Introdução ao Game Design através da modificação de Unreal 2004

Prof. Dr. Roger Tavares & Prof. Felipe Neves
Senac-SP – GP:Games
E-mail: {rogertavares, felipeneve}@gmail.com

Resumo

Pode-se observar que os professores e alunos de educação para games, e mesmo game designers, têm perdido um tempo precioso, como meses debruçados sobre *engines* comerciais, para se chegar à resultados que através das técnicas de modificação podem ser atingido em dias. O mesmo pode-se dizer do investimento financeiro envolvido na compra de tais engines, normalmente vultoso se comparado ao preço de um jogo, e suas ferramentas gratuitas de modificação. Tais técnicas, que podem ser usadas para prototipagem e mesmo alguns produtos finais, têm sido obscurecidas pelo uso de engines populares, auto-proclamadas de profissionais, enquanto relegam as técnicas de modificação ao "amadorismo" de comunidades de modificadores. Além de apresentar tal técnica, este tutorial visa tratar os principais conceitos de Game Design e Gamecultura.

Abstract

Normally is possible to observe teachers and students in education for games, and game designers too, loosing a precious time, like months using commercial game engines to get some kind of results that could be faster and easier using game modification techniques. Something closer could be talked about the amount of money involved in these engines, very different if compared to a game and some free tools only. Such techniques, those could be used to make prototypes, and even some final products, they are hidden by the use of such engines, self-called professionals, at the same time they accuse modification techniques, and their users, of amateurs. This tutorial intends to show a bit of this techniques, while applying basic concepts from game design and gameculture.

Conceituando *Game Design*

Neste trabalho, game design será tratado pelo seu ponto de vista projetual, relacionando-se às mecânicas e estratégias para a obtenção do prazer próprio do jogo, e não no sentido de construção, ou mesmo programação dos elementos que compõe o game. Para se entender a posição do game designer nesse *workflow,* podemos observar a constituição de uma equipe média de desenvolvimento de games, adaptada de Rolling e Morris [1].

1. Gerenciamento e Design
    1.1 Game Designer

1.2 Level Designer (Designer de níveis)
1.3 Character Designer (Designer de Personagens)
1.4 Gerente de Projeto
1.5 Gerente de Software

2. Programação
2.1 Lead Programmer (Programador de conceito)
2.2 Programadores

3. Arte Visual
3.1 Lead Artist (Artista de conceito)
3.2 Artistas visuais (modeladores, ilustradores etc)

4. Música
4.1 Músico
4.2 Efeitos Sonoros e diálogos
4.3 Programador de áudio

5. Controle de Qualidade
5.1 Q.A. Lead (Condutor de controle de qualidade)
5.2 Q.A. Technicians (Controladores de qualidade)
5.3 Playtesters (jogadores avaliadores)

6. Outros
6.1 Especialistas em outras áreas (educadores, consultores etc.)
6.2 Técnicos em áreas diretamente relacionadas (captura de movimento, roteiristas etc.)

Tabela 1. *Estrutura de um desenvolvimento baseado em projeto, modificada de Rollings e Morris (2004, 246).*

Uma visão geral da tabela apresentada nos mostra inicialmente quantos profissionais seriam necessários para se fazer um jogo adequadamente, sem que um profissional ficasse transitando superficialmente em diversas áreas. Rollings e Morris [2] reforçam ainda que problemas sérios podem ocorrer nesse trânsito desqualificado, sobretudo pelo fato de que diversos profissionais estão envolvidos, em especial na função de gerenciamento.

Ainda do ponto de vista geral, pode-se observar o quão interdisciplinar é a produção de um jogo digital. Entretanto, a maior parte das áreas envolvidas, especialmente música ou artes visuais, embora tenham alguns de seus conhecimentos específicos, redirecionados para a atuação em um jogo como esses, tais conhecimentos não são nativos dessa área. A exceção é o profissional de *game design,* que trabalha em conceitos específicos, embora beba também em outras áreas, direta ou indiretamente relacionadas como outros tipos de jogos, como é o caso da Psicologia ou da Semiótica.

Segundo Tavares (2005) "Pode-se observar portanto que o game design não está tão relacionado às artes visuais, ou à programação, como costuma-se pensar. O game designer tem a visão do jogo como um todo, embora toda equipe deva tê-la. Ele é o profissional responsável pelos conhecimentos específicos da área."[3]

Em outro trabalho [4] foi possível mostrar como é importante que toda a equipe envolvida

na produção de um jogo tenha ciência de tudo que se desenrolará nesse processo, ainda que caiba ao game designer, acima de todos, balancear e dosar as mecânicas de sorte, habilidade, dificuldade das regras, fator de diversão, além de ficar atento a outros elementos que os trabalhos de toda uma equipe estão gerando.

Princípios básicos em game design

Para facilitar o entendimento e estruturar um discurso acerca do game design pontuamos alguns princípios que devem ser levados em conta na hora de se produzir um game. É claro que o game design não se resume a uma "receita de bolo" a ser seguida, entretanto ao analisar alguns pontos importantes, poderemos ter uma visão mais abrangente e confiável para se produzir um game mais agradável.

Os princípios listados abaixo partem de uma reportagem de uma das mais tradicionais revistas sobre o assunto, *Next Generation,* em 1997 [5] que se tentou responder à pergunta, "O que faz um bom game?". Partimos destas 6 regras e acrescentamos elementos que podem também contribuir para uma visão mais abrangente, como os problemas de gênero, que não foi considerada na reportagem, mas que pode ser encontrada em diversos jogos campeões de vendas.

Provavelmente estes 7 princípios básicos serão capazes de ajudar muitos leigos, e mesmo os especialistas, na hora de se ponderar sobre um título que possa agradar a todos. Procuramos trabalhar com exemplos de jogos e referências a um trabalho autoral, a gamearte *Dominação:Fase1,* para ilustrar tais regras.

1. Um bom *game design* deve ser balanceado, ou seja, não pode ser muito fácil para que o jogador se entedie e perca o interesse nele, nem tão difícil a ponto do jogador se desmotivar e dessa maneira, desistir. O balanceamento não deve ser confundido com os ajustes de dificuldade particular de cada jogo, como fácil, média, ou difícil. O balanceamento deve ser independente dos níveis de dificuldade. Da mesma maneira, não confunda o balanceamento com a linguagem inerente de cada jogo. Caso nunca tenha jogado um jogo de estratégia em tempo real (RTS), por exemplo, primeiro habitue-se ao mecanismo comum a todos esses jogos, como as interfaces complexas e manipulação das diversas unidades. Para isso basta jogar um ou outro jogo mais conhecido, para depois da devida identificação desses elementos, partir para a análise do balanceamento deles. Um exemplo bastante ilustrativo de problemas no balanceamento, são alguns RPGs que apresentam minigames nou puzzles dentro da história principal, cuja resolução é obrigatória para poder se avançar na narrativa (**Final Fantasy** XI, Square Enix, 2003). Embora a maioria das soluções desses puzzles possa ser encontrada no próprio jogo ou mesmo

Ambiente a ser explorado em Final Fantasy IX

através de dicas na internet, muitas pessoas, além de desistirem, ficam tão desestimuladas que evitam jogos semelhantes. Outro exemplo menos comum é possível ser encontrado em uma classe de adultos que não têm mais velocidade ou coordenação motora suficientes para jogarem certos jogos que dependem de reflexos rápidos.

