Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais

DESENVOLVIMENTO DE JOGOS DIGITAIS 2D COM UNITY3D

NELSON TORETTI JUNIOR

Americana, SP
2013

Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais

DESENVOLVIMENTO DE JOGOS DIGITAIS 2D COM UNITY3D

NELSON TORETTI JUNIOR

ntorettijr@gmail.com

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais, sob a orientação do Prof. José William Pinto Gomes.

Área de concentração: Jogos Digitais

Americana, SP
2013

FICHA CATALOGRÁFICA elaborada pela
BIBLIOTECA – FATEC Americana – CEETPS

Toretti Junior, Nelson
T649d Desenvolvimento de jogos digitais 2D com Unity 3D. / Nelson Toretti Junior. – Americana: 2013.
50f.

Monografia (Graduação em Tecnologia em Jogos Digitais). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. José William Pinto Gomes

1. Multimídia 2. Jogos eletrônicos 3. Desenvolvimento de software I. Gomes, José William Pinto II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.6
681.3.05

Bibliotecária responsável pela FC: Ana Valquiria Niaradi – CRB-8 região 6203

Faculdade de Tecnologia de Americana

Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais

Nelson Toretti Junior – RA 0040441023001

Desenvolvimento de Jogos Digitais 2D com Unity3D

Trabalho de Graduação aprovado como requisito parcial para a obtenção do título de Tecnólogo em Desenvolvimento de Jogos Digitais da Faculdade de Tecnologia de Americana. PUBLIQUE-SE.

Banca Examinadora

Orientador: _____

Prof. José William Pinto Gomes

Professor Convidado: _____

Prof. Me. Ivan Menerval Gomes

Professor Convidado: _____

Prof. Gustavo Carvalho Gomes de Abreu

Americana, SP

2013

AGRADECIMENTOS

Primeiramente, agradeço a Deus por todas as oportunidades que tive nessa vida.

Agradeço aos meus pais que me apoiaram em todas as decisões, acreditando em mim e nos meus objetivos.

Ao professor José William Pinto Gomes por todo o suporte a este trabalho.

Agradeço também a todos os colegas de classe e professores que me acompanharam e apoiaram em toda essa jornada.

DEDICATÓRIA

Dedico este trabalho a todos os artistas e programadores que deram vida aos fantásticos mundos encontrados nos jogos digitais.

RESUMO

O presente texto visa explorar a possibilidade de uso do motor de jogo Unity 3D, uma plataforma focada no desenvolvimento de jogos digitais com gráficos e ambiente tridimensional, para a criação de títulos que utilizem jogabilidade e estética em duas dimensões. Com uma abordagem ampla, o estudo contido nesta pesquisa faz uma análise sobre a criação e manipulação de conteúdo para jogos digitais com estética 2D e a interação com o ambiente do motor de jogo Unity3D. Por fim, aplica-se o conteúdo para a criação de um projeto real com controles e elementos gráficos em duas dimensões.

Palavras chaves: Unity3D, Jogos Digitais, Jogos em Duas Dimensões

ABSTRACT

This paper aims to explore the possibility of using the Unity3D game engine, a platform used mainly to develop games with three-dimensional graphics and space, for creating titles in two dimensions. With a broad approach, the study contained in this research performs an analysis about the creation and management of assets for two-dimensional digital games and the interaction with the Unity3D workspace. Finally, it applies the concepts in a real example of a two-dimensional game both control and graphics wise.

Keywords: Unity3D, Games, two-dimensional games

SUMÁRIO

LISTA DE ILUSTRAÇÕES

INTRODUÇÃO

Uma vez necessidade devido ao fraco processamento gráfico dos computadores e consoles das décadas anteriores, o uso da orientação e elementos gráficos em duas dimensões nos jogos digitais se tornou uma opção estética e é encontrado principalmente em títulos lançados para dispositivos móveis e serviços de distribuição digital, favorecidos pelos desenvolvedores independentes que possuem recursos limitados para a criação de conteúdo.

Diferentemente dos jogos digitais tridimensionais, para se desenvolver um projeto em duas dimensões as ferramentas disponíveis no mercado são limitadas na questão de portabilidade e flexibilidade, além disso, muitas delas possuem pouca documentação ou uma linguagem própria de programação, dificultando o acesso a desenvolvedores iniciantes e adicionando mais um nível de dificuldade no projeto. A intenção deste trabalho é avaliar se é possível o uso de uma ferramenta focada no desenvolvimento em 3D, no caso o motor de jogo Unity3D, e aplicar os conceitos e técnicas para a criação de títulos 2D utilizando-se de sua estrutura flexível.

Este trabalho está dividido em sete capítulos, com o primeiro contextualizando o termo jogo digital na atualidade; o segundo introduz conceitos matemáticos e geométricos para compreensão e manipulação de dados dos objetos no espaço; a terceira parte mostra a evolução e uma explanação dos principais elementos gráficos utilizados nos jogos digitais; no quarto capítulo temos uma breve apresentação do motor de jogo Unity3D; seguindo no quinto capítulo é demonstrada a aplicação dos conceitos apresentados no ambiente tridimensional da Unity enquanto que a sexta parte apresenta um projeto desenvolvido utilizando dessas técnicas. Por fim, o último capítulo conclui o trabalho com a apresentação de resultados obtidos e indica novas possibilidades de pesquisa.

1. JOGOS DIGITAIS NA ATUALIDADE

Algumas décadas se passaram desde a introdução dos primeiros jogos digitais e nesse tempo, uma grande indústria se formou. De acordo com os dados fornecidos pela empresa Microsoft, o mercado dos jogos digitais movimentou 65 bilhões de dólares no ano de 2012, liderados pelos consoles que proporcionaram 27 bilhões desse total, crescendo num ritmo mais acelerado do que outros tipos de entretenimento como televisão e cinema e quebrando recordes como o título Grand Theft Auto V (Rockstar Games, 2013) que alcançou a marca de um bilhão de dólares em apenas três dias de vendas de acordo com Capelas (2013).

As estatísticas da associação ESA[1] sobre o estado da indústria dos jogos digitais no corrente ano mostram que 58% dos americanos têm o costume de utilizar jogos eletrônicos, sendo que sua idade média é de trinta anos, com 55% do público masculino e 45% feminino. Na média, cada residência americana possui um aparelho capaz de reproduzir jogos e pelo menos 51% tem um equipamento dedicado como um console. Um fato interessante é o crescimento das vendas por meio digital, que não passavam de 20% do total em 2009 e alcançou impressionantes 40% no ano passado.

Quanto ao panorama dos jogos digitais no Brasil, a pesquisa Game Pop do IBOPE[2], datada do ano 2012, revela que 11,8 milhões de brasileiros costumam jogar jogos eletrônicos, a grande maioria em consoles ou computadores. Dentre os entrevistados que declararam utilizar smartphones ou tablets, 45% baixam aplicativos de jogos.

1.1. Definições

Segundo Huizinga (2000), o jogo precede a cultura humana, pois mesmo os animais participam de atividades que apresentam muitos dos elementos característicos do jogo conhecido por nossa espécie, sendo algo que transcende os limites da atividade puramente física ou psicológica. No jogo sempre existe algo “em jogo” que confere um sentido a ação.

Observando os termos usados para descrever os conceitos de jogo e diversão em diversas línguas e dialeto, podemos observar que é uma atividade oposta ao conceito de seriedade, mas mesmo assim com suas próprias regras que precisam ser respeitadas. Essas

[1] Entertainment Software Association: associação americana das empresas desenvolvedoras de jogos digitais.

[2] Instituto Brasileiro de Opinião Pública e Estatística

atividades também são ações voluntárias, sendo um ato opcional, muitas vezes praticado num momento de ócio do indivíduo. Como o jogo não pertence à vida comum, ele se situa num intervalo de tempo com um fim determinado, num lugar determinado e só existe para a realização do jogador. Assim se dá a existência do proposto círculo mágico, em que o jogador estará imerso durante a experiência de jogo.

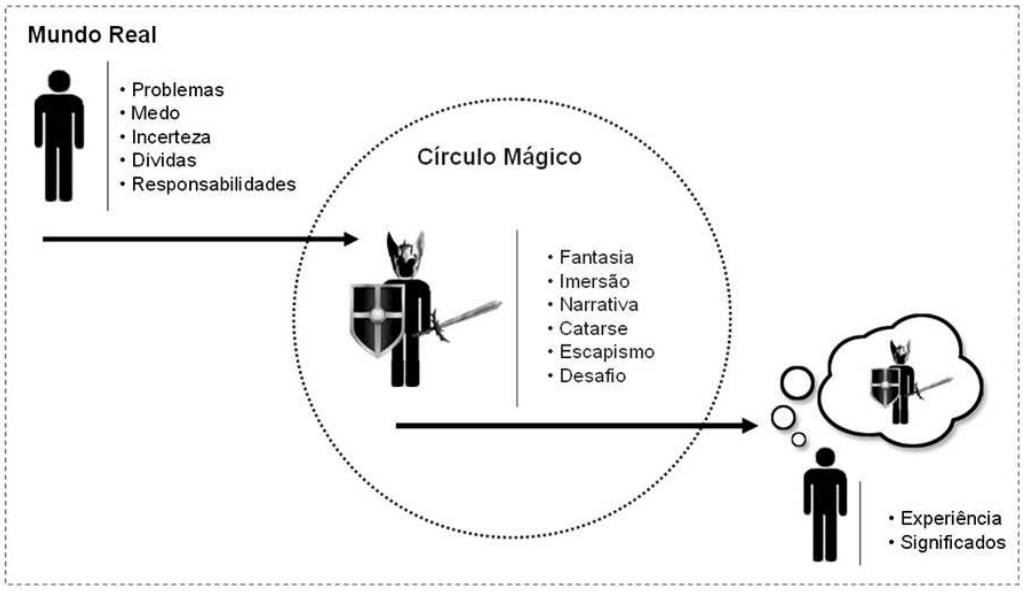

Figura 1 - Ilustração da teoria do círculo mágico de Huizinga (TREVISAN, 2012)

Para Salen e Zimmerman (2004), os jogos digitais são sistemas, como todos os outros tipos de jogos. Os elementos físicos como o computador e a mídia que armazena os dados são apenas uma parte desse sistema, materiais os quais o jogo digital é composto e devem ser usados para desenvolver uma experiência.

2. CONCEITOS GEOMÉTRICOS

Rabelo (2005) afirma que a geometria é o campo da matemática que estuda as formas encontradas na natureza, as observando e criando padrões visando a facilidade de uso para posteriores pesquisas e também para outros ramos de estudo.

Na geometria, existem três elementos considerados como primitivos: o ponto, sendo a forma mais simples e é a base para a criação de todas as formas geométricas; a reta, uma linha infinita que segue em uma direção específica, mas que pode ser parcialmente contida em segmentos envolvidos por dois pontos; e o plano, que pode ser formado dentre outras combinações, por três pontos que em conjunto não compartilhem a mesma reta.

