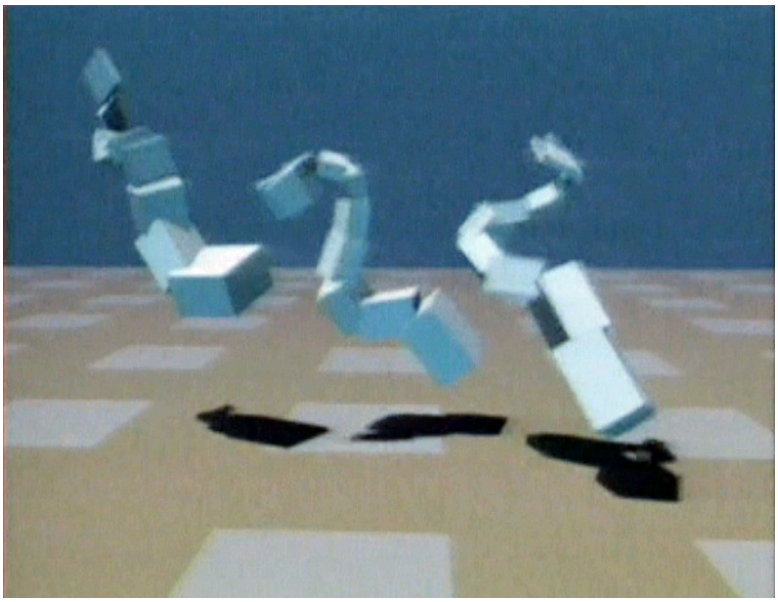


Figura 2.13. Frame do vídeo *Evolved Virtual Creatures*, de Karl Sims, 1994.

As criaturas não foram planejadas no sentido tradicional. Morfologias e comportamentos evoluíram para atender às funções de avaliação. É importante entender o que isso significa. As regras que definem o movimento foram vinculadas a uma estrutura genética, que nada mais faz além de permitir uma ampla gama de combinações de regras diferentes – regras cegas, sem objetivos pré-definidos e, em sua imensa maioria, inúteis. Ao programador coube definir esse código genético (que, por sua vez, define as regras) e a função de avaliação. Regras aleatórias e suas respectivas sequências de movimentos evoluíram em várias gerações, através de hibridização e mutação do código genético, selecionadas pela função de avaliação específica. E, nesse processo, soluções eficientes emergiram. O mesmo código genético básico resultou em regras adaptadas para nadar, andar, saltar e perseguir, dependendo apenas da função escolhida – que avaliou, respectivamente, a velocidade no líquido, a velocidade no solo, a altura de salto ou o tempo para atingir uma distância mínima do objetivo.

A aplicação de conceitos de emergência e evolução no desenvolvimento de softwares apresenta-se como possível solução para a criação de comportamentos complexos baseados em regras. Enquanto o genoma digital permite ampla gama de regras diferentes, a função de avaliação propicia ao algoritmo genético os meios para selecionar as regras que gerem comportamentos mais eficientes. Esse laço cibernético, criado pelo algoritmo entre o comportamento final e o genoma das

regras, elimina a necessidade de se programar as próprias regras. Nesse sentido, a codificação genética pode ser considerada um meta-programa. Ela define todo um conjunto de possíveis programas e cabe ao algoritmo genético selecionar exemplares bem sucedidos. As técnicas de hibridização e mutação permitem a seleção dos melhores candidatos sem que seja necessário examinar todas as possibilidades da codificação genética.

2.8. Confrontando o programador

Vimos que um programa pode gerar resultados imprevisíveis. Que pode também ser escrito para evoluir, desenvolvendo automaticamente estratégias para atingir objetivos, de forma independente da capacidade de seu programador. Os exemplos anteriores respondem às perguntas do início deste capítulo: uma sequência fixa de instruções pode, sim, gerar resultados não previstos e coerentes com objetivos pré-determinados.

Artigos de Otto Rössler sugerem ainda outra pergunta, mais ousada. Rössler é considerado o fundador da *endofísica*, a “física de dentro”, que estuda a influência do observador, enquanto parte integrante do universo, sobre sistemas ou fenômenos observados. Trata-se de uma linha de raciocínio diferente da utilizada na física mais tradicional, que normalmente isola observador e sistema observado. A pergunta sugerida por Rössler é a seguinte: *pode um programa forçar o programador a responder*? (RÖSSLER, 1998)

O contexto em que essa pergunta é levantada difere do deste trabalho, mas há pontos em comum. Rössler aponta uma inconsistência fundamental entre o pensamento racionalista, dominante na área cientifica, e a mecânica quântica, arcabouço teórico bem sucedido da física desde os anos 1920. Em 1927, Wolfgang Pauli propôs o princípio da chance primária que, de forma muito resumida, descreve a impossibilidade de se prever eventos em nível quântico. Um "click" de um contador Geiger, por exemplo, não pode ser previsto de modo racional, apenas a probabilidade de ele acontecer em determinado período de tempo pode ser calculada. Baseada numa interpretação da mecânica quântica sugerida originalmente por Hugh Everett, em 1957 (ver capítulo 3), a endofísica seria então

uma maneira de conciliar aparentes inconsistências entre a mecânica quântica e o pensamento racionalista. Seria possível explicar porque o universo se comporta desse modo em nível quântico se incluirmos o observador no mesmo nível que o sistema observado. Rössler propõe então a programação de um universo separado, em computador, perfeitamente simétrico em suas leis fundamentais, que englobaria tanto eventos como observadores, e que poderia ser estudado externamente pelo seu programador, para testar as propostas da endofísica. A pergunta está inserida nesse contexto complexo, que pode lembrar filmes de ficção como *Matrix* ou *13º Andar*:

> "O titereiro está segurando os barbantes." Tal pensamento religioso é necessário para entender a física. Universos low-level artificiais em um computador podem ser usados para se ver se os habitantes podem descobrir o segredo e começar a confrontar o programador: O programa pode forçar o programador a responder? (RÖSSLER, 2006, p. 164)[34].

Rössler argumenta que sim: criaturas programadas podem, analisando seu universo, descobrir que foram programadas e confrontar seu programador. E mais: experiências como as que tais criaturas poderiam desenvolver para testar a hipótese do programador seriam aplicáveis no nosso universo. Por exemplo, seria possível alterar o valor percebido de constantes fundamentais da física, como a velocidade da luz ou a constante de Plank, através de ajustes extremamente precisos da temperatura do cérebro de um observador humano. Segundo Rössler (2006), o universo como um todo seria forçado a responder, alterando tudo o que dependesse de tais constantes desde o início dos tempos – inclusive a memória de outras pessoas e livros que tivessem tais valores publicados. Dessa forma, apenas o observador que fez parte da experiência saberia que as constantes haviam mudado. Pode parecer estranho, mas essa proposta de experimento endofísico está de acordo com a interpretação de Everett da mecânica quântica (RÖSSLER, 2006). Voltaremos a possíveis implicações dessa experiência no próximo capítulo.

---

[34] "The puppeteer is holding the strings": such religious thinking is necessary to understand physics. Artificial lower-level universes in a computer can be used to see whether the inhabitants can get behind the secret and start confronting the programmer: Can a program force the programmer to reply? (RÖSSLER, 2006, pg. 164)

Em 2006, durante a concepção e os preparativos para a exposição e o simpósio *Emoção Art.ficial 3.0 – Interface Cibernética* (Itaú Cultural, 2006), tive a oportunidade de conhecer Otto Rössler. Durante sua palestra, ele declarou:

> Pode um programa forçar o programador a responder? Esta é a questão de causa. E a resposta é sim! É muito surpreendente (Otto Rossler, Simpósio Emoção Art.ficial 3.0, Itaú Cultural, São Paulo, 20 de julho de 2006, tradução nossa)[35].

As criaturas artificiais de Rössler precisariam ser conscientes para propor e testar hipóteses. Essas criaturas conscientes precisariam, ainda, evoluir de uma programação básica, que governasse o comportamento rigorosamente simétrico de partículas fundamentais artificiais, evoluindo com uma mecânica talvez similar à que vimos sobre algoritmos genéticos, mas sem dúvida exigindo capacidades de processamento e memória difíceis de se imaginar nos dias de hoje. Se nossa tecnologia atual não permite a programação de seres conscientes como esses, é de se supor que a resposta afirmativa de Rössler permanecesse sem comprovação. Entretanto, algumas horas antes, nessa mesma noite de 20 de julho de 2006 e no piso imediatamente superior ao da Sala Itaú Cultural (onde Rössler proferiria sua palestra), uma obra de Jon McCormack respondia a uma pergunta mais simples, mas similar: Pode um programa forçar *uma pessoa* a responder?

2.9. Eden

A obra *Eden* (Jon McCormack, 2000) apresenta um exemplo particularmente interessante de emergência dirigida. As criaturas de *Eden*, de formato circular, evoluem num ambiente virtual e competem por recursos limitados, podendo “morrer” quando gastam mais energia do que a que conseguem obter, ou após cumprir um período máximo de vida. Elas obtêm energia alimentando-se da biomassa, matéria de coloração azul clara (figura abaixo). No mundo de *Eden*, esse alimento estabelece a ligação entre o real e o virtual, uma vez que a biomassa cresce nos locais do ambiente que correspondem a posições reais ocupadas por visitantes humanos em frente às telas. Assume-se que os visitantes ficarão mais tempo

---

[35] Can a program force the programmer to reply? This is the cause question. And the answer is yes! It is very surprising (Otto Rossler, simpósio emoção art.ficial 3.0, Itaú Cultural, São Paulo, 20 de julho de 2006)

parados em frente a porções das telas que lhes agradem. Nesses locais, a produção de alimento será maior. Caso a obra permaneça sem visitação por muito tempo, as criaturas tendem a morrer de fome.

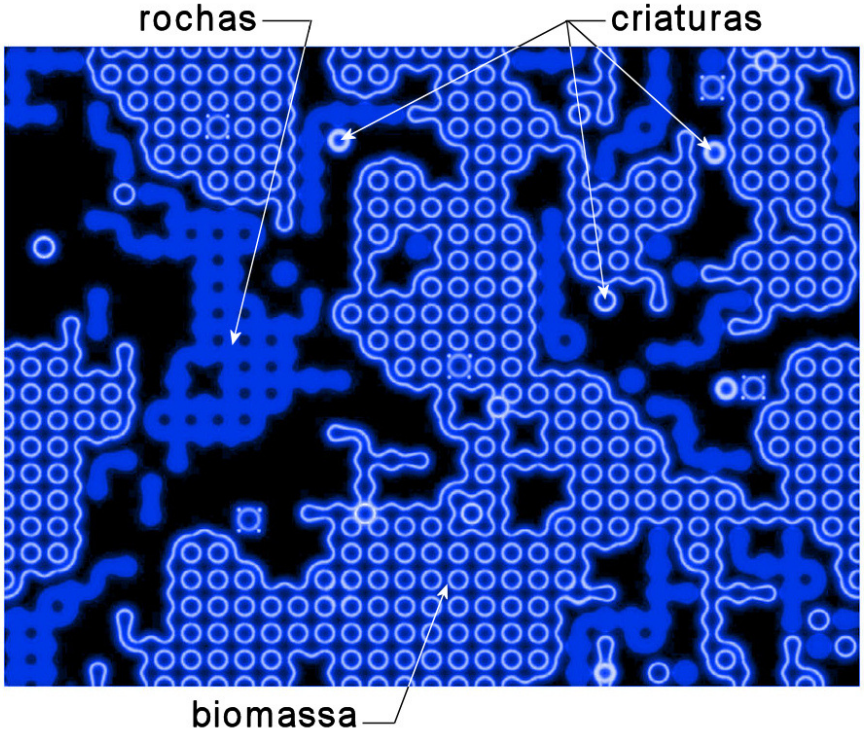


Figura 2.14. Ecossistema virtual de *Eden* (adaptado sobre imagem de Jon McCormack).

As criaturas monitoram o ambiente ao seu redor, processam informações e agem. O monitoramento inclui um sistema simples de visão, um detector de alimento e uma audição relativamente sofisticada. As ações incluem movimentar-se, emitir sons (cantar), alimentar-se e acasalar-se. O processamento ocorre através da aplicação de um conjunto limitado de regras simples, aleatórias no início, a partir do qual o software da criatura seleciona a melhor ação para as condições correntes. Com base em experiências anteriores, o software avalia como melhores regras aquelas que mais contribuíram para a sobrevivência da criatura. Quando duas criaturas geram uma terceira através da ação de acasalamento, um novo conjunto de regras é definido como uma mistura das regras mais bem sucedidas dos conjuntos dos pais. Há também mutações, variações aleatórias no código que podem gerar novas regras. Desse modo, novas regras e novos conjuntos de regras emergem e aprimoram-se, ciclicamente, durante a evolução das criaturas.

Eden emprega algoritmos genéticos de forma um tanto diferente do que vimos até aqui, pois não há uma função de avaliação pré-definida pelo programador. No lugar de explicitar alguma característica esperada das criaturas, McCormack construiu um ecossistema virtual que, a exemplo de ecossistemas reais, tende a estimular sobrevivência e adaptação em termos de conjunto, não apenas individualmente. A adaptação ocorre com base na interação entre diferentes criaturas com seu ambiente, bem como na interação entre seres humanos com o ambiente das criaturas através da geração de alimento sob forma de biomassa.

> Os ecossistemas afastam-se de abordagens anteriores no sentido de que consideram *o próprio meio* o sistema em que a evolução e a adaptação ocorrem. Isso contrasta com a visualização ou sonorização de um processo, que pode não ter conexão direta com as metáforas e ideias incorporadas à obra de arte. A atração de obras de arte ecossistêmicas reside na revelação de suas mudanças e dinâmicas complexas, o que chamei de "o sublime computacional" (MCCORMACK, 2010, p. 59, grifos do autor)[36].

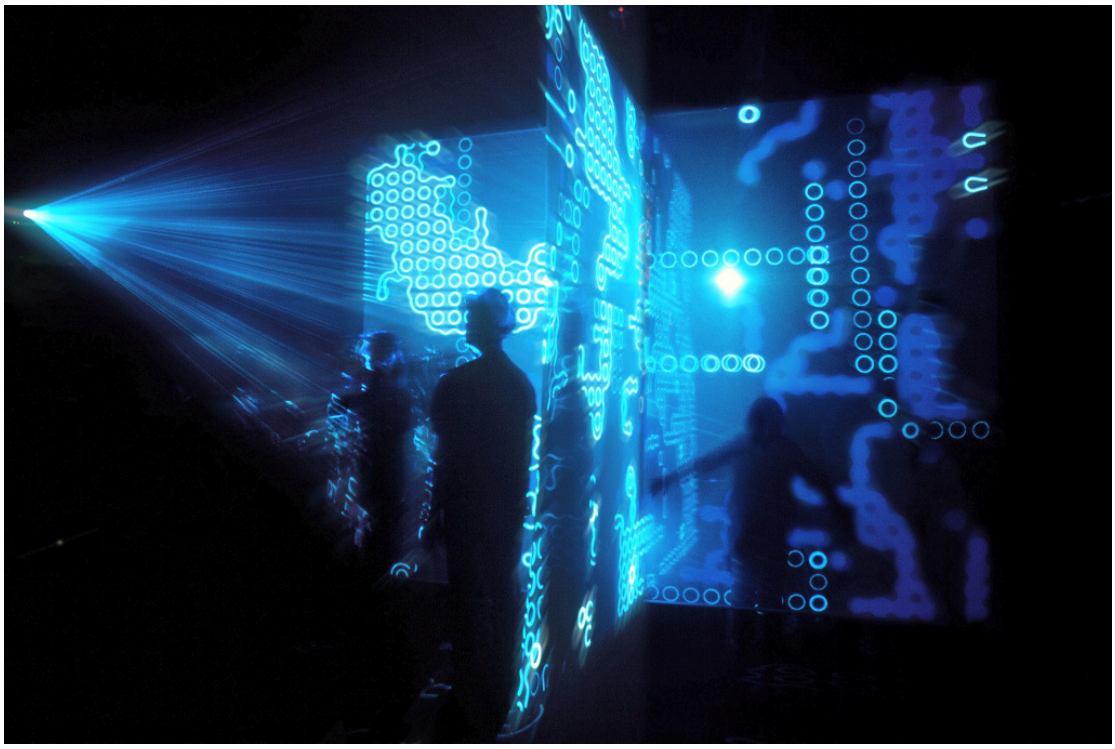


Figura 2.15. *Eden* (imagem: Jon McCormack).

[36] Ecosystems depart from previous approaches in that they consider *the medium itself* the system within which evolution and adaptation occurs. This is in contrast to the visualization or sonification of a process, which may lack a direct connection to the metaphors and ideas embodied in the artwork. Ecosystemic artworks appeal through their revealing of complex dynamics and changes, in a similar way that we find interest or inspiration in complex natural systems, what I have termed "the computational sublime" (MCCORMACK, 2010, p. 59).

As criaturas mais eficientes no uso de energia e na obtenção de alimento terão maiores chances de passar seu código genético para descendentes. Após várias gerações, elas tendem a adaptar-se criando e selecionando regras surpreendentemente eficazes.

Acompanhando a obra durante sua montagem em 2006 no Itaú Cultural, pude presenciar diversos comportamentos interessantes. No início de cada dia, quando o trabalho era ligado, notava-se pouca adaptação das criaturas tanto visualmente como pelos sons emitidos por elas. Nesse estágio, era comum identificar-se criaturas que batiam repetidamente contra pedras do ambiente virtual – a cada choque, uma parcela de energia era desperdiçada, até que a criatura acabasse morrendo. Outras praticamente não se moviam, gastando pouquíssima energia, mas dependendo inteiramente da aproximação de pessoas que gerassem biomassa para sua alimentação. Com o passar das horas, era normalmente possível classificar as criaturas entre “presas” e “predadoras”. Algumas atacavam (as criaturas também podem se alimentar de suas semelhantes), outras fugiam, especialmente em situações ocasionais de baixa biomassa no ambiente. De modo geral, ao final do dia, o ecossistema apresentava movimentos e sonorização muito mais fluidos, enquanto as criaturas pareciam atarefadas em cumprir, de forma eficiente, funções de suas vidas virtuais. Mas o comportamento mais surpreendente que pude acompanhar dizia respeito ao uso que as criaturas faziam dos sons.

A geração de sons em *Eden* está intimamente ligada ao processo de evolução, e os usos que as criaturas fazem desses sons depende dos benefícios que eles possam trazer a sua sobrevivência durante esse processo. Chamados de aviso, gritos de acasalamento ou indicações da presença de comida estão entre os usos que surgem durante a evolução do ecossistema (MCCORMACK, 2010). Entretanto, em algumas linhas evolutivas, as criaturas desenvolvem o que poderia ser chamado de simbiose com os visitantes da exposição: através de sons elas atraem a atenção dos visitantes humanos que, sem o perceberem e apenas com suas presenças, aumentam a biomassa do ambiente. Mais interessante, as criaturas às vezes parecem “gritar” após vários visitantes abandonarem o espaço simultaneamente, como que para trazê-los de volta, ainda que os sensores de presença do público não tenham ligação direta com o software de seleção de regras (as criaturas podem

apenas sentir, localmente, a variação indireta da biomassa). E a estratégia funciona: várias vezes presenciei pessoas voltando à frente das telas após ouvir os gritos das criaturas.

Sem que seja necessário qualquer tipo de consciência artificial, o ecossistema de *Eden* possibilita a evolução de criaturas muito simples, mas que ainda assim são capazes de manipular seres humanos, forçando-os a responder às suas necessidades.

2.10. Criaturas deterministas

A programação de criaturas que se comportam de forma interessante faz parte do desenvolvimento de games há décadas. Ferramentas para se alcançar esse objetivo não incluem nada além de sequências de instruções que consigam atingir certo equilíbrio entre caos e previsibilidade, na grande maioria dos casos, e adaptabilidade, conforme mostrado nos últimos exemplos. Mas existe um aspecto desse tipo de produção que talvez seja ainda menos compreendido que os algoritmos utilizados: a perspectiva do programador.

Durante o trabalho de programação de comportamento dos personagens de *Incidente em Varginha*, percebi-me muitas vezes tendo que emprestar verbos de ações humanas para descrever o comportamento dessas criaturas. Dizer que um *Drakalis* me "conduziu" para uma emboscada, por exemplo, era algo comum. Antes disso, lembro-me de comemorar o dia em que essas criaturas "aprenderam" a caçar em grupos de três ou mais. Ou de dizer a Odair Gaspar, que desenvolveu o game comigo: "olha só! o *Drakalis* está 'fingindo' que não me vê, só para que eu chegue mais perto!" É evidente que um personagem programado através de máquina de estados finitos, mesmo que otimizada por algoritmos genéticos, não "conduz", não "aprende" e não "finge". Esses comportamentos são resultados de sequências de instruções, adaptações do próprio programa ao que se exige dele. Mas a sensação de presenciar um programa que criei chegar a resultados que eu próprio não imaginaria é difícil de descrever. Ter sido "enganado" por alguns *Drakalis* tem muito valor para mim. Foram momentos em que o programa parecia estar, por falta de expressão melhor, “me dizendo alguma coisa”. Instintivamente, eu respondi às

“provocações” dos *Drakalis* e caí nas emboscadas, antes mesmo de perceber que a ação deles era um resultado emergente do código que criei.

Perguntei a Jon McCormack se ele havia passado por situações similares com as criaturas de *Eden*. Não surpreende que ele use verbos de ações humanas para descrever o comportamento de suas criaturas:

> Certamente elas me fizeram modificar meu comportamento com base em pressuposições sobre o que elas estavam fazendo. Por vezes o que eu pensava que elas estivessem fazendo mostrava-se incorreto, então nesse sentido eu fui "enganado". Certamente penso que o comportamento delas me surpreendeu (inventando coisas sobre as quais eu não teria pensado), isso me fez pensar de modo diferente sobre como eu encaro a programação (Jon McCormack, email pessoal de 14/12/2011, tradução nossa)[37].

Mesmo com as limitações atuais de nossa tecnologia, um programa pode forçar o programador a responder. Os exemplos acima não têm valor para confirmar previsões da endofísica, pois as criaturas de *Eden* e de *Incidente em Varginha* absolutamente não se dão conta da existência de um programador. Mas, para cumprir seus objetivos programados de sobrevivência, elas foram capazes de forçar seus programadores a responder – seja para obter alimento, seja para garantir a sobrevivência do grupo. A pergunta de Otto Rössler pode ser respondida com um sim.

---

[37] Certainly they have made me modify my behaviour based on assumptions about what they were doing. Sometimes what I thought they were doing turned out to be incorrect, so in that sense I was "tricked". Certainly I think their behavior has surprised me (inventing things I wouldn't have thought of), that's made me think differently about how I approach programming (Jon McCormack, email pessoal de 14/12/2011).

CAPÍTULO 3

3.1. A textura do real

O comportamento de personagens de games 3D pode ser programado de forma não previsível e adaptativa. O programa, ainda que mera sequência de instruções, não nos limita necessariamente a personagens de comportamentos previsíveis ou aleatórios. Mas há outro fator limitante que não depende da programação: o modelo matemático utilizado. Boa parte da programação de um game 3D dedica-se a converter em imagens e sons os valores numéricos que compõem o ambiente virtual. Esse conjunto de números e regras define um "modelo de realidade"[38], que rege o mundo virtual. Falamos sobre isso no capitulo 1: a simulação física baseada na mecânica clássica é o modelo de realidade predominante em games 3D atuais.

Vimos também que podemos desrespeitar as leis de Newton de modo a tornar a realidade do game mais interessante, menos previsível e mais adaptável. Vejo isso como indicador de que o modelo da mecânica clássica talvez não seja ideal para uma mídia não linear como a dos games 3D, especialmente no que diz respeito ao comportamento de personagens. Conceitos como o de trajetória, de posições definidas no espaço e no tempo, limitam a realidade de um game 3D ao trazer o foco para uma verdade matemática que independeria das interações entre o jogador e os objetos do game. Um modelo mais apropriado deveria construir a realidade a partir dessas interações, e não apenas monitorá-las para obter dados de entrada e aplicá-los a equações de movimento.

Neste capitulo analisaremos um modelo de realidade diferente, já utilizado com sucesso na física, e, com base nele, faremos propostas para a criação de um novo modelo matemático para games 3D. Vale dizer que, no campo da física, o modelo

[38] A expressão “modelo de realidade” será usada neste texto tanto para se referir a modelos desenvolvidos pela física para estudar e entender a realidade, como para modelos matemáticos que regem games 3D, determinando algo parecido com a realidade nesses ambientes virtuais. A aproximação desses dois significados é importante para a compreensão de modelos matemático de games 3D com base na física.

escolhido para essa análise suplantou o da mecânica clássica já nas primeiras décadas do século vinte. Trata-se da mecânica quântica.

3.1.1. Ondas e partículas

Proposta originalmente por Thomas Young no início do século 19, a "experiência das duas fendas" foi projetada para testar a natureza da luz. Acreditava-se que essa natureza estivesse limitada a duas possibilidades: a luz poderia ser corpuscular – ou seja, formada por pequenas partículas individuais – ou ondulatória, efeito visível de oscilações de energia num meio ainda desconhecido.

A experiência de Young tinha uma configuração bastante simples: um feixe de luz era projetado em uma tela após atravessar duas fendas estreitas. Caso a luz fosse corpuscular, o resultado deveria ser uma distribuição uniforme, consistente com a simples soma dos resultados individuais para cada fenda, conforme ilustrado abaixo.

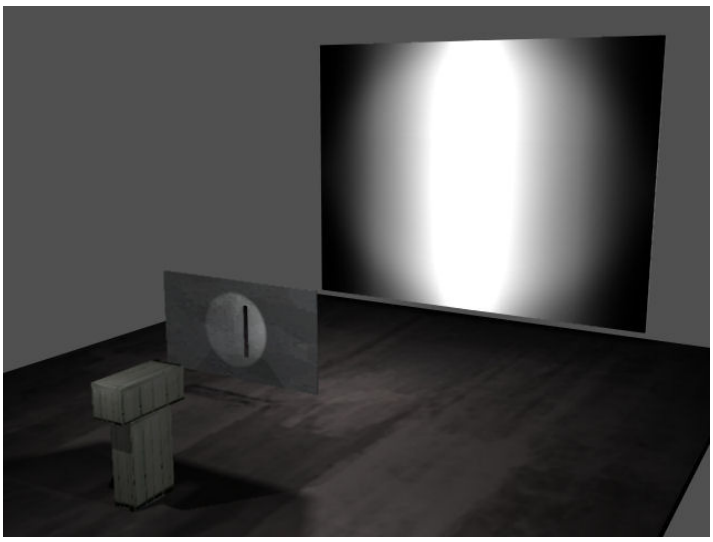


Figura 3.1. Padrão obtido ao se projetar a luz por uma única fenda. Se a luz fosse de natureza corpuscular, supunha-se que o resultado com duas fendas seria um padrão muito similar, formado pela sobreposição dos efeitos devidos a cada uma delas.