2. Um bom *game design* deve ser criativo. Talvez essa seja uma das grandes dificuldades em uma indústria milionária, em que um bom videogame surge e imediatamente é copiado por dezenas de outras empresas que vão em busca do mesmo sucesso, muitas vezes copiando-os de maneira equivocada. Grandes games sempre adicionaram "um algo mais" a seus antecessores. Esse "algo mais" não são meros detalhes técnicos, como mais resolução, canais de som, mais velocidade, mas sim novos desafios, novas regras, mais dramaticidade, mais abertura a decisões, mais suspense e emoção. Nesse tópico despontam não apenas jogos revolucionários, de empresas poderosas com muitas pessoas investidas, mas também jogos GPL e os conhecidos mod-games, que são modificações que os próprios usuários fazem de seus jogos e disponibilizam gratuitamente na rede. Muitos deles primam pelo bom humor, outros por propostas arrojadas, como *Warcraft Mod para Counter Strike,* que adiciona elementos de RPG ao famoso jogo de terroristas, ou a modificação *Death Ball,* que transforma o jogo de arenas de tiro *Unreal Tournament 2004*, em uma espécie de jogo de futebol com diferentes mapas, e um desses mapas, GreenBaize, de RevBillyG, é uma mesa de sinuca.

3. Um bom *game design* deve ser focado, ou seja, ele deve manter o jogador entretido com os objetivos de sua partida, sem que ele se distraia com outros elementos. A maneira mais comum dos game designers fazerem isso, é descobrir quais são os elementos mais atrativos em seu jogo, além de possibilitarem que os jogadores tenham acesso facilitado a esses elementos. Um caso muito interessante, bastante recorrente em jogos de estratégia, é o jogo ***Rome Total War*** (The Creative Assembly Ltd., 2004), um jogo de estratégia que fornece as possibilidades de manipulação de recursos, de batalhas cinematográficas, além de possibilidades de leituras de fichas sobre personalidades, política e tecnologia do auge do Império Romano, e simulação de conhecidas batalhas épicas. Tudo isso possibilita que diversos tipos de jogadores se utilizem do mesmo jogo, explorando aspectos diversos deste.

A interface de construção e administração de cidades em Rome Total War.

4. Um bom *game design* deve ter personagens que cativem, ou mesmo aflijam, o seu público. Deve-se acrescentar que uma grande parte do público pode ser cativada por cenários, transportes e edifícios que, além de fornecerem o *gamespace* para que o jogo aconteça em toda sua jogabilidade (*gameplay*), podem fornecer um componente estético que, muitas vezes, agrada mais do que as personagens do jogo. Um exemplo clássico é **Lara Croft** (Tomb Raider Legend, Crystal Dynamics, 2005), ou Mario Bros, nomes que, sozinhos, já garantem uma empatia com o jogo. Ou os jogos de corridas, como a série *Gran Turismo* (Polyphony Digital), que não apresentam personagem algum, mas que cativam pelo realismo de seus carros, pistas e cenários.

Lara Croft em um cenário submerso.

5. Um bom *game design* deve ter tensão, que é um dos aspectos mais difíceis de se explicar, mas um dos mais fáceis de se sentir, em especial quando o jogador se recosta novamente na cadeira, ou volta a respirar aliviado.

Um aparte deve ser feito aos jogos de terror, suspense, ou mesmo de tiro, que podem incomodar públicos mais sensíveis, embora mesmo jogos de corrida ou simulações de esportes, como basquete ou futebol, trabalhem muito com este elemento. Tornar os objetivos difíceis de serem alcançados, como desarmar uma bomba contra o pouco tempo restante (*Counter Strike Source*, Valve Software, 2004), também faz aumentar a tensão. Contudo, não se deve descuidar com o balanceamento.

6. Um bom *game design* deve ter energia, ou seja, deve levar o jogador a querer jogar sempre mais. Pequenos objetivos e desafios, misturados a pequenas pausas para descanso, embalados por uma trilha sonora adequada, sem que tudo isso atrapalhe os objetivos do jogo (manter focado, como no item 2), como descobrir um tesouro, salvar uma cidade, ou mesmo possuir bens materiais. ***Sid Meyer's Pirates!*** (Firaxis Games, 2004) e *The Sims* (Maxis, 2000) são ótimos exemplos de como manter o jogador entretido a noite toda.

Sid Meyer's Pirates!

7. Um bom *game design* deve ser livre de gênero, um aspecto que pode não estar presente em muitos jogos campeões de bilheterias, que são feitos decididamente para homens ou para mulheres, mas que certamente aparecem na lista dos jogos mais vendidos até hoje, como ***Myst*** (Cyan Worlds, 1993) e *The Sims* (Maxis, 2000). Um jogo que seja livre de gênero pode ser jogado sem maiores problemas tanto por homens como por mulheres. Muitas vezes, entretanto, isso não é problema, uma vez que um jogador pode se representar por um personagem de outro sexo, embora algumas meninas não gostem de jogar com personagens masculinas, da mesma maneira que nem sempre gostam de ser "o objeto a ser salva do dragão". Alguns jogos permitem escolher a sua personagem entre opções masculinas ou femininas e, às vezes, o sexo do jogador não tem a menor importância nesse processo, como nos casos de jogos em primeira pessoa, em que a personagem normalmente nada mais é do que uma casca vazia a ser ocupada pelo jogador.

As docas em Myst.

Aplicando os conceitos

Decerto que a prática e o conhecimento de diversos títulos e gêneros de jogos é fator fundamental não só apenas para o game designer, mas para todos que estejam envolvidos com a produção e utilização dessa mídia, do modelador e do músico que constroem os conteúdos, a professores e treinadores que iniciam a aplicá-los na sala de aula ou nas empresas.

O resultado da fluência de um jogo, o seu gameplay, advém de uma complexidade de diversos fatores outros, que muitas vezes necessitam serem testados. O problema é que nem sempre o game designer têm acesso a essas ferramentas, seja por seu preço ou sua elevada curva de aprendizado, com necessidade de códigos e scripts que mesmo os profissionais de informática tem dificuldade.

Para minimizar esse problema, e poder trazer parte dos testes, nem todos claro, para a alçada do game designer, estamos realizando já há algum tempo, pesquisas, trabalhos e testes com ferramentas de modificações de diversos jogos. Ano passado, 2005, nos concentramos nas engines de Half-Life e Counter-Strike, como apresentamos no SBGames 2005, e este ano de 2006 nas engines de Neverwinter Nights e Unreal, esta última a qual apresentamos neste momento.

A seguir veremos uma pequena introdução ao acesso básico a engine do Unreal, uma das engines mais poderosas atualmente, através da ferramenta UnrealEd, que é distribuída gratuitamente com diversos games que se utilizam dessa engie. Durante essa introdução técnica já aplicaremos alguns conceitos básicos de game design, em especial level design, e por fim, avançaremos nas

possibilidades de aplicação de mod-games como ferramenta de prototipagem, e mesmo de alguns produtos finais, ao alcance do game designer, equipes reduzidas, ou prazos de desenvolvimento muito curtos.

O editor UnrealEd

Quase todos os jogos produzidos desde 1998 para o sistema operacional MS Windows pela Epic Games, desenvolvedora da poderosa engine Unreal, costumam desde então virem acompanhados do editor de levels UnrealEd. As versões para Linux, Macintosh e X-Box, infelizmente ainda não apresentam este editor.

Ícones do jogo Unreal e do seu editor UnrealEd.

Além dos títulos da empresa desenvolvedora, diversas outras empresas também licenciam essa poderosa tecnologia para produzir seus títulos [6], e algumas chegam a distribuir o editor UnrealEd com seus games, como é o caso do jogo *Pariah* (Groove Games, 2005), detentor do título do melhor jogo da feira E3 2004 [7]. A cada dia mais empresas, como a Electronic Arts [8] e a Sony [9], têm licenciado essa tecnologia para uso em seus jogos high-end, o que torna o conhecimento de uma ferramenta como essa, imprescindível para quem tem interesse sobre as tecnologias usadas na produção de games desse nível.