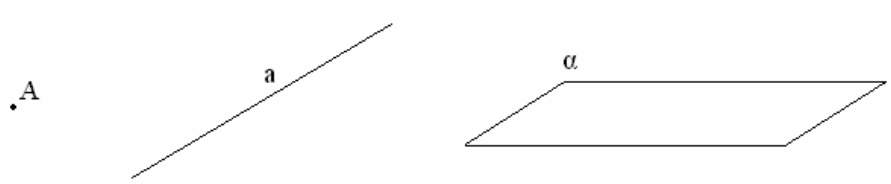

Figura 2 - Representação dos elementos primitivos da geometria: o ponto, determinado por uma letra maiúscula; a reta, indicada por uma letra minúscula; o plano, assinalado por uma letra do alfabeto grego.

2.1. Vetores

Segundo Vince (2006), na maioria das situações cotidianas, utilizamos um número único para representar valores, o que chamamos de grandezas escalares. Porém, alguns elementos necessitam de mais que apenas um número para representá-los, como a força do vento ou velocidade de um corpo, em que temos um valor e uma direção. Essa quantidade é definida como uma grandeza vetorial.

Na matemática, vetores são descritos por uma seqüência de números envolvidos por chaves. Geometricamente, o vetor é representado por um segmento de reta com uma seta em uma das pontas, sendo o comprimento do segmento a sua intensidade e a seta a sua direção (DUNN e PARBERRY, 2011).

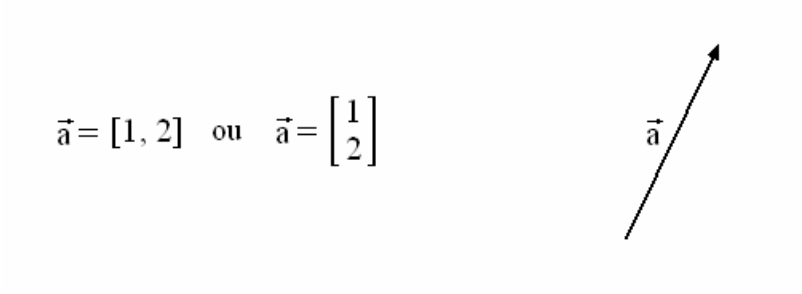

Figura 3 - Notação matemática e notação geométrica de um vetor.

2.2. Definindo espaço 2D e 3D

Para representar graficamente um espaço bidimensional e suas coordenadas, utilizamos o chamado plano cartesiano. Descrito por Descartes (1637), o plano cartesiano possui duas retas que interseccionam num ponto de origem, denominadas por eixo “x” e eixo “y”, que por convenção são dispostas em orientação horizontal e vertical respectivamente em um plano. Os valores são determinados por um par de variáveis, denominados por x e y e sua localização no plano, sendo os valores em x positivos a direita do eixo y, e negativos a sua esquerda e os valores em y positivos acima da linha do eixo x e negativos abaixo da mesma (VINCE, 2006).

Dunn e Parberry (2002) afirmam que para descrever um espaço tridimensional, utilizamos três eixos para criar um sistema de coordenadas, geralmente conhecidos por “x”, “y” e “z” e perpendiculares um ao outro. Diferentemente do sistema cartesiano em duas dimensões, a direção de cada eixo não é padronizada e pode divergir de acordo com o campo de trabalho.

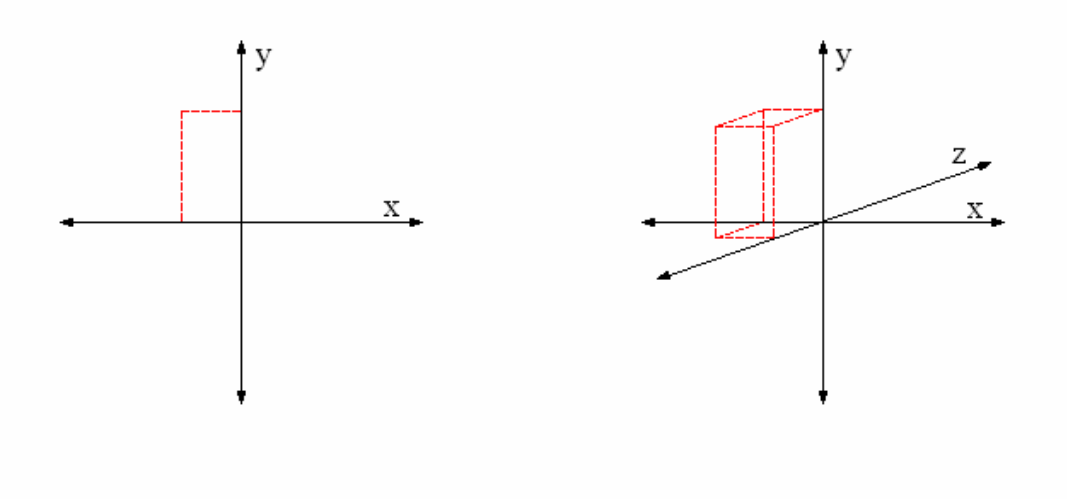

Figura 4 - Plano cartesiano de duas e três dimensões.

Gantzler (2005) declara que a maioria dos desenhos e pinturas representa elementos em três dimensões em uma mídia de apenas duas dimensões, utilizando métodos para referenciar a dimensão faltante. Um desses métodos é chamado de desenho isométrico, em que todas as três dimensões estão claramente representadas em uma única imagem 2D. Outro modo de visualização é a utilização de um conjunto de projeções ortográficas, exibindo todos os lados de um mesmo objeto 3D em imagens diferentes.

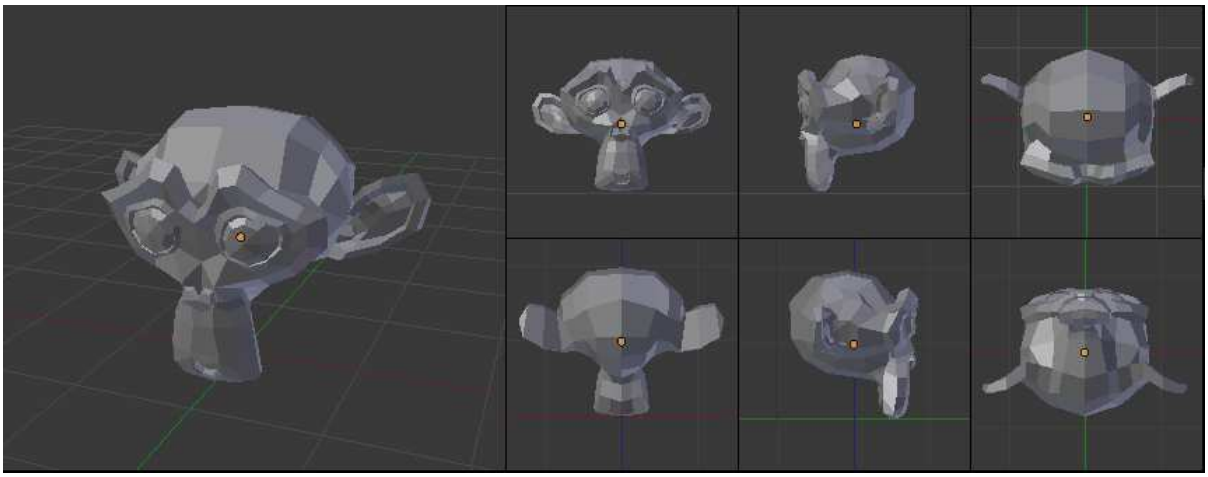

Figura 5 - Visualização isométrica e projeções ortográficas de um modelo tridimensional.

3. ARTE NOS JOGOS DIGITAIS

Bethke (2003) afirma que a qualidade dos aspectos gráficos de um jogo digital influencia sua viabilidade comercial, visto que é o elemento no qual o consumidor tem o primeiro contato, como na sua embalagem. Por esse motivo, os artistas gráficos estão cada vez mais envolvidos em todas as etapas do processo de criação dos jogos, executando novas funções além do tradicional desenho em duas dimensões, como roteiro, modelagem tridimensional e animação.

3.1. Evolução da arte

Segundo Egenfeldt-Nielsen, Heide e Tosca (2013), definir qual foi o primeiro jogo digital é uma tarefa difícil, mas podemos considerar a criação do programa Noughts and Crosses como o ponto de partida. Desenvolvido em 1952 para o computador EDSAC[3] da Universidade de Cambridge na Inglaterra, pelo então estudante de Doutoramento A.S. Douglas, Noughts and Crosses colocava o jogador contra a inteligência artificial rústica do aparelho em uma partida do popularmente conhecido como jogo da velha.

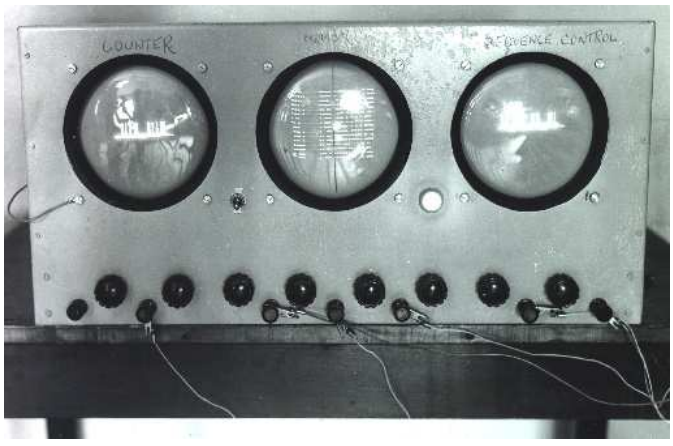

Figura 6 - Fotografia do painel de um computador EDSAC (UNIVERSIDADE DE CAMBRIDGE, 1999).

Pardew (2007) afirma que a arte nos jogos digitais começou até antes que a utilização de monitores convencionais nos computadores, que nesse momento possuíam outras formas

[3] Electronic Delay Storage Automatic Calculator, conhecido como o primeiro computador capaz de executar programas armazenados.

de se comunicar com os usuários, dentre elas um aparelho chamado osciloscópio. Alguns dos jogos criados nessa época para teletipos usavam texto ou figuras compostas com letras para descrever ações e situações que eram enviados de jogador para jogador.

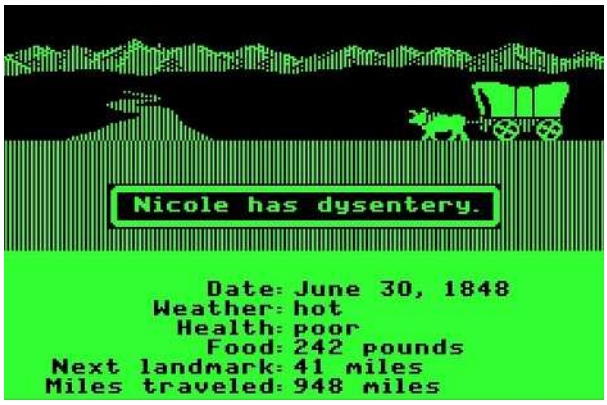

Figura 7 - The Oregon Trail (Minnesota Educational Computing Consortium), um exemplo de jogo baseado em texto.