Caso a luz se comportasse de maneira ondulatória, um padrão de interferência, formado por franjas de luz e sombra, deveria ser projetado na tela. O resultado desses primeiros experimentos foi o padrão de interferência ilustrado abaixo e, por quase um século, cientistas aceitaram a experiência como evidência de que a luz não era corpuscular, mas ondulatória.

Figura 3.2. A luz que atravessa uma das fendas supostamente interfere com a que atravessa a outra, formando o padrão de franjas da figura.

O padrão de interferência apresentado pelo experimento realmente sugere uma natureza ondulatória para a luz. De fato, como aconteceria com ondas na água que atravessassem duas fendas, as variações de amplitude das duas novas frentes de onda combinam-se, conforme ilustrado abaixo à direita.

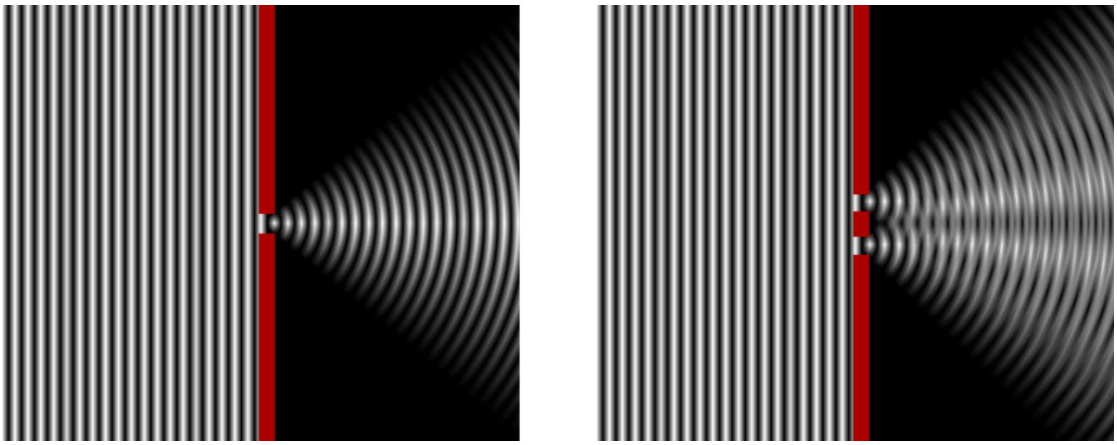


Figura 3.3. Visão superior de ondas atravessando fendas. À direita, as ondas que atravessam uma fenda interferem com as que passam pela outra, criando “franjas”.

Em alguns lugares, as duas ondas que se originam ao passar pelas fendas estão “em fase” (cristas e vales coincidem), reforçando a intensidade luminosa quando atingem a tela nas faixas mais claras. Essa é a chamada interferência construtiva. Em outros locais, uma onda anula a outra, pois as fases se opõem, gerando interferência destrutiva e resultando em faixas escuras.

É relativamente simples entender esses padrões de interferência quando pensamos em oscilações de um meio físico, como o ar ou a água. Parece natural supor que qualquer fenômeno de interferência dependesse de alguma coisa interferindo com outra, talvez moléculas de ar ou de água interagindo. No caso da luz, Isaac Newton, James Clerk Maxwell e Hendric Antoon Lorentz, entre outros, consideraram uma substância hipotética denominada *éter* como meio de propagação. O éter facilitava a compreensão dessa natureza aparentemente ondulatória da luz, mas existiam outros aspectos que não dependiam de um meio de propagação. O próprio Newton, em *Optik*s (1704), propôs que a luz fosse formada por numerosas partículas. Outras características observadas experimentalmente, como o deslocamento da luz em linha reta e seus ângulos de reflexão, podiam ser explicadas dessa forma.

Em 1905, com base em trabalho desenvolvido anteriormente por Max Plank, Albert Einstein demonstrou que a luz poderia ser compreendida como composta por pequenos "pacotes" individuais (ou *quanta)* de energia radiante, partículas sem massa que se movem exclusivamente à velocidade da luz (aproximadamente 300 mil quilômetros por segundo). Mais tarde, G. N. Lewis viria a denominá-los *fótons*, nome pelo qual são conhecidos ainda hoje. Porém, foi apenas após a publicação de dados experimentais sobre o espalhamento de raios X por Arthur H. Compton, em 1923, que o conceito de fótons passou a ser amplamente aceito pela comunidade científica (MORRISON 1990). A comprovação de que a luz era composta por partículas individuais não parecia compatível com os resultados da experiência das duas fendas. Ainda assim, a luz era vista como um caso especial pelo fato de suas partículas não possuírem massa e de se deslocarem à máxima velocidade conhecida. Talvez o fenômeno da interferência pudesse vir a ser explicado pela interação entre um grande número de fótons atravessando as fendas.

Entretanto, outra experiência, conduzida em 1927 por Clinton Davisson e Lester Germer, do Bell Labs, com elétrons – partículas com massa definida e velocidades sempre inferiores à da luz –, também apresentava padrões de interferência. Mais recentemente, a experiência das duas fendas foi recriada utilizando elétrons individuais, emitidos um a um, no lugar de fótons (Tonomura, Matsuda, Kawasaki e Ezawa, 1989). O resultado dessa experiência é ilustrado nas figuras abaixo. Os elétrons que atravessam as fendas colidem com a tela fosforescente, deixando

pontos brilhantes. No início, os pontos parecem assumir posições aleatórias (à esquerda). Entretanto, enquanto o número de colisões de elétrons com a tela aumenta, percebe-se a formação do típico padrão de franjas (à direita). Como explicar que partículas individuais, deslocando-se no vácuo, apresentassem o padrão de interferência da experiência das duas fendas?

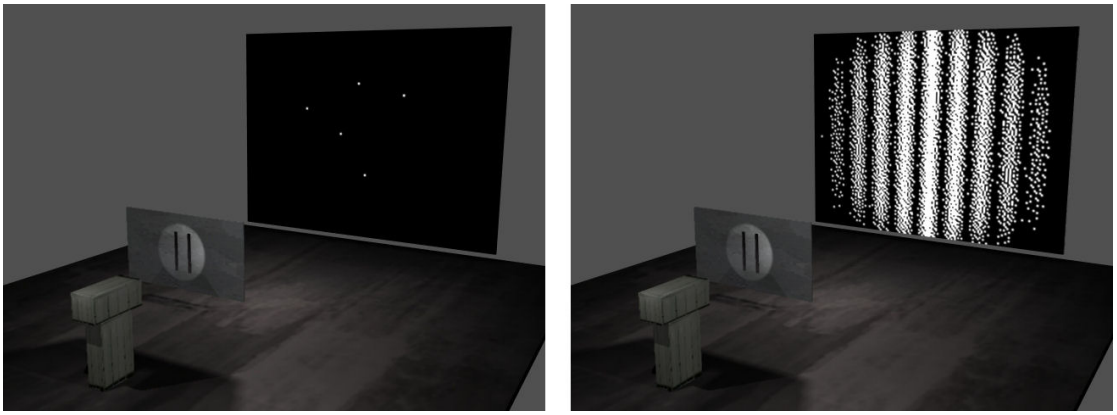


Figura 3.4. Elétrons, emitidos um a um através das duas fendas, também formam um padrão de interferência.

Tais resultados levantam questões sobre a natureza dessas partículas. Se cada elétron atravessa individualmente uma das fendas para depois ser detectado ao colidir com a tela, como pode existir algum tipo de interferência? As faixas escuras desaparecem quando apenas uma das fendas está aberta. Desse modo, como a existência de uma segunda fenda pode interferir com o resultado das colisões de elétrons com a tela se, em princípio, eles deveriam atravessar apenas uma das fendas? É como se cada partícula sofresse a influência das duas fendas durante sua trajetória. Ainda mais desconcertante é o fato de que o padrão de interferência *desaparece* quando um sensor é instalado em uma das fendas para determinar por qual delas cada elétron passou (figura abaixo). A mera medição da passagem de elétrons por uma das fendas destrói o padrão de interferências. É, no mínimo, intrigante: se soubermos por onde cada elétron passou, a interferência deixa de existir. Por quê?

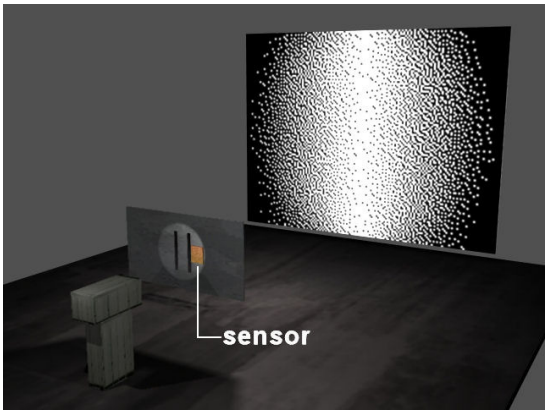


Figura 3.5. A simples introdução de um sensor para determinar por qual das fendas cada elétron passa destrói o padrão de interferência do experimento.

A experiência das duas fendas ilustra uma característica importante das partículas elementares, a *dualidade onda-partícula*. O fato de que cada elétron gere um ponto luminoso definido na tela fosforescente sugere o comportamento de partícula, com massa, posição e trajetória definidas. Entretanto, o padrão de interferência é típico do modelo clássico de ondas, que transmitem energia em “frentes” esféricas, sem posição definida no espaço. A dualidade é apenas uma maneira de expressar esse comportamento estranho – na realidade, elétrons não são nem uma coisa, nem outra:

> Assim, um elétron atômico *não* é um minúsculo "planeta" orbitando um "sol nuclear", como nos modelos do átomo de Rutherford e Bohr. Mas ele também não é uma coisa "nebulosa", espalhada sobre uma região do espaço, como é retratado em muitos textos introdutórios de física. O elétron é *alguma outra coisa*, nem partícula nem onda, mas estranhamente reminiscente de ambas (MORRISON, 1990, p. 7, tradução nossa).[39]

Tanto o modelo clássico de partículas como o de ondas falham ao descrever o comportamento de elétrons, ou de qualquer outro elemento de mesma ordem de grandeza. A compreensão dessa estranha mecânica do mundo microscópico exigiu o desenvolvimento de uma nova teoria, conhecida como *mecânica quântica*.

Antes de prosseguirmos, faz-se necessário um pequeno ajuste de vocabulário. A palavra *partícula* é utilizada regularmente em textos de física com sentidos radicalmente diferentes. Em *dualidade onda-partícula*, por exemplo, o sentido é o de

[39] Thus, an atomic electron is *not* a tiny “planet” orbiting a “nuclear sun”, as in the Rutherford and Bohr models of the atom. But neither is it a “fuzzy” thing, smeared out over a region of space, as it is portrayed in many introductory physics texts. The electron is *something else*, neither particle nor wave, but eerily reminiscent of both (MORRISON, 1990, p. 7).

partícula da mecânica clássica, ou seja, aquela governada pelas leis da mecânica de Newton, com massa, posições definidas, trajetória, etc. Por outro lado, quando elétrons ou fótons são denominados *partículas*, o sentido é o de partícula quântica, elementos que não obedecem à mecânica clássica. Para evitar essa dualidade desnecessária de sentidos, utilizaremos neste trabalho a expressão *partículas quânticas*, ou a palavra *quanta*, para nos referir a partículas não newtonianas. Nos textos de outros autores citados adiante, entretanto, *partícula* refere-se sempre a partículas quânticas.

3.1.2. Natureza incerta

Quando pensamos em elétrons sendo disparados um a um, atravessando a barreira com duas fendas e colidindo com uma tela fosforescente, é praticamente inevitável assumir que cada elétron siga uma trajetória bem definida, que se inicia no emissor, atravessa uma das duas fendas e finalmente colide com a tela num ponto determinado. Essa é a visão clássica de trajetórias de partículas: se determinarmos a exata posição inicial (no caso, a do próprio emissor de elétrons), a massa (também conhecida) e a velocidade com que cada partícula é emitida, poderíamos calcular novas posições em qualquer tempo futuro, prevendo não apenas por qual fenda o elétron passaria, mas também o ponto exato de sua colisão com a tela fosforescente. Esse raciocínio é válido para objetos de dimensões razoáveis, como, por exemplo, bolas de futebol ou esferas de rolamento sendo arremessadas através de duas fendas. Para elétrons e outros quanta, entretanto, as lei do movimento de Newton não se aplicam.

Em 1927, o físico Werner Heisenberg formulou um enunciado impondo restrições sobre a precisão com que se pode efetuar medições simultâneas entre pares de *observáveis* (propriedades do estado do sistema que podem ser determinadas por uma sequência de operações físicas). Um exemplo de par de observáveis é a posição $(x(t_0))$ e o momento linear[40] $(p_x(t_0))$ de um elétron no instante de sua emissão (t_0) na experiência das duas fendas. Segundo o princípio da incerteza de

[40] De maneira simples, podemos dizer que momento linear combina massa e velocidade numa única variável. Na mecânica clássica, o momento linear é o produto da massa pela velocidade de um corpo.

Heisenberg, esse par de observáveis precisa necessariamente satisfazer a seguinte condição:

$$\Delta x(t_0)\Delta p_x(t_0) \geq h/4\pi$$

Onde h é a constante de Plank:

$$h \equiv 6.626 \times 10^{-34} J - sec$$

Em palavras, o produto das incertezas de posição e de momento linear (representadas por $\Delta x(t_0)$ e $\Delta p_x(t_0)$) nunca pode ser zero. Isso equivale a dizer que nenhuma das incertezas pode ser zero (caso contrário, o produto também seria zero). Além disso, a redução da incerteza na posição de um elétron, por exemplo, necessariamente aumenta a incerteza sobre seu momento. Conhecer com maior precisão o momento linear de um elétron, por outro lado, significa ignorar sua posição. Vale notar que o valor limite para o produto dessas duas quantidades ($h/4\pi$) é um número muito pequeno. Por essa razão, o princípio da incerteza não se faz notar no mundo macroscópico, embora continue válido. Pode-se medir a posição de um planeta, por exemplo, com precisão de poucos metros, e ainda assim determinar-se seu momento de forma relativamente precisa, respeitando o princípio de Heisenberg. Uma incerteza de poucos metros na medição da posição de um planeta, de alguns quilômetros por hora na determinação de sua velocidade ou de toneladas no cálculo de sua massa são suficientes para satisfazer o princípio, mas ainda conferem números relativamente precisos para essas quantidades. O problema real aparece em escalas diminutas, com valores muito pequenos para posição e momento linear.

Mas há uma implicação ainda mais crítica no princípio da incerteza de Heisenberg. A teoria quântica revela que a limitação que o princípio impõe sobre nossa capacidade de conhecer simultaneamente a posição e o momento de uma partícula está *implícita na natureza*. A incerteza dessas medições não resulta de nossos equipamentos imperfeitos, nem da técnica utilizada nessas experiências. A mecânica quântica *prova* que uma partícula não pode apresentar, simultaneamente,

valores precisos para x e para p_x (MORISSON, 1990). Ou seja, o universo é intrinsecamente incerto:

> De acordo com a física quântica, não importa quanta informação obtenhamos nem quão poderosas sejam nossas habilidades de computação, os resultados de processos físicos não podem ser previstos com certeza, porque eles não são determinados com certeza. Em vez disso, dado o estado inicial de um sistema, a natureza determina seu estado futuro através de um processo que é fundamentalmente incerto. Em outras palavras, a natureza não dita o resultado de qualquer processo ou experimento, mesmo na mais simples das situações. Antes, ela permite uma série de eventualidades diferentes, cada uma com certa probabilidade de ser realizada. Parafraseando Einstein, é como se Deus jogasse dados antes de decidir o resultado de cada processo físico (HAWKING e MLODINOW, 2010, posição 680 de 2386[41], tradução nossa)[42].

Ora, se um elétron comprovadamente não pode ter posição e momento linear definidos num instante t_0, não há como prever essas quantidades para qualquer momento futuro. É exatamente nesse ponto que toda a construção matemática da mecânica clássica desaba: como esses valores não podem existir simultaneamente, o conceito de trajetória não faz sentido na mecânica quântica. Um elétron não "passa" exclusivamente por uma das duas fendas. De alguma forma, *ele passa pelas duas*. Nesse processo, forma-se o padrão de interferência apresentado pela experiência.

Em sua dissertação de PhD de 1923, Prince Louis Victor Pierre Raymond de Broglie propôs que partículas de matéria (como os elétrons) estariam associadas a ondas que guiariam seu comportamento quântico. De Broglie chamou-as de *ondas de matéria*, desenvolvendo equações que permitiam atribuir características normalmente associadas a radiações eletromagnéticas (à luz, por exemplo), como frequência e comprimento de onda, a partículas de massa diferente de zero, como

---

[41] A indicação "680 de 2386" refere-se à posição do texto citado na edição Kindle, que não posui números de páginas

[42] According to quantum physics, no matter how much information we obtain or how powerful our computing abilities, the outcomes of physical processes cannot be predicted with certainty because they are not determined with certainty. Instead, given the initial state of a system, nature determines its future state through a process that is fundamentally uncertain. In other words, nature does not dictate the outcome of any process or experiment, even in the simplest of situations. Rather, it allows a number of different eventualities, each with a certain likelihood of being realized. It is, to paraphrase Einstein, as if God throws the dice before deciding the result of every physical process (HAWKING e MLODINOW, 2010, posição 680 de 2386).

elétrons. Esse foi um passo historicamente importante para a compreensão da dualidade onda-partícula.

Ainda sobre o problema da dualidade onda-partícula, Niels Bohr propôs o *princípio da complementaridade* em 1927. Segundo esse princípio, os comportamentos de onda e de partícula não são contraditórios, mas mutuamente exclusivos e complementares. Segundo Bohr, aspectos mutuamente exclusivos das partículas quânticas não se revelam numa mesma observação – por exemplo: não é possível observar um elétron comportando-se como onda e como partícula simultaneamente. Em outras palavras, o princípio da complementaridade propõem que o problema da dualidade não existe. Por mais óbvio que possa parecer esse artifício de solução, nenhum resultado experimental foi ainda capaz de contradizê-lo (MORISSON, 1990, p. 41).

3.1.3. Uma maneira diferente de entender a realidade

Com base no que foi exposto até aqui, faço a seguir um resumo sobre o desenvolvimento posterior da mecânica quântica, tomando como exemplos resultados da experiência das duas fendas. Detalhes e explicações matemáticas desse processo fascinante podem ser vistos em Morisson (1990, capítulos de 3 a 7).

A mecânica quântica descreve eventos observáveis. Uma observação, nesse sentido, nunca é passiva. Para localizarmos a posição de um elétron, por exemplo, é necessário que exista interação desse elétron com outro elemento (normalmente um fóton). Essa interação altera o estado do elétron observado. Entre observações/interações, não faz sentido perguntar onde esse elétron realmente estaria, ou qual seria o seu momento linear. Podemos entender que, nas interações para medir sua posição, elétrons se comportam como *partículas*, enquanto que em sua propagação entre interações eles se comportam como *ondas*.

Outro ponto importante é o caráter não determinista do comportamento das partículas quânticas. Como vimos, segundo o princípio da incerteza de Heisenberg, os quanta não podem apresentar posição e momento linear definidos simultaneamente. Quando elétrons são emitidos um a um na experiência das duas

fendas, o padrão que se forma na tela fosforescente parece totalmente aleatório em princípio. Apenas depois que um número considerável de elétrons colide com a tela é que o padrão de interferência começa a ser reconhecível. Esse comportamento probabilístico, em que o padrão é reconhecível após um grande número de eventos, também foi determinante na formulação de um modelo matemático para a mecânica quântica.

Do desenvolvimento da proposta original das ondas de matéria (por de Broglie), um primeiro postulado generalizava o conceito para qualquer variável que pudesse ser medida experimentalmente:

Primeiro postulado da mecânica quântica: todo o estado fisicamente realizável de um sistema é descrito, na mecânica quântica, por uma função de onda Ψ que contém toda a informação fisicamente acessível sobre o sistema naquele estado.

Ou seja: qualquer informação que pode ser observada de partículas quânticas é definida por uma *função de onda* (chamada também de *função de estado*). Desse modo, funções de onda são *descritores de estado* na mecânica quântica – algo equivalente em importância às trajetórias na mecânica clássica.

Entretanto, não há consenso sobre o que as funções de onda realmente representam. Essas funções não descrevem ondas no sentido clássico, como ondas na água ou no ar, nem mesmo ondas eletromagnéticas. Ondas “clássicas” transportam energia, mas funções de onda não o fazem. Na mecânica quântica, as funções de onda resultam em números complexos, que têm uma parte real e outra imaginária (ver capítulo 2). Para todos os efeitos, números complexos são abstrações matemáticas: não se pode medir um número complexo numa experiência física! Funções de onda estão abertas a diversas interpretações diferentes. Uma das primeiras, que ainda tem boa aceitação entre os físicos, é a *Interpretação de Copenhague*, fundamentada na interpretação de Max Born. Expressa em termos de

posição da partícula[43], essa interpretação pode ser enunciada no seguinte postulado:

Segundo postulado da mecânica quântica: se um sistema está num estado quântico representado pela função de onda Ψ, então $P\,dv = |\Psi|^2 dv$ é a probabilidade de que, numa medição de posição no tempo t, a partícula quântica seja detectada no volume infinitesimal dv.

A função de onda resulta em valores complexos para determinada posição no espaço e no tempo. Tais valores possuem uma parte real e outra imaginária. Como dissemos acima, não se pode medir números imaginários em laboratório. Entretanto, o quadrado do módulo de um número complexo (no caso $|\Psi|^2$) é necessariamente um número real – que pode, portanto, ser confirmado em laboratório. A Interpretação de Copenhague propõe que a probabilidade de se encontrar uma partícula cuja posição seja representada pela função de onda Ψ no volume de espaço dv depende do quadrado do módulo da função de onda. A probabilidade P é um valor real que pode ser comprovado experimentalmente.

3.1.4. Modelo determinista para realidade não determinista

Mas ainda existe uma etapa a ser vencida neste breve resumo de equações matemáticas. Se funções de onda guardam alguma similaridade com os descritores de estado da mecânica clássica – as trajetórias –, ainda se faz necessária uma equação que descreva a evolução desses descritores de estado no tempo. Na mecânica clássica, essas seriam as equações de movimento, mas já vimos que a mecânica de Newton simplesmente não funciona para sistemas quânticos. Uma *equação quântica de movimento* foi publicada em 1926 por Erwin Schrödinger e ficou conhecida como *Equação de Schrödinger dependente do tempo*[44]:

$$\widehat{H}\Psi(x,t) = i\hbar\frac{\delta}{\delta t}\Psi(x,t)$$

[43] A posição é apenas um entre vários estados fisicamente realizáveis (o momento, por exemplo, seria outro). Usaremos a posição por simplicidade.

[44] Representada aqui apenas para a uma coordenada do espaço (x). Em três dimensões, é preciso considerar os dois outros eixos de coordenadas.

Onde i é o número imaginário ($\sqrt{-1}$),$\hbar$ é a constante de Planck dividida por 2π, enquanto que o Hamiltoniano $\widehat{H}(t)$ representa a energia total do sistema. A equação descreve completamente a evolução de estados quânticos de um sistema, para velocidades não relativistas (posteriormente, desenvolveram-se novas equações que funcionam também para velocidades próximas ou iguais à da luz, mas a equação de Schrödinger basta-nos como ilustração). Por exemplo, na experiência das duas fendas, a probabilidade de encontramos elétrons em determinada posição do espaço, no tempo t, pode ser calculada através da equação de Schrödinger. Resolvendo-se a equação para as regiões da tela fosforescente, pode-se prever com exatidão o padrão de interferência apresentado pela experiência.

Ao se resolver a Equação de Schrödinger para uma região do espaço e para um tempo qualquer, obtém-se um valor que é denominado *amplitude*, normalmente representado por ϕ, um número complexo. Pela Interpretação de Copenhague, o quadrado do módulo desse número ($|\phi|^2$) é a probabilidade, valor real e não complexo, de localizarmos o elétron nessa região do espaço e nesse tempo.

Chegamos a um ponto importante deste trabalho: *a equação de Schrödinger é determinista, mas a interpretação das probabilidades que dela decorrem, não*:

> Nós descobriremos que, na mecânica quântica, as amplitudes ϕ são soluções para uma equação totalmente determinista (a equação de Schrödinger). Conhecer ϕ em *t=0* implica seu conhecimento em todos os momentos subsequentes. A interpretação de $|\phi|^2$ como sendo a probabilidade de um evento é uma interpretação não determinista. Ela implica que o resultado de um experimento não é exatamente previsível. É especialmente notável que essa interpretação não acarrete qualquer inconsistência. A veracidade dessa afirmação foi demonstrada em análises de muitas situações particulares por Heisenberg, Bohr, Born, von Neumann e muitos outros físicos. Apesar de todas essas análises, o fato de que nenhuma inconsistência possa surgir não é totalmente óbvio. Por essa razão, a mecânica quântica apresenta-se como matéria difícil, e um tanto misteriosa, para um iniciante (FEYNMAN e HIBBS, 2010, p. 22).[45]

[45]We shall find that in quantum mechanics, the amplitudes ϕ are solutions of a completely deterministic equation (the Schrödinger equation). Knowledge of ϕ at t=0 implies its knowledge at all subsequent times. The interpretation of $|\phi|^2$ as the probability of an event is an indeterministic interpretation. It implies that the result of an experiment is not exactly predictable. It is very remarkable that this interpretation does not lead to any inconsistencies. That this is true has been amply demonstrated by analyses of many particular situations by Heisenberg, Bohr, Born, von Neumann, and many other physicists. In spite of all these analyses the fact that no inconsistency can arise is not thoroughly obvious. For this reason quantum mechanics appears as a difficult and somewhat misterious subject to a beginner (FEYNMAN, HIBBS, 2010, p. 22).

Que uma equação determinista tenha o quadrado do módulo de seu resultado interpretado como a probabilidade de um evento ocorrer, sem gerar quaisquer inconsistências entre o que é previsto e o que é observado, é uma forte evidência a favor do modelo de realidade da mecânica quântica, por mais estranho que ele seja. O porquê de o modelo funcionar e o que exatamente a matemática significa são ainda questões abertas a diversas interpretações diferentes.