Diferente das ferramentas para a criação de novos mapas, que acompanham muitos jogos, o UnrealEd é uma ferramenta para design de Levels, ou seja, os mapas, arquiteturas, paths e nodes para a inteligência artificial, disposição de armas, munições, e portais, entre outras capacidades, além de acessar a linguagem UnrealScript.. Com ele também é possível visualizar todo o conteúdo dos games, e editar grande parte deles. Entretanto, um game é feito de uma quantidade muito grande de *assets*, como sons, modelos 3d, e texturas, e o UnrealEd não é capaz de produzir todos esses tipos de mídias. Ele é capaz de criar mapas, importar e gerenciar esses diferentes tipos de conteúdos, mas para produzi-los deve se utilizar softwares de terceiros, como o Autodesk Maya [10] para os modelos 3D, ou o Audacity [11] para a edição sonora.

Este pequeno tutorial introduz as funcionalidades e as técnicas mais comuns no uso desta poderosa e desconhecida ferramenta. Para as pessoas que quiserem entendê-la em um nível mais profundo, recomendamos o livro *Mastering Unreal Technology* [12], além dos inúmeros sites e fóruns, alguns citados ao fim deste artigo.

Conhecendo o editor

Inicialmente a movimentação pelo UnrealEd causa alguma estranheza para quem tem contato com outros editores 3D, mas a curva de aprendizado é muito rápida.

Outra característica que pode causar estranheza é a sua maneira de trabalhar subtrativamente. Diferente da grande maioria dos modeladores 3D, que são aditivos, o UnrealEd inicia a sua modelagem com um volume sólido, que deve ser escavado com as ferramentas apropriadas, os BSP brushes. Esse método de criação de level subrativo, apesar de não-intuitivo é bastante interessante, pois ele evita um problema muito comum em criação de levels em games, mesmo em algumas engines ditas profissionais, no qual a personagem podia cair fora do level, e o jogador assistia assim a sua personagem caindo infinitamente dentro do *world space*.

A anatomia básica de um level é muito simples. Ela consiste de 3 elementos básicos: (1) um CSG, Constructive Solid Geometry, ou seja, um espaço escavado; (2) um Player Start, o lugar no qual o jogador inicia a sua partida; e (3) uma luz, para que ele possa enxergar o que está

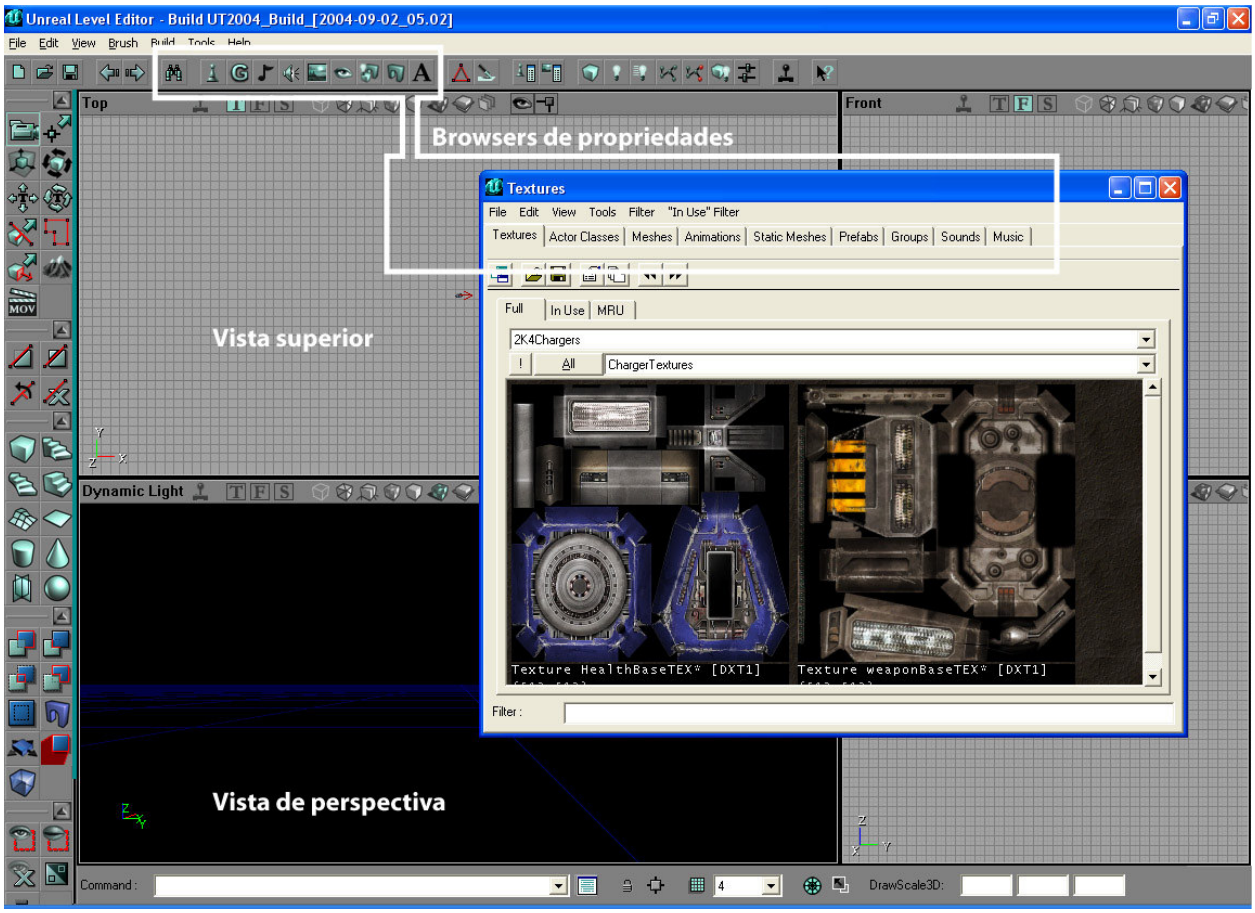


Figura 1: Vistas e browser de propriedades dos actors.

acontecendo.

Vamos fazer esse level básico então.

Clique com o botão direito do mouse na ferramenta de BSP Brush cúbica (um cubinho azul próximo à três ferramentas de escadas), e coloque as medidas da seguinte sala:

Height = 320
Width = 480
Breadth = 640

(Valores em unidades Unreal.)

Esses valores foram escolhidos por serem múltiplos de 16 (16 x 20; 16 x 30; 16 x 40), o que equivale a quatro vezes o tamanho do grid padrão do UnrealEd, e facilita um pouco na hora de construir e texturizar.

Caso você queira fazer uma sala gigantesca, cuidado com o tamanho, porque um level em Unreal é limitado a "apenas" 524.288 unidades Unreal, o que equivale a 216 milhas cúbicas, ou seis milhas (pouco mais de onze quilômetros) em cada direção (X, Y, Z).

Clique em OK e veja aparecer um BSP brush, um cubo vermelho, nas vistas ortogonais (Top, Front, e Side). Ajuste esse cubo nessas vistas, para que a câmera fique dentro dele. Isso tornará mais fácil a sua visualização na vista de perspectiva. Para movimentá-lo nessas vistas utilize a tecla Control, mais o botão direito do mouse. Para observá-lo na janela de perspectiva, movimente-o com o botão direito do mouse, o esquerdo, ou ambos simultâneamente.