Com a evolução da tecnologia, os computadores começaram a utilizar monitores, que eram inicialmente monocromáticos, ou seja, conseguiam exibir apenas uma cor. Mesmo assim, artistas começaram a desenvolver métodos para trabalhar com essa limitação, dentre eles o Sketchpad (1963), que permitia a criação de desenhos vetoriais, que são imagens criadas por pontos e formas geométricas simples. Na mesma década, as primeiras ferramentas para desenho do tipo raster, imagens representadas por pontos coloridos, foram introduzidas.

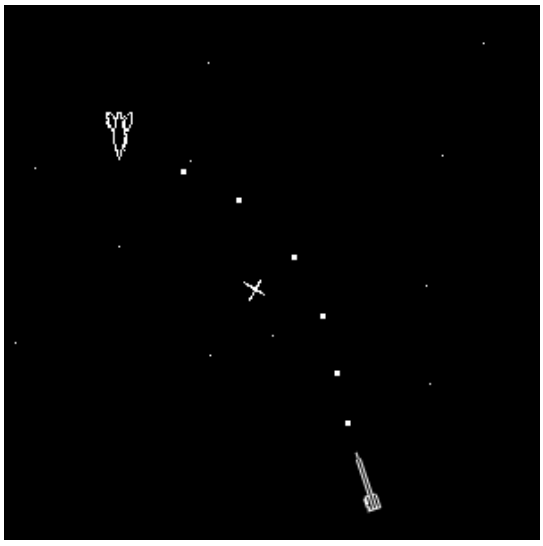

Figura 8 - Spacewar! (Steve Russell) um dos precursores dos jogos digitais, possuía gráficos vetoriais.

As primeiras imagens em raster eram primitivas e quadradas, pois os computadores não tinham capacidade suficiente para projetar muitos pontos na tela. Logo, monitores coloridos foram criados e o computador pessoal foi lançado no mercado, colocando-o nas mãos de milhares de usuários, alguns nos quais eram interessados em criar arte. Com a disseminação do computador pessoal e a introdução dos primeiros consoles caseiros, as pessoas começaram a ver os computadores como uma ferramenta multimídia invés de apenas uma máquina para ser usada nos negócios. Empresas começaram a lançar partes e programas no mercado que antes eram exclusivos para outras empresas e a tecnologia neste campo evolui drasticamente a cada ano.

Figura 9 - Evolução dos gráficos 2D: A, Street Fighter (Capcom) de 1987; B, Street Fighter II: The World Warrior (Capcom) de 1991; C, Street Fighter Alpha: Warrior's Dreams (Capcom) lançado em 1995; D, Street Fighter III: New Generation (Capcom) de 1997.

De acordo com Pardew, a tecnologia para a criação de imagens tridimensionais surgiu em simuladores de uso militar, para replicar a sensação de pilotar um tanque ou jato sem risco e poupando milhares de dólares. Artistas e designers começaram a usar essa nova tecnologia em filmes e jogos, com formas poligonais simples. Nos anos 90, com a queda de preço dos componentes gráficos devido à competição entre fabricantes, os computadores pessoais e consoles começaram a substituir o uso de imagens bidimensionais por modelos tridimensionais, uma tendência que continua até os dias atuais.

Figura 10 - Super Mario 64 (Nintendo), um dos primeiros títulos a proporcionar ao jogador liberdade de movimentação num ambiente tridimensional.

3.2. Imagens 2D

De acordo com Shirley e Marschner (2009), a maioria das imagens utilizadas nos processos de computação gráfica são consideradas imagens do tipo raster. Imagens raster são compostas por milhares de minúsculos retângulos coloridos chamados pixels, dispostos em pontos específicos da área de desenho, preenchendo os espaços com cores diferentes para criar figuras. Matematicamente, podemos representar uma imagem raster como uma matriz na qual os valores dos pixels são armazenados, geralmente como uma cor com três números.

A maioria dos televisores, monitores, câmeras e impressoras, dentre outros dispositivos de entrada e saída gráfica, utilizam o formato raster para criar ou exibir imagens, sendo com tinta ou com uma grade de pontos emissores de luz. Lembrando também que em quase todos os casos a exibição da imagem é uma aproximação, devido à diferença entre o número de pixels existente no equipamento.

3.2.1. Sprite

Sprites são objetos gráficos utilizados em jogos digitais que tem como principal característica a desvinculação ao fundo do cenário, proporcionando liberdade de movimento na tela de desenho, útil para elementos móveis, como os personagens controlados pelo jogador.

O tamanho e a forma das imagens podem variar mesmo enquanto o jogo está em andamento, possibilitando a representação de praticamente qualquer objeto. Essas imagens podem ser estáticas ou animadas.

Figura 11 - Exemplo de sprites de vários tamanhos

Quanto maior o sprite, menor a sua velocidade de renderização, visto que o computador precisa manipular uma quantidade maior de dados. Para otimizar o processamento dos jogos e padronizar o tamanho dos sprites, uma técnica bastante utilizada é a criação de grades com dimensões pré determinadas que serve como guia para desenvolver os desenhos utilizados (Feldman, 2011).

3.2.2. Textura

Para Pardew (2007), texturas são imagens em duas dimensões aplicadas sobre uma superfície tridimensional para adicionar detalhes, desde os mais simples como a coloração de um objeto, até desenhos mais complexos como a casca de uma árvore. No mundo dos jogos digitais, texturas podem ser utilizadas para reduzir detalhes no momento da criação do objeto utilizado, poupando o processamento para tarefas prioritárias.

Cada superfície possui uma série de características visuais, tais como a cor, rigidez, transparência, refletividade e iluminação, que podem ser replicadas com o uso de texturas.

3.3. Modelos 3D

Segundo Vaughan (2011), existem três categorias diferentes de modelos tridimensionais, definidos pela forma de sua construção. A primeira categoria é a dos modelos poligonais, formados por centenas ou milhares de polígonos, formas geométricas criadas pela

união de pontos que delimitam uma superfície. Apesar da existência de polígonos formados por um ou dois pontos, a maioria dos programas utiliza formas com três ou mais vértices.

O segundo tipo de modelo 3D utiliza superfícies NURBS (Non-Uniform Rational B-Splines) definidas por um conjunto de splines, que são curvas criadas pela união mais de dois ou mais pontos em um espaço 3D. Essas curvas são naturalmente arredondadas, sendo possível a criação de formas orgânicas e a representação precisa de circunferências. Os modelos criados com superfícies NURBS podem ser convertidos para modelos poligonais.

Apesar de compartilhar elementos estruturais iguais aos dos modelos poligonais, a subdivisão de superfícies pode ser considerada uma categoria separada, pois aproveita alguns dos atributos do sistema NURBS. Utilizando os vértices de uma malha poligonal como base, o algoritmo de subdivisão gera uma nova grade arredondada, chamada de grade de controle. O número de subdivisões pode ser alterado para proporcionar uma variedade de níveis de detalhamento.

3.4. Conceitos de Animação

Para Feldman (2001), animação é um processo que produz a ilusão do movimento, que consiste em exibir duas ou mais imagens com alterações, chamadas de frames ou células, em rápida sucessão. Pardew (2007) menciona que a maioria das pessoas relaciona esse processo principalmente com os desenhos animados, mas na realidade o conceito é utilizado para todos os tipos de mídia que envolvem imagens que se movem, como filmes e até mesmo jogos digitais.

Figura 12 - Imagens utilizadas para a criação de uma animação.

Nos filmes para cinema, cada segundo de vídeo consiste em vinte e quatro quadros exibidos em uma velocidade imperceptível ao olho humano, enquanto que a televisão trabalha com uma média de trinta quadros. Para os jogos digitais, a média de quadros é determinada

pela potência dos componentes e a complexidade dos processos que são realizados, sempre tentando chegar próximo aos sessenta frames por segundo.

Feldman (2001) afirma que para criar animações de boa qualidade, é preciso isolar os elementos básicos dos movimentos utilizados em frames com o auxílio da observação dos detalhes e dos conhecimentos básicos sobre os conceitos de movimentação.

4. GAME ENGINE

Conforme Gregory (2009), o termo “game engine” foi introduzido na década de 90 pelos jogos de tiro em primeira pessoa como o jogo Doom (id Software, 1993). Esse título foi arquitetado de uma maneira que separava o núcleo do software, como os sistemas de controle e renderização, dos elementos de arte e design, facilitando a sua inserção e modificação no jogo. Logo outros títulos foram criados apenas alterando-se os elementos gráficos e de jogabilidade, mantendo o núcleo do programa intacto. Essa característica deu início a um movimento de criadores profissionais e amadores que criavam novos títulos a partir de jogos existentes, os chamados "mods".

Figura 13 – Mods criados a partir do jogo Doom (id Software), indicado pela letra A; Chex Quest (Digital Café) na imagem B; Ghostbusters (Steve Browning), inspirado pelo filme do estúdio Columbia Pictures na imagem C; Mega Man 8-bit Deathmatch (Cutstuff), baseado na série Mega Man (Capcom) na imagem D.

Percebendo que existia um mercado para essa comunidade, as desenvolvedoras começaram a criar títulos com o foco na criação e modificação de conteúdo. Atualmente, essas mesmas empresas licenciam as suas engines para outros estúdios, gerando uma fonte secundária de receita, visto que é uma opção muito mais econômica para os clientes do que desenvolver um jogo por completo.

A principal característica que separa uma game engine de um jogo digital propriamente dito é a sua arquitetura, planejada para que o desenvolvedor possa criar outros produtos sem alterações drásticas no código do núcleo do programa, sendo otimizada pra um gênero ou plataforma específica.

4.1. Unity 3D

Segundo Felicia (2013), Unity 3D é uma game engine acessível a criadores amadores e profissionais e permite a criação de jogos digitais focando nas mecânicas de jogabilidade ao invés de inúmeras camadas de código. No mercado há vários anos, essa plataforma de desenvolvimento possui uma quantidade significativa de usuários e vários títulos comerciais lançados, devido à facilidade de uso e sua portabilidade para diversos aparelhos e sistemas operacionais.

As ferramentas e técnicas utilizadas na Unity 3D simplificam o processo de criação de jogos, ajudando os desenvolvedores em áreas complexas como inteligência artificial, iluminação e animações. O uso de múltiplas linguagens de programação de alto nível para a criação de scripts, tais como Javascript e C#, é possível. Além das ferramentas nativas da Unity, o usuário pode contar com plugins e extensões criadas por terceiros que adicionam outras funcionalidades ao programa principal.