3.1.5. Soma de histórias

Essa é uma característica da mecânica quântica, ela admite interpretações e até mesmo formas matemáticas diferentes, que ainda assim chegam às mesmas previsões do modelo original. Por exemplo, na experiência das duas fendas, um elétron assume uma posição definida apenas quando é observado, ao colidir com a tela. Como vimos, não se pode dizer que ele execute uma trajetória específica entre o emissor e a tela. Então, uma interpretação direta é assumir que o elétron não percorra qualquer caminho definido, que se propague como uma onda (e essa interpretação funciona corretamente). Nos anos 40 do século XX, entretanto, Richard Feynman propôs uma ideia intrigante para modelar o comportamento peculiar que os elétrons exibem na experiência das duas fendas:

> Feynman percebeu que não se tem de interpretar isso como significando que as partículas não tomam *nenhum* caminho enquanto transitam entre a fonte e a tela. Isso poderia significar que as partículas tomassem *todos* os caminho possíveis que conectam esses pontos. Isso, Feynman afirmou, é o que torna a física quântica diferente da física newtoniana. A situação nas duas fendas é relevante porque, ao invés de seguir um único caminho definido, as partículas seguem todos os caminhos, e seguem-nos simultaneamente! Isso soa como ficção científica, mas não é. Feynman formulou uma expressão matemática – a soma de histórias de Feynman – que reflete essa ideia e reproduz todas as leis da física quântica. Na teoria de Feynman a matemática e a imagem física são diferentes da formulação original da física quântica, mas as previsões são as mesmas (HAWKING e MLODINOW, 2010, POSIÇÃO 726 de 2386, grifos do autor, tradução nossa)[46].

---

[46] Feynman realized one does not have to interpret that to mean that particles take *no* path as they travel between source and screen. It could mean instead that particles take *every* possible path connecting those points. This, Feynman asserted, is what makes quantum physics different from Newtonian physics. The situation at both slits matters because, rather than following a single definite path, particles take every path, and they take them all simultaneously! That sounds like science fiction, but it isn't. Feynman formulated a mathematical expression – the Feynman sum over histories – that reflects this idea and reproduces all the laws of quantum

Surpreendentemente, em termos de resultados, é indiferente utilizar-se equações de Schrödinger ou a soma de histórias de Feynman. Ambas podem descrever os mesmos resultados, sob interpretações diferentes. É inevitável supor que exista algo de fundamental por trás dessas abstrações matemáticas. Elas descrevem a realidade de modo muito diferente ao que nossas experiências e senso comum nos levam a acreditar. Mas as diversas interpretações da mecânica quântica podem ser ainda mais estranhas, como veremos adiante.

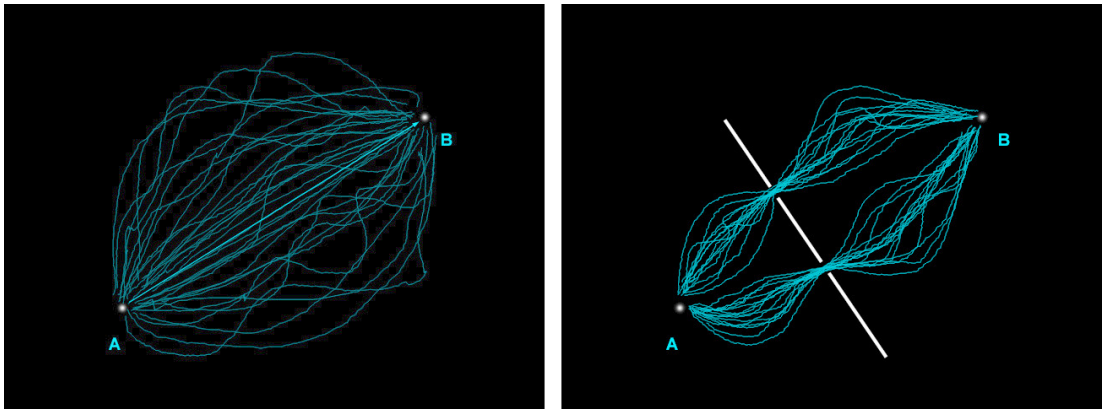


Figura 3.6. Soma de Histórias de Feynman: cada trajetória possível contribui para o cálculo da probabilidade de *um* elétron sair de A e atingir B em determinado tempo. A experiência das duas fendas é ilustrada à direita.

3.1.6. Diagramas de Feynman

Richard Feynman desenvolveu ainda, em 1948, um método para descrever graficamente expressões matemáticas utilizadas no cálculo de amplitudes de probabilidade em interações quânticas. Diagramas de Feynman são utilizados ainda hoje como auxílio no cálculo de interações entre partículas subatômicas, permitindo que físicos visualizem alternativas graficamente, no lugar de simplesmente seguirem sequências de fórmulas abstratas.

Nesses diagramas, o tempo corre da esquerda para a direita e as linhas representam partículas quânticas. Como exemplo, tomemos os dois tipos de partículas subatômicas mencionadas até aqui – elétrons e fótons – e uma terceira, o

---

physics. In Feynman's theory the mathematics and physical picture are different from that of the original formulation of quantum physics, but the predictions are the same (HAWKING e MLODINOW, 2010, posição 726 de 2386).

pósitron. O pósitron, uma antipartícula, é indistinguível de um elétron com o sentido de direção do tempo invertido (FEYNMAN 1988):

Símbolo	Partícula	Obs.
	elétron	elétron avançando no tempo
	pósitron	elétron retornando no tempo
	fóton	

A dimensão vertical representa o deslocamento espacial das partículas quânticas, medidas num único eixo, por simplicidade. Os vértices representam interações, ou "acoplamentos". Por exemplo (do website Feynman Online, www.feynman.com):

Símbolo	Interação
	elétron emite fóton
	elétron absorve fóton
	pósitron emite fóton
	pósitron absorve fóton
	fóton gera par elétron-pósitron = pósitron absorve fóton e reverte no tempo
	aniquilação de par elétron-pósitron, gerando fóton = elétron emite fóton e reverte no tempo

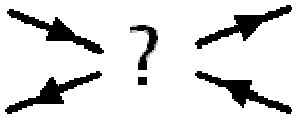
Nesses diagramas verificam-se as seguintes condições (FEYNMAN, 1988):

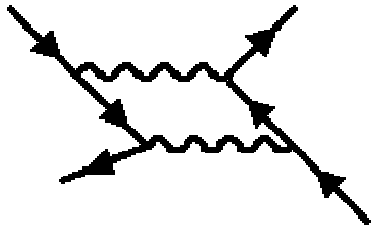
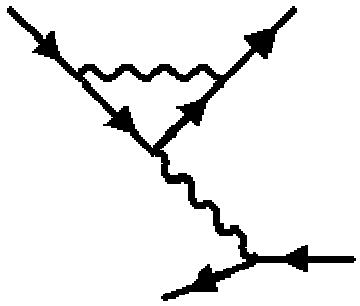
1. Conservação de energia e momento linear para todos os vértices.

2. Partículas que entram ou saem do diagrama são partículas reais, que obedecem à conservação de energia e momento linear e que podem ser observadas em experimentos de laboratório.

3. Partículas cuja vida é restrita ao diagrama são partículas virtuais, que não obedecem, necessariamente, a conservação de energia e momento linear. Entretanto, quando ocorre violação de conservação de energia ou momento, tal partícula nunca pode ser observada num experimento real.

Para calcular-se a amplitude de probabilidade de um evento quântico, usa-se os diagramas para representar todas alternativas possíveis (soma de histórias). Cada diagrama contribui com uma parcela da probabilidade de o evento geral ocorrer. Considerando, por exemplo, uma interação entre elétron e pósitron, teríamos as alternativas abaixo:

	Interação elétron / pósitron
	Possibilidade 1 elétron emite fóton pósitron absorve fóton
	Possibilidade 2 pósitron emite fóton elétron absorve fóton
	Possibilidade 3 elétron emite fóton e retorna no tempo pósitron absorve fóton e avança no tempo

	Possibilidade 4 elétron emite dois fótons e retorna no tempo pósitron absorve dois fótons e avança no tempo
	Possibilidade 5 elétron emite dois fótons elétron re-absorve fóton pósitron absorve fóton

E assim sucessivamente, para qualquer número de fótons e para qualquer ordem de interações possíveis. Todas as “histórias” possíveis devem ser levadas em conta para que os cálculos descrevam, com precisão absoluta, a probabilidade de um processo real medido em laboratório. Na prática, entretanto, considera-se um número restrito de realidades superpostas, dependendo da precisão requerida. Por exemplo, restringindo-se o número de acoplamentos (emissões ou absorções) entre elétrons (ou pósitrons) e fótons a oito, obtém-se precisão de 12 dígitos significativos ao se comparar o cálculo com dados experimentais (FEYNMAN, 1988).

Assim como a grande variedade de caminhos que os elétrons percorrem virtualmente na experiência das duas fendas, segundo a soma de histórias de Feynman, cada alternativa afeta o valor da amplitude de probabilidade calculado. Mas, podemos nos perguntar, essas alternativas todas existem ou são apenas virtuais?

Tanto a função de onda como a soma de histórias conduzem à conclusão de que, enquanto uma medição não é feita, as alternativas se superpõem: os elétrons percorrem todos os caminhos entre o emissor e a tela simultaneamente. Um elétron que interaja com um pósitron o faz de todos os modos possíveis, também simultaneamente. Mas, se medirmos por qual fenda um elétron passou, por exemplo, a superposição de alternativas se desfaz e o efeito de interferência

desaparece da tela do experimento. Estados de superposição de alternativas são destruídos pela simples interferência de um observador, fenômeno denominado de *colapso da função de onda* pela interpretação mais tradicional da mecânica quântica, a interpretação de Copenhague.

3.1.7. O gato de Schrödinger

Num ensaio publicado originalmente em 1935, Erwing Schrödinger ressaltou essa aparente inconsistência da função de onda em relação ao que se espera da realidade, ao propor um experimento de pensamento hoje muito conhecido.

> Alguém pode até mesmo configurar casos bem ridículos. Um gato é trancado numa câmara de aço, junto com o seguinte dispositivo (que deve ser protegido contra a interferência direta do gato): num contador Geiger, há uma pequena porção de substância radioativa, *tão* pequena, que *talvez* no curso de uma hora um dos átomos decaia, mas também, com igual probabilidade, talvez nada aconteça; se acontecer, o tubo do contador descarrega e, através de um relê, libera um martelo que despedaça um pequeno frasco de ácido cianídrico. Se alguém deixar esse sistema por conta própria por uma hora, pode-se dizer que o gato ainda vive *se*, nesse meio tempo, nenhum átomo tiver decaído. A função psi [Ψ] de todo o sistema expressaria isso por indicar um gato vivo e morto (perdoem a expressão) misturados ou borrados em partes iguais (SCHRÖDINGER, 1983, p. 157, grifos do autor, tradução nossa)[47].

É compreensível que a imagem de um gato *vivo e morto simultaneamente* seja mais sugestiva que, por exemplo, a de um elétron atravessando duas fendas ao mesmo tempo. Mas o problema é exatamente o mesmo. Se abrimos a câmara de aço e observamos, a sobreposição se desfaz e encontramos o gato vivo *ou* morto. De modo similar, se adicionamos um sensor à experiência das duas fendas, observamos que os elétrons passam apenas por uma delas e o padrão de interferência é destruído. Por que isso acontece? Poderíamos questionar a validade da mecânica quântica (como fizeram Einstein e outros), mas evidências experimentais, até o momento, são unânimes em favor da teoria. Resta-nos inverter

---

[47] One can even set up quite ridiculous cases. A cat is penned up in a steel chamber, along with the following device (which must be secured against direct interference by the cat): in a Geiger counter there is a tiny bit of radioactive substance, *so* small, that *perhaps* in the course of the hour one of the atoms decays, but also, with equal probability, perhaps none; if it happens, the counter tube discharges and through a relay releases a hammer which shatters a small flask of hydrocyanic acid. If one has left this entire system to itself for an hour, one would say that the cat still lives *if* meanwhile no atom has decayed. The psi-function of the entire system would express this by having in it the living and dead cat (pardon the expression) mixed or smeared out in equal parts (SCHRÖDINGER, 1983, p. 157).

a questão: que poderes especiais permitem ao observador colapsar, pelo simples ato de observação, a superposição de histórias numa única realidade válida?

3.1.8. A interpretação dos muitos mundos

Segundo a interpretação de Copenhague, a observação de um fenômeno quântico colapsa a função de onda do sistema observado. Nas primeiras páginas de sua tese, de 1956, Hugh Everett questiona uma consequência dessa interpretação: um sistema com mais de um observador pode apresentar contradições. Imaginemos um observador *A* que realiza o experimento do gato de Schrödinger. O aparato do experimento obedece a uma função de onda Ψ_A, que admite dois resultados superpostos: o gato está igualmente vivo e morto ao final da primeira hora. Quando o observador *A* abre a câmara, a função Ψ_A colapsa e o gato apresenta-se, por exemplo, vivo. Digamos que esse observador *A* tenha realizado o experimento dentro de uma sala completamente isolada, e que exista um segundo observador, *B*, do lado de fora da sala. Para *B*, *A* faz parte de outro experimento que obedece à função de onda Ψ_B e admite dois resultados superpostos: *A* encontrou o gato vivo e *A* encontrou o gato morto, simultaneamente, em partes iguais. Entretanto, a função Ψ_B não poderia entrar em colapso quando o observador *A* abre a câmara e vê o gato vivo, mas apenas quando *B* entra na sala e observa o resultado do experimento de *A*. Ou seja, o fato do observador *A* finalizar sua experiência não poderia alterar a realidade do observador *B*, para o qual as duas opções possíveis ainda estariam superpostas. Everett (1956) pondera que algo não está certo: ou funções de onda não são aplicáveis a observadores, ou existe alguma coisa errada com o colapso dessas funções durante as observações.

> Fica claro que a interpretação da mecânica quântica com a qual começamos é insustentável se quisermos considerar um universo que contenha mais de um observador. Devemos, portanto, procurar uma modificação adequada desse esquema, ou um sistema de interpretação totalmente diferente (EVERETT, 1956, p. 6, tradução nossa)[48]

[48] It is now clear that the interpretation of quantum mechanics with which we began is untenable if we are to consider a universe containing more than one observer. We must therefore seek a suitable modification of this scheme, or an entirely different system of interpretation (EVERET, 1956, p. 6)

A teoria da *função de onda universal*, de Everett, levou a uma interpretação da mecânica quântica que explica o colapso da função de onda de maneira elegante e surpreendente. Simplificando, no exemplo anterior, existiriam *dois* universos simultâneos: num deles, o observador A encontra o gato vivo; no outro, ele constata que o gato está morto. Antes que o observador B entre na sala, esses dois universos estariam superpostos. Dessa maneira, o aparente colapso da função de onda seria mera percepção de cada uma das diferentes versões dos observadores: um epifenômeno, nada mais. As observações não afetariam as funções de onda. Todas as realidades alternativas que interferem no cálculo de amplitudes de probabilidade seriam reais, mas isoladas. Acreditar nessa interpretação implica aceitar um número muito grande de universos superpostos e incomunicáveis, embora nossas múltiplas consciências percebam, cada uma, apenas um deles.

Ao interpretar que todas as alternativas realmente ocorrem, em universos separados, e que diferentes versões de observadores experimentam cada possibilidade, o valor do quadrado do módulo da amplitude deixa de ser interpretado como a probabilidade de um evento ocorrer para tornar-se a probabilidade de estarmos nesse universo específico durante a medição. Isso tem um implicação séria: *os múltiplos universos formariam um sistema determinista*. Uma possível maneira de se testar essa interpretação foi mencionada no capítulo anterior. A experiência tem por base uma característica de duas constantes físicas, a de Plank (*h*) e a velocidade da luz (*c*): elas seriam invariáveis durante todo o passado e o futuro de um universo específico (RÖSSLER, 2006).

> Essa percepção pode dar uma motivação adicional para realizar um "experimento crucial", em nosso próprio mundo físico, que provaria que apenas a versão de Everett – e não a de Copenhague – da mecânica quântica é correta como teoria (RÖSSLER, 2006, p. 166)[49].

Propõe-se então a experiência já mencionada de alterar os valores das constantes *h* e *c* através de ajustes muito precisos sobre a temperatura do cérebro de um observador. Quando esses valores fossem alterados para o observador, ele

[49] This insight can give one an additional motivation to perform a “crucial experiment” in our own physical world that would prove that only Everett’s – and not the Copenhagen – version of quantum mechanics is correct as a theory (RÖSSLER, p. 166, 2006).

necessariamente se encontraria em outro universo, no qual as duas constantes sempre mantiveram os novos valores. Rössler (2006) indica que o observador ainda lembraria da mudança, enquanto que para todas as outras pessoas os novos valores seriam lembrados como os de sempre, imutáveis. Caso a proposta de Everett venha a ser provada, ela deixaria de ser uma simples interpretação para se tornar uma teoria.

Não há consenso sobre a chamada *interpretação dos muitos mundos*, iniciada com o trabalho de Everett. Enquanto não se realizar com sucesso uma experiência como a proposta por Rössler (2006), a interpretação não altera em nada as regras da mecânica quântica – como aliás acontece com outras interpretações. E mesmo que a experiência viesse a ser realizada com sucesso, uma única pessoa, o próprio observador, saberia da verdade e não teria qualquer prova disso, apenas sua memória. Mas a ideia de universos diferentes, sobrepostos, sem comunicação e selecionados através da interação com um observador pode ter aplicações interessantes como modelo matemático para games 3D. Talvez essa seja a melhor maneira de modelarmos uma realidade virtual com base na interação.

3.2. Estados superpostos: proposta de modelo matemático de realidade para games 3D

> (...) *Não existe conceito de realidade independente de uma imagem ou teoria*. Em vez disso, adotaremos uma visão que chamaremos de realismo dependente do modelo: a ideia de que uma teoria física ou visão de mundo é um modelo (geralmente de natureza matemática) e um conjunto de regras que conectam os elementos do modelo com observações. Isto fornece uma estrutura com o qual podemos interpretar a ciência moderna (HAWKING e MLODINOW, 2010, posição 409 de 2386, grifos do autor, tradução nossa)[50].

[50] (...) *There is no picture – or theory – independent concept of reality*. Instead we will adopt a view that we will call model-dependent realism: the idea that a physical theory or world picture is a model (generally of a mathematical nature) and a set of rules that connect the elements of the model to observations. This provides a framework with which to interpret modern science (HAWKING e MLODINOW, 2010, posição 409 de 2386)

Um modelo matemático de realidade normalmente tem como função prever ou simular eventos que aconteceriam no mundo real. As equações de movimento da mecânica clássica permitem, por exemplo, que se calcule com relativa precisão a trajetória de um projétil, desde que conheçamos seus dados em determinado momento. Existem maneiras de se aprimorar esse modelo: podemos considerar efeitos aerodinâmicos sobre o projétil, a força externa aplicada pelo vento, a variação do vetor de aceleração da gravidade devida à curvatura do planeta, etc., para que o cálculo seja o mais preciso possível. Teremos então uma descrição matemática que nos permitirá prever o comportamento do projétil real.

Existe uma diferença fundamental entre o exposto acima e a utilização de um modelo matemático de realidade num game 3D: no ambiente virtual, o modelo *define* a realidade. Objetos passarão a se comportar, não de maneira aproximada, mas exatamente conforme o ditado por ele. Mais precisamente, o que experimentamos num game 3D é a aplicação de um modelo sobre dados internos e externos, esses últimos coletados através de interfaces. Ou seja, não há realidade a ser simulada. Antes, a aplicação do modelo sobre dados ocupa o lugar do real sob a perspectiva do jogador. Na citação que abre esta seção, Hawking e Mlodinow (2010) mencionam o conjunto de regras que conectam os elementos do modelo com observações, através das quais é possível verificar se o modelo está de acordo com a realidade. Games 3D possuem um conjunto de regras com objetivo diferente, pois elas constroem o que se apresenta como real de acordo com o modelo.

Ao esquecer disso, um designer pode limitar seriamente as possibilidades de um game 3D, tentando encaixar, artificialmente, um modelo projetado para prever o real como "realidade" do ambiente virtual. Entretanto, se abandonarmos a prática de adaptar simulações do que acreditamos ser real, que tipo de modelo poderia ser mais apropriado aos games 3D? Como seriam os “descritores de estado” de personagens? Quais características devem ser priorizadas? Como garantir a coerência do que acontece no ambiente virtual? Que vantagens um modelo diferente poderia trazer?

A seguir, apresento possíveis respostas a essas perguntas com base no que foi exposto até aqui. É importante notar que não existe a intenção de se apresentar um

modelo completo e funcional, uma vez que isso exigiria todo um desenvolvimento de algoritmos que fugiria do escopo deste trabalho. Mas um modelo funcional pode resultar do desenvolvimento e codificação das considerações apresentadas a seguir.

3.2.1. Similaridades

Se compararmos o modelo da mecânica quântica com alguns exemplos de programas analisados anteriormente, poderemos destacar duas similaridades:

Campos de possibilidades: nos algoritmos genéticos, o genoma define um vasto número de possibilidades. Basta imaginar todas as configurações que as criaturas de *Evolving Virtual Creatures* podem assumir, ou os modos diferentes com que elas podem se movimentar. Todas essas criaturas diferentes estão potencialmente presentes na estrutura do código genético. O mesmo vale para a grande variedade de comportamentos diferentes que as criaturas de *Eden*, ou de *Incidente em Varginha*, podem adotar. Consideraremos que esses campos de possibilidades, definidos pelo código, tenham algum paralelo com a soma de histórias de Feynman ou com a superposição de estados no cálculo de funções de onda.

Interação: o que causa o colapso de um campo de possibilidades numa expressão específica é alguma forma de interação. Não se trata apenas da interação no sentido mais tradicional, entre um ser humano e a máquina, mas também de interações entre trechos de código que funcionam de maneira independente (objetos). Pode ser, por exemplo, o "acasalamento" entre duas criaturas de *Eden*, ou as novas gerações de criaturas de *Evolving Virtual Creatures*, também elas construídas com técnicas de hibridização e mutação sobre os códigos dos "pais". Essas interações podem ser comparadas com a observação em sistemas quânticos. Vale lembrar que qualquer observação ou medição experimental de um sistema quântico nunca é passiva, mas exige interação. A interação com o observador, segundo a interpretação de Copenhague, acarreta o colapso da função de onda do sistema. Segundo a interpretação dos muitos mundos, a mesma interação está nos pontos de bifurcação entre universos possíveis. De modo similar, a interação entre trechos de um programa, ou entre o programa e um ser humano, pode definir uma opção especifica entre as muitas possíveis no código.

Com base nesse pequeno paralelo, a proposta é descrever, em linhas gerais, um modelo que explore as características acima. Um modelo que não apenas priorize a interação como seletora do campo de possibilidades, mas que também permita a adaptação da realidade do game às habilidades do jogador, tornando mais provável que este atinja o estado de *flow*, por exemplo. Com isso buscamos a construção de uma realidade virtual a partir da interação e da experiência de jogo, e não o contrário.

3.2.2. Descritores de estados finitos

Máquinas de estados finitos são relativamente próximas ao modelo almejado. Como vimos no capítulo 2, a alternância entre estados diferentes, sujeita a interações, pode gerar comportamentos interessantes e verossímeis.

Ainda assim, máquinas de estados finitos são essencialmente lineares, pois um personagem sempre tem posição, velocidade e estado definidos em determinado momento. Personagens podem ficar em estado de espera, ou de procura, durante grande parte da partida, sem ter qualquer interação com o avatar do jogador – e isso gera problemas. Ocasionalmente, esses personagens que vagam pelo ambiente virtual podem ficar "presos" em locais insólitos, como atrás de portas, em becos estreitos ou atrás de algum tipo de mobiliário. Cabe ao *level designer* reconfigurar essas potenciais armadilhas do mapa do game, mas isso não evita que ocorram exemplos desses erros até mesmo em super-produções como a primeira versão de *Half-Life*, onde era possível notar soldados presos em alguns locais das instalações de *Black Mesa*. Lembro-me que no nível da Praça da Sé, em *Incidente em Varginha*, um personagem conseguia escapar do mapa por uma pequena falha entre duas paredes e ficava vagando fora do ambiente do game, mas ainda visível, como se caminhasse pelo ar. Era difícil reproduzir o problema, pois o personagem precisava se movimentar por muito tempo até ocasionalmente passar pela falha. Encontrá-la exigiu dias de trabalho analisando todo o mapa do nível.

Estados superpostos seriam simples de se programar. Basta que, para cada estado, o personagem tenha atribuída uma variável de probabilidade. Por exemplo, um personagem poderia estar 20% no estado de ataque e 80% no de defesa, em

estratégia não dissimilar à da lógica *fuzzy*[51], mas não apenas limitada a dois estados. No momento da interação, quando o personagem vê ou é visto pelo jogador, ou de alguma outra forma interage com ele, o programa calcularia o "colapso" dessa superposição num estado definido, com base nas probabilidades atribuídas a cada um. Se o colapso fosse repetido 100 vezes, a tendência seria de que o personagem se encontrasse no estado de defesa em 80 delas, e no de ataque nas outras 20. O emprego de probabilidades, além de permitir a superposição de diversos estados, deixaria margem para que o comportamento variasse de forma menos previsível que numa máquina de estados finitos tradicional.

Mas a principal limitação que conduz ao tipo de erro mencionado anteriormente, entretanto, é o fato de máquinas de estados finitos ainda serem aplicadas a personagens newtonianos, que têm posição e velocidade definidas durante todos os momentos do game. Não se trata apenas da tentativa de correção de um erro comum, de evitar que personagens fiquem presos ou perdidos no mapa do game. Há maneiras mais simples de se contornar esses problemas. Mas posição e velocidade também são estados que precisam ser descritos, assim como "atacar", "defender" ou "fugir". Se quisermos estender a ideia de máquina de estados, criando o que poderíamos chamar de *máquina de estados superpostos*, será necessário tratar posição e velocidade como campos de possibilidades.