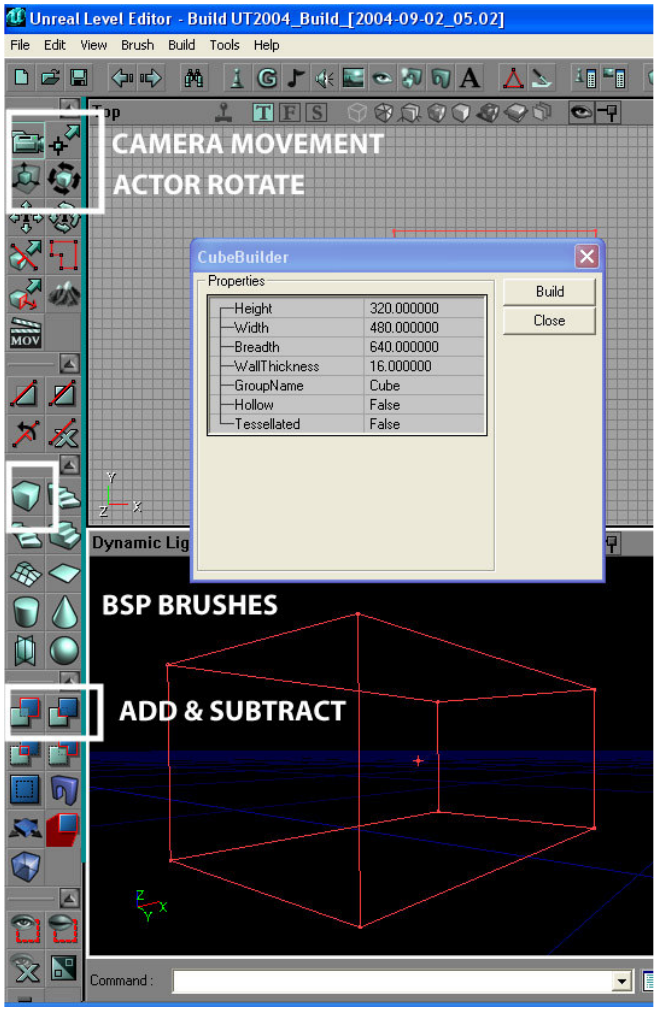


Figura 2: Criando e posicionando a sala.

Uma vez que o brush esteja visível, clique na ferramenta *Subtract,* e veja aparecer algumas texturas padrão.

A sala já está pronta. Agora basta adicionar o Start Point e a Luz para fazer o primeiro teste na engine, e na sequência colocar algumas texuras.

Adicionar tanto o StartPoint, quanto a luz, é muito fácil e muito parecido. Clique com o botão direito no chão da sala, e depois com o botão esquerdo escolha *Add Player Start Here.* Depois selecione o teto, e na seqüência escolha *Add Light Here*. Veja que esses dois elementos, Actors, têm algumas propriedades que podem ser ajustadas. O StartPoint tem uma flecha que indica aonde o jogador vai parecer olhando. Aponte-a para onde desejar que o jogador comece olhando. Para fazer isso, basta clicar sobre ela em uma das vistas, e movimentá-la com o mouse e a tecla

Ctrl pressionada. Se quiser rotacioná-lo, basta mudar a ferramenta de Camera Movement, que normalmente é a ferramenta padrão, para Actor Rotate. Depois volte à Camera Movement.

Para a luz, vamos apenas dar um duplo clique sobre ela, e no quadro que aparecer, escolha *Light Color*, e ajuste a intensidade do brilho para 128, e Color para um azul índigo.

Você já está pronto para testar o seu primeiro level. Antes disso basta ir até o menu Build e clicar em ***Build All.*** Esse procedimento de *rebuild* é sempre necessário antes de se testar qualquer level, pois é nele são realizados os cálculos para os testes. Caso a sua janela de perspectiva esteja com o *real-time render* regulado para *Dynamic Light* (Alt+5), você irá ver a incidência da cor azul atuando nas paredes da sua sala. Agora basta salvar, voltar em Build, e clicar em *Play Level* (Ctrl+P). Quando salvar este level, salve-o na pasta Maps. O jogo será inicializado. Divirta-se por alguns instantes.

Você irá correr por uma sala com luz azul, armado com um Assalt Rifle, e quando acabarem as balas, provavelmente vai enjoar e querer sair. Para sair aperte a tecla de ~ (til) ou "aspas" (ns teclados ABNT) para baixar o console, e digite EXIT. O jogo será encerrado e você voltará ao UnrealEd.

Parabéns, você criou e já testou o seu primeiro level.

Agora vamos implementá-lo para que fique um pouco mais divertido e um pouco mais bonito.

Adicionando as texturas

Adicionar as texturas também é muito simples, e você pode inicialmente partir de algumas texturas que o próprio Unreal traz com ele. Abra o browser de texturas, clicando num ícone que lembra uma fotografia (ver Figura 1).

Esse browser apresenta as texturas compactadas em pacotes .utx. É um formato de compressão da engine para as texturas. Vá ao texture browser e role o menu pull-down até o pacote chamado AbaddonArchitecture, que possui algumas texturas que poderemos utilizar neste momento. Para aplicar uma textura dessas, basta, na ajnela de perspectiva, selecionar a parede a qual irá receber a textura, e clicar sobre a textura. Ela será imediatamente aplicada. Para ajustar o tamanho, direção e posição da textura aplicada, selecione-a e vá em *View / Surface Properties* (F5). Experimente mudar a cor da luz para um verde claro, com um clique duplo na lâmpada e ajustar LightColor no quadro Light Properties. Depois veja o resultado com *Build All, Save, e Play All.* Agora já começa a ficar com cara de videogame.

Figura 3: Primeiro teste do level rodando na engine.

Criando texturas personalizadas

No entanto, nem sempre as texturas do Unreal, por melhores que sejam, serão suficientes para o seu projeto. Normalmente você irá necessitar de texturas próprias, quer sejam fotografias, ou imagens feitas em um software de desenho. O importante é que essas texturas tenham um tamanho, em pixels, cuja altura e largura sejam sempre potência de 2. 16x64, 128x256, até um máximo de 2048 pixels, para computadores mais poderosos. Normalmente 512 pixels são suficientes para texturas de alta-resolução, dependendo da superfície a serem aplicadas, claro.

Para importar suas próprias texturas será necessário antes fazê-las em algum editor de imagens, como o *Photoshop*, desde que este possa salvá-las no formato BMP ou TGA.

Depois de volta ao UnrealEd, é necessário fazer **um novo pacote .utx,** sem textura alguma por enquanto, apenas para guardar as suas texturas sem estragar as do jogo.

Para criar um pacote desses, abra o browser de texturas (veja a Figura 1), e em *File / New,* crie um material novo, nomeando o campo *Package,* com um nome não muito criativo, como Teste, e no campo *Material* escreva outro nome qualquer, como Teste. Para um ambiente simples como esse não é necessário criar Grupos de texturas, então deixe esse campo no padrão Base, ou em branco. Clique em New, e apenas feche o próximo quadro. Esse procedimento é apenas para criar um novo pacote .utx, quando você salvá-lo *(File / Save).* Não é necessário, por enquanto ter textura

alguma dentro dele, visto que as importaremos a seguir. Não se esqueça de salvar esse pacote na pasta de Textures, do contrário o jogo não conseguirá localizá-lo, e salve-o sempre que atualizar as texturas dele, pois isso recomprime e atualiza o pacote, permitindo então qua a engine utilize as texturas.

Crie algumas texturas com o *tile* aparente, como uma borda mais clara, por exemplo. Isso vai facilitar enquanto você estiver aprendendo. Aqui vemos 3 texturas .tga, 128x256 pixels, sem um canal alpha, portanto em 24 bits de resolução.

Embora isso seja comum, não custa lembrar, de que espaços e caracters especiais, como acentos, não devem ser utilizados ao se nomear esse tipo de arquivos.