4.2. Características da Unity 3D

Uma das principais características da engine Unity 3D é utilizar conceitos que podem ser aplicados em qualquer tipo de jogo digital, proporcionando uma seqüência lógica de passos para o desenvolvimento. Os arquivos usados para criar o jogo, sendo imagens, modelos 3D, dentre outros, são chamados de Assets e ficam em uma pasta de mesmo nome dentro do projeto. As telas, fases ou menus são referidos como Scenes. Todo elemento usado em uma Scene é considerado um GameObject, sendo composto por vários Components, e estes por sua vez, possuem atributos que podem ser modificados para uma variedade de efeitos. Um dos Components necessários para a existência de um GameObject é o Transform, uma coordenada global que posiciona o objeto na cena de jogo (GOLDSTONE, 2009).

A interface do editor de projetos da engine Unity 3D é customizável, ou seja, o usuário pode alterar a posição, tamanho ou até adicionar e desativar os componentes da tela.

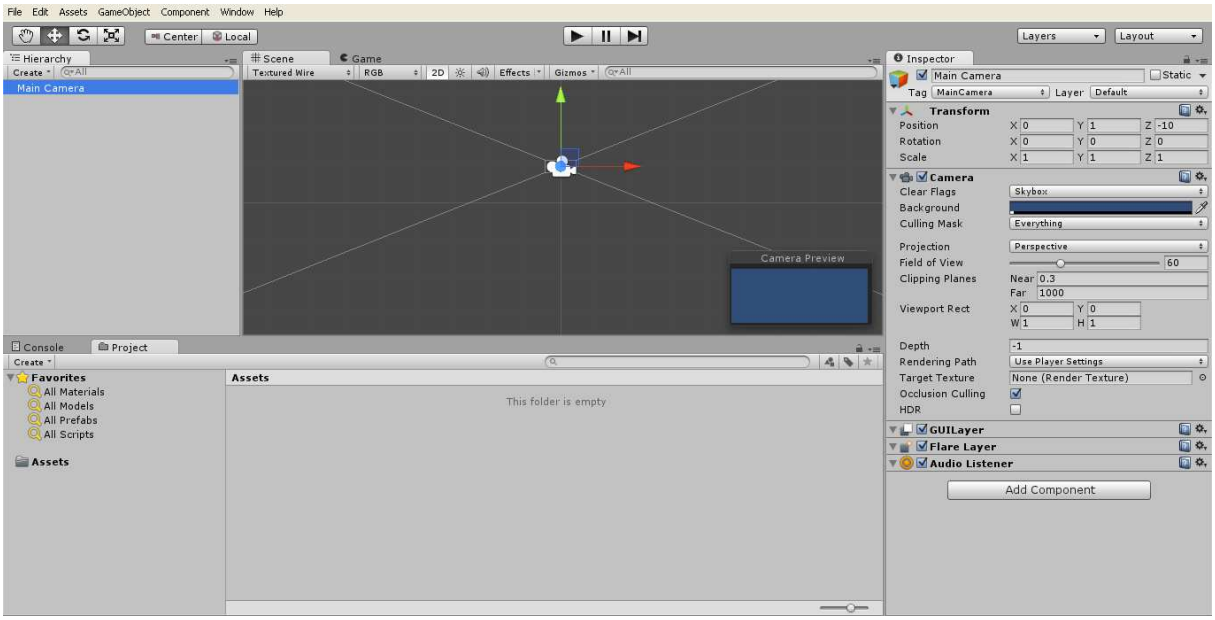

Figura 14 – Interface do editor de jogos da engine Unity 3D (Unity)

A chamada Scene View é a parte da interface em que o projeto de jogo é construído e permite a visualização dos objetos em cena em posições diferentes. Sempre acompanhada e ligada pela Scene View está a janela de Hierarchy, que lista os objetos presentes em cena. Selecionando um objeto na hierarquia, o mesmo é ativado na cena, o mesmo ocorre na situação oposta.

O Inspector é a ferramenta que apresenta informações sobre o GameObject selecionado e seus componentes e atributos. Todos os atributos contidos nessa janela podem ser diretamente modificados, até mesmo enquanto o jogo está em funcionamento (UNITY, 2013a)

Para organizar os assets usados no projeto existe o Project Manager. A janela a esquerda mostra a hierarquia das pastas e arquivos existentes enquanto o painel os apresenta de uma maneira mais detalhada. O usuário também pode contar com a função de marcar favoritos e pesquisar por assets segundo nome ou extensão de arquivo.

A janela Game entra em ação quando o botão Play é pressionado e coloca o projeto em execução. Atributos podem ser ajustados enquanto o jogo está em andamento, porém todas as alterações são descartadas quando o botão é pressionado novamente (GOLDSTONE, 2009).

5. DESENVOLVIMENTO DE JOGOS DIGITAIS 2D NO AMBIENTE 3D DA UNITY ENGINE

O termo jogos digitais em 2D possui dois significados diferentes, jogabilidade em duas dimensões ou gráficos em duas dimensões. A primeira situação se aplica aos jogos que possuem um ambiente ou câmera que utiliza apenas dois eixos de movimentação, enquanto que a segunda se refere a jogos nos quais os elementos gráficos são constituídos na maior parte por imagens, compreendendo desde sprites a cenários. Ambos os casos serão tratados nos capítulos a seguir.

A engine Unity 3D é especializada em criação de jogos com ambiente e gráficos tridimensionais, porém suas características adaptáveis permitem a criação de projetos que ultrapassam essa limitação e são acessíveis a estúdios profissionais e amadores. Alguns dos títulos mais rentáveis criados com a Unity possuem a combinação de jogabilidade e gráficos em duas dimensões, como os demonstrados na imagem abaixo.

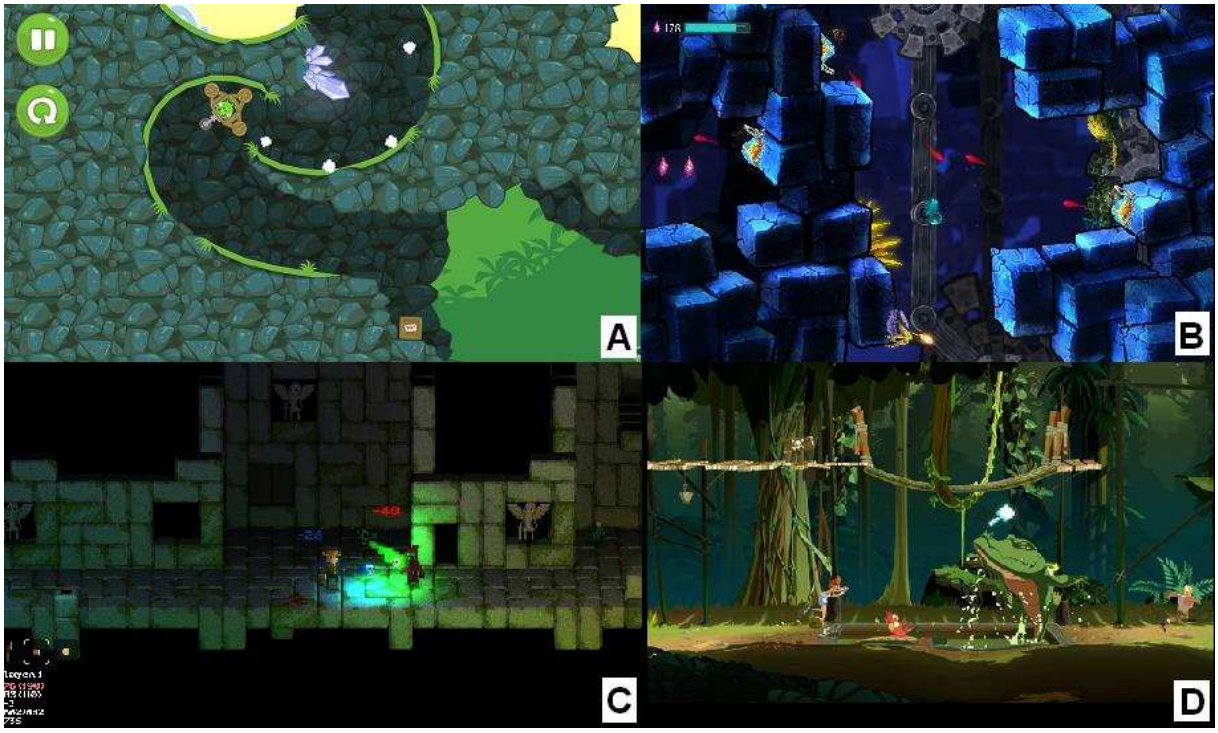

Figura 15 – Exemplos de jogos com gráficos em duas dimensões desenvolvidos em Unity: A (Bad Piggies, Rovio), B (Beatbuddy: Tale of the Guardians, Threaks), C (Legend of Dungeon, RobotLovesKitty) e D (Harold, Moon Spider Studio)

5.1. Motivação

São muitas as vantagens que a engine Unity 3D proporciona sobre suas concorrentes, principalmente em questão de flexibilidade e portabilidade, tornando-se uma opção para jogos digitais em duas dimensões mesmo quando engines especializadas nesse nicho existem. A Unity é compatível com a grande maioria dos formatos utilizados pelos programas criadores de conteúdo visual e áudio, importando modelos, animações, texturas, efeitos sonoros e músicas. Por padrão, a programação é feita por meio de scripts no editor Mono Developer, mas existe a opção de substituição e comporta o uso de três linguagens de programação: C#, Javascript e Boo, sendo as três linguagens de alto nível orientadas a objeto com uma base de usuários e documentação significativa, o que coloca a Unity numa posição mais favorável do que vários dos seus competidores que utilizam linguagens próprias, e removendo um nível na curva de aprendizagem.

Jogos digitais criados na Unity podem ser exportados para quase todos os aparelhos atualmente no mercado, como computadores pessoais, consoles, smartphones e até mesmo incorporados em websites para uso em navegadores de internet com o uso de um plugin proprietário. Grandes empresas do ramo como a Nintendo, que formou um acordo disponibilizando licenças profissionais gratuitas para desenvolvedores cadastrados no seu programa de desenvolvimento para o console Wii U (GILBERT, 2012), e a Microsoft, com uma parceira envolvendo o desenvolvimento de ferramentas para seu novo console, o X-Box one e o suporte ao ecossistema do sistema operacional Windows 8, anunciada na conferência Build Day do ano corrente (UNITY, 2013b).

Outra vantagem no uso da Unity são as ferramentas para teste do editor, que permitem ao usuário fazer modificações enquanto o seu jogo está em andamento, poupando tempo ao realizar alterações simples, porém é necessário frisar que todas essas alterações são descartadas quando o processo de teste do programa é encerrado. O Remote Play permite que o desenvolvedor conecte um dispositivo móvel à estação de trabalho para testar um jogo em andamento sem que necessite exportar um arquivo.