3.2.3. Descritores de estados: posição e velocidade

Para que a posição de cada personagem seja definida apenas em momentos de interação com o jogador, é preciso definir uma estrutura de dados diferente das variáveis de posição *x*, *y* e *z*. Primeiro, é importante reduzir o número de alternativas de posição para simplificar o processamento de probabilidades. Para isso, os pisos do ambiente virtual seriam subdivididos em matriz de elementos, pequenos quadrados de dimensões inferiores à do próprio personagem, que representam apenas posições válidas. A definição dessa matriz já evitaria que qualquer personagem fosse encontrado em posições inválidas – por exemplo, preso no mapa ou fora dele. No lugar das tradicionais variáveis de posição, os dados do

[51] A lógica *fuzzy* lida com valores intermediários entre "sim" e "não", possibilitando aproximações e raciocínios que não seriam possíveis com a lógica tradicional, ou booleana.

personagem incluiriam vetores para elementos da matriz. A cada elemento de posição associaríamos um valor limitado a uma casa decimal (0.1), representando a probabilidade de aquela área estar ocupada pelo personagem durante uma observação. Utilizando-se valores de uma única casa decimal garante-se que a menor probabilidade a ser considerada seria de 0.1% e que o maior número de posições ocupadas simultaneamente por personagem não ultrapassaria 1000, visando reduzir o processamento necessário para o cálculo das possíveis posições ocupadas.

A figura abaixo ilustra valores de probabilidade de posição para um personagem ao longo do tempo. Nesse exemplo, em t=1 a posição é conhecida, representada pelo tom mais saturado de cor, que equivale a 100% de probabilidade. Com o passar do tempo, outras possíveis posições ocupadas pelo personagem são atualizadas, sendo que probabilidades mais baixas correspondem a tons gradualmente mais transparentes.

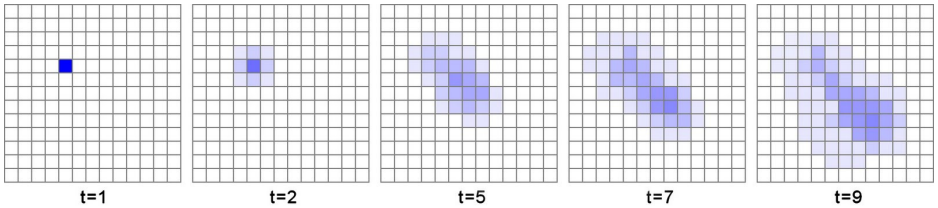


Figura 3.7. Atualização do campo de possibilidades de posição de um personagem.

Podemos pensar nessas diversas posições simultâneas como uma “soma de histórias” bastante simplificada. O programa consideraria diversos possíveis caminhos que o personagem poderia percorrer durante o tempo em que ele não interage com o avatar do jogador, gerando as “nuvens” de posição ilustradas na figura acima. Para cada elemento de posição, além da probabilidade de que o personagem o ocupe, teríamos também alguns vetores de velocidade associados, que indicariam possíveis ângulos e velocidades absolutas. Todos esses valores seriam atualizados durante o processamento principal, e é interessante notar que essa atualização não exigiria um período crítico de ciclo. Não seria necessário, por exemplo, atualizar os valores a cada *frame* do game (como se faz normalmente em modelos tradicionais). A diferença de processamento de alguns *frames* acarretaria

mudanças mínimas nas “nuvens” de posição, afetando principalmente as bordas, e com valores relativamente pequenos.

3.2.4. O observador

Quando qualquer elemento da matriz, ocupado por probabilidade de posição diferente de zero, estiver em linha visual com o avatar do jogador[52], o programa estabelecerá se o personagem se encontra ou não nesse elemento, utilizando para isso a probabilidade a ele associada. Se o resultado for positivo, o personagem passa a ocupar aquele elemento de posição e todos os outros elementos anteriormente ocupados serão zerados. Caso contrário, o elemento em linha visual com o avatar do jogador será zerado e os valores de probabilidade a eles atribuídos serão distribuídos pelos outros elementos com valores diferentes de zero, aumentando a probabilidade de o personagem se encontrar nessas outras posições.

Na figura abaixo, podemos acompanhar um mapa com campos de possibilidade de posição de cinco personagens. Para efeito de ilustração, os campos em azul representam personagens em estado predominantemente de defesa, buscando posições protegidas, enquanto que os em vermelho representam maior proporção do estado de ataque, quando os personagens buscam linha visual com o avatar do jogador (ver capítulo 2). As áreas cinzas representam paredes e colunas.

[52] Quando a posição da câmera virtual não coincidir com a do avatar do jogador (como nos games denominados “em terceira pessoa”, por exemplo), a referência de linha visual será a própria câmera. Uso a posição do avatar do jogador para simplificar a descrição das propostas.

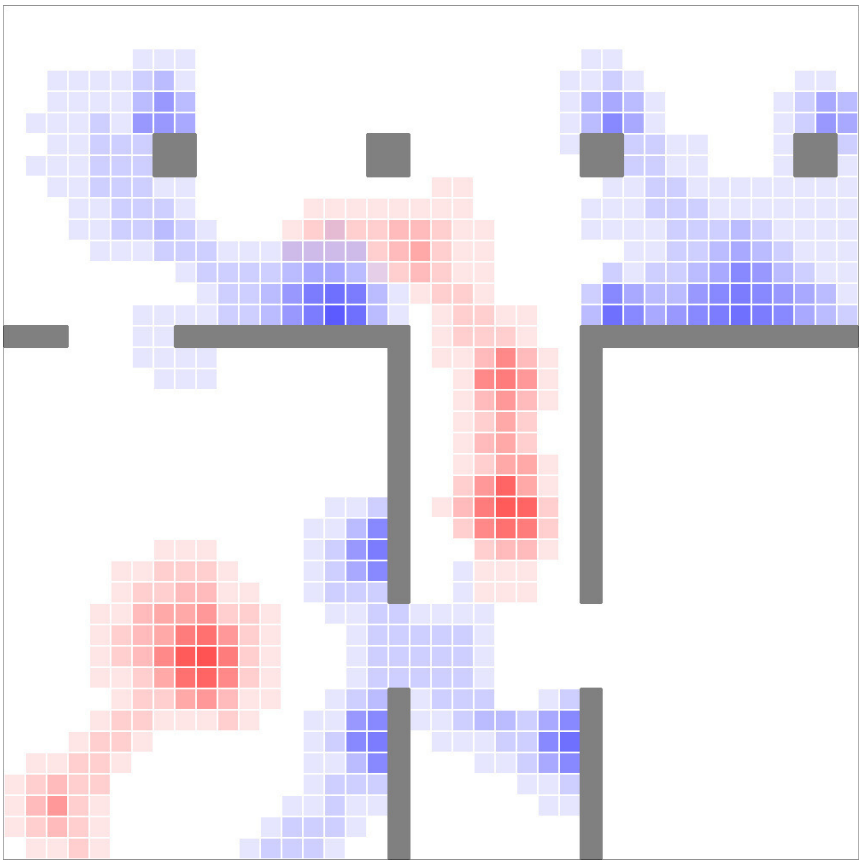


Figura 3.8. Campos de possibilidades de posição de cinco personagens.

Definiremos também o *setor de contato visual*, um setor esférico que se estende a partir do centro do avatar do jogador[53] e que é bloqueado por paredes e outros objetos de cenário. Personagens que se encontrem no interior desse setor podem potencialmente ver o jogador, ou serem vistos por ele. Quando os elementos da matriz entram nesse setor de contato visual, têm sua probabilidades de posição avaliadas pelo programa. No exemplo abaixo, um dos personagens atacantes é localizado nesse processo, enquanto que um personagem defensor tem seu campo de possibilidades condensado. Para simplificar a analise do modelo, consideraremos que os campos de possibilidades de posição serão sempre formados por quadrados contíguos. Caso o setor de contato visual separe o campo original em dois, apenas o que tiver maior probabilidade total será mantido, com a distribuição dos valores de probabilidades dos elementos que serão zerados.

[53] Ou da posição da câmera do game. Em games multi-usuários, teríamos um setor de contato visual para cada jogador.

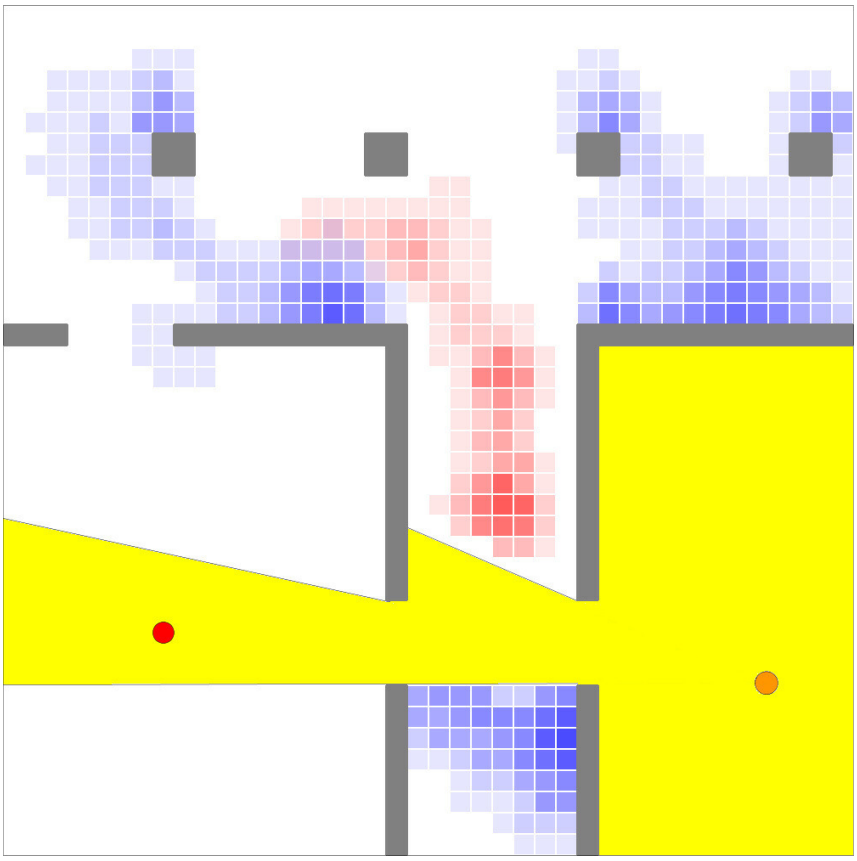


Figura 3.9. O avatar do jogador, em laranja, o setor de contato visual, em amarelo, e o personagem localizado, em vermelho

O colapso do campo de possibilidades de posição pode acontecer também com outros tipos de interação, não apenas com o possível contato visual. Ao emitir qualquer som audível pelo jogador, o personagem terá sua posição definida mesmo que não esteja no setor de contato visual. O colapso da posição será feito, nesses casos, antes da execução da rotina que emite o som, pois a localização definida é necessária para o cálculo de volume e balanceamento de canais. A abertura de portas, o acionamento de mecanismos, ou mesmo o ruído de passos próximos, são eventos que podem conduzir ao colapso de posição e velocidade de um personagem. Imediatamente após esses eventos, os campos de possibilidades voltam a crescer a partir do local do colapso.

3.2.5. Consistência

Avaliar os elementos da matriz assim que eles entrem no setor de contato visual força que um personagem só possa aparecer nas bordas do campo visual do jogador ou fora dele. Um personagem não sofreria o colapso de sua posição no

meio da tela, surgindo do nada, pois isso representaria uma descontinuidade na avaliação dos elementos da matriz de probabilidades de posições[54].

Evitar inconsistências enquanto um personagem permanece no setor de contato visual (mudanças inverossímeis de velocidade e posição, por exemplo), exigiria mais do programa. A atualização de posição, velocidade e estado poderia ser feita de modo tradicional nessas situações, respeitando equações de movimento e máquina de estados finitos. Assim que o personagem saísse do setor de contato visual, os campos de possibilidades voltariam a ser atualizados como descrito anteriormente. Melhor seria, entretanto, que o algoritmo de atualização de campos de possibilidades apresentasse, como caso especial, mudanças de posição e velocidades visualmente consistentes, sem a necessidade de se recorrer a outro tipo de algoritmo enquanto o personagem estiver no interior do setor de contato visual. Isso é plenamente possível se utilizarmos algo parecido com a soma de histórias de Feynman. Dentro do setor de contato visual, apenas uma história seria atualizada, e tudo funcionaria de modo tradicional. Fora dele, histórias diferentes se sobreporiam, criando campos de possibilidades.

3.2.6. Diagramas de personagens

Sempre que um personagem tiver o avatar do jogador em sua linha de visão, ele obviamente estará no interior do setor de contato visual. Nesses casos, além das já descritas atualizações de posição e velocidade, pode acontecer também o chaveamento da máquina de estados do personagem, visto que a existência de linha visual com o avatar do jogador é uma das principais condições que conduzem à mudança de estado. De fato, a mudança de estados pode acontecer no exato momento em que o personagem entra no setor de contato visual, se seu ângulo de visão for apropriado. Nessas situações, tanto o estado atual como o novo estado devem ser calculados.

[54] Isso poderia ocorrer se a posição do avatar do jogador também fosse descontínua: por exemplo, ao ser teleportado para algum outro local do game. Mesmo nesse caso, o aparecimento do personagem em seu campo visual seria consistente com a mudança repentina de cenário.

O número de estados finitos de um personagem normalmente está na casa de poucas dezenas, mas ainda assim as combinações possíveis seriam numerosas e difíceis de se acompanhar unicamente pelo código. Na mecânica quântica, especificamente na soma de histórias, os diagramas de Feynman são ferramentas utilizadas para resolver problemas similares, ainda que muito mais complexos. Vejamos então a viabilidade de se utilizar diagramas para acompanhar possíveis interações de personagens com o avatar do jogador, propiciando assim o cálculo de suas respectivas probabilidades de modo mais visual e intuitivo.

A movimentação de personagens será representada por linhas contínuas com setas e o contato visual por linhas pontilhadas. Estados serão representados por círculos coloridos com uma letra, a inicial em inglês do nome do estado[55]. O jogador, ou melhor, seu avatar, é representado por um círculo branco. Temos então a legenda abaixo, que considera apenas sete dos estados possíveis para um game de combate, como exemplo.

Símbolo	Descrição
→	**personagem**
.....	**contato visual**
player	**jogador**
P	**estado de patrulha (*patrol*)**
S	**estado de busca (*search*)**
I	**estado de espera (*idle*)**
A	**estado de ataque (*attack*)**
D	**estado de defesa (*defense*)**

[55] O uso do inglês é sugerido unicamente pela simplicidade de se associar estados a variáveis de personagens, pois linguagens de programação normalmente não aceitam acentos e caracteres especiais em nomes de variáveis.

C	**estado: chamar ajuda (*call for help*)**
F	**estado de fuga (*flee*)**

Figura 3.10. Legenda de sete dos estados possíveis para um game de combate.

Os diagramas são então construídos com o tempo correndo no eixo vertical, de baixo para cima, e com uma dimensão no eixo horizontal, da esquerda para a direita. Uma única dimensão de espaço pode parecer pouco, mas já é suficiente para indicar se um personagem se aproxima ou se afasta do avatar do jogador, por exemplo. Outra convenção que adotaremos é a direção do contato visual: quem está à esquerda vê e quem está à direita é visto. Como na atualização de estados interessa-nos o que o personagem vê, teremos o avatar do jogador sempre do lado direito do diagrama. O contato visual será considerado instantâneo, por isso a linha pontilhada que o representa será sempre horizontal. A mudança de estado do personagem, entretanto, não deve ser instantânea e coincidente com o contato visual, por um motivo simples: um atraso de um a dois décimos de segundo entre o contato visual e a mudança de estado tende a ser percebido como "tempo de reação" do personagem virtual – mudanças instantâneas nunca parecem convincentes. Um exemplo de diagrama é ilustrado na figura abaixo.

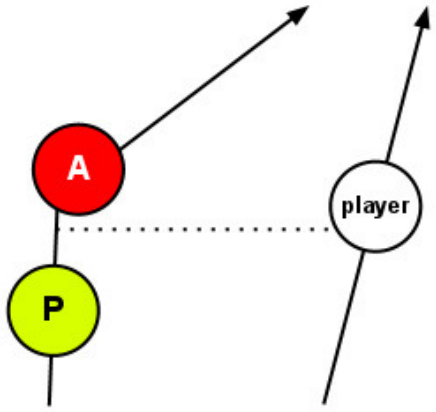


Figura 3.11. Exemplo de diagrama.

No exemplo, o personagem está no estado "patrulha", vê o avatar do jogador, muda seu estado para "ataque" (após um ou dois décimos de segundo) e passa a persegui-lo. Essa seria apenas uma possibilidade de interação entre o personagem

e o avatar do jogador, e teria uma determinada probabilidade de ocorrer. Suponhamos agora um personagem não newtoniano que apresentasse três estados superpostos: espera, patrulha e busca. Ao localizar o personagem no campo de possibilidades de posição e velocidade, o programa estabelece que o avatar do jogador está visível. É necessário então calcular o colapso dos três estados superpostos e, ao mesmo tempo, considerar uma eventual mudança de estado.

Podemos ver na tabela a seguir as diversas possibilidades dessa interação. Para definir qual delas ocorrerá, é preciso avaliar a probabilidade associada a cada uma. O uso de diagramas facilita a compreensão dessas possibilidades, além de ajudar no preparo do código correspondente. Existe, entretanto, outra vantagem importante. No exemplo, os diagramas apresentam dez realidades sobrepostas e qualquer delas pode ser definida no momento da interação. Isso nos deixa a liberdade de definir a probabilidade de cada uma de acordo com o contexto atual do game.

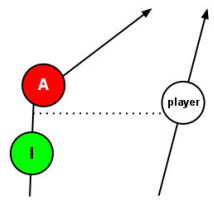
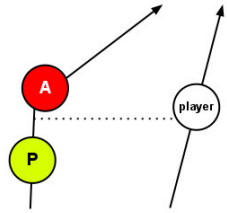
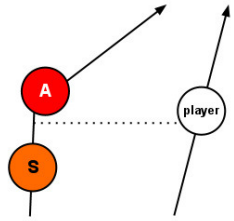
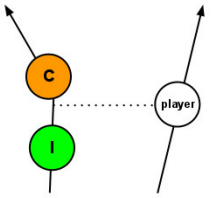
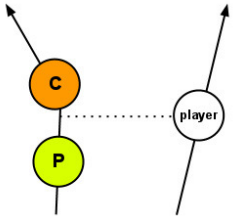
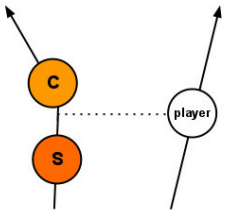
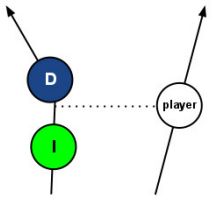
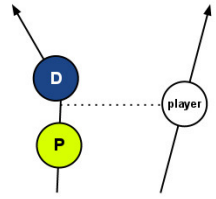
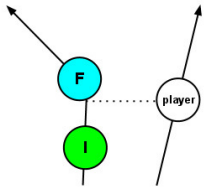
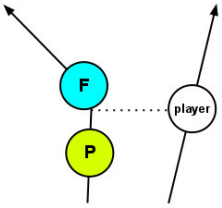
de: ESPERA	de: PATRULHA	de: BUSCA	Para:
			**ATAQUE**
			**CHAMAR AJUDA**
		**N/A**	**DEFESA**
		**N/A**	**FUGA**

Figura 3.12. Mapa de interação: possíveis mudanças de estados a partir de três estado iniciais superpostos.

3.2.7. Cálculo de probabilidades

Mas como deveríamos calcular as probabilidade desses estados, posições e outras variáveis? É importante não repetirmos aqui o erro de importar um modelo do mundo real de maneira indiscriminada, calculando probabilidades como na mecânica quântica, por exemplo. Mesmo porque isso não faria sentido para a nossa percepção de realidade, que é aparentemente bastante limitada.

O principal objetivo da proposta de estados superpostos é permitir que o programa escolha a possibilidade mais adequada durante momentos de interação. Nos

algoritmos genéticos, a função de avaliação tem objetivo similar, selecionando os exemplares mais adequados de uma população. A proposta de estados superpostos segue nesse sentido, superpondo histórias possíveis durante o desenrolar do game e selecionando as que forem mais adequadas nos momentos de interação.

Consideremos, por exemplo, que o objetivo seja adaptar a reação de personagens a mudanças de contexto no desenrolar de um game de combate em primeira pessoa. Através do cálculo de probabilidades seria possível equilibrar a dificuldade do game com as habilidades do jogador, potencializando a experiência que Csikszentmihalyi (1997) denominou *flow* (ver capítulo 1). Alguns exemplos podem tornar a proposta mais clara: quando o jogador estivesse em desvantagem no game, o cálculo de probabilidades poderia priorizar posições desfavoráveis, desprotegidas, para os personagens inimigos. O estado de alerta e a precisão de tiro também poderiam ser reduzidos nessa situação. Mesmo em estado de ataque, personagens inimigos poderiam traçar, com maior probabilidade, caminhos que não se cruzassem com o do jogador. No outro extremo, se o jogador estivesse em situação muito vantajosa no game, o cálculo de probabilidades poderia favorecer as melhores posições e ângulos de ataque para os personagens inimigos, aumentando-lhes o estado de alerta e a precisão de seus ataques. Nos dois extremos, o programa distribuiria probabilidades de acordo com uma medida de desempenho do jogador – que, para um jogo de combate, pode ser definida pela fórmula:

desempenho = saúde do jogador (1 - média da saúde dos inimigos enfrentados / 2)

Onde tanto as saúdes como o desempenho variam[56] de 0.0 até 1.0. O desempenho seria máximo para um jogador com valor de saúde máxima que tenha eliminado todos os inimigos enfrentados até o momento. Um jogador com saúde total mas que não tenha causado qualquer dano aos inimigos teria um desempenho de 0.5, e assim por diante. Na matriz de elementos do mapa, desempenhos próximos a 1.0 favoreceriam posições bem protegidas e ângulos ideais, enquanto que desempenhos próximos de 0.0 favoreceriam exatamente o contrário. Além disso, o valor de desempenho interferiria diretamente no cálculo de probabilidade de estados

[56] Números de ponto flutuante entre 0.0 e 1.0 são utilizados para representar variações entre 0 a 100% por simplicidade de cálculo.

durante interações entre o personagem e o jogador. Estados de fuga e defesa teriam probabilidades maiores quando o desempenho do jogador fosse baixo. Ataques e posicionamentos táticos (chamar ajuda, por exemplo) seriam mais prováveis com valores altos de desempenho.

3.2.8. Enredos interativos

Selecionar histórias durante interações, com base no contexto atual do game, permitiria equilibrar o nível de dificuldade com as habilidades específicas do jogador, mas essa seria apenas uma aplicação de um modelo de estados superpostos. Desse uso direto e localizado talvez seja possível extrapolar a construção de enredos verdadeiramente interativos.

Para isso, seria necessário desenvolver um modo de construir histórias que podem ser modificadas em tempo real, adaptando novas linhas narrativas a uma estrutura básica que garanta o interesse do enredo. Um processo similar ao da construção de algoritmos pode ser visualizado: das interações possíveis entre personagens comporiam-se pequenos trechos de histórias, que por sua vez formariam trechos maiores, até que a história completa se consolidasse ao final do game. A dificuldade está justamente em planejar essa história geral de baixo para cima, a partir das pequenas ações. Um programa deveria gerenciar essa construção.

Um *drama manager*, como proposto pelo Oz Project, tem por objetivo criar sentido a partir de eventos isolados, “costurando-os” de forma coerente ao direcionar os possíveis eventos futuros:

> O *drama manager* vigia o estado do mundo (incluindo a interação do usuário). Enquanto o usuário está se movendo ao redor e interagindo com personagens em algum ponto de enredo particular, o sistema não está fazendo nada, apenas observando. Eventualmente, uma sequência de atividades no mundo será reconhecida como causa de uma transição de enredo. O *drama manager* entra em ação. Existe um histórico de pontos de enredo passados. Neste momento, as histórias futuras consistem de todas as sequências possíveis de pontos do enredo restantes. (...) De modo semelhante a programas de jogos (tais como programas de xadrez), o *drama manager* analisa todos os movimentos possíveis que o usuário poderia fazer para causar uma transição de enredo, cada movimento possível do sistema a partir desse novo estado do mundo, etc. até que ele

> tenha experimentado as histórias possíveis (Mateas, 1997, tradução nossa)[57].

A técnica da "força bruta", de analisar todos os movimentos possíveis, dificilmente seria eficiente ou bem sucedida. Mas a proposta do Oz Project tem um ponto muito forte: o gerenciamento de possibilidades. No lugar da análise exaustiva de todas as histórias possíveis, uma ferramenta que trabalhasse de forma adaptativa, como os algoritmos genéticos, seria indicada. E os resultados dessa análise poderiam ser aplicados diretamente ao cálculo de probabilidades de uma máquina de estados superpostos. Ao alimentar o cálculo de probabilidades com informações do *drama manager*, estaríamos reforçando sequências de eventos que se encaixassem de forma mais natural e coerente numa história construída em tempo real.

Podemos imaginar características gerais para um *drama manager* desse tipo. Cada personagem teria múltiplas histórias possíveis, superpostas. Como resultado de interações específicas, incluindo aqui o conhecimento sobre algum fato importante para o personagem, essas histórias seriam apropriadamente chaveadas (como na soma de histórias em sistemas quânticos). Ao final do game, independente do número de histórias que continuassem válidas para cada personagem, a sequência de cada uma delas teria feito sentido. O mesmo deveria ocorrer com o enredo geral, vivenciado pelo jogador.

3.2.9. Pensar diferente

Para ilustrar diferenças entre modelos, Hawking e Mlodinow (2010) usam o exemplo de um peixe dourado no interior de um aquário esférico. O peixinho pode ver o mundo exterior, mas tudo é distorcido por efeitos óticos da água e do vidro do aquário. Algo que se desloque do lado de fora em linha reta, por exemplo, parece mover-se numa curva quando visto do interior do aquário. Os movimentos de objetos

---

[57] The drama manager watches the state of the world (including the user interaction). While the user is moving around and interacting with characters within some particular plot point, the system isn't doing anything but whatching. Eventually, some sequence of activities in the world will be recognized as causing a plot transition. The drama manager springs into action. There exists some past history of plot points. At this point in time, the future histories consist of all possible sequences of remaining plot points. (...) In a manner similar to game playing programs (such as chess programs), the manager examines every possible system move the user could make to cause a plot transition, every possible system move from that new state of the world, etc. until it has played out the possible stories (MATEAS, 1997).

fora do aquário podem ser descritos do ponto de vista do peixe dourado, mas de acordo com um modelo diferente do necessário para o lado de fora. Teríamos então dois modelos, um que funcionaria dentro do aquário e outro válido para o lado de fora. Ambos descreveriam a mesma realidade, sob pontos de vista diferentes. O modelo do interior do aquário, mais complexo, seria tão válido quanto o do lado de fora para explicar os mesmos eventos (HAWKING e MLODINOW, 2010). Se os dois modelos apresentam resultados equivalentes e se o modelo interno exige maior trabalho, é óbvio que seria mais prático utilizar o modelo externo. Só que essa escolha pode não parecer óbvia para o peixe dourado, que provavelmente nunca teve acesso a qualquer ponto de vista exterior. Para ele, a sala de estar do lado de fora do aquário seria composta por curvas que se deformam e se compõem de modo naturalmente complexo. Linhas retas seriam aberrações difíceis de se imaginar.