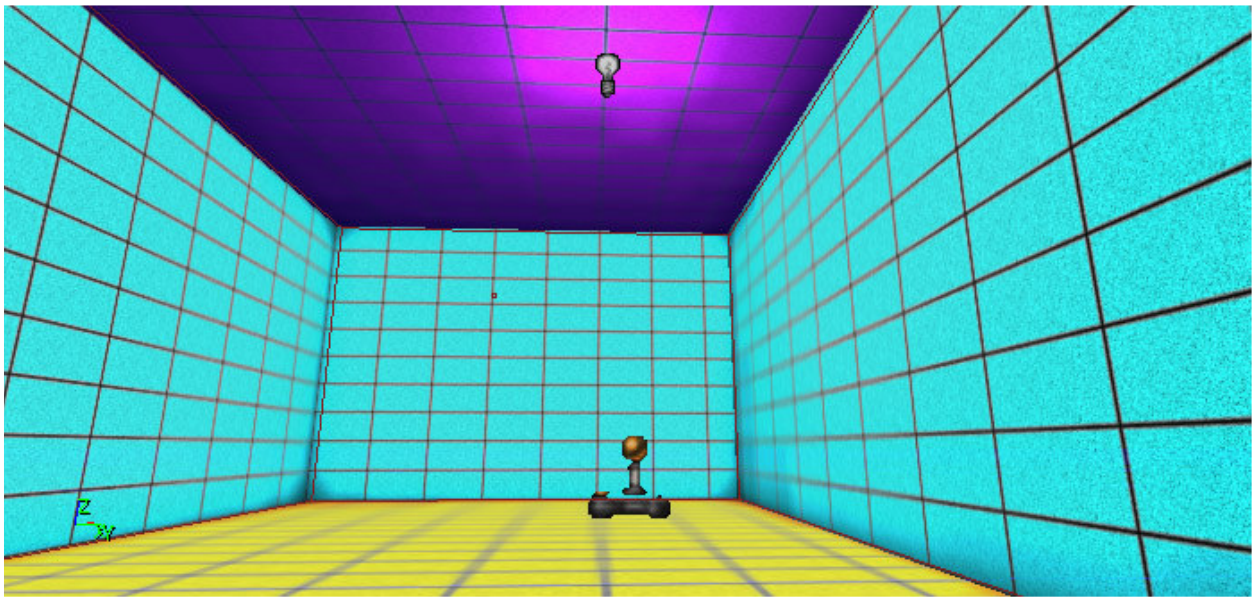
Uma vez que você esteja de posse desses aquivos, no browser de texturas selecione o pacote que você criou (o Teste, por exemplo) e importe as texturas para dentro dele, com *File / Import.*

No request de importação, você deve neste momento deixar o LOD, Level of Detail, em 0 (zero), que é o máximo de qualidade para a sua textura, deixar o gerador de MipMaps clicado, para a engine calcular a melhor viusalização dependendo da distância, e se houver algum canal alpha na sua imagem, selecione-o também.

Depois você pode deletar o material bobo que você criou, como o *Teste,* clicando com o botão direito do mouse, e escolhendo a opção *delete.* Quando acabar de editar esse pacote .utx, não esqueça de salvá-lo, com File / Save.

Agora aplique as texturas na sua sala, paredes, teto e piso, e ajuste-as com o editor de Surface Properties, em View / Surface Properties (F5). O tab de Pan/Rot/Scale será importante em todas as tarefas de ajuste de texturas. Experimente-o bastante, e não se esqueça de selecionar a textura (janela de perspectiva, botão esquerdo do mouse) que você estará atuando. A janela de perspectiva também deve estar regulada para fazer os shadings em tempo real (Dynamic Light, Alt+5) caso contrário você não perceberá as atualizações acontecendo no seu level.

Agora você já pode fazer o rebuild, salvar o seu level, e testar novamente. Goze do seu level por mais alguns instantes e vamos deixá-lo um pouco mais complexo.



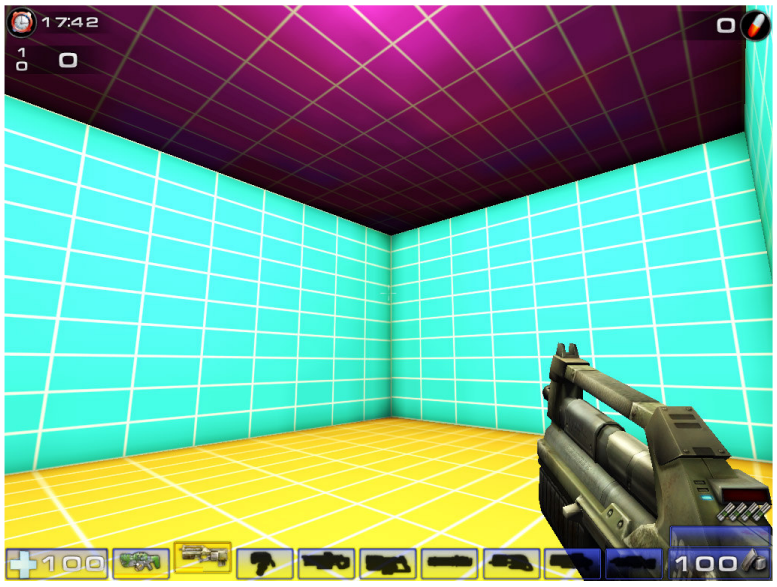


Figura 5: O level rodando na engine, com as suas texturas.

Auntentando a complexidade do level

Agora que temos um primeiro level pronto, vamos aproveitá-lo para iniciar um level de duelo.

Um level de duelo é um pouco mais complexo do que uma sala apenas. Ele comporta dois jogadores, tem uma simetria ou assimetria que permitem alguns esconderijos simples para se esconder das balas, e alguma recompensa ou objeto motivador para que o jogo não vire um chato esconde-esconde.

Iremos aproveitar essa pequena sala, como uma pequena ligação para duas salas com o dobro do tamanho dessa. Assim, crie um novo BSP Brush, com o dobro das medidas, e recorte as salas aproximadamente como na *Figura 6.*

Quando estiver fazendo o level, baseie-se nas vistas ortogonais, Top, Front, Side, pois a vista de perspecitva normalmente tem de ser reconstruída. Quando você aplicar as subtrações, faça o rebuild (Build All), e no final teste na engine com Play Level.

Nesse momento você irá notar duas coisas importantes. A primeira é que você precisa de mais iluminação para as salas novas, uma lâmpada central em cada, ou lâmpadas mais fracas espalhadas pelo teto, dependendo do efeito que você quiser proporcionar. A segunda é que as texturas que o UnrealEd utilizou durante a subtração eram as que estavam selecionadas no seu browser de texturas. Assim, você vai ter de retexturizar algumas paredes. Invista um tempo nisso e faça vários testes de iluminação antes de acrescentarmos alguns elementos para aumentar a jogabilidade.