5.2. Suporte oficial

O suporte nativo a jogos que utilizam gráficos em duas dimensões para a engine foi anunciado no evento UNITE 2013 e está atualmente em fase de testes, com a previsão de

lançamento da versão estável para o final do ano de 2013. A versão, denominada de Unity 4.3, dentre outros aprimoramentos gerais, adiciona ferramentas que permitem uma maior facilidade de manipular as imagens contendo sprites, a integração com o sistema de animação nativo possibilitando a criação de estados para controlar as transições entre essas animações e um conjunto de componentes de física especializados para uso com objetos em duas dimensões (GOLDSTONE, 2013).

5.3. Add-ons

Uma das vantagens da engine Unity é a possibilidade de usar expansões e scripts de terceiros para complementar as funções nativas do editor, com uma loja incorporada chamada Asset Store. Na Asset Store podemos encontrar add-ons gratuitos ou pagos para diversas finalidades, incluindo alguns para melhorar a compatibilidade da engine com jogos em duas dimensões, como o 2D Toolkit (Unikron Software), SpriteManager (AB Software) e Orthello (Wyrmtale Games). Esses frameworks adicionam recursos que simplificam a forma de se lidar com o gerenciamento de imagens quando se desenvolve um jogo em duas dimensões e dois deles, 2D Toolkit e Orthello, contém seus próprios scripts de controle de câmera. Orthello tem a característica principal de usar tiles, ou seja, blocos pré-construídos para a criação de fases, uma vantagem se todos os elementos de um nível forem padronizados para o mesmo tamanho, mas dificulta a criação de objetos com tamanhos variáveis.

5.4. Definindo planos no ambiente 3D

Como a Unity 3D é uma engine focada para o desenvolvimento de jogos digitais com ambientes tridimensionais, o seu editor trabalha com três eixos de movimentação, chamados de X, Y e Z, que podem ser observados na parte superior da guia Scene. Cada GameObject em cena possui um trio de coordenadas de posição e um trio de valores de rotação que correspondem aos três eixos e determinam sua situação no espaço, representados por variáveis do tipo Vector3. Para criar a ilusão de um jogo em duas dimensões é preciso utilizar a câmera principal da cena de uma forma que ela abstraia um dos eixos de movimentação.

As câmeras são os elementos que capturam as informações do ambiente e as exibem ao jogador de acordo com sua configuração. Para a criação de títulos com jogabilidade em duas dimensões, a opção fundamental é chamada de Projection (projeção) que controla a noção de profundidade da câmera, por padrão, a opção ativa é chamada de Perspective e mostra a cena em perspectiva, ou seja, os elementos que estão mais próximos a câmera são exibidos em tamanho maior, enquanto que os objetos ao fundo ficam menores; o segundo valor é chamado de Orthographic e mostra todos os objetos com o mesmo tamanho independente de sua posição em cena (UNITY, 2013c).

Figura 16 – Demonstração das diferenças entre a câmera Perspective (A) e a câmera Orthographic (B) (UNITY, 2013)

Com a câmera no modo Orthographic, é possível posicioná-la de uma maneira que oculte um dos três eixos, utilizando os valores como um controlador de camadas, ou seja, os

objetos que possuem os números mais próximos à origem da projeção no eixo abstraído são desenhados na frente dos outros elementos (SUTPHIN, 2012).

Figura 17 – Câmera no modo Orthographic vista pelo inspetor do editor da engine Unity 3D

A imagem acima mostra a câmera no modo Orthographic, com suas configurações particulares, com destaque as opções Size e Clipping Planes. A opção Size controla o tamanho da projeção da câmera na cena e é inversamente proporcional ao tamanho dos objetos que ela exibe. Para alcançar fidelidade quanto as dimensões dos sprites, um cálculo deve ser feito determinando o valor do campo Size em relação à resolução usada. Dunk (2013) demonstra uma fórmula simples para essa finalidade:

Tamanho da câmera = x / (((x / y) * 2) * s)

Onde:
x = Largura da tela
y = Altura da tela
s = Dimensão dos sprites

Figura 18 – Fórmula para determinar o valor da variável Size (DUNK, 2013)

O atributo Clipping Planes possui dois valores que respectivamente são o valor mínimo e o valor máximo em que a câmera pode capturar valores, mostrando apenas os objetos que estiverem entre as duas marcas, o que pode ser útil para criar recursos visuais ou até mesmo permitir ações com o fim de melhorar o processamento do jogo.

5.5. Modelos na Unity 3D

Para utilizar modelos tridimensionais na Unity, temos duas opções diferentes. A primeira é importar um modelo pronto criado com um programa especializado em modelagem e a segunda é usar um dos seis modelos primitivos criados pelo editor da engine que correspondem a formas geométricas simples, o cube (cubo), capsule (cápsula), cylinder (cilindro), sphere (esfera), plane (plano) e quad, que é um plano simples medindo apenas uma unidade (UNITY, 2013d). Desses, os dois últimos, o plano e o quad são os mais interessantes quando se desenvolve um projeto 2D, pois eles possuem apenas um lado visível no qual as imagens dos sprites podem ser sobrepostas. A fim de otimizar o jogo, recomenda-se que estes sejam criados em um programa externo e importados para o editor, pois o plano criado nativamente pelo Unity é formado por muitos triângulos, gastando recursos de processamento, enquanto que o plano importado pode ser criado com apenas dois triângulos (SUTHPIN, 2012).

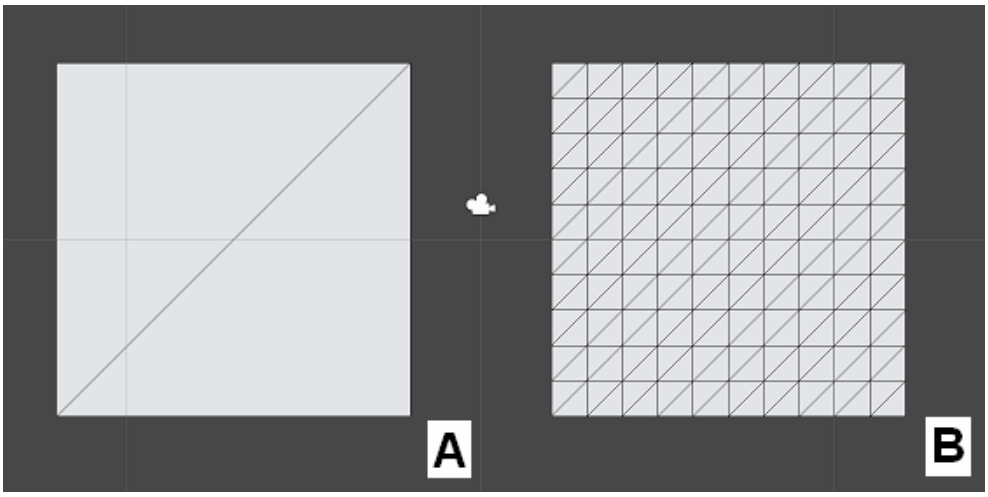

Figura 19 – Comparação dos modelos: O plano A foi criado utilizando o programa de modelagem Blender (Blender Foundation), enquanto o plano B foi gerado pelo editor da engine Unity.

A interação dos modelos com o ambiente e outros objetos se dá pelo uso de Colliders, caixas de colisão de formas geométricas simples com tamanhos e posições variáveis, que podem ser aglomeradas em um único GameObject para a criação de formas mais complexas (UNITY, 2013). O Collider inicial dos objetos do tipo plano se chama Mesh Collider e tem a mesma forma que o modelo, impossibilitando interações por todos os lados que não sejam sua face principal. Para adicionar volume ao objeto no qual os sprites estão anexados e permitir a interação com o ambiente por todos os lados, recomenda-se a substituição do Mesh Collider pelo Box Collider, a caixa de colisão em formato de cubo (WITTAYABUNDIT, 2011).

5.6. Preparação e Utilização de Imagens 2D na Unity 3D

O primeiro passo para se trabalhar com imagens num projeto de jogos digitais é determinar o formato de arquivo utilizado, pois cada um deles possui características diferentes. Os formatos mais utilizados para a criação de jogos digitais no momento são BMP, PNG, GIF e JPEG.

O formato do tipo BMP (Bitmap) é nativo do sistema operacional Windows (Microsoft) e salva exatamente as informações disponíveis na tela com milhões de cores. Sua maior vantagem é a compatibilidade com todas as versões do Windows e sua principal desvantagem se dá no tamanho dos arquivos criados. GIF, ou Graphics Interchange Format, foi criado para ser utilizado em diversos sistemas operacionais na década de 80, tendo como suas maiores vantagens a capacidade de incluir áreas com transparência na imagem e a utilização de frames sobrepostos para criar animações simples. JPEG não é um formato de imagem propriamente dito, mas sim um mecanismo de compressão de imagens que permite a criação de arquivos leves com muitas cores, com o efeito colateral de perder qualidade e definição, não recomendado para imagens com linhas e cores sólidas como sprites e elementos do menu de um jogo. O formato PNG foi criado para substituir o GIF quando este se tornou um formato proprietário compartilhando muitas das suas características enquanto aprimorando algumas delas como o suporte a níveis diferentes de transparência e adicionando outras como correção de cor embutida. Cabe ao desenvolvedor escolher um ou mais formatos que se encaixem no seu projeto (FELDMAN, 2001).

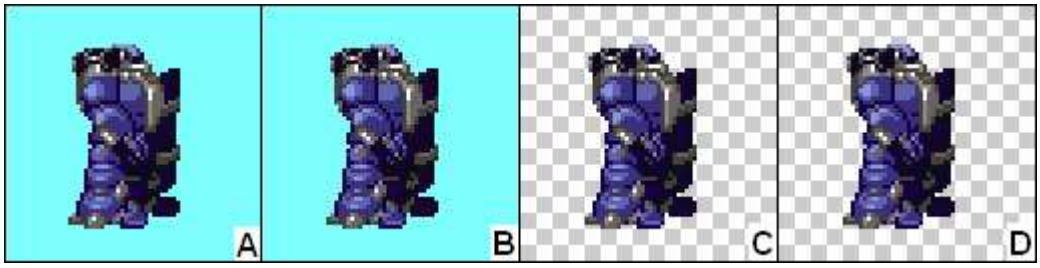

Figura 20 – Comparação de uma mesma imagem salva em diferentes formatos com transparência se suportada: A(BMP), B(JPG), C(PNG), D(GIF)

Nome	Tamanho	Tipo	Dimensões
robo.bmp	49 KB	Imagem de bitmap	128 x 128
robo.JPG	3 KB	Imagem no formato JPEG	128 x 128
robo.PNG	3 KB	Imagem PNG	128 x 128
robo.gif	2 KB	Imagem no formato GIF	128 x 128

Figura 21 – Características das imagens utilizadas na comparação.