Universos superpostos; o elétron que percorre, simultaneamente, todos os caminhos possíveis entre dois pontos; ou o gato que estaria ao mesmo tempo vivo e morto, em partes iguais, também podem nos parecer aberrações difíceis de imaginar. Não é natural pensarmos dessa maneira, pois estamos sujeitos a pontos de vista muito diferentes, como se estivéssemos dentro do aquário. A consciência que experimentamos parece estar presa a uma única linha de realidade, como indica a experiência proposta por Rössler (2006). Pensar diferente, entretanto, já provou tanto sua necessidade como sua utilidade no campo da física: entre outros exemplos, a mecânica quântica está por trás do desenvolvimento do transístor e do laser, ou seja, de toda a revolução digital iniciada no século XX. Não existiriam games 3D sem essas maneiras diferentes de entender a realidade.

O pleno uso de superposição de possibilidades em games 3D, com seleção a partir da interação, pode exigir um poder de processamento ainda não disponível. Mas, como demonstramos, esse modelo é potencialmente muito mais apropriado a mídias não lineares que o da mecânica clássica. A proposta também pode aproximar os games 3D de um “realismo” bem diferente do que é perseguido pela indústria de games – afinal, segundo a mecânica quântica, superposições de possibilidades formam a base do real, mesmo que não sejamos capazes de percebê-las.

A proposta apresentada nesta tese visa abandonar um paradigma herdado de outras mídias – que favorece uma história linear com a adaptação de pontos de interatividade – pela sobreposição das histórias possíveis selecionadas pela própria interação. Existem vantagens que poderiam ser exploradas imediatamente, como a adaptação automática do nível de dificuldade às habilidades específicas do jogador. Para outros objetivos, este seria apenas um ponto de partida: muitas técnicas diferentes serão necessárias para que seja possível escrever enredos interativos de modo processual e adaptativo. Porém, como acontece com os algoritmos genéticos, é possível que as estratégias emergentes para a construção de enredos coerentes em games 3D venham a surpreender os próprios autores desses novos sistemas.

CONCLUSÃO

Quando Otto Rössler (2006) propõe a criação de “universos low level artificiais”, ele descreve uma configuração de experimento que visa verificar se seus habitantes artificiais poderiam descobrir que foram programados. Não é difícil perceber que games 3D têm potencial para evoluir em direção a objetivos similarmente ambiciosos. Mesmo hoje, com tantas limitações, eles formam a mídia de massa que mais se aproxima da descrição “mundos artificiais”. Mas, para que esses produtos deixem de ser meros testes de habilidade intercalados com sequências narrativas pré-determinadas, é preciso que o desenvolvimento de games 3D assuma modelos matemáticos diferentes dos atuais. Este trabalho mostrou uma possibilidade.

No capítulo 1, vimos que o modelo matemático mais utilizado em games é o da mecânica clássica, mas também que é possível criar experiências interessantes que não respeitam esse modelo. Exemplos de códigos e análises de trajetórias características foram mostrados como evidência do uso generalizado das equações de movimento em games 3D e 2D. A seguir, vimos exemplos de exceções à regra, de games que, de uma forma ou de outra, "burlam" as equações de movimento da mecânica clássica. Em *Super Mario Brothers*, as alterações propiciam um gameplay mais ágil e agradável, enquanto que em *Micro Scooter Challenge* a modificação da segunda lei de Newton resulta no equilíbrio automático entre o nível de dificuldade do game e as habilidades específicas do jogador. Essa experiência indica que o equilíbrio automático do nível de dificuldade deveria ser parte integrante do modelo matemático de games 3D. Vimos também exemplos de que mesmo em games considerados realistas é possível obter resultados positivos com o uso de interfaces pouco usuais, nada realistas, e com alterações do modelo da mecânica clássica. Analisamos a interface pouco realista de *IL-2*, que contorna limitações experimentadas por pilotos na vida real; o *bullet-time* de *Max Payne*, que define, no gameplay, uma nítida impressão de maior velocidade de reação, aumentando a precisão dos movimentos do jogador; e a violação de inércia na antecipação invisível

do personagem de *Incidente em Varginha 2*, que estranhamente confere um aspecto de maior realismo aos movimentos do personagem virtual.

No capítulo 2, vimos que um programa não limita necessariamente os comportamentos de criaturas virtuais a reações previsíveis ou meramente aleatórias. Examinamos a capacidade de se gerar, a partir de uma sequência de instruções, comportamentos imprevisíveis e adaptativos que podem ser coerentes com objetivos definidos. A imprevisibilidade, comum em sistemas caóticos de origem determinista, como o conjunto de Mandelbrot., também pode ser resultado das chamadas máquinas não triviais, propostas por von Foester (1972). Máquinas não triviais têm a capacidade de modificar os próprios estados internos que determinam seu funcionamento e, com isso, geram resultados que não podem ser previstos, apenas simulados. A alteração de estados internos também é a base de funcionamento do algoritmo das máquinas de estados finitos, muito utilizado na programação de comportamentos artificiais em games 3D. Vimos como a alternância entre estados nesses algoritmos, causada por regras de transição, pode criar comportamentos complexos de personagens. A descrição de uma experiência que utilizou algoritmos genéticos no ajuste de variáveis de máquinas de estados finitos, em *Incidente em Varginha*, mostra que é possível obter comportamentos plenamente adaptados às respectivas funções de avaliação. Criaturas artificiais de dois trabalhos artísticos – *Evolved Virtual Creatures*, de Karl Sims, e *Eden*, de Jon McCormack – que usam algoritmos genéticos para sua evolução, também foram apresentadas como exemplos de que é possível, através de programação, gerar comportamentos não previsíveis e, ainda assim, plenamente adaptados a objetivos. Por fim, analisamos uma pergunta de Otto Rössler (1998): *Pode um programa forçar o programador a responder?* Depois de explicar o contexto diferente em que a pergunta foi proposta originalmente, mostramos que uma criatura programada não precisa evoluir a um estágio de consciência para forçar uma pessoa a responder. Isso pode acontecer como resultado das estratégias que emergem do programa para cumprir objetivos pré-determinados. Assim, criaturas de *Eden* desenvolvem estratégias para atrair a atenção dos seres humanos que, apenas com suas presenças, aumentam o alimento disponível para elas. De modo similar, personagens de *Incidente em Varginha* atraem o jogador para emboscadas, cumprindo objetivos estabelecidos nas funções de avaliação de seus programas. Numa perspectiva dos autores desses

dois trabalhos citados, mostramos que a pessoa forçada a responder pode muito bem ser o próprio programador, o que nos leva a responder a pergunta de Rössler com um sim.

No capítulo 3, o contexto desta tese aproxima-se um pouco mais do da pergunta de Otto Rössler. Apresentamos uma breve revisão da mecânica quântica, até chegarmos ao trabalho de Hugh Everett (1956), que propõe uma nova interpretação dessa teoria, considerando a superposição de muitos universos incomunicáveis. A interpretação de Everett visa solucionar um problema da interpretação de Copenhague, que não funciona de modo satisfatório quando da existência de mais de um observador para um mesmo evento quântico. Se todas as possibilidades quânticas realmente acontecem em universos incomunicáveis, como sugere a interpretação de Everett, o que percebemos como incerteza quântica, num único universo, pode ser entendido como determinismo num sistema de múltiplos universos. Esse é o princípio que guia a proposta de um novo modelo matemático para games 3D: a superposição de muitas possibilidades diferentes, que são selecionadas em momentos de interação com o avatar do jogador. O mecanismo de seleção, ou de “colapso” do campo de possibilidades, é proposto segundo princípios similares aos das funções de avaliação dos algoritmos genéticos. Objetivos como o equilíbrio automático entre a dificuldade do game e as habilidades do jogador, ou como a criação de enredos realmente interativos, determinariam as probabilidades desses campos de possibilidades, com a finalidade de tornar mais prováveis as opções que conduzem aos objetivos determinados.

A proposta de linhas gerais para um novo modelo matemático, apresentada no capítulo 3, leva em conta limitações de processamento e memória porque tem como objetivo a construção de um modelo funcional, com tecnologia disponível hoje. De fato, uma sequência óbvia deste trabalho é o desenvolvimento de um software funcional, que já iniciei. O trabalho no software do modelo está em estágio muito incipiente para ser apresentado, mas foi durante esses primeiros passos que algumas restrições de processamento tornaram-se evidentes, gerando informações que ajudaram a criar propostas mais viáveis. Por exemplo, os mapas do game são subdivididos em matriz de posições porque é muito mais rápido processar possibilidades numa matriz, composta por um número limitado de posições, do que

se considerássemos qualquer posição possível no mapa. Pelo mesmo motivo, as probabilidades são limitadas a uma casa decimal, artifício que reduz o máximo número de elementos da matriz de posições a 1.000 por personagem – e isso no caso improvável de um personagem ter seu campo de possibilidades de posição distribuído igualmente em elementos com a probabilidade mínima, de 0.1%. A utilização de gráficos similares aos diagramas de Feynman também surgiu durante o trabalho inicial no software de estados superpostos, pela necessidade prática de visualização de um número muito grande de possibilidades diferentes.

Mas é importante também considerar o que poderemos esperar de um modelo como esse no futuro, sem os mesmos limites de processamento e memória que nortearam a proposta. Talvez essa seja uma imagem mais direta para descrever o que motivou o desenvolvimento do modelo do capítulo 3. Assim como na soma de histórias de Feynman, onde todas as possibilidades precisam ser consideradas para se calcular a probabilidade de uma delas acontecer, um game poderia simular, *simultaneamente*, um grande número de possibilidades diferentes. Um mesmo personagem estaria, por exemplo, em diversos lugares, assumindo estados diferentes, sem que fosse necessário gerenciar campos de possibilidade de posição e de estados. Diversas versões do personagem estariam realmente sendo computadas pelo software do game, que poderia então escolher a melhor opção no momento de qualquer interação com o avatar do jogador. Se compararmos os campos de possibilidade descritos no capítulo 3 com as possibilidades *virtuais* da interpretação de Copenhague, que entram em colapso sob o efeito de uma observação, computar realmente diversas versões dos personagens e escolher a mais apropriada seria equivalente à interpretação dos muitos mundos: as possibilidades estariam realmente ali, mesmo que só pudéssemos acompanhar, através de nossas interações, *uma* linha de realidade. Posições e velocidades possíveis não seriam apenas valores armazenados em matrizes para serem ou não selecionados. Teríamos a capacidade de processamento para calcular um mesmo personagem em diversas opções diferentes, e tudo o que precisaria ser feito em momentos de interação seria determinar qual a melhor opção para aquele momento, para aquele contexto, segundo objetivos pré-determinados.

Desse modo, o modelo de estados superpostos seria, possivelmente, uma simulação muito próxima de uma realidade que não conseguimos perceber através de nossos sentidos, mas que, ainda assim, poderíamos recriar através de software. Um simulador da *interpretação dos muitos mundos* ou, talvez, como implica Rössler (2006), um simulador da *teoria quântica de Everett*. Games 3D tornar-se-iam, então, muito mais que meros produtos de entretenimento. Eles seriam simuladores da realidade num nível difícil, ou mesmo impossível, de ser imaginado. Games que realmente simulem estados superpostos abrigarão personagens extremamente sofisticados, que terão a capacidade de mudar nossa compreensão do universo.

REFERÊNCIAS

Publicações

ABRASH, Michael. *Zen of Graphics Programming: The Ultimate Guide to Writing Fast PC Graphics*. Scottsdale: Coriolis Group, 1994, 750 p.

AHEARN, Luke. *3D Game Textures: Create Professional Game Art using Photoshop*. Oxford: Focal Press, 2006, 369 p.

ANDERS, Peter. *Envisioning cyberspace: Designing 3D electronic spaces*. New York: McGraw-Hill, 1999, 228 p.

ANDERSON, Greg *et al*. *More Tricks of the Game Programming Gurus*. Indianapolis: Sams Publishing, 1995, 1000 p.

BEER, Randall D. "Autopoiesis and Cognition in the Game of Life", In: *Artificial Life*, Vol. 10, No. 3, MIT Press, 2006, pp. 309–326.

BRIGGS, John (1992). *Fractals:The Patterns of Chaos*. London : Thames and Hudson, 1992.. p. 148. ISBN 0500276935, 0500276935.

CHAPMAN, Paul (November 11, 2002), *Life Universal Computer* (http://www.igblan.free-online.co.uk/igblan/ca/). Acessado em 02/10/2011.

CLOSTERMANN, Pierre. *Le Grand Cirque 2000*. Flammarion, 2001, 469 p.

COAD, Peter; NICOLA, Jill. *Object Oriented Programming*. Upper Saddle River: Pearson Education, 1993, 583 p.

COUCHOT, Edmond. *A Tecnologia na Arte: da Fotografia à Realidade Virtual*. Porto Alegre: Editora UFRGS, 2003, 320 p.

COUCHOT, Edmond; HILLAIRE, Norbert. *l'Art Numérique: comment la technologie vient au monde de l'art*. Paris, Editions Flamarion, 2003, 260 p.

CRAWFORD, Chris. *apud* SCHIESEL Seth. "Redefining the Power of the Gamer". *The New York Times*. New York, 7 jun. 2005, Arts.

CSIKSZENTMIHALYI, M., *Finding Flow – The psychology of engagement with everyday life*. New York, Basic Books, 1997.

CUTTING, J. E. “Blowing in the wind: Perceiving structure in trees and bushes”. *Cognition*, v. 12, n. 1, pp. 25–44, 1982.

CUTTING, J. E. “Generation of synthetic male and female walkers through manipulation of a biomechanical invariant”. *Perception*, v. 7, n. 4, pp. 393–405, 1978.

CUTTING, J. E.; KOZLOWSKI, L. T. “Recognizing friends by their walk: Gait perception without familiarity cue”. *Bulletin of the Psychonomic Society*, v. 9, n. 5, pp. 353–356, 1977.

CUTTING, J. E.; PROFFITT, D. R. “Gait perception as an example of how we may perceive events”. In: *Intersensory perception and sensory integration*. WALK, R. D. e PICK, H. L. (ed.). New York: Plenum, 1981, pp. 249–273.

CUZZIOL, M. *Games 3D: aspectos de desenvolvimento*, dissertação de Mestrado, Universidade de São Paulo, São Paulo, 2007, 95 p.

DE PAULA ASSIS, Jesus. “Roteiros em ambientes virtuais interativos”. *Cadernos da Pós-Graduação*, Campinas, v. 3, n. 1, 1999, pp. 93–110.

EVERETT, Hugh. *Theory of the Universal Wavefunction*, Tese, Universidade de Princeton, (1956, 1973), pp. 1–140

FEYNMAN, Richard P. e HIBBS, Albert R. *Quantum Mechanics and Path Integrals – Emended Edition*, Dover, 2010.

FEYNMAN, Richard P. *QED: The Strange Theory of Light and Matter*, Princeton University Press, 1988.

FULLERTON, Tracy; SWAINAND, Christopher; HOFFMAN, Steven. *Game Design Workshop: Designing, Prototyping, and Playtesting Games*. São Francisco: CMP Books, 2004.

GALLAND, Adolf. *The First and the Last*, Mesa, Champlin Museum Press, 1986, 370 p.

GAME Brasilis – Catálogo de jogos eletrônicos brasileiros. São Paulo: Senac, 2003.

GARDNER, Martin (Outubro 1970), *Mathematical Games – The fantastic combinations of John Conway's new solitaire game "life"*. 223. pp. 120–123. ISBN 0894540017

GOMES, R. *Agentes verossímeis: uma investigação sobre a construção dos personagens autônomos nos videogames*, tese de Doutorado, Pontifícia Universidade Católica de São Paulo, São Paulo, São Paulo, 2008, 171 p.

GRAU, Oliver. *Virtual Art: From Illusion to Immersion*. Cambridge: Leonardo Books, 2003, 416 p.

HAWKING, Stephen W. e MLODINOW, Leonard. *The Grand Design*. New York, Bantam Books, 2010 (versão para Kindle).

HOLLAND, J. H. *Adaptation in natural and artificial system*. Ann Arbor: The University of Michigan Press, 1975, 228 p.

JENKINS, Henry. "Art Form for the Digital Age". Technology Review. Cambridge: MIT, set./out. 2000.

JOHANSSON, G. "Visual motion perception". *Scientific American*, v. 232, n. 6. pp. 76–88, 1975.

JOHANSSON, G. "Visual perception of biological motion and a model for its analysis". *Perception and Psychophysics*, v. 14, n. 2, pp. 201–211, 1973.

JOHANSSON, G. *Configuration in event perception*. Uppsala: Almqvist & Wiksell, 1950.

JOHNSON, Steven. *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*. New York: Scribner, 2001, 288 p.

KELLERT, Stephen H. *In the Wake of Chaos: Unpredictable Order in Dynamical Systems*, University of Chicago Press, 1993, p. 32, ISBN 0-226-42976-8.

KELMAN, Nic. *Jeux vidéo: L'art du XXIe siècle*. Paris: Assouline, 2005, p. 319.

LAMOTHE, Andre; RATCLIFF, John; TYLER Denise. *Tricks of the Game Programming Gurus*. Indianapolis: Sams Publishing, 1994, 768 p.

LAMPTON, Christopher. *Gardens of Imagination: Programming 3D Maze Games in C/C++*. Corte Madera: Waite Group Press, 1994, 500 p.

MACHADO, Arlindo. *Máquina e imaginário: o desafio das poéticas eletrônicas*. São Paulo: Edusp, 1993, 316 p.

MANDELBROT, B.B. (1982). *The Fractal Geometry of Nature*. W.H. Freeman and Company..ISBN 0-7167-1186-9.

MANOVICH, Lev. *The language of new media*. Cambridge: MIT Press, 2001, 352 p.

MATEAS, Michael. *An Oz-Centric Review of Interactive Drama and Believable.*Agents. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/papers/CMU-CS-97-156.html#serious_character. 1997. Acessado em 02/10/2011.

McCORMACK, J. “Eden: An Evolutionary Sonic Ecosystem”, in Kelemen, J. & P. Sosík (eds), *Advances in Artificial Life, 6th European Conference, ECAL 2001*, Springer, Prague, Czech Republic, 2001, pp. 133–142.

MCCORMACK, Jon. “Arte Evolucionista: Pirâmides Cósmicas de Baixo para Cima”, In Itaulab (org.), *Emoção art.ficial 5.0 : Autonomia Cibernética*, São Paulo, Itaú Cultural, 2010, pp. 47–59.

MINSKY, Marvin. *Computation: Finite and Infinite Machines*. Upper Saddle River: Prentice Hall, Inc., 1967, 317 p.

MORRISON, Michael A. *Understanding Quantum Physics: A User’s Manual*, Prentice Hall, 1990.

MURRAY, Janet H. *Hamlet on the Holodeck: The future of narrative in cyberspace*. New York: The Free Press, 1997, 326 p.

NEWMAN, James. *Videogames*. London: Routledge, 2004, 198 p.

PRADO, Gilbertto. *Arte Telemática: dos intercâmbios pontuais aos ambientes virtuais multiusuário.* São Paulo: Itaú Cultural, 2003, 125 p.

RENDELL, Paul (January 12, 2005). "A Turing Machine in Conway's Game of Life". http://rendell-attic.org/gol/tm.htm Acessado em 09/10/2011.

ROLLINGS, Andrew; MORRIS, Dave. *Game Architecture and Design*. Scottsdale: Coriolis Group, 2000, 742 p.

RÖSSLER, Otto E., “Can a program force the programmer to reply?”, In SOMMERER, Christa e MIGNONNEAU, Laurent (ed.), *art@science*, New York; Springer – Verlag Wien – New York, 1998, pp. 193–196.

RÖSSLER, Otto E., A Endobolha: “Virando o Submarino Amarelo às Avessas”, In Itaulab (org.), *Emoção art.ficial 3.0 : Interface Cibernética*, São Paulo, Itaú Cultural, 2007, pp. 164–170.

SCHRÖDINGER, Erwin. “The present situation in quantum mechanics”. Versão inglesa de John D. Trimmer do original alemão. In: WHEELER, J. A.; ZUREK, W. H. (Ed). *Quantum Theory and Measurement*. New Jersey: Princeton University Press, 1983, I-11, pp. 152–167.

SIMS, Karl. “Evolving 3D Morphology and Behavior by Competition”. In: BROOKS, R.; MAES, P. (ed.). *Artificial Life IV Proceedings*. Cambridge: MIT Press, 1994, pp. 28–39.

SIMS, Karl. “Evolving Virtual Creatures”. In: *SIGGRAPH 1994. Computer Graphics Annual Conference Series*, jul. 1994, pp. 15–22.

SWINK, Steve. *Game Feel: a Game Designer’s Guide to Virtual Sensation*, Morgan Kaufmann Publishers, 2009, 358 p.

TAVARES, Mônica B. “A Leitura da Imagem Interativa”, in: *XXVI Congresso Brasileiro de Ciência da Comunicação*, 2001.

TONOMURA, J. Endo, T. Matsuda, T. Kawasaki e H. Ezawa, “Demonstration of Single-Electron Buildup of an Interference Pattern”, in: *American Journal of Phisics* 57, 1989, pp. 117–120.

TURING, Alan, *On computable numbers, with an application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, Series 2, 42 (1936), pp. 230–265.

ULLMAN, S. *The interpretation of visual motion*. Cambridge: MIT Press, 1979, 229 p.

VENTURELLI, Suzete. *Arte: espaço_tempo_imagem*. Brasília: Editora UnB, 2004, 186 p.

VON FOERSTER, “Perception of the Future and the Future of Perception”, In: *Instructional Science* 1 (1), pp. 31–43, 1972.

VON FOERSTER, Heinz (ed.). *Cybernetics of Cybernetics*. Urbana: University of Illinois, 1974.

VON NEUMANN, John; BURKS, Arthur W., *Theory of Self-Reproducing Automata*, University of Illinois Press, Urbana e Londres, 1966.

WILLIAMS, Lance. “Pyramidal Parametrics”. In: *SIGGRAPH 1983. Proceedings of the 10th annual conference on Computer graphics and interactive techniques*. New York: ACM Press, jul. 1983. pp. 1–11.

WILLIAMS, Richard. *The Animator's Survival Kit: A Manual of Methods, Principles, and Formulas for Classical, Computer, Games, Stop Motion, and Internet Animators*. London: Faber & Faber, 2002, 352 p.

Obras digitais

Angry Birds Rio, Rovio Mobile, 2011.

Angry Birds, Rovio Mobile, 2009.

Battlefield 1942, EA, 2002.

Call of Duty 2, Infinity Ward, 2005.

Crysis 2, Electronic Arts, 2011.

Crysis, Electronic Arts, 2008.

Delta Force, NovaLogic, 1998.

Doom, Id Software, 1993.

Eden, Jon McCormack, 2000.

Evolved Virtual Creatures, Karl Sims, 1994.

Far Cry, Crytek, 2004.

Half-Life 2, Valve, 2004.

Half-Life, Valve, 1998.

IL-2 Sturmovik, 1C: Madox Games*,* 2001.

Incidente em Varginha 2, Perceptum, inédito.

Incidente em Varginha, Perceptum, 1998.

Iracema Aventura, Perceptum, 2005.

Max Payne, Remedy Entertainment*,* 2001.

Micro Scooter Challenge, Perceptum, 2001.

Pac-Man, Namco, 1980.

Quake, Id Software, 1996.

Rainbow Six, Red Storm Entertainment, 1998.

Sim City, Maxis, 1989.

Sonic the Hedgehog, Sega, 1991.

Super Mario Brothers, Nintendo, 1986.

Super Mini Racing Ice, Perceptum, 2006.

Super Mini Racing, Perceptum, 2001.

Tetris, Alexei Pajitnov, 1984.

The Sims, Maxis, 2000.

Wolfenstein 3D, Id Software, 1992.

Sites

3D Game Studio: http://www.3dgamestudio.com/

3DRad: http://www.3drad.com/

44 Bico Largo: http://www.44bicolargo.com.br/

ABRAGAMES: http://www.abragames.com.br/

Casual Game Design: http://www.casualgamedesign.com/

Délirus: http://www.delirus.com.br/

Devworks: http://www.devworks.com.br/

Espaço: http://www.hades2.com/

Gamasutra: http://www.gamasutra.com/

Game Developer Magazine: http://www.gdmag.com/

Game/AI: http://www.ai-blog.net/

Gamecultura: http://www.gamecultura.com.br/

GarageGames: http://www.garagegames.com/

Genesis3D: http://www.genesis3d.com/

Geo-metriks: http://www.geo-metricks.com/

Journal of Game Development: http://www.jogd.com/subguide.html

Jynx: http://www.jynx.com.br/

Locz: http://www.locz.com.br/loczgames/

Oniria: http://www.oniriasoft.com.br/

Perceptum: http://www.perceptum.com/

Projeto QuickMAN http://quickman.sourceforge.net

SouthLogic: http://www.southlogic.com/

The Game AI Page: http://www.gameai.com/

GLOSSÁRIO

Advergame: game produzido como veículo publicitário. Financiado por anunciantes, é normalmente distribuído de forma gratuita.

Algoritmo genético: ferramenta de design evolutivo baseada nos princípios da seleção natural. Capaz de gerar soluções de forma automática com base em codificação genética. O algoritmo gera uma população de candidatos e seleciona os mais aptos, de acordo com uma função de avaliação pré-definida. Nova população é gerada com base na hibridização, mutação ou clonagem dos códigos genéticos dos selecionados. O processo se repete até que uma solução adequada seja encontrada.

Captura de movimentos: processo de digitalização de movimentos de atores reais para posterior aplicação em personagens virtuais. A captura de movimentos possibilita que um personagem tenha suas articulações animadas exatamente de acordo com dados reais digitalizados, conferindo precisão e realismo razoáveis ao movimento, de forma quase automática.

Comportamento artificial: comportamento de personagens e objetos virtuais gerado por software. Simula ações ou tomadas de decisão e está intimamente ligado à interatividade nos games. Os resultados de diferentes comportamentos artificiais vão das ações mais simples (o abrir e fechar de uma porta, por exemplo) até simulações complexas (como decisões táticas ou estratégicas de personagens virtuais). Entre os algoritmos mais utilizados no comportamento artificial estão as máquinas de estados finitos.