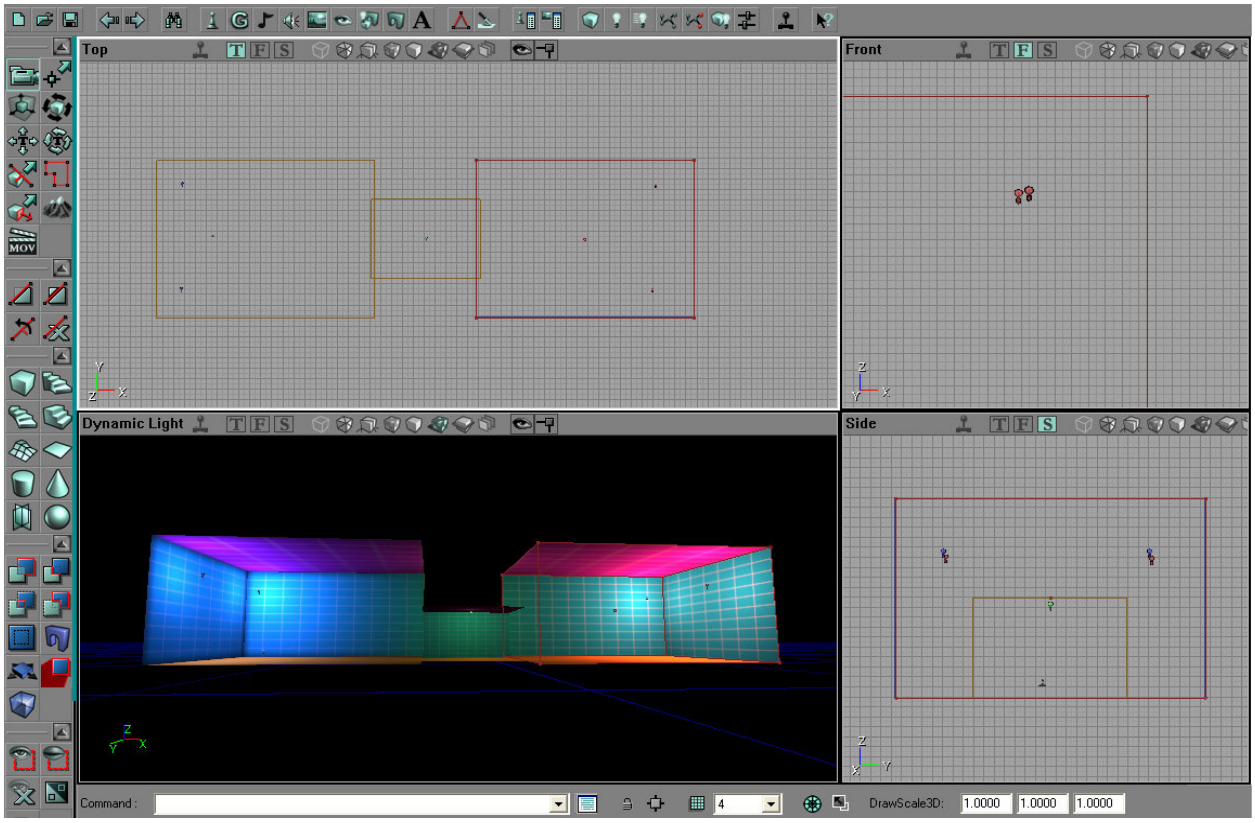


Figura 6: Level de Duelo.

Um pouco de game design

No exemplo acima, como a geometria das salas é bastante simples, procurei criar um jogo de sombras, para que os jogadores pudessem se esconder nas sombras. Para isso posicionei duas lâmpadas médias de cada lado (LightBrightness = 128), com cor azul de um lado, e vermelha do outro, para que o jogador tenha alguma noção da sala na qual ele se encontra. Isso será importante no decorrer do jogo, pois para criar um elemento de motivação, deixarei uma arma

forte de cada lado, mas a munição dela estará no lado do inimigo, e não do lado da arma, como de costume. Isso fará com que o jogador seja obrigado a invadir o campo inimigo para pegar munição, ou ficará apenas com uma arma fraca, o Assault Rifle, padrão do jogo.

Isso tornará o jogo um pouco mais interessante, mas ele ainda está bastante linear. Os comportamentos padrões de cada jogador podem ser facilmente assimilados e o jogo ficará chato em questão de minutos, ainda mais porque cada jogador não deve demorar muito para morrer. Para diminuir essa linearidade, iremos acrescentar um elemento de recompensa, um objeto de desejo de ambos os jogadors que aparecerá randomicamente, em um lugar perigoso, bem no centro da arena: um repositor de saúde. De posse desse elemento, o jogador dura mais tempo, e pode assim aumentar o seu número de frags. Esse fator ficará bem no meio da arena, num lugar propositadamente pouco iluminado, para que a sua luz chame a atenção de ambos os jogadores, que devem então se arriscar para possuir esse elemento, ou se posicionar para evitar que o outro o possua, ao mesmo tempo que aumenta as suas chances de pegá-lo.

Assim, com esses poucos elementos temos a oportunidade de testar um level interessante, que permitirá avançarmos em outras oportunidades a levels mais complexos, se bem que sua complexidade apenas, não seja garantia alguma de um jogo interessante. Ao contrário, pode confundir os jogadores a maior parte do tempo, e levar a ação na direção do tédio, e fim prematuro do jogo como um todo.

De volta ao level

Vamos acrescentar inicialmente o fator de saúde bem no centro da arena.

Todos esses pequenos objetos de programação, como a luz e o Start Point, que acrescentamos em nosso level, a engine do Unreal os chama de Actors. Assim, os outros elementos que iremos acrescentar agora, são nada mais do que outros Actors, ou seja, outros objetos pré-programados.

Vamos iniciar acrescentando um repositor de saúde, chamado HealthCharger. Todos esses objetos que podem ser pegados do chão, o UnrealEd agrupa em uma classe chamada *xPickUpBase,* ou seja, uma base para pegar alguma coisa, x.

Para adicioná-lo ao centro da arena, é o mesmo procedimento de adicionar os outros elementos, com a diferença que você deverá estar com o browser de actors abertos, e com o objeto HealthCharger selecionado. Você vai achar esse objeto bem no fim da lista de actors, debaixo do link xPickUpBase. Então basta clicar com o botão direito na vista de perspectiva, no lugar aonde você quer que o HeathCharger apareça. A base irá aparecer e você pode posicioná-la mais adequadamente nas vistas ortogonais. Tome cuidado para que a base fique encostada no chão. Do contrário, quando você fizer o rebuild, o debugger irá dar uma mensagem que a sua base está *floating,* flutuando, e você não conseguirá rodar o seu level.

Teste o seu level, e vamos acrescentar as armas.

Da mesma maneira que você acrescentou o HealthCharger, acrescente agora o xWeaponBase, ele está bem próximo no browse actors, na mesma categoria do anterior. Lembre-se de colocá-lo encostado no chão, para não dar o erro de *floating*.

Agora precisamos explicar quais são as armas que aparecerão sobre essas bases. O procedimento é muito simples e bem semelhante aos outros que você já realizou neste tutorial. Basta acionar a janela de xWeaponBase Properties com um duplo clique sobre o objeto, e expandir a opção xWeaponBase. Clique no campo ClassXWeapons e escolha a arma que você achar mais indicada. Neste level eu optei por armas que não fizessem repetição automática, para que os jogadores possam durar mais tempo na arena, que é bem pequena. No caso, escolhi o ShockRifle e o Rocket Launcher. As duas armas apresentam vantagens e desvantagens semelhantes, característica de um excelente balanceamento de armas que este jogo apresenta.

Vamos então às munições.

As munições encontram-se na categoria *PickUp,* no browser de *actors.* Para ficar um pouco mais difícil de pegar as munições, iremos colocá-las no alto de uma caixa. Isso vai aumentar o dinamismo do jogo, e serve também para que possamos ver um pouco da CSG aditiva.

Para construir as caixas ou cilindros, pegue as ferramentas apropriadas e faça os BSP Brushes. Posicione os brushes e clique no icone de Add, ao lado do subtract que temos usado até agora.

Apenas para aumentar o desafio de pegar a munição, eu preferi construir duas caixas, e não apenas uma. Uma tem 200 unidades de altura, a outra menor, que terá de ser usada como degrau, 100 unidades. Também coloquei o degrau de frente para o território inimigo, ou seja, para pegar a munição deve-se dar as costas ao inimigo e suportar as conseqüências disso. E para piorar ainda mais, um pulo simples não deve ser suficiente, devido à altura da caixa, e será necessário que seja um daqueles pulos duplos do Unreal, bem dado, senão o jogador não alcancará a munição. Claro que para compensar tanto esforço, devemos fazer valer a pena, e então aumentei o número de munições de 10 para 20, no quadro *ShockAmmoPickup Properties.* A "cereja do bolo" fianl são as texturas que fiz para dar bem a idéia da roleta russa que se enfrenta para poder se pegar a munição: um par de dados.

Se estiver difícil de visualizar as texturas para aplicar nas caixas, recomendo que regule o *render* da janela de perspectiva para Textures (Alt+6). Isso cancela a influência da luz e permite uma melhor visualização da colocação das texturas.

Por fim, refaça as caixas do outro lado da arena, e prepare-se para colocar o seu inimigo.

Conhecendo o Inimigo

Os inimigos, ou outros jogadores, no caso de jogos multiplayer, são inseridos em outros Start Points. Mas no caso dos inimigos, os bots, não é necessário apenas o Start Point, mas paths que indiquem as suas possibilidades de deslocamento.