Na versão estável corrente, o editor da Unity trata qualquer formato de imagem como uma textura, disponibilizando diversos modos de visualização e opções avançadas. Sutphin (2012) recomenda que se use o modo avançado com o filtro de renderização Point, respeitando os pixels individuais ao invés de tentar corrigir as cores, para manter a fidelidade visual quando trabalhando com sprites. Goldstone (2009) afirma que o tamanho das imagens utilizadas como texturas devem ser preferencialmente um número que seja divisível por dois.

A fim de otimizar o projeto, muitos desenvolvedores utilizam um recurso chamado de Atlas de Texturas, uma compilação de todas as texturas utilizadas por um mesmo tipo de objeto em uma única imagem, considerando que todas as imagens, independente do tamanho, precisam do mesmo número de chamadas de desenho. Dentro do Atlas, cada sprite individual possui coordenadas específicas, permitindo sua localização pelo programa (BYL, 2011).

Figura 22 – Exemplo de um atlas de sprites.

A versão 4.3 da Unity introduzirá os objetos do tipo Sprite, facilitando a manipulação de imagens em lote, utilizando um único arquivo com todas as imagens do elemento e o passando pelo Editor de Sprites, que pode cortar as imagens individuais e inserir as caixas de colisão utilizando um dos tipos de corte automático ou manual. O uso dessas imagens torna desnecessário o uso de modelos para intermediar sua exibição na cena.

5.6.1. Materiais

Materiais são os componentes que ligam as imagens aos modelos na Unity, contendo atributos para alterar o tamanho, posição e orientação das mesmas. Os materiais são influenciados pelos Shaders, que possuem informações de cor, luminosidade, transparência e reflexão, ditando como o objeto deve aparecer na cena de jogo. O editor da Unity possui dezenas de shaders diferentes, mas também podem ser criados pelos usuários com a linguagem ShaderLab ou obtidos pela Asset Store (UNITY, 2013e).

No desenvolvimento com gráficos 2D na Unity, os atributos Tiling e Offset do componente Material devem ser usados para localizar os sprites requeridos por uma animação quando se usa um atlas de texturas. O shader utilizado deve suportar transparência para separar o fundo do objeto nos sprites, como os nativos da categoria Transparency.

5.7. Animação em Unity 3D

Na Unity, temos três maneiras principais de trabalhar com animações usando imagens que não necessitam do uso de componentes externos. A primeira opção utiliza o método de legado do editor e envolve adicionar um componente denominado Animation ao modelo, anexando Animation Clips para cada animação diferente. Estes Animation Clips podem manipular as variáveis de outros componentes do modelo, incluindo materiais e podem ser acionados por eventos determinados pelo desenvolvedor. (UNITY, 2013f).

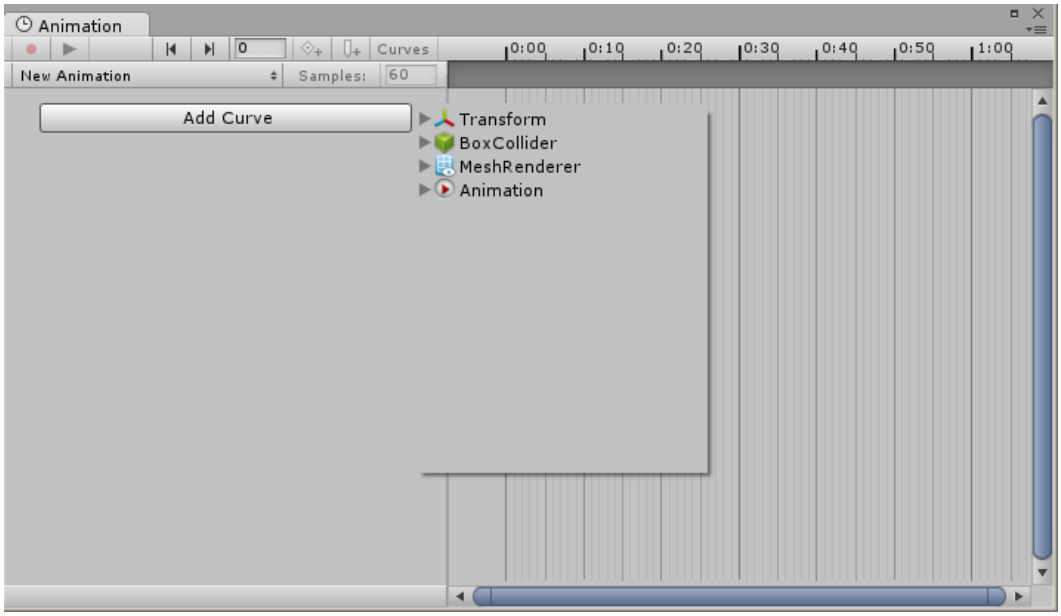

Figura 23 – Aba Animation do editor da Unity 3D

O sistema de animação Mecanim, introduzido nas versões mais recentes da Unity é a segunda opção. Mecanim é uma evolução do método acima que mantém os Animation Clips e introduz um componente chamado Animation Controller, uma máquina de estados que controla o fluxo das mudanças de animação, ou seja, o desenvolvedor pode criar um mapa contendo todos os clipes de animação usados por certo objeto e construir ligações entre eles com condições de transição customizáveis como tempo decorrido ou o estado de uma variável de controle.

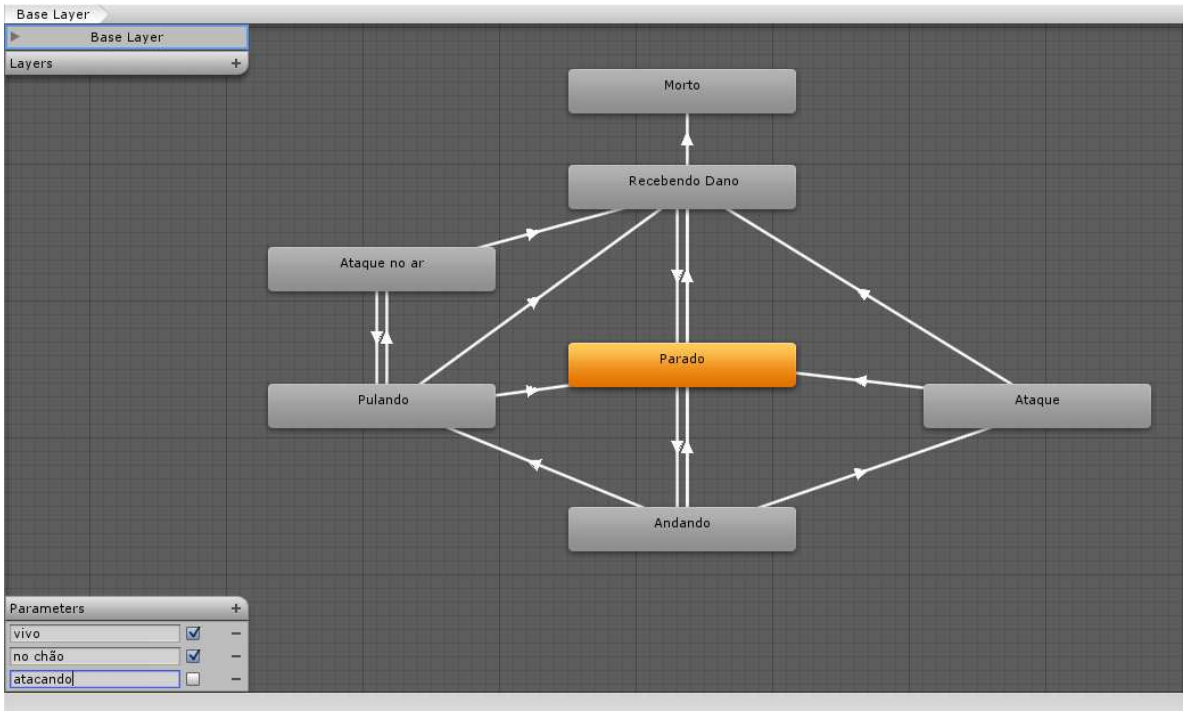

Figura 24 – Animation Controller com um fluxo simples de animações.

A terceira opção envolve utilizar scripts para manipular o componente Material do objeto exibindo imagens seqüenciais por um curto período de tempo como uma animação tradicional em duas dimensões conforme o exemplo abaixo.

```
using UnityEngine;
using System.Collections;

public class CharacterAnimation : MonoBehaviour {

    public SingleAnimation[] animations;

    IEnumerator Play(int anim){
        if(animations[anim]!=null){
                for(int i=0;i<animations[anim].frames.Length;i++){
                    renderer.material.mainTexture = animations[anim].sprites[i];
                    yield return new WaitForSeconds (animations[anim].frames[i]);

                }
        }
    }

    [System.Serializable]
    public class SingleAnimation{
        public Texture2D[] sprites;
        public float[] frames;
    }
}
```

Figura 25 – Script CharacterAnimation.cs

O script CharacterAnimation.cs cria um vetor de um novo tipo de variável chamado SingleAnimation utilizando o recurso [System.Serializable], que por sua vez contém dois vetores distintos, um que suporta a entrada de texturas nomeado de sprites e um vetor de variáveis numéricas do tipo float para controle de tempo chamado frames. O método Play ao ser iniciado procura a SingleAnimation correspondente no vetor e checa sua condição de existência, terminando o processo se ela não estiver presente. O próximo e mais importante passo é a criação de uma repetição para alternar entre as texturas do vetor sprite pausando o processo de acordo com os valores determinados no campo frames, se encerrando assim que todas as texturas do grupo passar pelo menos uma vez pelo ciclo.

6. ESTUDO DE CASO: DESENVOLVENDO UM PROJETO 2D EM UNITY 3D

Para demonstrar a viabilidade da criação de um jogo com jogabilidade e elementos gráficos em duas dimensões no ambiente em três dimensões da engine Unity 3D será utilizado o projeto Robo Force, desenvolvido como projeto de conclusão de curso no segundo semestre de 2012. O uso da Unity ao invés de uma engine própria para jogos digitais bidimensionais foi decidido devido à familiaridade dos componentes do grupo com a ferramenta.

6.1. Definições do Projeto

Robo Force é um jogo no estilo plataforma no qual o jogador deve atravessar níveis com vários inimigos espalhados ao decorrer do terreno acidentado, chegando ao final e enfrentando o vilão principal, parcialmente inspirado por títulos clássicos do estilo. O personagem principal é um policial pilotando a armadura robótica denominada Peacemaker que investiga o roubo de peças e outros artefatos por uma gangue que assola a cidade. O projeto utiliza um estilo de arte com influências futurísticas e industriais, evidenciado pelos personagens e cenários com características mecânicas.