Emergência: geração de ordem e complexidade a partir da interação constante entre elementos que compõem um sistema. Um exemplo clássico de emergência é o cérebro humano, formado por bilhões de células relativamente simples que se comunicam entre si e de onde emergem o raciocínio e a consciência.

Endofísica: a “física de dentro”, campo que estuda a influência do observador sobre os sistemas observados, desenvolvido principalmente por Otto Rössler. De certo modo, Rössler trabalha em oposição à física mais tradicional, que normalmente isola o observador do sistema observado.

Engine (ou Game Engine): componente de software utilizado em games e outras aplicações gráficas em tempo real. Programa responsável principalmente pela geração de imagens, pode incluir o gerenciamento de sons, animações e física.

Fog: técnica que reproduz em ambientes virtuais efeitos semelhantes à neblina real. Basicamente o algoritmo de fog mescla uma cor com os pixels da imagem, de forma progressiva, dependendo de sua distância ao observador.

Frame: em games 3D, um *frame* é uma imagem do ambiente virtual gerada a partir do ponto de vista principal. Normalmente são gerados entre 20 e 60 *frames* por segundo para garantir a interação com personagens e objetos em tempo real.

Gameplay: defino *gameplay* como o conjunto de experiências interativas entre o jogador e o game, entre ser humano, hardware e software. Gameplay envolve não apenas a experiência do ser humano com o game, mas também a capacidade que um game tem de dirigir ações em resposta às do ser humano, através de comportamentos artificiais atribuídos a objetos e personagens virtuais.

Level designer: profissional responsável pela construção de níveis e mapas de jogo em games 3D.

Lógica fuzzy: lógica que trabalha com valores intermediários entre "sim" e "não", possibilitando aproximações e raciocínios que não seriam possíveis com a lógica tradicional, ou booleana.

Mapeamento de normais: em sua estrutura de dados, a textura de mapeamento normal é uma imagem comum, mas as componentes R, G e B são interpretadas como coordenadas de vetores normais, não como pixels. É possível então calcular como a luz é refletida em cada elemento da matriz e conferir uma impressão muito realista de relevo a uma superfície virtualmente plana.

Mapeamento de texturas: coordenadas que definem a orientação de aplicação de textura sobre os polígonos que constituem um modelo 3D.

Mapeamento MIP: técnica utilizada para reduzir erros de *aliasing* gerados por texturas. Emprega várias versões da mesma textura em dimensões diferentes. Quanto mais próxima a distância da textura ao ponto de observação, maior a dimensão da versão exibida.

Máquina de estados finitos: algoritmo utilizado na criação de comportamento artificial. Seleciona entre uma série de estados pré-definidos com base em regras de transição e informações sobre o ambiente virtual.

Níveis de detalhe: técnica de otimização que emprega várias versões de um modelo 3D com resoluções (contagem de polígonos) diferentes.

Observáveis: propriedades do estado de um sistema quântico que podem ser determinadas através de uma sequência de operações físicas. Também chamados de "estados fisicamente realizáveis".

Pixel: unidade básica de imagens digitais. Contração de *picture element*, representa um pequeno retângulo de cor na matriz digital da imagem.

Polígonos: blocos construtores dos modelos 3D poligonais. Game Engines e placas aceleradoras normalmente lidam com a forma mais simples de polígono, o triângulo (qualquer polígono pode ser subdividido em triângulos).

Renderização de imagens: processo que converte os modelos numéricos que compõem um ambiente virtual na matriz de pixels que será exibida como imagem.

Sprites: figuras criadas em mapa de bits. Normalmente representam personagens ou objetos nos ambientes virtuais que se utilizam do algoritmo raycaster ou em games 2D.

APÊNDICE 1:
AUTONOMIA CIBERNÉTICA

A ideia de *autonomia cibernética*, que guiou três edições da bienal internacional de arte e tecnologia do Itaú Cultural, influenciou a construção desta tese. As exposições foram concebidas pelo Itaulab, núcleo que eu coordenava no período (hoje renomeado como núcleo de inovação), responsável pela área de arte e tecnologia da instituição. Descrevo abaixo tanto a ideia como seu processo de desenvolvimento, realizado de forma colaborativa principalmente por três pessoas, Paul Pangaro (ciberneticista americano), Guilherme Kujawski (do Itaú Cultural) e eu próprio.

Autonomia em sistemas deterministas?

A sequência de três exposições *Emoção Art.ficial* iniciada em 2006 (*Interface Cibernética*, 2006; *Emergência!*, 2008; e *Autonomia Cibernética*, 2010) reflete, em sua concepção, parte importante das pesquisas do núcleo de mídias interativas do Itaú Cultural, Itaulab: a busca pelo que as novas tecnologias realmente trazem de novo quando aplicadas ao campo artístico.

Interatividade – palavra que, de tão utilizada, perdeu muito de seu sentido objetivo original – é frequentemente apontada como principal novidade. O termo não está restrito às “novas” tecnologias – para que exista interação bastam sistemas que ajam entre si –, mas processadores e softwares progressivamente mais velozes permitiram o desenvolvimento de sistemas que interagem de modo cada vez mais complexo e interessante. Não há duvida de que a interatividade tenha propiciado o desenvolvimento de novas poéticas, mas uma investigação mais científica e menos baseada no mero aspecto lúdico desses processos parecia-nos necessária. Para tanto, a cibernética foi escolhida como ferramenta de análise.

Interface Cibernética

A cibernética trata de sistemas que agem para atingir objetivos. Os elementos de tais sistemas podem ser denominados *agentes* e não há distinção entre elementos biológicos, maquínicos, reais ou virtuais nessa denominação: robôs, trechos de software, células e organismos biológicos são exemplos de agentes. São entidades que, por definição, podem agir sobre seu ambiente, mas que, para atingir seus objetivos, precisam estabelecer algum tipo de comunicação com ele. Um termostato de aquecedor é frequentemente utilizado como exemplo de agente num sistema cibernético. O dispositivo é formado por um sensor de temperatura, um comparador (que mede o erro entre a temperatura ambiente e o objetivo) e um contato de relê que liga ou desliga o aquecedor. O sensor e o contato moldam a interface desse agente: definem o que pode ser percebido e as ações que ele pode ter em relação ao ambiente. O comparador do termostato processa duas regras simples para que o objetivo seja mantido: “se a temperatura for maior que o objetivo, então desligue o aquecimento” e “se a temperatura for menor que o objetivo, então ligue o aquecimento”.

De fato, interações entre agentes exigem as três etapas do ciclo cibernético: monitoramento, processamento e ação. Frequentemente essas etapas são contínuas, ou seja, não são interrompidas pela etapa seguinte. O monitoramento pode continuar a ser efetuado enquanto dados anteriores estão sendo processados, assim como uma ação não interrompe necessariamente o processamento de outras informações, por exemplo. O processamento envolve aplicação de regras, que podem ser tão simples quanto as executadas por uma célula virtual do Game of Life, de John Conway (ver capítulo 2), ou tão complexas quanto os diversos intercâmbios moleculares que ocorrem no interior de uma célula biológica.

Um termostato é um agente que trabalha cegamente para manter baixo o erro entre a temperatura medida e a estabelecida como objetivo. Ele continuará fazendo exatamente a mesma coisa se o seu sensor estiver desregulado, mostrando temperaturas erradas, ou se o aquecedor deixar de funcionar. Trata-se de um sistema cibernético de primeira ordem.

Mas quem ajusta a temperatura objetivo de um termostato? Outro agente, normalmente uma pessoa. Essa pessoa também tem algum objetivo, e age para atingi-lo e mantê-lo. A temperatura ambiente é percebida por sensores orgânicos da pele, uma comparação é feita dependendo do nível de conforto causado pela temperatura atual e a ação de regular o termostato para cima ou para baixo pode seguir como consequência. Pessoa e termostato seguem, dessa maneira, ciclos cibernéticos similares, mas distribuídos hierarquicamente: o ser humano observa o sistema como um todo e age sobre as regras do termostato quando acha necessário. Dessa maneira, os dois agentes formam um sistema cibernético de segunda ordem – o humano como observador e o conjunto termostato/ambiente como observado.

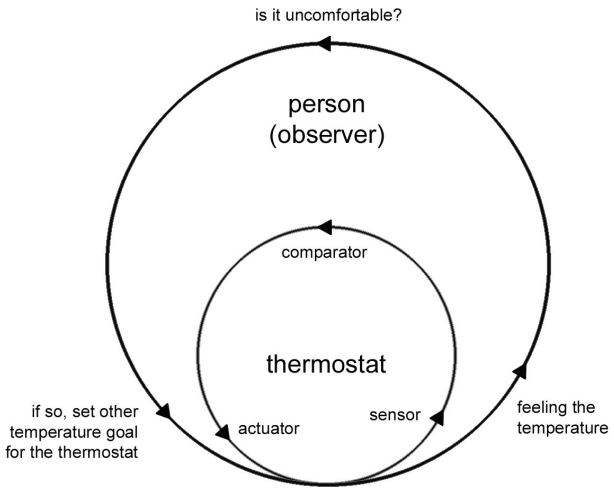


Figura A1.1. Exemplo de sistema cibernético de segunda ordem.

Emergência!

Da aplicação de regras e das interações entre um número suficientemente grande de agentes, ainda que limitados a ciclos cibernéticos de primeira ordem, comportamentos complexos podem emergir. Autômatos celulares como o *Game of Life*, descrito no capítulo 2, podem exibir grupos de células que se comportam como indivíduos, com base apenas na aplicação de regras muito simples. Como exemplo biológico de comportamento emergente, uma colônia de formigas emprega sempre um número eficiente de insetos para recolher determinado alimento como consequência da aplicação de regras individuais para a distribuição de feromônios.

Tais comportamentos não são previsíveis, mas, uma vez identificados, são diretamente explicáveis com base nas regras executadas pelos agentes. Em outras palavras, não existe mágica no fenômeno da emergência. O que existe é uma aparente incapacidade humana para prever resultados de interações simples em tão grande número.

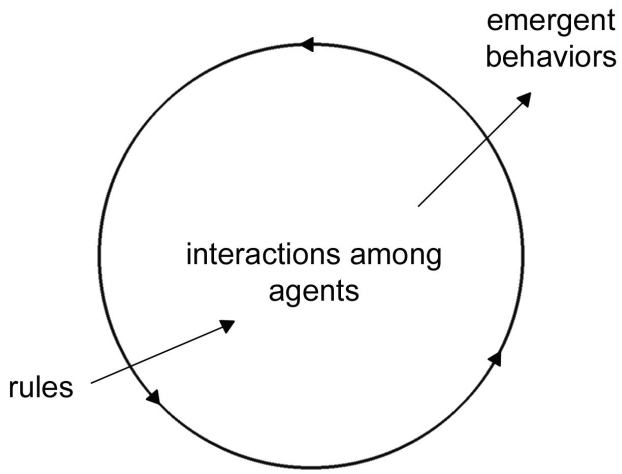


Figura A1.2. Emergência de comportamentos com base em regras de interação.

Conforme ilustrado na figura 2, a emergência propriamente dita é um fenômeno fortuito. Comportamentos emergentes não são adaptativos, pelo menos não por conta própria. Isso se deve à ausência de um laço de realimentação entre os resultados emergentes e as regras que os causam. Deve-se também à ausência de um agente que funcione como observador, capaz de comparar os resultados emergentes com um objetivo maior, de segunda ordem, para então agir apropriadamente sobre as regras do sistema.

Além da emergência

Várias obras do ciclo de três exposições Emoção Art.ficial, iniciado em 2006, extrapolam as limitações descritas acima. *Evolving Sonic Environments* (Usman Haque e Robert Davis, 2006), *Performative Ecologies* (Ruairi Glynn, 2007) e *Silent Barrage* (Guy Ben-Ary, Philip Gamblen, Peter Gee, Nathan Scott, Brett Murray, Dr. Steve Potter, 2009) são alguns exemplos. Duas obras em particular, *Eden* (Jon

McCormack, 2000) e *Evolving Virtual Creatures* (Karl Sims, 1994), ambas descritas no capítulo 2, ilustram alguns processos comuns nesses diversos trabalhos.

Eden apresenta um exemplo particularmente interessante de emergência dirigida que se desenvolve num ecossistema, gerando criaturas que parecem “saber” o que estão fazendo. Não faz sentido atribuir intencionalidade a criaturas tão simples, mas as estratégias que são evoluídas pelo algoritmo funcionam, e é exatamente por funcionarem que as regras mais adaptadas perpetuam-se entre a população de agentes.

Evolving Virtual Creatures aplica algoritmos genéticos para a evolução de criaturas que se movimentam num espaço virtual 3D, sujeitas a simulação física. O resultado são criaturas que se movimentam de forma orgânica sem que tenham sido explicitamente programadas para isso, quase como se soubessem o que estão fazendo. Assim como em *Eden*, somos tentados a atribuir intencionalidade às criaturas de Karl Sims, mesmo sabendo que isso não deveria ser minimamente possível.

Autonomia cibernética

Os dois exemplos extrapolam o fenômeno da emergência. Existe um novo agente (nos dois casos, parte do software) que monitora resultados emergentes, compara-os com objetivos gerais e age selecionando novas regras (ou novos agentes) cujos resultados mais se aproximem dos objetivos, realimentando assim todo o processo através de um ciclo cibernético de segunda ordem. Para diferenciar esse processo realimentado do fenômeno da emergência, denominemos como *autonomia cibernética* a capacidade de um sistema cibernético evoluir comportamentos através da geração de novas regras (do grego “autonomia”: a capacidade de fazer suas próprias leis). Lembremos, entretanto, que os resultados desse processo, por serem emergentes, são virtualmente imprevisíveis e não estão restritos à geração de novas regras. Novos agentes e novos grupamentos funcionais de agentes, por exemplo, podem também ser resultados de emergência num ciclo cibernético de segunda ordem (figura 3).

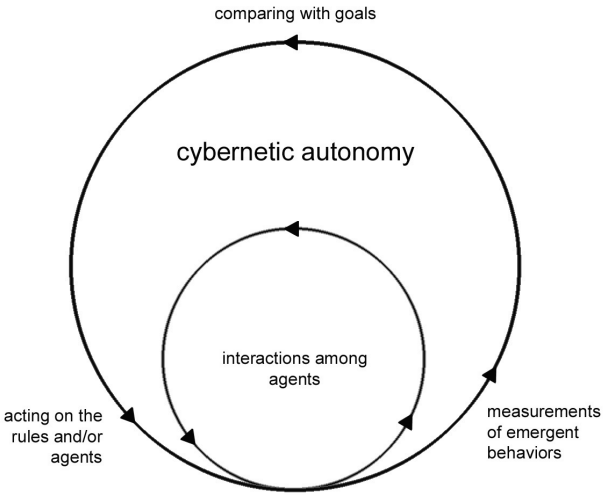


Figura A1.3. Evolução de comportamentos emergentes em sistema cibernético de segunda ordem.

O próprio software atua como observador do sistema que apresenta emergência, fato que reforça a característica cibernética de segunda ordem. Em *Eden*, os objetivos de segunda ordem estão embutidos no algoritmo genético: adaptar-se e procriar (ou, numa formulação mais adequada à poética do trabalho: “crescei e multiplicai-vos”). As regras que prevalecem pertencem às criaturas mais adaptadas, que sobrevivem o suficiente para gerar uma descendência. Inicialmente, essas regras são aleatórias. Entretanto, após sucessivas interações de segunda ordem, as novas regras autonomamente cumprem objetivos diversos de primeira ordem: movimentar-se, alimentar-se, poupar energia, acasalar, etc., sem que seja necessária a intervenção de um programador. Em *Evolving Virtual Creatures*, os objetivos de segunda ordem são mais diretos: “deslocar-se o máximo possível” é um deles. Mas tanto as regras de movimentação como a morfologia das criaturas são controladas pelo código genético. Esses códigos são aleatórios a princípio e desenvolvem-se para atender os objetivos de segunda ordem. Qual o formato das criaturas e como elas devem movimentar seus membros são resultados que emergem para atender esses objetivos, novamente sem a intervenção de um programador ou designer.

Ao contrário do fenômeno da emergência, os comportamentos de sistemas que apresentam autonomia cibernética não podem ser explicados diretamente com base em regras iniciais, programadas. Tanto *Eden* como *Evolving Virtual Creatures* partem de regras aleatórias, o que sugere que as regras iniciais não sejam

fundamentais para a complexidade de resultados das diferentes linhas evolutivas. É preciso levar em conta que regras e interações entre agentes evoluem com o tempo a partir de outro ciclo cibernético, que direciona a adaptação do sistema a objetivos de segunda ordem, mais abertos e menos específicos, a ponto de que nos surpreendamos (na verdade, de que os próprios autores das obras se surpreendam) com as estratégias desenvolvidas pelo sistema para atingi-los.

Talvez mais surpreendente seja o fato de que esses resultados decorram, nos dois exemplos, de meras sequências lógicas de instruções, nada mais que disso. Softwares que geram novas sequências de instruções, que conduzem à emergência e à adaptabilidade de resultados complexos sem a necessidade de programação explícita e detalhada. Softwares que, como reflexos simplificados de organismos biológicos autônomos, geram, testam e selecionam suas próprias regras.

Notas sobre o desenvolvimento da ideia

Como mencionado na abertura deste apêndice, a ideia de autonomia cibernética foi detalhada em conjunto com Paul Pangaro e Guilherme Kujawski. Por exemplo, as figuras deste apêndice, criadas por mim, estão em inglês porque foram utilizadas em trocas de email com Paul Pangaro e Guilherme Kujawski. Essas trocas de email foram numerosas entre os anos de 2006 e 2010.

A decisão de realizar um ciclo de três exposições (ao qual nos referíamos como “trilogia cibernética”) ocorreu ainda durante os trabalhos de preparação de Emoção Art.ficial 3.0 – Interface Cibernética, em 2006. Eu havia proposto uma linha de raciocínio que descrevia o aumento de complexidade em sistemas deterministas com base em agentes e regras. Essa linha de raciocínio tinha três etapas: Primeira: a interação entre diversos agentes deterministas, que obedecem a regras simples, pode conduzir ao fenômeno da emergência. Segunda: o fenômeno da emergência pode resultar em novas regras, e até mesmo em novos agentes. Terceira: regras emergentes, através de um ciclo cibernético, podem evoluir de acordo com objetivos de segunda ordem. Sugeri então que essa capacidade de criação de regras próprias poderia ser denominada de “autonomia cibernética”. Muito dessa proposta veio da análise de obras de arte e tecnologia e de minha experiência com comportamentos

artificiais de personagens de games 3D, ambas já discutidas no capítulo 2. Nas trocas de email com Pangaro e Kujawski, descrevi essas etapas através do que eles viriam a chamar informalmente de “as equações do Marcos”:

Interação entre agentes + regras simples => Emergência!

Interação entre agentes + regras emergentes => Autonomia Cibernética

A partir desse esboço, a ideia foi ganhando força em nossas trocas de email, até assumir sua forma final. Como exemplo, durante a exposição de 2008, sobre o fenômenos da emergência, estávamos discutindo a próxima, sobre autonomia cibernética, e Paul Pangaro iniciava, na época, uma longa e produtiva discussão sobre escopo e objetivos através do seguinte email:

> **De:** Paul Pangaro
> **Enviada em:** quinta-feira, 17 de julho de 2008 00:52
> **Para:** Marcos Fernandez Cuzziol
> **Cc:** Paul Pangaro; Guilherme Kujawski Ramos
> **Assunto:** Re: RES: Food for thought
>
> fantastic
>
> now for the hardest part: deciding scope, agreeing on goals
>
> for some reason the first thing that came to mind is marcos' 'equations' which may be the best framework for discussing interactive art in cybernetics terms. the progression of interaction/emergence/autonomy is a great classification scheme... but...
>
> what will be our major 'point' or 'thrust'? what are the key insights we want to convey?[58]

Quando começamos a conversar sobre escopo e objetivos, tínhamos ainda aproximadamente dois anos de trabalho antes da abertura prevista para a exposição de 2010. Considero um privilégio ter trabalhado colaborativamente com Pangaro e Kujawski, durante tanto tempo, sobre o conceito dessas exposições.

---

[58] Paul Pangaro, email pessoal recebido em 17/07/2008

Uma influência do desenvolvimento da ideia de autonomia cibernética sobre a presente tese foi a comprovação de que sequências de instruções não limitavam, necessariamente, a complexidade que um sistema programado pode apresentar. Isso reforçou minha crença de que talvez a limitação não estivesse no programa, mas no modelo matemático que guia a criação do programa.

O uso do qualificador em “autonomia cibernética” visava originalmente diferenciar a ideia do termo muito mais amplo, autonomia, bem como explicitar o *framework* utilizado: a própria cibernética. Mas não pude evitar de ponderar que, se algum tipo de autonomia fosse possível em sistemas deterministas, talvez a realidade, como a conhecemos, também fosse, de alguma forma, determinista. Pesquisas nesse sentido levaram-me à interpretação de Everett da mecânica quântica, que se desenvolveu na interpretação dos muitos mundos, descrita no capítulo 3. Se todas as opções possíveis realmente acontecem em universos superpostos, então esse “super sistema” composto de múltiplos universos pode ser determinista, enquanto que a impressão de aleatoriedade dos fenômenos quânticos passaria a ser mera percepção das várias versões de nossas consciências, que experimentam, cada uma, apenas uma linha de possibilidades específica. Percebi que um sistema como esse seria passível de simulação, e que isso poderia estender o conceito de autonomia cibernética. A noção de que talvez a autonomia humana seja apenas um epifenômeno, resultado de superposições das diversas escolhas possíveis que se realizam em universos diferentes, é realmente assustadora – mas ela serviu de base para a proposta do modelo matemático para games 3D descrita nesta tese.

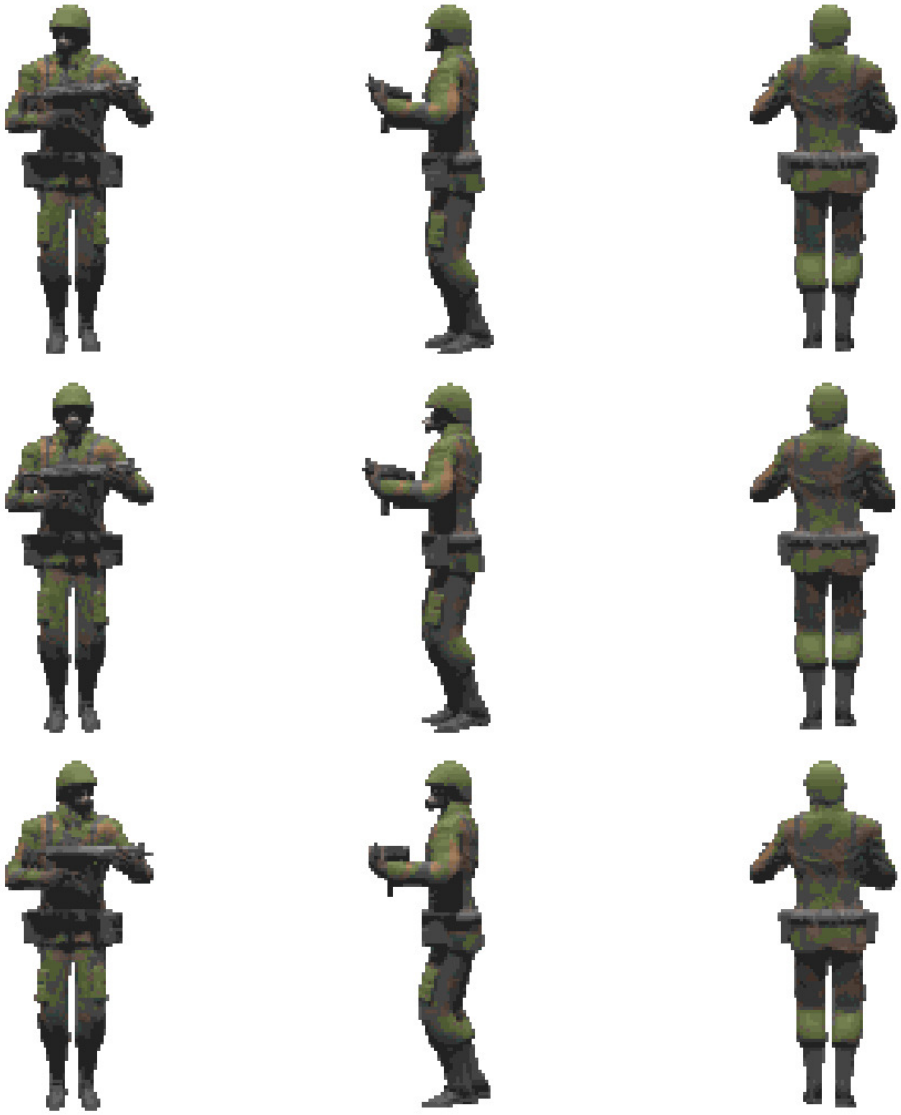
APÊNDICE 2:
DEFINIÇÃO COMPLETA DE PERSONAGEM

Trooper (soldado)