Adicione um outro Player Start na outra sala. Lembre-se que para orientá-lo na direção certa deve-se utilizar a ferramenta de Actor Rotate com o botão direito do mouse. Vamos adicionar agora um tipo de actor chamado PathNode, que explica para os bots aonde eles podem andar. Esse é apenas um dos tipos de paths que podem ser utilizados. Existem vários outros, como pegar elevadores ou defender pontos estratégicos, que podem fazer seus bots parecerem muito inteligentes, outra característica importante dessa engine.

Comece acrescentando um *PathNode* próximo ao Start Point, se possível entre ele e uma primeira arma. Isso permitirá ao seu bot se armar rapidamente. Depois crie outro Start Point passando por trás das caixas, entre os cantos, e cruzando para o outro lado da arena.

Você vai ver pequenas maçãs aparecendo no mapa, mas você pode ver também os *paths* que elas estão gerando. Com o botão direito clique sobre o nome da vista que você quer ver os paths, e escolha *View / Show Paths.* Você vai ver que existe um path automático que liga os Start Points, as armas e as munições, mas os que foram criados agora não aparecem. Para que eles apareçam, a engine precisa calculá-los através de um rebuild. Então faça um Build All e veja os paths aparecerem.

Finalmente vamos ao jogo. Vá em *Play Level,* e entre no jogo, você vai ver que não há inimigos ainda. Desça o console com ~ (til) ou "aspas" para teclados ABNT. Escreva *addbots 1* e *Enter.* Recolha o console e veja o seu bot já atirando em você. Na próxima vez que for fazer isso, esconda-se antes atrás das caixas, porque o tempo de recolher o console é suficiente para levar uns tiros bem chatos. Se você quiser mais emoção, adicione mais bots, com *addbots 2, addbots 3, addbots 4,* dependendo da sua coragem, só não se esqueça de criar os Start Points necessários para cada um deles.

Concluindo a prática

Esse foi um pequeno tutorial para vermos como funciona uma engine como essa. Com esses elementos já é possivel se prototipar e fazer diversos testes para verificar com facilidade conceitos de design, que necessitariam de programação bem pesada. Com esse pequeno protóptipo foi possível se verificar que os bots não subiam nas caixas para pegar munição, eles preferiam ficar no chão e pegar a munição diretamente das armas, e não se arricar a subir nas caixas, deixando assim o jogo mais linear. É bem possível que jogadores humanos tivessem a mesma atitude. Para resolver isso, deixei as armas em cima das caixas, e a munição no chão, além de colocar um *JumpDes* para forçar o pulo do bot, caso contrário ele não subiria nas caixas, e o level ficaria muito fácil com o bot armado apenas com um rifle. Isso é apenas um pequeno exemplo de uma técnica de modificação sendo usada como prototipagem, teste de conceitos, e até mesmo como produto final. Agora vamos retomar alguns conceitos e ver como eles podem ser testados dentro dessa engine.

Usando a modificação para testar conceitos

É claro que a utilização das ferramentas de modificação para a criação de mapas e *levels* novos, pode ser uma atividade divertida e uma ótima experimentação do procedimento para criação de games, embora não seja sua única função. Para os game designers, a modificação pode ser um excelente campo de provas para testar conceitos, experimentar novas formas de jogo, variar o balanceamento ou mesmo se tornar um ambiente de prototipagem.

Se em um primeiro momento, é preciso conhecer esse ambiente de produção utilizando os elementos que são fornecidos com o próprio game, a segunda etapa é partir para mudanças mais profundas. É esse o caso das chamadas *total convertions*, como o **Red Orchestra** [13], ganhador de vários prêmios, e uma das mais famosas modificações completas para Unreal Tournament®. A partir da recriação dos cenários, dos personagens, da história e interface, o jogo propõe uma atmosfera ligada à 2ª Guerra Mundial. Hoje o Red Orchestra, seguindo o exemplo de outros mod-games famosos como Counter Strike, se tornou um jogo independente.

Para evidenciar o potencial das modificações, podemos elencar algumas que chegam inclusive a mudar o gênero como é o caso do **UnWheel** [14] que parte da engine do Unreal e o transforma em um jogo de corrida ou o *Alien Swarm* que transforma o *Unreal Tournament* em um jogo de ação e estratégia.

A prototipagem também é uma possibilidade de uso dos mod-games ajudando a visualizar melhor uma idéia com custo baixo e menor tempo de desenvolvimento, já que envolve menos etapas que o processo padrão de produção de um game. Ao partir de uma estrutura semi-pronta, é possível ao game designer aprimorar suas idéias e obter resultados muito mais rápidos, economizando tempo e recursos e, o mais importante, possibilitando maiores experimentações. Um dos grandes casos de prototipagem é a utilização do Neverwinter Nights, para a experiência educacional do **Revolution** [15], projeto do Educational

Arcade, onde os game designers puderam trabalhar com uma estrutura conhecida e experimentar as possibilidades de adequação de uma proposta educacional ao ambiente de um jogo. Após esta etapa, o grupo procurou uma engine para efetivamente produzir um novo jogo.

Outro campo que têm crescido com a possibilidade de modificação de games, é justamente o campo da Arte, já que a utilização das ferramentas de modificação tornam um campo acessível de trabalho e principalmente um espaço interessante para se testar conceitos, matéria-prima dos artistas.

O trabalho de artistas como o de Anne-Marie Schleiner's do Velvet Strike [16], se apropria de elementos presentes nos games para discutir conceitos relativos não apenas ao próprio game, como também ao mundo. O trabalho **Realidades Alternativas** [17] também parte de ambientes conhecidos pelos gamers e procura resignificar elmentos e questionar alguns elementos como a capacidade de detectar perigos, alterações estruturais e mesmo gerar dificuldades como a falta de iluminação, envolvendo impulsos sensoriais visuais e sonoros distintos, questionando a percepção do jogador.

Testando alguns conceitos no Unreal

É a partir de agora que procuramos testar algumas situações contrastantes para refletir sobre a criação da modificação do **Unreal Tournament 2004**, para testar alguns conceitos.

O jogo Unreal, em sua versão 2004, é sem sombra de dúvida um dos jogos mais modificados de todos os tempos. A versão em DVD é acompanhada de 11 modificações, mais as centenas, sem exagero, de mapas e modificações que podem ser achadas em comunidades de modificadores, como a *Beyond Unreal* e a *Planet Unreal*.

Inicialmente não podemos confundir a criação de mapas, as arquiteturas dos terrenos e construções, com a criação de levels. Os levels, incluem os paths e os nodes aonde serão

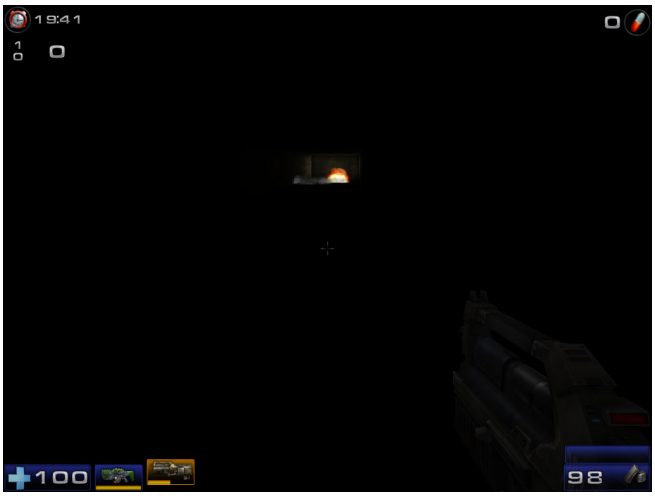
dispostas a inteligência artificial, além dos pontos de saída do jogador e dos bots, disposição das armas, munições, puzzles e saúde, entre outros ítens.