Do projeto inicial de quatro níveis, um foi desenvolvido segmentado em dois momentos introdução, transição entre as partes e fim animados. A primeira parte consiste em percorrer um cais com veículos e plataformas metálicas enquanto que o segundo momento dá espaço ao confronto com o vilão principal em um hangar. Nos capítulos a seguir serão demonstrados os recursos utilizados para a criação e utilização dos elementos gráficos e preparação do ambiente bidimensional do jogo.

6.2. Aplicando os Conceitos do Jogo 2D no Ambiente 3D

No primeiro passo do projeto, foram criadas as imagens individuais utilizando os programas Paint (Microsoft) e a versão gratuita do Graphics Gale (Human Balance) para confecção dos sprites, com os arquivos em formato PNG devido ao suporte a transparência. As dimensões

das imagens foram de 1024x1024 pixels no caso de cenário, 64x64 pixels para os personagens móveis e tamanhos variáveis para os efeitos especiais como explosões e projéteis. Os arquivos foram organizados por pastas e importados para o editor da Unity3D onde foram submetidos ao processo de alterar a classificação da textura para Advanced e o filtro de renderização para Point. Cada tipo de objeto possui um Material no qual um Shader customizado foi usado para manter a fidelidade gráfica dos sprites.

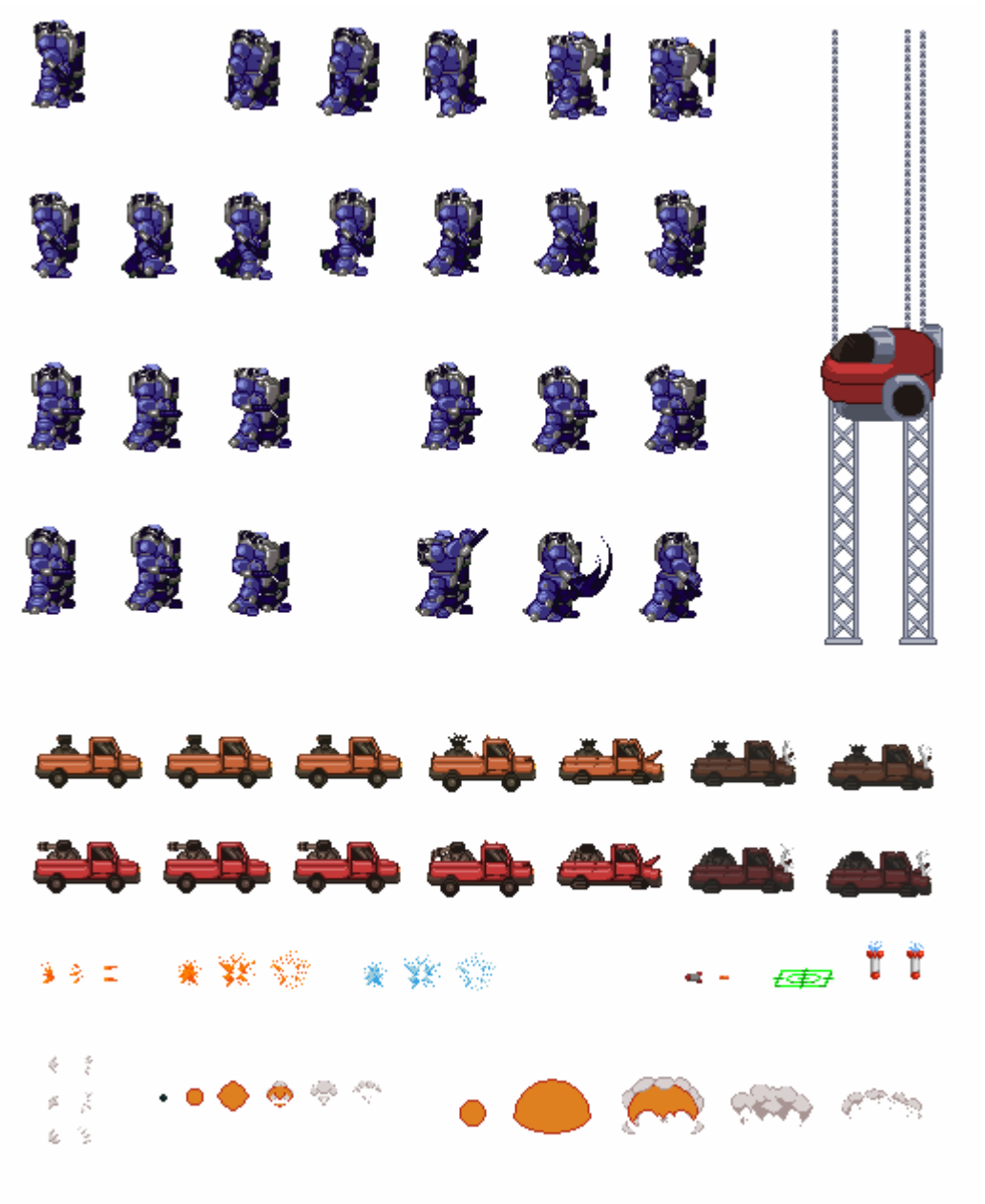

Figura 26 – Sprites utilizados no primeiro nível do projeto Robo Force

Para criar a ilusão de um ambiente bidimensional foi utilizada a câmera principal no modo Orthographic apontada para a direção do eixo Z, com a intenção de ocultá-lo e ao

mesmo tempo usa-lo como um gerenciador de ordem de desenho na tela, colocando o personagem principal em evidência, seguido pelos outros objetos móveis como inimigos e projéteis, as partes interativas do cenário e duas camadas diferentes de fundo.

O método utilizado para exibir os sprites e realizar a interação deles com o ambiente foi projetá-los em modelos, especificamente o primitivo Plane nativo do editor, com uma caixa de colisão do tipo Box Collider para melhores resultados e possibilidade de interação com objetos com valores diferentes no eixo Z. Utilizando esse recurso, os elementos de colisão, física e interação com o ambiente são realizados como se o objeto fosse um modelo convencional em três dimensões.

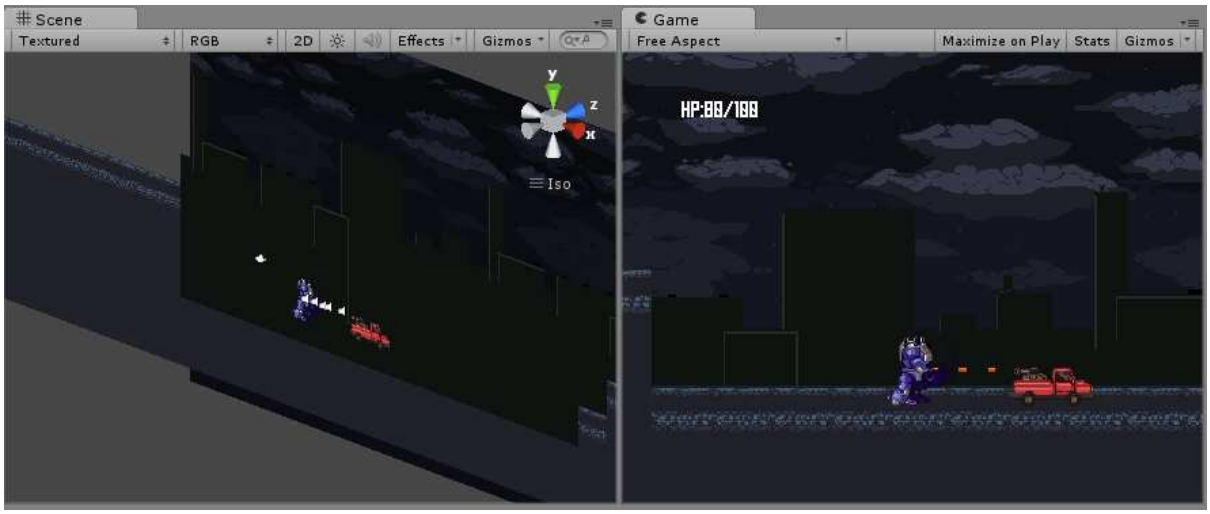

Figura 27 – Demonstração da montagem das camadas na aba Scene com o resultado dentro do jogo na aba Game.

6.3. Codificação

Para criar os elementos de estética do jogo foram utilizados um shader customizado e um sistema de animação por scripts em C#.

A imagem abaixo demonstra um script desenvolvido por Suthpin (2012) na linguagem ShaderLab com a intenção de compor um Shader que possua apenas o necessário para manter a fidelidade gráfica nos objetos em que os sprites são projetados, consistindo em um shader principal com uma base na cor branca e transparência ligada e um shader secundário no qual são desabilitadas a interação com a iluminação do ambiente e a renderização em profundidade.

```
Shader "Sprite" {
Properties {
    _MainTex ("Base (RGB) Trans (A)", 2D) = "white" {}
}

SubShader {
    Tags {"Queue"="Transparent" "IgnoreProjector"="True" "RenderType"="Transparent"}
//    LOD 100

    ZWrite Off
    Blend SrcAlpha OneMinusSrcAlpha
    Lighting Off

    Pass {
        SetTexture [_MainTex] { combine texture }
    }
}
}
```

Figura 28 – Script contido no arquivo Sprite.shader (SUTHPIN, 2012)

A figura 29 apresenta um script de gerenciamento de imagens a fim de criar animações que foi desenvolvido com a aplicação dos conceitos apresentados no capítulo anterior, criando dois métodos para uso em situações diferentes. O método PlayOnce é utilizado para animações em que o ciclo de imagens se encerre assim que todas passarem pelo menos uma vez pelo componente Material do objeto. Play é um método focado em animações recorrentes, ou seja, estados em que a animação continue por um tempo indeterminado até que seja cancelada por outra ação. Os outros dois métodos, StopAnim e ChangeTex são utilizados para o controle de animação, o primeiro com a função de alterar a variável de controle que indica se existe uma animação tocando e o segundo pode enviar apenas um frame para estados que necessitem apenas de uma imagem.

```
using UnityEngine;
using System.Collections;

public class CharacterAnimation : MonoBehaviour {

    public SingleAnimation[] animations;
    protected bool stopped = true;

    IEnumerator PlayOnce(int anim){
        StopCoroutine("Play");
        SendMessage("AnimInProgress",true);
        if(animations[anim]!=null){
            StopAnim(false);
            for(int i=0;i<animations[anim].sprites.Length;i++){
                renderer.material.mainTexture = animations[anim].sprites[i];
                yield return new WaitForSeconds (animations[anim].frames[i]);
            }
            StopAnim(true);
            SendMessage("AnimInProgress",false);
        }
        else{
            Debug.Log ("Animação não existente");
        }
    }

    IEnumerator Play(int anim){
        if(animations[anim]!=null){
                while(stopped==true){
                    for(int i=0;i<animations[anim].frames.Length;i++){
                        renderer.material.mainTexture = animations[anim].sprites[i];
                        yield return new WaitForSeconds (animations[anim].frames[i]);
                    }
                }
        }
        else{
            Debug.Log ("Animação não existente");
        }
    }

    void StopAnim(bool s){
        stopped = s;
    }

    void ChangeTex(int mat, int frame){
        StopAnim(false);
        renderer.material.mainTexture = animations[mat].sprites[frame];
    }

    [System.Serializable]
    public class SingleAnimation{
        public Texture2D[] sprites;
        public float[] frames;
    }
}
```

Figura 29 – Script CharacterAnimation.cs usado no projeto.