Incidente em Varginha, 1998

Simulação 3D em sprites com animação em frames individuais

Máquina de Estados escrita em C-Script para Acknex game engine

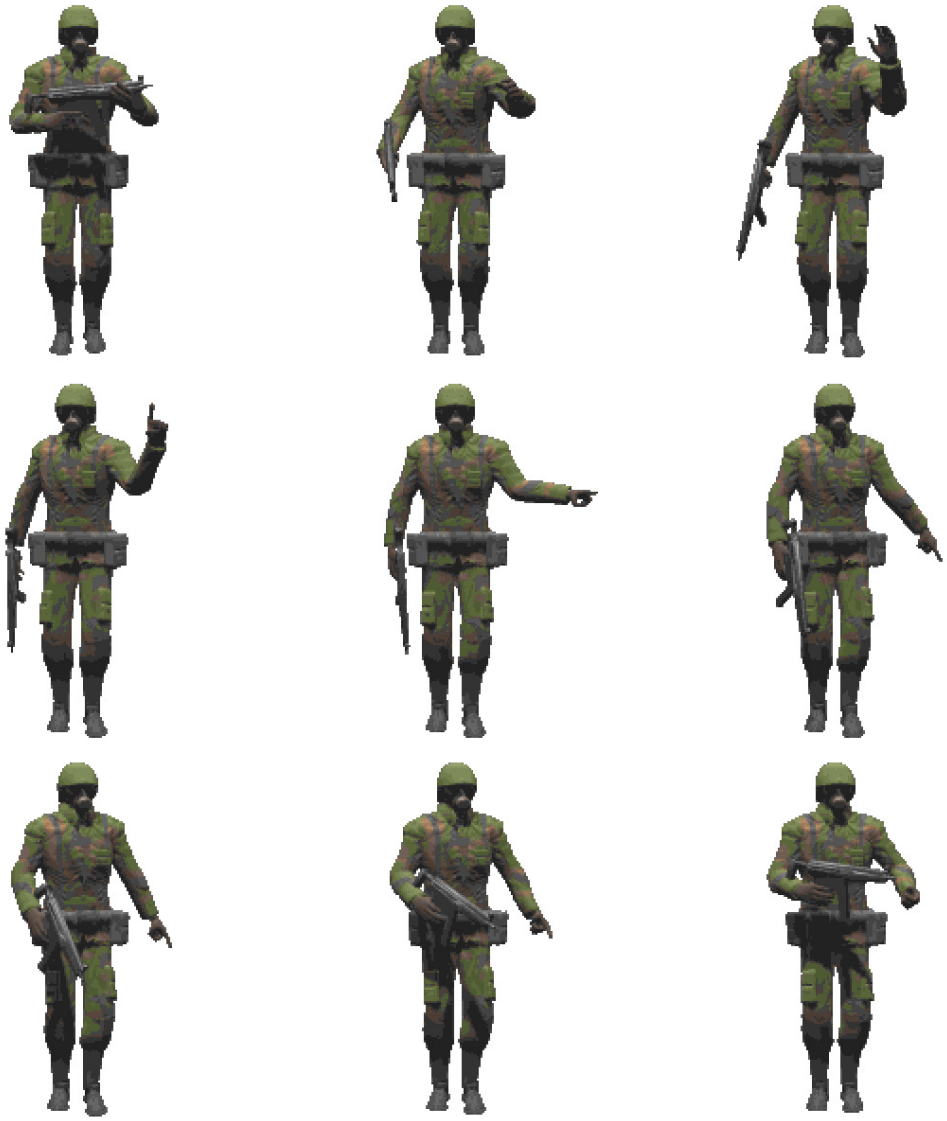


Trop0g.pcx

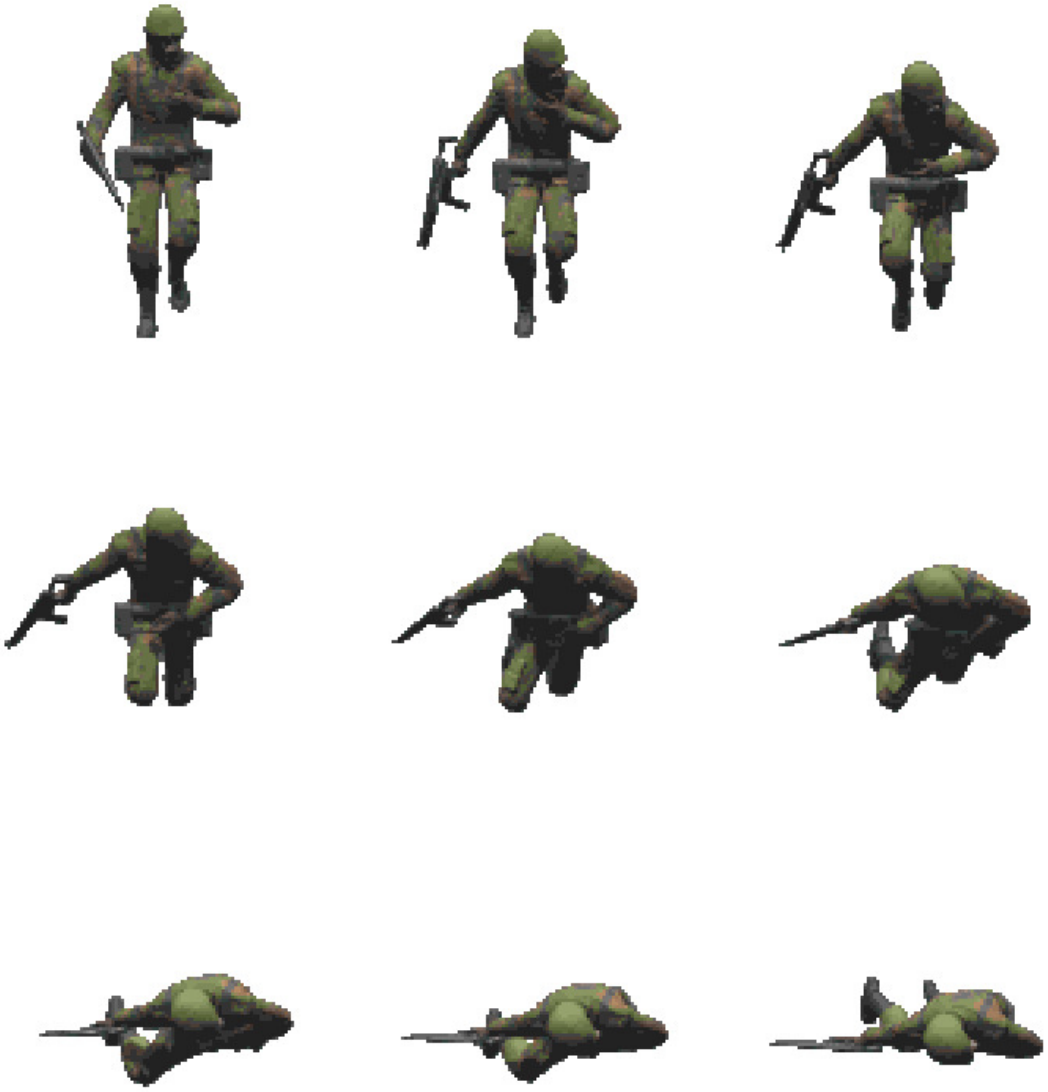
Trop1g.pcx

Trop3g.pcx

Trop5g.pcx



Trop7g.pcx



Trop8g.pcx

TropAg.pcx

```
//////////////////////////////////////////////////////////////////
// Trooper Actor
// Copyright © 1997 Perceptum Informática Ltda
// All Rights Reserved
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
// Bitmaps
//////////////////////////////////////////////////////////////////

BMAP Trop0001, <Trop0g.pcx>,  35,   5,  51, 111;
BMAP Trop0002, <Trop0g.pcx>, 153,   5,  55, 111;
BMAP Trop0003, <Trop0g.pcx>, 275,   5,  51, 110;
BMAP Trop0101, <Trop0g.pcx>,  35, 125,  51, 111;
BMAP Trop0102, <Trop0g.pcx>, 152, 125,  57, 111;
BMAP Trop0103, <Trop0g.pcx>, 275, 125,  51, 110;
BMAP Trop0201, <Trop0g.pcx>,  35, 245,  51, 111;
BMAP Trop0202, <Trop0g.pcx>, 154, 245,  53, 110;
BMAP Trop0203, <Trop0g.pcx>, 275, 245,  51, 110;

BMAP Trop1001, <Trop1g.pcx>,  38,  10,  45, 106;
BMAP Trop1002, <Trop1g.pcx>, 157,  10,  47, 104;
BMAP Trop1003, <Trop1g.pcx>, 271,  10,  59, 106;
BMAP Trop1004, <Trop1g.pcx>, 391,  10,  59, 107;
BMAP Trop1005, <Trop1g.pcx>, 518,  10,  45, 107;
BMAP Trop1101, <Trop1g.pcx>,  38, 125,  45, 110;
BMAP Trop1102, <Trop1g.pcx>, 163, 125,  35, 110;
BMAP Trop1103, <Trop1g.pcx>, 278, 125,  45, 109;
BMAP Trop1104, <Trop1g.pcx>, 392, 125,  57, 109;
BMAP Trop1105, <Trop1g.pcx>, 519, 125,  43, 109;
BMAP Trop1201, <Trop1g.pcx>,  38, 250,  45, 106;
BMAP Trop1202, <Trop1g.pcx>, 162, 250,  37, 105;
BMAP Trop1203, <Trop1g.pcx>, 272, 250,  57, 104;
BMAP Trop1204, <Trop1g.pcx>, 391, 250,  59, 106;
BMAP Trop1205, <Trop1g.pcx>, 519, 250,  43, 107;
BMAP Trop1301, <Trop1g.pcx>,  38, 365,  45, 110;
BMAP Trop1302, <Trop1g.pcx>, 163, 365,  35, 111;
BMAP Trop1303, <Trop1g.pcx>, 278, 365,  45, 111;
BMAP Trop1304, <Trop1g.pcx>, 392, 365,  57, 110;
BMAP Trop1305, <Trop1g.pcx>, 518, 365,  45, 109;

BMAP Trop3001, <Trop3g.pcx>,  36,  11,  49, 106;
BMAP Trop3101, <Trop3g.pcx>, 158,  14,  45, 104;
BMAP Trop3201, <Trop3g.pcx>, 280,  16,  41, 103;
BMAP Trop3301, <Trop3g.pcx>,  43, 137,  35, 102;
BMAP Trop3401, <Trop3g.pcx>, 162, 136,  37, 103;
BMAP Trop3501, <Trop3g.pcx>, 282, 136,  37, 103;

BMAP Trop5001, <Trop5g.pcx>,  25,  12,  71, 106;
BMAP Trop5002, <Trop5g.pcx>, 155,  13,  51, 103;
BMAP Trop5003, <Trop5g.pcx>, 269,  13,  63, 104;

BMAP Trop7001, <Trop7g.pcx>,  58,   8,  85, 185;
BMAP Trop7101, <Trop7g.pcx>, 258,   8,  85, 185;
BMAP Trop7201, <Trop7g.pcx>, 448,   8, 105, 186;
BMAP Trop7301, <Trop7g.pcx>,  53, 208,  95, 187;
BMAP Trop7401, <Trop7g.pcx>, 226, 208, 149, 187;
BMAP Trop7501, <Trop7g.pcx>, 444, 208, 113, 187;
BMAP Trop7601, <Trop7g.pcx>,  51, 408,  99, 187;
BMAP Trop7701, <Trop7g.pcx>, 250, 408, 101, 186;
BMAP Trop7801, <Trop7g.pcx>, 456, 408,  89, 185;
```

```
BMAP Trop8001, <Trop8g.pcx>,  33,   9,  55, 106;
BMAP Trop8101, <Trop8g.pcx>, 141,  17,  79,  98;
BMAP Trop8201, <Trop8g.pcx>, 255,  27,  91,  86;
BMAP Trop8301, <Trop8g.pcx>,  10, 169, 101,  63;
BMAP Trop8401, <Trop8g.pcx>, 133, 177,  95,  57;
BMAP Trop8501, <Trop8g.pcx>, 256, 187,  89,  47;
BMAP Trop8601, <Trop8g.pcx>,  19, 317,  83,  36;
BMAP Trop8701, <Trop8g.pcx>, 136, 320,  89,  32;
BMAP Trop9001, <Trop8g.pcx>, 253, 318,  95,  28;

BMAP TropA001, <TropAg.pcx>,  36,  21,  49,  94;
BMAP TropA002, <TropAg.pcx>, 161,  21,  39,  95;
BMAP TropA003, <TropAg.pcx>, 276,  21,  49,  95;
BMAP TropA004, <TropAg.pcx>, 388,  21,  65,  94;
BMAP TropA005, <TropAg.pcx>, 516,  21,  49,  93;
BMAP TropA101, <TropAg.pcx>,  33, 144,  55,  92;
BMAP TropA102, <TropAg.pcx>, 157, 144,  47,  90;
BMAP TropA103, <TropAg.pcx>, 271, 144,  59,  92;
BMAP TropA104, <TropAg.pcx>, 388, 144,  65,  93;
BMAP TropA105, <TropAg.pcx>, 515, 144,  51,  93;
BMAP TropA201, <TropAg.pcx>,  36, 261,  49,  94;
BMAP TropA202, <TropAg.pcx>, 161, 261,  39,  94;
BMAP TropA203, <TropAg.pcx>, 276, 261,  49,  93;
BMAP TropA204, <TropAg.pcx>, 388, 261,  65,  93;
BMAP TropA205, <TropAg.pcx>, 516, 261,  49,  93;
BMAP TropA301, <TropAg.pcx>,  37, 384,  47,  92;
BMAP TropA302, <TropAg.pcx>, 162, 384,  37,  91;
BMAP TropA303, <TropAg.pcx>, 272, 384,  57,  90;
BMAP TropA304, <TropAg.pcx>, 389, 384,  63,  92;
BMAP TropA305, <TropAg.pcx>, 516, 384,  49,  93;

//////////////////////////////////////////////////////////////////
// Sounds
//////////////////////////////////////////////////////////////////
SOUND Trop01Snd, <trop01.wav>; // "Ok Charlie Delta Tango..."
SOUND Trop02Snd, <trop02.wav>; // "Over there..."
SOUND Trop03Snd, <trop03.wav>; // "Get down!"
SOUND Trop04Snd, <trop04.wav>; // MP5 firirng
SOUND Trop05Snd, <trop05.wav>; // "Uh!"
SOUND Trop06Snd, <trop06.wav>; // "Help me over here!"
SOUND Trop07Snd, <trop07.wav>; // "This is a restricted area!"
SOUND Trop08Snd, <trop08.wav>; // death
SOUND Trop09Snd, <trop09.wav>; // "X-ray Tango! Come in, over"
SOUND Trop10Snd, <trop10.wav>; // "He's over there"
SOUND Trop11Snd, <trop11.wav>; // "Son of a..."
SOUND Trop12Snd, <trop12.wav>; // "Hey, scumbag!"

//////////////////////////////////////////////////////////////////
// Animated Textures
//////////////////////////////////////////////////////////////////
TEXTURE Trop0Tex {
        SCALE_XY        23,23;
        SIDES           4;
        CYCLES          4;
        BMAPS   Trop0001, Trop0101, Trop0201, Trop0101,
                Trop0002, Trop0102, Trop0202, Trop0102,
                Trop0003, Trop0103, Trop0203, Trop0103,
                Trop0002, Trop0102, Trop0202, Trop0102;
        DELAY   8, 8, 8, 8;
        RANDOM  0.7;
```

```
        MIRROR  0,0,0,1;
        FLAGS   CLIP;
}
TEXTURE Trop0aTex {
        SCALE_XY        23,23;
        SIDES           4;
        CYCLES          3;
        BMAPS   Trop0001, Trop0101, Trop0201,
                Trop0002, Trop0102, Trop0202,
                Trop0003, Trop0103, Trop0203,
                Trop0002, Trop0102, Trop0202;
        DELAY   4, 4, 4;
        MIRROR  0,0,0,1;
        RADIANCE 0.7;
        FLAGS   CLIP;
}
TEXTURE Trop1Tex {
        SCALE_XY        23,23;
        SIDES           8;
        CYCLES          4;
        BMAPS   Trop1001, Trop1101, Trop1201, Trop1301,
                Trop1002, Trop1102, Trop1202, Trop1302,
                Trop1003, Trop1103, Trop1203, Trop1303,
                Trop1004, Trop1104, Trop1204, Trop1304,
                Trop1005, Trop1105, Trop1205, Trop1305,
                Trop1204, Trop1304, Trop1004, Trop1104,
                Trop1203, Trop1303, Trop1003, Trop1103,
                Trop1202, Trop1302, Trop1002, Trop1102;
        DELAY   4, 5, 4, 5;
        MIRROR  0,0,0,0,0,1,1,1;
        FLAGS   CLIP;
}
TEXTURE Trop3Tex {
        SCALE_XY        23,23;
        SIDES           1;
        CYCLES          9;
        BMAPS   Trop3301, Trop3401,
                Trop3301, Trop3401,
                Trop3301, Trop3401,
                Trop3301, Trop3401,
                Trop3501;
   SOUND   Trop04Snd;
   SDIST   300;
   SVOL    0.4;
   SVDIST  25;
   DELAY   2, 2, 2, 2, 2, 2, 2, 3, 4;
   SCYCLES 0, 1, 0, 0, 0, 0, 0, 0, 0;
   FLAGS   CLIP;
   AMBIENT 0.2;
}
TEXTURE Trop3aTex {
        SCALE_XY        23,23;
        SIDES           1;
        CYCLES          3;
        BMAPS   Trop3001, Trop3101, Trop3201;

        DELAY   4, 3, 3;
        FLAGS   CLIP;
}
TEXTURE Trop5Tex {
        SCALE_XY        23,23;
```

```
        SIDES           4;
        CYCLES          1;
        BMAPS   Trop5001, Trop5002, Trop5003, Trop5002;
        MIRROR  0, 0, 0, 1;
        FLAGS   CLIP;
}
TEXTURE Trop5aTex {
        SCALE_XY        23,23;
        SIDES           4;
        CYCLES          1;
        BMAPS   Trop5001, Trop5002, Trop5003, Trop5002;
        MIRROR  1, 1, 1, 0;
        FLAGS   CLIP;
}
TEXTURE Trop7Tex {
        SCALE_XY        39,39;
        SIDES           1;
        CYCLES          9;
        BMAPS   Trop7001, Trop7101, Trop7201, Trop7301,
                Trop7401, Trop7501, Trop7601, Trop7701,
                Trop7801;
        DELAY   5, 5, 5, 5, 5, 5, 5, 5, 5;
        SOUND   Trop07Snd;
        SDIST   300;
        SVDIST  25;
        SCYCLES 0, 1, 0, 0, 0, 0, 0, 0, 0;
        FLAGS   CLIP;
}
TEXTURE Trop7aTex {
        SCALE_XY        39,39;
        SIDES           1;
        CYCLES          9;
        BMAPS   Trop7001, Trop7101, Trop7201, Trop7301,
                Trop7401, Trop7501, Trop7601, Trop7701,
                Trop7801;
        DELAY   5, 5, 5, 5, 5, 5, 5, 5, 5;
        MIRROR  1;
        SOUND   Trop07Snd;
        SDIST   300;
        SVDIST  25;
        SCYCLES 0, 1, 0, 0, 0, 0, 0, 0, 0;
        FLAGS   CLIP;
}
TEXTURE Trop8Tex {
        SCALE_XY        23,23;
        SIDES           1;
        CYCLES          8;
        BMAPS   Trop8001, Trop8101, Trop8201, Trop8301,
        Trop8401, Trop8501, Trop8601, Trop8701;
        DELAY   5, 4, 3, 3, 3, 3, 3, 3;
        SOUND    Trop08Snd;
        SCYCLES 0,1,0,0,0,0,0,0;
        SVOL 0.3;
        SDIST 200;
        SVDIST 25;
        FLAGS   CLIP;
}
TEXTURE Trop9Tex {
        SCALE_XY        23,23;
        SIDES           1;
        CYCLES          1;
```

```
        BMAPS   Trop9001;
        FLAGS   CLIP;
}
TEXTURE TropATex {
        SCALE_XY        23,23;
        SIDES           8;
        CYCLES          4;
        BMAPS   TropA001, TropA101, TropA201, TropA301,
                TropA002, TropA102, TropA202, TropA302,
                TropA003, TropA103, TropA203, TropA303,
                TropA004, TropA104, TropA204, TropA304,
                TropA005, TropA105, TropA205, TropA305,
                TropA204, TropA304, TropA004, TropA104,
                TropA203, TropA303, TropA003, TropA103,
                TropA202, TropA302, TropA002, TropA102;
        DELAY   4, 5, 4, 5;
        MIRROR  0,0,0,0,0,1,1,1;
        FLAGS   CLIP;
}

//////////////////////////////////////////////////////////////////
// Actions
//////////////////////////////////////////////////////////////////

ACTION TropTurn {
   RULE waitTime = 64 * RANDOM + 16;
   IF_BELOW RANDOM, 0.07;
      BRANCH TropListen;
   IF_ABOVE RANDOM, 0.95;
      RULE rightTurnTrop = (rightTurnTrop-1)*(rightTurnTrop-1);
   IF_MAX rightTurnTrop;
      BRANCH TurnRight;
   BRANCH TurnLeft;
}

ACTION TropBackoff {
        SET MY.EACH_TICK,       NULL;
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,   NULL;
        SET MY.IF_HIT,    TropHit;
        SET MY.CAREFULLY,   1;

        SET MY.SKILL4, 1;    // skill4 = state
        SET MY.TEXTURE, Trop1Tex;
        SET MY.SPEED, 0.3;
        SET MY.TARGET, REPEL;
        WAITT 64;
        SET MY.TARGET, FOLLOW;
        WAITT 2;
        IF_ABOVE MY.SKILL1, 5;
            END;
        BRANCH TropWait;
}

ACTION TropWait {
        SET MY.EACH_TICK,       NULL;
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,   CycleTropAttack;
        SET MY.IF_HIT,    TropHit;
```

```
        SET MY.CAREFULLY,   0;
        SET MY.GROUND,    0;

        SET MY.SKILL4, 1;    // skill4 = state
        SET MY.TEXTURE, Trop0Tex;
        SET MY.SPEED, 0.0;
        SET MY.TARGET, NULL;
}
ACTION TropWander {
        SET MY.EACH_TICK,       NULL;
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,   CycleTropAttack;
        SET MY.IF_HIT,    TropHit;
        SET MY.CAREFULLY,   1;
        SET MY.BERKELEY,   0;

   SET MY.SKILL4, 1;    // skill4 = state
        SET MY.TEXTURE, Trop1Tex;
        SET MY.SPEED, 0.3;
        SET MY.TARGET, BULLET;
}
ACTION TropSearch {
        SET MY.EACH_TICK,       NULL;
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,      CycleTropAttack;
        SET MY.IF_HIT,    TropHit;
        SET MY.CAREFULLY,   1;
        SET MY.BERKELEY,   0;

        SET MY.TEXTURE, TropATex;
        SET MY.SPEED, 0.3;
        SET MY.TARGET, BULLET;
}
ACTION TropHide {
        SET MY.EACH_TICK,       NULL;
        SET MY.IF_NEAR,         TropEscape;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,   NULL;
        SET MY.IF_HIT,    TropHit;
        SET MY.CAREFULLY,   1;
        SET MY.BERKELEY,   0;

        SET MY.SKILL4, 4;    // skill4 = state
        RULE MY.ANGLE = PLAYER_ANGLE + (RANDOM-0.5) * 2.4;
        SET MY.TEXTURE, Trop1Tex;
        SET MY.DIST,  20;
        SET MY.SPEED, 0.6;
        SET MY.TARGET, BULLET;
        WAITT 384;
        IF_BELOW MY.SKILL1, 9;
           BRANCH TropWander;
}

ACTION TropListen {
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,      CycleTropAttack;
        SET MY.IF_HIT,          TropHit;
        SET MY.CAREFULLY,  0;
```

```
        SET MY.BERKELEY,   0;

        SET MY.SKILL4,     0;    // skill4 = state
        SET MY.TEXTURE, Trop0Tex;
        SET MY.SPEED, 0.0;
        SET MY.TARGET, NULL;
        WAITT 120;
        IF_BELOW MY.SKILL1, 9;
           BRANCH TropWander;
}

ACTION TropWarning {
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,      TropBackoff;
        SET MY.IF_HIT,          TropHit;

        SET MY.SKILL4, 7;    // skill4 = state
        SET MY.TEXTURE, Trop7Tex;
   IF_BELOW RANDOM, 0.5;
         SET MY.TEXTURE, Trop7aTex;
   SET TROP_TEX, MY.TEXTURE;
   RANDOMIZE randomTrop, 1;
   IF_BELOW randomTrop, 0.2;
         GOTO overThere;
   IF_BELOW randomTrop, 0.4;
         GOTO getDown;
   SET TROP_TEX.SOUND, Trop07Snd;
   GOTO cont;
getDown:
   SET TROP_TEX.SOUND, Trop03Snd;
   GOTO cont;
overThere:
   SET TROP_TEX.SOUND, Trop02Snd;
   GOTO cont;
cont:
   SET MY.SPEED, 0.0;
   WAITT 32;
   IF_ABOVE MY.SKILL1, 5;
   END;
   RULE MY.ANGLE = MY.ANGLE + 2;
   IF_BELOW RANDOM, 0.5;
        RULE MY.ANGLE = MY.ANGLE - 2;
   BRANCH TropBackoff;
}

ACTION TropLookFor {
   IF_EQUAL MY.SKILL4, 10;
      END;
   IF_ABOVE MY.DISTANCE, 100;
      END;
   IF_EQUAL MY.SKILL4, 10;
      END;
   IF_EQUAL MY.TARGET, FOLLOW;
      END;
   IF_ABOVE MY.SKILL1, 5;
      END;
   SET MY.IF_NEAR,         NULL;
   SET MY.IF_FAR,          NULL;
   SET MY.EACH_CYCLE,      TropSearch;
   SET MY.IF_HIT,         TropHit;
```

```
   SET MY.CAREFULLY,      0;
   SET MY.BERKELEY,       0;
   IF_BELOW MY.DISTANCE,  40;
         SET MY.EACH_CYCLE, TropFollowAttack;

   SET MY.SKILL4, 10;    // skill4 = state
   SET MY.TEXTURE, TropATex;
   SET MY.SPEED, 0.0;

   SET MY.TARGET, BULLET;
}
ACTION TropFollow {
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,      CycleTropAttack;
        SET MY.IF_HIT,    TropHit;
        SET MY.CAREFULLY,   1;
        SET MY.BERKELEY,   0;

        SET MY.SKILL4, 1;    // skill4 = state
        SET MY.TEXTURE, Trop1Tex;
        SET MY.SPEED, 0.4;
        SET MY.TARGET, FOLLOW;
}
ACTION TropFollowAttack {
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,      CycleTropShoot;
        SET MY.IF_HIT,          TropHit;
        SET MY.CAREFULLY,       1;
        SET MY.BERKELEY,        0;

        SET MY.SKILL4, 2;    // skill4 = state
        SET MY.TEXTURE,         TropATex;
        SET MY.SPEED,           0.4;
        SET MY.TARGET,          FOLLOW;
}
ACTION TropFollowWarning {
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,      CycleTropWarning;
        SET MY.IF_HIT,          TropHit;
        SET MY.CAREFULLY,       1;
        SET MY.BERKELEY,        0;

   SET MY.SKILL4, 1;    // skill4 = state
        SET MY.TEXTURE,         Trop1Tex;
        SET MY.SPEED,           0.4;
        SET MY.TARGET,          FOLLOW;
}
ACTION TropAim {
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,      TropShoot;
        SET MY.IF_HIT,          TropHit;
        SET MY.CAREFULLY,       0;
        SET MY.BERKELEY,        0;

        SET MY.SKILL4, 3;    // skill4 = state
        SET MY.TEXTURE,         Trop3aTex;
        SET MY.SPEED, 0.0;
```

```
        SET MY.TARGET, FOLLOW;
        CALL TropShout;
}
ACTION TropShoot {
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.EACH_CYCLE,      CycleTropShoot;
        SET MY.IF_HIT,          TropHit;
        SET MY.CAREFULLY,       0;
        SET MY.BERKELEY,        0;

   SET MY.SKILL4, 3;    // skill4 = state
        SET MY.TEXTURE, Trop3Tex;
        SET MY.SPEED, 0.0;
        SET MY.TARGET, FOLLOW;
        SET SHOOT_SECTOR, 6.28;
        SET SHOOT_FAC,    tropShootFactor;
        SET SHOOT_RANGE,  100;
        SET SHOOT_Y,   0.0;
        SHOOT MY;
        IF_EQUAL HIT_DIST, 0;
           GOTO miss;
        SET UP_REGION, MY.REGION;
        IF_EQUAL HERE, UP_REGION.BELOW;
           GOTO miss;
        IF_EQUAL UP_REGION, HERE.BELOW;
           GOTO miss;
        SET PLAYER_RESULT, RESULT;
        BRANCH HitPlayerDelay;
        END;
miss:
        BRANCH TropFollowAttack;
}
ACTION TropEscape {
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          TropHide;
        SET MY.EACH_CYCLE,      CycleTropHide;
        SET MY.IF_HIT,          TropHit;
        SET MY.CAREFULLY,       1;
        SET MY.BERKELEY,        0;

        SET MY.SKILL4, 4;    // skill4 = state
        SET MY.TEXTURE, Trop1Tex;
        SET MY.DIST,  30;
        SET MY.SPEED, 0.4;
        SET MY.TARGET, REPEL;
}
ACTION TropDead {
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.IF_HIT,          NULL;
        SET MY.EACH_CYCLE,      NULL;

        SET MY.SKILL4, 9;    // skill4 = state
        SET MY.TEXTURE, Trop9Tex;
        SET MY.SPEED, 0.0;
        SET MY.TARGET, NULL;
        SET MY.PASSABLE, 1;
        SET MY.CAREFULLY,   0;
        SET MY.BERKELEY,   1;
}
```

```
ACTION TropDie {
   SET MY.SKILL1,     10;   // certify actor is dead
        SET MY.IF_NEAR,         NULL;
        SET MY.IF_FAR,          NULL;
        SET MY.IF_HIT,          NULL;
        SET MY.EACH_CYCLE,      TropDead;

        IF_NEQUAL MY.FLAG4, 0;
           CALL TropDropKey;
        SET MY.SKILL4, 8;    // skill4 = state
        SET MY.TEXTURE, Trop8Tex;
        SET MY.SPEED, 0.0;
        SET MY.TARGET, NULL;
}
ACTION TropImplode {
   RULE distX = (MY.X - EXPLOSION_CENTER.X)*(MY.X -
EXPLOSION_CENTER.X)+
      (MY.Y - EXPLOSION_CENTER.Y)*(MY.Y - EXPLOSION_CENTER.Y);
   SQRT distX, distX;
   RULE distZ = MY.HEIGHT - EXPLOSION_CENTER.HEIGHT;
   IF_ABOVE distX, 10;
      BRANCH BeamReact;
   SET MY.PASSABLE, 1;
   SET MY.CAREFULLY,   0;
   SET MY.IF_NEAR,         NULL;
   SET MY.IF_FAR,          NULL;
   SET MY.IF_HIT,          NULL;
   SET MY.EACH_CYCLE,      VanishStop;

   SET MY.SKILL4, 8;    // skill4 = state
   SET MY.TEXTURE, Trop0aTex;
   SET MY.SPEED, 0.0;
   SET MY.TARGET, NULL;
   IF_NEQUAL MY.FLAG4, 0;
       CALL TropDropKey;
}

ACTION TropHit {
   IF_EQUAL SHOOT_FAC, 0;
      GOTO obstacle;
   IF_EQUAL HIT, MY;
      GOTO hit;
   IF_EQUAL EXPLOSION_ON, 0;
      GOTO obstacle;
   RULE distX = (MY.X - EXPLOSION_CENTER.X)*(MY.X - EXPLOSION_CENTER.X)+
      (MY.Y - EXPLOSION_CENTER.Y)*(MY.Y - EXPLOSION_CENTER.Y);
   SQRT distX, distX;
   IF_ABOVE distX, 25;
      GOTO obstacle;
hit:
   IF_NEQUAL EXPLOSION_ON, 2;
      GOTO cont;
   IF_EQUAL MY.VISIBLE, 1;
      BRANCH TropImplode;
cont:
   ADD MY.SKILL1, SHOOT_FAC;
   IF_ABOVE MY.SKILL1, 5;
       GOTO die;
   IF_BELOW RANDOM, 0.05;
       GOTO die;
   SET TROP_TEX,  MY.TEXTURE;
```

```
   SET MY.IF_NEAR, NULL;
   SET MY.IF_FAR, NULL;
   SET MY.IF_HIT, NULL;
   IF_ABOVE RANDOM, 0.7;
      GOTO sonOfa;
   SET MY.TEXTURE, Trop5Tex;
      PLAY_SOUND Trop05Snd, 0.2, MY;
   GOTO wait;
sonofa:
   SET MY.TEXTURE, Trop5aTex;
   PLAY_SOUND Trop11Snd, 0.2, MY;
wait:
   WAITT 4;
   SET MY.IF_NEAR, TropEscape;
   SET MY.IF_FAR, TropHide;
   SET MY.IF_HIT, TropHit;
   SET MY.TEXTURE, TROP_TEX;
   IF_ABOVE MY.SKILL1, 4;
       BRANCH TropHide;
   IF_NEQUAL MY.SKILL4, 4;
       BRANCH TropFollowAttack;
   END;
die:
   SET MY.SKILL1, 10;
   BRANCH TropDie;
   END;
obstacle:
   BRANCH TropTurn;
}
ACTION TropDropKey
{
    RULE Key2Tng.X = My.X;
    RULE Key2Tng.Y = My.Y;
    SET Key2Tng.GROUND,    1;  // ensures proper region for LOCATE
    RULE Key2Tng.HEIGHT = My.FLOOR_HGT-0.01;

    SET Key2Tng.INVISIBLE, 0;
    LOCATE Key2Tng;
    SET Key2Tng.GROUND,    0;
}

ACTION CycleTropAttack {
   CALL LocateActor;
   IF_EQUAL SHOT_SOUND_ON, 0;
       goto cont;
   CALL TropLookFor;
cont:
   IF_NEQUAL MY.VISIBLE, 0;
       GOTO distance;
   IF_BELOW MY.DISTANCE, 25;
       GOTO distance;
   SET SHOOT_SECTOR, 3;
   SET SHOOT_FAC,    0;
   SET SHOOT_RANGE,  200;
   SHOOT MY;
   IF_EQUAL HIT_DIST, 0;
       END;
   SET UP_REGION, MY.REGION;
   IF_EQUAL HERE, UP_REGION.BELOW;
      END;
  IF_EQUAL UP_REGION, HERE.BELOW;
```

```
      END;
distance:
  IF_NEQUAL MY.FLAG2, 0;  // flag 2 = player's got a gun
      GOTO attack;
  IF_NEQUAL GUN_ON, 0;
      GOTO attack;
  IF_BELOW MY.DISTANCE, 20;
      BRANCH TropFollowWarning;
  END;
attack:
  SET MY.FLAG2, 1;   // flag 2 = player's got a gun
  CALL TropTalk;
  IF_BELOW MY.DISTANCE, 50;
      BRANCH TropFollowAttack;
}
ACTION CycleTropHide {
   CALL LocateActor;
   SET SHOOT_SECTOR, 6.283;
   SET SHOOT_FAC,    0;
   SET SHOOT_RANGE,  200;
   SHOOT MY;
   IF_EQUAL HIT_DIST, 0;
           GOTO stop;
   SET MY.SPEED, 0.6;
   SET MY.TARGET, BULLET;
   END;
stop:
   SET MY.SPEED, 0;
   SET MY.TARGET, NULL;
}

ACTION CycleTropShoot {
   CALL LocateActor;
   IF_ABOVE MY.SKILL1, 9;
       BRANCH TropDie;
   SET SHOOT_SECTOR, 6.28;
   SET SHOOT_FAC,    0;
   SET SHOOT_RANGE,  100;
   SHOOT MY;
   IF_EQUAL MY.VISIBLE, 1;
       GOTO shoot;
   IF_EQUAL HIT_DIST, 0;
       GOTO attack;
   SET UP_REGION, MY.REGION;
   IF_EQUAL HERE, UP_REGION.BELOW;
       GOTO attack;
   IF_EQUAL UP_REGION, HERE.BELOW;
       GOTO attack;
shoot:
   SET SHOT_SOUND_ON, 1;
   SET shotSecCount,  0;
   IF_ABOVE RANDOM, 0.8;
        BRANCH  TropAim;
   BRANCH  TropShoot;
attack:
   BRANCH  TropFollowAttack;
}

ACTION CycleTropWarning {
   CALL LocateActor;
   IF_NEQUAL GUN_ON, 0;
```

```
        BRANCH TropFollowAttack;
   IF_ABOVE MY.DISTANCE, 10;
        GOTO cont;
   SET MY.EACH_CYCLE,      NULL;
   BRANCH TropWarning;
cont:
   IF_ABOVE MY.DISTANCE, 40;
   BRANCH TropWander;
}

ACTION TropTalk {
   RANDOMIZE randomTrop, 1;
   IF_ABOVE randomTrop, 0.9;
        GOTO talk2;
   IF_ABOVE randomTrop, 0.8;
        GOTO talk1;
   END;
talk1:
   PLAY_SOUND Trop01Snd, 0.3, MY;
   END;
talk2:
   PLAY_SOUND Trop09Snd, 0.3, MY;
}

ACTION TropShout {
   RANDOMIZE randomTrop, 1;
   IF_ABOVE randomTrop, 0.8;
        GOTO hey;
   IF_ABOVE randomTrop, 0.35;
        GOTO overThere;
   PLAY_SOUND Trop03Snd, 0.3, MY;
   END;
overThere:
   PLAY_SOUND Trop10Snd, 0.3, MY;
   END;
hey:
   PLAY_SOUND Trop12Snd, 0.4, MY;
}

//////////////////////////////////////////////////////////////////
// Actor
//////////////////////////////////////////////////////////////////
ACTOR TropAct {
      TEXTURE   Trop1Tex;
      HEIGHT   -0.05;
      DIST      20;
      SPEED     0.3;
      FLAGS     CAREFULLY,FRAGILE;
      IF_HIT    TropHit;
      EACH_TICK TropWait;
}
ACTOR Trop1ct {
      TEXTURE   Trop0Tex;
      HEIGHT   -0.05;
      SPEED     0.0;
      FLAGS     CAREFULLY,FRAGILE;
      IF_HIT    TropHit;
}
```