O **balanceamento** pode ser testado de maneira efetiva através da construção de mapas, levando em consideração elementos como tempo de resposta do usuário, habilidade motora, quantidade de inimigos e outros. Na imagem acima, temos um mapa que foi gerado para reproduzir uma condição extrema. Muitos inimigos, plataformas separadas, um vão enorme, e sem nenhuma arma. Será que alguém consegue atravessar este nível?

Podemos também imaginar a utilização de ambientes claustrofóbicos e testar a reação do jogador dentro destes mapas. Pouco espaço para movimentação, visão interrompida e um ambiente fechado ajudam a caracterizar ambientes claustrofóbicos.

Níveis muito fáceis também podem ser extremamente frustrantes ao usuário, pois não geram tensão e não motivam o usuário. Grandes espaços a serem percorridos, poucos inimigos, nenhum desafio. Prontos para criar mais um fracasso de vendas. :)

O tempo de jogo é extremamente importante, em um FPS (*First Person Shooter)* por exemplo, ao exigir que o usuário percorra distâncias enormes, com poucos atrativos e baixíssima dificuldade, quebramos o ritmo que o jogo se apresenta.



Efeitos de ambiente como a **iluminação** ajudam a caracterizar situações com maior ou menor tensão. Clássicos como *Silent Hill* e o recente *F.E.A.R.* utilizam tais elementos com maestria.

Pensar tais elementos pode facilitar o trabalho do game designer, trazendo elementos muito mais acessíveis para testar conceitos importantes, sem desperdiçar tempo iniciando todo o trabalho a partir do zero em uma engine dedicada.

DOWNLOAD

Este material, assim como seus arquivos e outros materiais, estão disponíveis para download no site Comunidade Gamecultura, seção de Downloads, no endereço: www.gamecultura.com.br.

Notas e Referências Bibliográficas

[1] ROLLINGS, Andrew & MORRIS, Dave (2004). *Game Architecture and Design.* A New Edition. Indianapolis: USA: New Riders.

[2] ibid, 247.

[3] TAVARES, Roger (2005). *Fundamentos de game design para educadores.* Texto apresentado no I Seminário Jogos Eletrônicos, Educação e Comunicação - construindo novas trilhas, no GT – Desenvolvimento de Games. UNEB, Salvador – Bahia, outubro/2005.

[4] TAVARES, Roger (2005). *Games Theory: when reading is not enough.* Poster apresentado no SBGAMES 2005. Universidade de São Paulo, 23 a 25 de Novembro de 2005

[5] PRENSKY, Mark (2001). *Digital Game-based learning.* Versão e-Book. USA: McGraw-Hill.

[6] Existem dezenas de títulos desenvolvidos nessa engine, para diversos gostos e públicos, indo do FPS *Star Wars: Republic Commando* (Activision Deutschland GmbH, 2005), ao infantil *Harry Potter and the prisioner of Azkaban* (Electronic Arts, 2004), passando por XIII (Feral Interactive, 2003), Deus Ex: Invisible War (Eidos, 2003), e America's Army (U.S. Army, 2002). Veja duas listas mais completas nos links a seguir (ambas, out/2006):

> Títulos desenvolvidos com a engine Unreal 1 e 1.5:
> http://www.mobygames.com/game-group/3d-engine-unrealengine1
>
> Títulos desenvolvidos com a engine Unreal 2, 2.5 e 2X:
> http://www.mobygames.com/game-group/3d-engine-unrealengine2

[7] Conforme anunciado no site oficial do título, em www.pariahgame.com (out/2006).

[8] *Epic and EA announce Unreal® Engine 3 adopted for select next-gen titles.* Release online em: http://www.info.ea.com/news/pr/pr811.pdf (out/2006).

[9] *Sony Online Entertainment licenses Unreal® Engine 3.* http://www.epicgames.com/press_releases/sony_ue3.html (out/2006).

[10] Pode-se obter a versão PLE do Maya (Personal Learning Edition), para a criação de conteúdos não comerciais neste link: http://usa.autodesk.com/adsk/servlet/item?siteID=123112&id=7679012 (out/2006).

[11] Software Open Source gratuito para a gravação e edição de sons. Disponível para Windows, Mac e Linux/Unix. http://audacity.sourceforge.net/download/ (out/2006).

[12] BUSBY, Jason; PARRISH, Zak, VAN EENWYK, Joel (2005). *Mastering Unreal Technology. The Art of Level Design.* Sams Publishing. Indianapolis, Indiana.

[13] http://www.redorchestragame.com/ (out/2006)

[14] http://unwheel.beyondunreal.com/ (out/2006)

[15] O Revolution faz parte de uma iniciativa do grupo Education Arcade para a utilização de games em sala de aula. http://www.educationarcade.org/revolution (out/2006)

[17] http://www.opensorcery.net/velvet-strike/

[16] TAVARES, Roger & NEVES, Felipe (2006). Realidades Alternativas. Gamearte exibida no evento do Sesc Consolação. Julho a agosto de 2005. São Paulo, SP. http://www.gamecultura.com.br/realt (out/2006)

Internet

A gamearte **Invisible War** *::* **Dominação :: Fase I**, pode ser baixada gratuitamente em http://gamecultura.blogspot.com ou em www.felipe.pro.br/gameart.zip

As referências sobre games, como data da publicação ou produtora, podem ser conferidas em www.mobygames.com

Fontes de referência para UnrealEd na web:

Epic Games: Site da desenvolvedora da tecnologia Unreal. http://www.epicgames.com (out/2006)

Unreal Technology: Site da tecnologia. Contém informações para licenciamento. http://www.unrealtechnology.com (out/2006)

UDN – Unreal Developer Network: http://udn.epicgames.com

Beyond Unreal. Um dos principais portais para modificadores de Unreal. Alguns fóruns contém

mais de 100 mil respostas. www.beyondunreal.com.

Unreal Wiki: http://wiki.beyondunreal.com/wiki/

Planet Unreal: Outro ativo portal para Unreal. http://www.planetunreal.com/

Fórum na Epic Games: fórum oficial suportado pela desenvolvedora.

http://gearsforums.epicgames.com/forumdisplay.php?f=249

Unreal Playground: Mais uma comunidade hiperativa de Unreal.

http://www.unrealplayground.com (out/2006)

Brute – Comunidade Brasileira de Unreal: http://www.brute.com.br (out/2006)

Expandindo Unreal

NextUp.com: Esta empresa produz e vende vozes mais naturais para Windows e games, através do AT&T Natural Voice Engine. http://nextup.com/nvgames.html

Roger Tavares, Msc, PhD, lecturer and educational designer, obtained a master degree in Education, Art and History of Culture at Mackenzie University, and your doctoral degree in Communication and Semiotics at Pontifícia Universidade Católica de São Paulo, PUC-SP, with the thesis Videogames: Toys of the Posthuman (www.gamecultura.com.br/thesis). He teaches 3D Animation in the School of Communication, Arts and Design at Senac Universitary Center at São Paulo – Brazil. He is, also, member of the research groups in games at the aforementioned institutions, and of the Self-Reference in Media project, videogames teamwork, in the School of Linguistics in Kassel Univeristy, Germany. He has regularly published in related books and congresses. His company, GameCultura, has supplied service to companies and public institutions in the consulting area and courses with videogames, in an attempt to reduce of prejudice against this media, and in this way, to allow its use as a tool for motivation as well as social and digital inclusion. E-mail: rogertavares@gmail.com

Felipe Neves, professor and multimedia designer is a Specialist in Design, who has been developing multimedia projects since 2000, and is also finishing his master's degree in Communications and Semiotics at Pontifícia Universidade Católica de São Paulo, PUC-SP. He teaches Multimedia Authoring at Centro Universitário Senac de São Paulo - Brazil, School of Communication, Arts and Design. Member of a Research Group in Games: Culture, Education and Design at Senac-SP; and also a member of an Non-Governmental Organization called Visual Litteracy [www.alfabetizacaovisual.com.br]. E-mail: felipeneve@gmail.com