Por fim, para se fazer uso do sistema de animações por script é preciso que os métodos sejam invocados por outro script de controle no mesmo objeto que envie os parâmetros

necessários. A figura 30 demonstra um trecho do script PlayerController.cs no qual a ação de mover o personagem com o teclado enquanto não realiza outras ações ao mesmo tempo, como atacar ou pular, envia uma mensagem com o parâmetro do número correspondente a animação de andar do personagem principal e seu efeito dentro do jogo, conforme visto na imagem 31.

```
if(!isFiring){
    movement = -(Input.GetAxis("Horizontal") * moveSpeed);
    movement *= Time.deltaTime;
    transform.Translate(movement,0.0f, 0.0f);
}

if(movement!=0 && canJump && !isWalking && !animInProgress){
    SendMessage("Play", 4);
    isWalking = true;
}
```

Figura 30 – Trecho do script PlayerController.cs

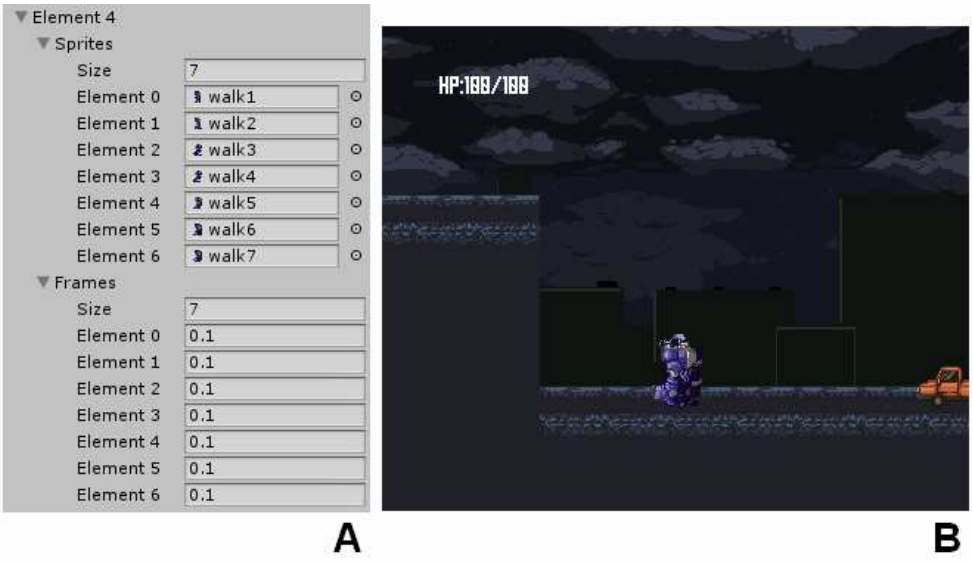

Figura 31 – Demonstração da animação no script Character Animation (imagem A) e seu efeito em jogo (imagem B).

7. CONSIDERAÇÕES FINAIS

A partir da pesquisa realizada é possível identificar os elementos gráficos e espaciais que compõem os jogos digitais em duas e três dimensões, sendo possível a aplicação desses conceitos para a criação e desenvolvimento de títulos com jogabilidade e estética 2D em ambientes 3D, neste caso, o motor de jogo Unity3D que se mostrou uma ferramenta muito flexível compensando o tempo gasto para a montagem das técnicas de compatibilidade, tornando-a mais uma opção para os usuários que desejarem criar jogos em duas dimensões em um prazo curto ou até mesmo como ferramenta de prototipagem sem precisar aprender uma linguagem própria de programação.

É necessário ressaltar que este trabalho não é um tutorial, mas oferece técnicas relacionadas com a manipulação de imagens para utilização em animações e sua utilização em jogos digitais, conceitos de localização espacial se tratando de jogos em duas e três dimensões e opções no uso das ferramentas incluídas no pacote padrão da Unity3D, alguns deles escolhidos para demonstração no caso prático do jogo Robo Force.

Como trabalhos futuros, evidenciam-se a necessidade de pesquisas que envolvam os aspectos técnicos de criação de conteúdo para jogos em duas dimensões, a aplicação dos conceitos utilizados em outros motores de jogo que possuam flexibilidade suficiente para tal ou com as novas ferramentas que a Unity3D introduza eventualmente.

8. REFERÊNCIAS BIBLIOGRÁFICAS

BETHKE, Erik. **Game Development and Production.** 1a. Ed. Plano. Wordware Publishing. 2003.

BYL, Penny de. **Holistic Game Development with Unity**. 1a. Ed. Oxford. Focal Press. 2011.

CAPELAS, Bruno. Sucesso de GTA V empolga indústria. 2013. Disponível em:<http://blogs.estadao.com.br/link/sucesso-de-gta-v-empolga-industria/> Acesso em: 5 Outubro 2013.

DUNK, Reggie. **Pixel Perfect Calculator for Orthographic Câmera – Unity 3D**. 2013. Disponível em: <http://indiehoodgames.wordpress.com/2013/07/27/pixel-perfect-calculator-for-orthographic-camera-unity3d/ >. Acesso em 14 Outubro 2013.

DUNN, Fletcher; PARBERRY, Ian. **3D Math Primer for Graphics and Game Development.** 2a. Ed. Boca Raton: CRC Press, 2011.

EGENFELDT-NIELSEN, Simon; HEIDE, Jonas; TOSCA, Susana Pajares. **Understanding Video Games: The Essential Introduction.** 2a. Ed. Nova Iorque: Routledge, 2013.

ESA. **2013 Sales, Demographic and Usage Data - Essential Facts About the Computer and Video Game Industry.** 2013. Disponível em: <http://www.theesa.com/facts/pdfs/ESA_EF_2013.pdf> Acesso em: 3 Outubro 2013.

FELDMAN, Ari. **Designing Arcade Computer Game Graphics**. 1a. Ed. Plano. Wordware Publishing. 2001.

FELICIA, Patrick. **Getting Started with Unity**. 1a Ed. Birmingham. Packt Publishing, 2013.

GANTZLER, Todd. **Game Development Essentials: Video Game Art**. 1a Ed. Clifton Park: Thomson Delmar Learning, 2005.

GILBERT, Ben. **Unity and Nintendo partner to bring Unity Engine, and its 1.2 million devs, to Wii U**. 2012. Disponível em <http://www.engadget.com/2012/09/19/unity-nintendo-partnership/> Acesso em 16 Outubro 2013

GREGORY, Jason. **Game Engine Architecture**. 1a. Ed. Boca Raton: A.K. Peters, 2009.

GOLDSTONE, Will. **Unity Game Development Essentials**. 1a. Ed. B Birmingham. Packt Publishing, 2009.

GOLDSTONE, Will. **Unity Native 2D Tools.** 2013. Disponível em: <http://blogs.unity3d.com/2013/08/28/unity-native-2d-tools/> Acesso em 16 Outubro 2013.

HUIZINGA, Johan. **Homo Ludens**. 4a. Ed. São Paulo, Perspectiva, 2000.

IBOPE, **Pesquisa GamesPop**. 2012. Disponível em: <http://www.ibope.com/pt-br/conhecimento/Infograficos/Paginas/Games-pop.aspx> Acesso em: 29 Setembro 2013.

PARDEU, Les. **Game Character Animation All in One**. 1a. Ed. Boston: Thomson Course Technology, 2007.

RABELO, Paulo Sérgio Brunner. **Geometria Descritiva Básica**. 2005. Disponível em: <http://magnum.ime.uerj.br/ensinoepesquisa/LIVROS%20DE%20GEOMETRIA/GDBASICA.pdf> Acesso em: 2 Outubro 2013.

SALEN, Katie; ZIMMERMAN, Eric. **Rules of Play – Game Design Fundamentals**. 1a. Ed. Cambridge: The MIT Press, 2004.

SHIRLEY, Peter; MARSCHNER, Steve. **Fundamentals of Computer Graphics.** 3a. Ed. Boca Raton: CRC Press, 2009.

SUTPHIN, Josh. **Making 2D Games with Unity 3D**. 2012. Disponível em: <http://www.third-helix.com/2012/02/making-2d-games-with-unity/>. Acesso em: 14 Outubro 2013

TREVISAN, Leonardo. **Entretenimento e Games**. 2012. Disponível em: <http://leonardotrevisan.com.br/blog/entretenimento-e-games/>. Acesso em: 5 Outubro 2013

UNITY. **Unity – Learning the Interface**. 2013a. Disponível em: <http://docs.unity3d.com/Documentation/Manual/LearningtheInterface.html> Acesso em: 30 Setembro 2013.

UNITY. **Unity – Unity Announces Strategic Collaboration With Microsoft**. 2013b. Disponível em: <http://unity3d.com/company/public-relations/news/unity-announces-strategic-collaboration-microsoft> Acesso em: 14 Outubro 2013

UNITY. **Unity – Cameras**. 2013c. Disponível em: < http://docs.unity3d.com/Documentation/Manual/Cameras.html> Acesso em: 14 Outubro 2013.

UNITY. **Unity – Primitive Objects.** 2013d. Disponível em: < http://docs.unity3d.com/Documentation/Manual/PrimitiveObjects.html> Acesso em: 14 Outubro 2013.

UNITY. **Unity – Shaders.** 2013e. Disponível em: <http://docs.unity3d.com/Documentation/Manual/Shaders.html> Acesso em: 14 Outubro 2013.

UNITY. **Unity – Animation.** 2013f. Disponível em: <http://docs.unity3d.com/Documentation/Components/class-Animation.html> Acesso em: 14 Outubro 2013.

UNIVERSITY OF CAMBRIDGE. **The Relics Project – Archives photos.** 1999. Disponível em: <http://www.cl.cam.ac.uk/relics/archive_photos.html> Acesso em: 5 Outubro 2013.

VINCE, John. **Mathematics for Computer Graphics**. 2a. Ed. Nova Iorque: Springer Verlag, 2005.

VAUGHAN, William. **Digital Modeling**. 1a. Ed. Berkeley: New Riders. 2011.

WITTAYABUNDIT, Jate. **Unity 3 Game Development Hotshot**. 1a Ed. Birmingham. Packt Publishing, 2011.

XBOX WIRE. **Hear Xbox's Aaron Greenberg Share the Latest on the Gaming Industry's Growth**. 2013. Disponível em: <http://news.xbox.com/2013/05/x360-aaron-greenberg-industry-growth> Acesso em: 5 Outubro 2013