APÊNDICE 3: BREVE HISTÓRICO DA PRODUÇÃO BRASILEIRA DE GAMES 3D

Este breve relato apresenta os oito anos iniciais da produção brasileira de games 3D[59]. O foco é o trabalho das quatro principais empresas dedicadas a esse tipo de desenvolvimento no Brasil: Perceptum, SouthLogic, Espaço e Greenland.

Criada por mim e por Odair Gaspar em outubro de 1995, a Perceptum Software Ltda. nasceu de experiências profissionais anteriores com o desenvolvimento de softwares em tempo real para aplicações de automação industrial.

A Perceptum lançou o primeiro Game 3D brasileiro de ação em primeira pessoa, *Incidente em Varginha*, em setembro de 1998, de forma independente, devido à virtual ausência de publicadoras interessadas em games nacionais na época. O game atingiu sucesso apenas relativo no mercado nacional, mas gerou vendas significativas na Europa e na Ásia sob o título *Alien Anarchy*; foi encartado em revistas especializadas nos Estados Unidos, além de ter sido vendido em alguns países da América do Sul, com o título *Misión Alien*. A distribuição das versões internacionais do game foi realizada pela Midas Interactive Entertainment.

O desdobramento mais insólito de *Incidente em Varginha* teve início com uma consulta do U.S. Army Special Forces Command (Comando das Forças Especiais do Exército Americano), feita por email em 11 de novembro de 1998. A mensagem sondava o possível uso do game no treinamento de soldados das forças especiais[60] como alternativa ao game *Doom*, da ID Software, utilizado na época para

---

[59] Existem games brasileiros importantes que não se utilizam de tecnologia 3D em tempo real (o pioneiro *Amazônia*, de Renato Degiovani, e o game de estratégia *Outlive*, da Continuum, entre eles). Devido às especificidades do desenvolvimento de games 3D, tema deste trabalho, o histórico limita-se a esse tipo de produção.

[60] É importante ressaltar que os soldados inimigos em *Incidente em Varginha* retratavam especificamente forças de elite do exército americano.

treinamento dos SEALS, grupo de elite da marinha americana. Descrevia ainda algumas missões de treinamento hipotéticas, incluindo ataques a centros de radar e a veículos de lançamento de mísseis SCUDs – curiosamente, todas as missões tinham como cenário o Iraque. Respondi pessoalmente ao email, esclarecendo que a Perceptum não era uma empresa americana, mas brasileira, e que um game muito similar ao solicitado estava sendo lançado nos Estados Unidos pela Novalogic: *Delta Force*. Não recebi qualquer retorno, mas uma versão especial de *Delta Force* foi posteriormente desenvolvida para o treinamento das forças de elite do exército americano.

Em 1999, a Perceptum foi contratada pelo Instituto Itaú Cultural para o desenvolvimento da exposição virtual *Imateriais*. Seis grandes ambientes virtuais multiusuários foram criados com a tecnologia desenvolvida para o game *IV2: Sombras da Verdade*, sequência não publicada de *Incidente em Varginha*. A exposição virtual *Imateriais* contava ainda com comunicação por voz em tempo real, algo que só viria a se popularizar em games online alguns anos depois.

Um demo de *IV2*, otimizado para as novas instruções do processador Pentium III, foi selecionado pela Intel para apresentação no evento *Pentium III Preview Day*, pré-lançamento do novo processador que ocorreu em San Jose, Califórnia (EUA), em 17 de fevereiro de 1999. Todo o trabalho de aperfeiçoamento do código foi realizado dentro do Programa de Desenvolvedores Intel.

Outra empresa brasileira participante do mesmo programa foi a Jack in the Box Computing, hoje conhecida como SouthLogic Studios. Fundada por Christian Lykawka em julho de 1996, a empresa lançou um game 2D, *Guimo*, em 1997. Em 1998, Christian ingressou na incubadora de novas empresas da Universidade Federal do Rio Grande do Sul e iniciou o desenvolvimento de um *engine* 3D proprietário, o *Aspen Engine*. Na mesma época, Adriano Ledur, Ricardo Coimbra da Rocha e Gustavo Goedert uniram-se a Christian. *Aquarius*, um protótipo de game desenvolvido com o *Aspen Engine*, foi também selecionado pela Intel e exibido no *Pentium III Preview Day*, em 1999. Impressionado com o trabalho da equipe, um dos engenheiros de software da Intel, Chris Kastensmidt, então consultor internacional do Programa de Desenvolvedores, deixou a empresa americana no mesmo ano para

unir-se à SouthLogic Studios. Sua participação como diretor comercial, especialmente na negociação junto a empresas do mercado norte-americano, seria importantíssima no futuro da empresa gaúcha.

Em maio de 2001, a Perceptum finalizou o desenvolvimento de *Micro Scooter Challenge*, lançado no mercado internacional no mês seguinte. O game, uma simulação de corrida de patinetes, foi criado a pedido da Incagold, publicadora baseada em Londres, e distribuído em 14 países por diferentes empresas, com versões em Inglês e Alemão. A distribuição na Europa foi particularmente bem sucedida, e *Micro Scooter Challenge* ficou durante semanas entre os títulos mais vendidos na Alemanha. Entretanto, o contrato com a Perceptum não foi honrado, desenvolvimento e *royaties* sobre as vendas não foram pagos e, como resultado, a empresa brasileira passou dois anos à beira da falência. Outro título que estava sendo desenvolvido pela Perceptum para a Incagold, *Beach Volleyball*, já em estágio de demo, teve seu desenvolvimento suspenso por falta de pagamento em 2001.

Em julho de 2001, outra empresa de games brasileira lançou seu primeiro título. Tratava-se de um game de tiro em primeira pessoa, *Hades 2*, da Espaço Informática. Criada em 1999, em Porto Alegre, pelos irmãos Augusto e Gustavo Bülow, a Espaço destacou-se desde o início pela dedicação e seriedade no processo de desenvolvimento. Soluções engenhosas e de baixo custo foram utilizadas para criar *Hades 2*, que empregava uma versão mais recente do mesmo engine de *Incidente em Varginha*. Os sprites de alguns personagens, por exemplo, foram gerados por Maurício Lucas a partir de imagens digitalizadas dos próprios membros da equipe, usando fantasias. *Hades 2* foi comercializado em quatro países: Brasil, Argentina, Inglaterra e Polônia.

A Perceptum produziu o advergame *Super Mini Racing*, um simulador de corrida com carros em miniatura, em meados de 2001. Trata-se do game 3D brasileiro com maior número de cópias distribuídas. Mais de meio milhão de exemplares foram encartados gratuitamente nos jornais *O Estado de S. Paulo* e *Jornal da Tarde*, nas edições de 12 de outubro de 2001. Em novembro do mesmo ano, a Espaço produziu o game natalino *Christmas Magic*, distribuído pela Incagold no Brasil, EUA, Inglaterra, Polônia e Coreia do Sul.

Outra produtora brasileira de games estreou com um título 3D em agosto de 2002. *Vampiromania* foi produzido pela Green Land Studios. Formada por Marcelo Duarte Oliveira, Leonardo A. Branco, Wagner Gomes Carvalho, Victor C. Strella e Vitor Gomes Rodrigues, a Green Land Studios foi aberta em outubro de 2001, em Santos, São Paulo. O game *Vampiromania* estava ligado à produção de *O Beijo do Vampiro*, da Rede Globo, e aparecia em diversas cenas, sendo jogado por personagens da novela. Os principais títulos 3D da Green Land foram resultados dessa parceria com a Rede Globo: *Big Brother Brasil 3D On-line*, de fevereiro de 2003; *Sandy e Júnior, Aventura Virtual*, de setembro de 2003; e *Aquária – O Jogo*, de dezembro de 2003.

Trophy Hunter 2003, game de simulação de caça desenvolvido pela SouthLogic Studios, foi publicado por Atari e Infogrames em setembro de 2002, no Canadá e nos Estados Unidos. O game resultou de diversas negociações com publicadores dos Estados Unidos, centradas na capacidade do Aspen Engine de simular ambientes abertos e florestas. A boa receptividade desse título gerou dois outros produtos similares da SouthLogic, *Deer Hunter 2004* e *Deer Hunter 2005*, ambos publicados pela Atari Americana.

A Espaço Informática lançou, em maio de 2003, o game de simulação de tênis *Matchball Tennis*, distribuído pela Brasoft / BVH no Brasil, Alemanha, Áustria e Suíça. Um detalhe interessante do desenvolvimento de *Matchball Tennis* foi o processo utilizado para a animação dos personagens. Para evitar os altos custos da captura de movimentos, a equipe da Espaço utilizou técnica baseada na rotoscopia: gravou em vídeo a atuação de um tenista, em dois ângulos, para depois animar manualmente seu personagens, quadro a quadro, com base no material gravado. Apesar de trabalhoso, o processo resultou em qualidade similar à da captura profissional de movimentos, mas a custo irrisório.

Pferderennstall, da Espaço, é um game de estratégia em tempo real sobre o gerenciamento de um haras de cavalos de corrida. Publicado em janeiro de 2005 no mercado alemão, *Pferderennstall* foi um sucesso de vendas e teve como desdobramento a versão *Hipodromme Tycoon*, publicada em na França em novembro de 2005, e *Thoroughbred Tycoon*, sequência publicada em 2006.

Finalmente, em 2006, a Perceptum publicou *Super Mini Racing Ice*, sequência de seu primeiro advergame, em conjunto com a Canal Kids. A SouthLogic, por sua vez, concluiu *Kelloggs Football Tiger*, publicado no mercado norte-americano pela Microtime.

ANEXO 1:
GUIAS RÁPIDOS PARA OS GAMES

Micro Scooter Challenge

SYSTEM REQUIREMENTS:
Windows 95/98/ME/XP/2000 Operating System

Pentium or AMD K6 200 MHz Processor or better (300 MHz Recommended)

32 MB RAM (64 MB Recommended)

2x Speed CD-ROM

8 MB 3D accelerated Graphics Card (16 MB Recommended)

TO INSTALL:

1. Insert the Scooter Challenge CD into the CD-ROM drive of your computer.
2. If auto-play is enabled the Scooter Challenge Installation screen will appear. Please follow the on screen instructions.
3. If auto-play is disabled double click on the My Computer icon on your desktop. Locate your CD-ROM drive with the Scooter Challenge disc and double click on the icon.
4. Locate the file "Setup.exe" on the Scooter Challenge disco Double click on the icon for this file.
5. The Scooter Challenge Installer will now run. Follow the on screen instructions.

TO RUN SCOOTER CHALLENGE:

1. Click on your "Start Menu" and select the Programs bar.
2. Find the Program group titled "Scooter Challenge" and select it.
3. Click on the Scooter Challenge icon to start the game.

STARTING SCOOTER CHALLENGE:
After the boot-up sequence you are prompted to select a mode of play. The three modes are:

PRACTICE - This mode lets you race on any of the 7 levels around the world without any opponents. This is good for getting used to each track and improving your limes before going to the big leagues!

SINGLE RACE - Single race allows you to further hone your Scooter Racing skills against 3 other Computer opponents on any single level. You can choose to race on any of the seven levels for 5 laps.

CHAMPIONSHIP - Championship is for the true Scooter Racing pros! This mode takes you through all the levels in a set order. First, Second and Third place on each level are awarded points. At the end of the Championship, the player with the most points is the winner!

There are a total of seven levels in Scooter Challenge. They are:

London, England - A last-paced romp through downtown London, with a great view to the palace!

Rio de Janeiro, Brazil - A winding track located in Rio's hottest resort. Lots of ramps and turns!

Olympia Stadium, Berlin - Extreme turns and speedy straight-a-ways can drive the crowd mad here.

Cairo, Egypt - Take a tour of the tombs and pyramids while out-racing your opponents.

Central Park, New York - There's no place like New York, especially for Scooter Racing!

Opera House, Sydney - This Australian landmark is a perfect place for some big jumps and tricks.

Lost City of the Incas - A hidden temple that only the bravest and best of Racers can find.

IMAGE IS EVERYTHING:
After selecting your mode and level, a good Scooter Player must choose his gear and ride! Enter your name by typing in the name prompt, and choose your color clothing with the left and right areas next to the corresponding garment. You can choose to be a male or female scooter pro, your type of scooter and your attitude.

Attitude affects the difficulty of your computer opponents. On nice, your opponents will go easy on you. Aggressive opponents will try hard to beat you and keep you from cutting in front of them. Extreme opponents will do their best to stop you from placing at all!

The Scooter Type you choose also affects play. Push scooters are the slowest, followed by the electric ones. The fastest type of scooters is the oil/gasoline driven type. Try to ride them on Extreme difficulty and see just how much 01 a Scooter Pro you are! Once all your options are selected, click on the arrow in the bottom right 01 the screen to proceed.

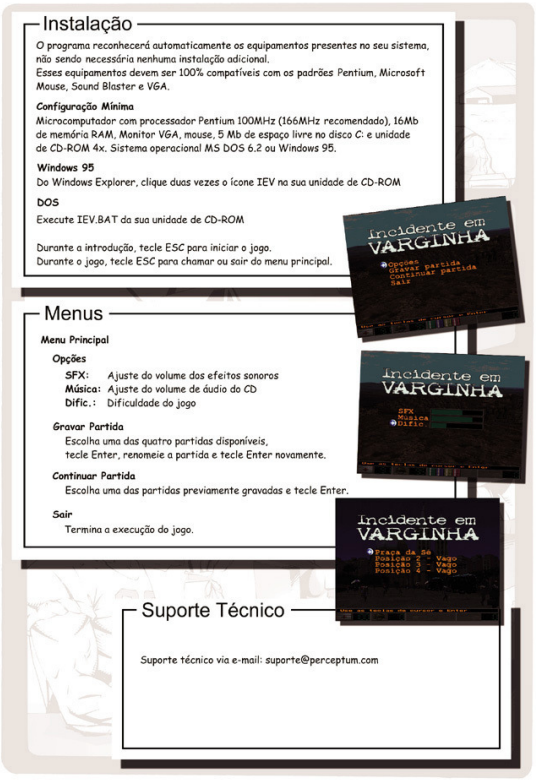
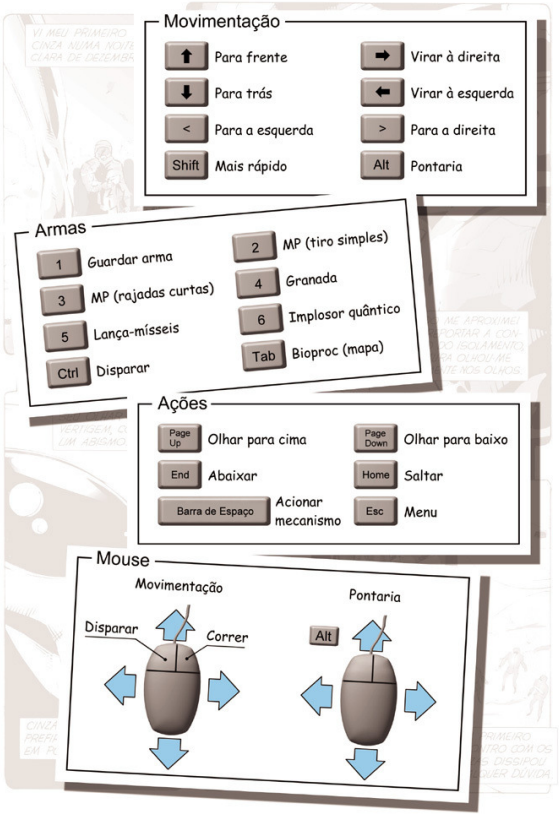
HOW TO PLAY:
Your Scooter and Rider can be controlled simply using the arrow keys on your keyboard.

Up Arrow - Forward Acceleration

Left Arrow - Steer Left

Right Arrow - Steer Right

Back Arrow - Slow down/Backward Acceleration (electric and oil scooters can only push backwards)

ESC - Press the escape key at any time to pause the race. You will be prompted Yes or No to continue. You can exit the race by pressing 'N' for No, or continue with 'Y' for Yes. At the main menu, ESC will prompt you to exit Scooter Challenge.

Incidente em Varginha