Capítulo 6

Aplicação das técnicas de Rigging e Normal Map na Unity para criar animações 2D com profundidade e iluminação dinâmica a partir de uma única imagem

Héder Pereira Rodrigues Silva, Isabele Rodrigues de Souza, Iallen Gábio de Sousa Santos

Abstract

In this chapter, we will explore the creation of 2D animations through the application of two techniques: rigging, to structure and animate characters, and the use of normal maps, to simulate surface relief and enable dynamic lighting effects. These techniques will be implemented using the Unity engine. Additionally, we will use artificial intelligence tools as creative support throughout the process. The course will have a visual and conceptual focus, with no programming involved. The animations produced can be used in game development, marketing materials, animated video production, among other applications.

Resumo

Neste capítulo, exploraremos a construção de animações 2D através da aplicação de duas técnicas: o rigging, para estruturar e animar personagens, e o uso de normal maps, para simular relevos e permitir efeitos de iluminação dinâmica. As técnicas serão implementadas na engine Unity. Adicionalmente vamos utilizar recursos de inteligência artificial como apoio criativo durante o processo. O curso terá enfoque visual e conceitual, sem programação. As animações produzidas podem ser utilizadas para o desenvolvimento de jogos, peças de marketing, produção de vídeo animado, entre outras aplicações.

6.1. Introdução

As animações 2D desempenham um papel fundamental em diversas mídias, desde jogos e aplicativos interativos até conteúdos digitais [Patel and Desai 2022]. Ademais, em setores

como o da publicidade e mídia, as animações vêm sendo utilizadas como ferramenta para a criação de propaganda desde o início da televisão até os dias de hoje [Mencari 2007]. Marcas e criadores de conteúdo recorrem a essa linguagem visual para capturar a atenção do público e transmitir ideias de forma criativa e memorável.

Nesse contexto, a animação 2D se mantém como uma ferramenta no desenvolvimento de experiências digitais. Jogos como Cuphead [MDHR 2017], por exemplo, mostram como esse estilo permanece relevante. Lançado em 2017, o título encantou o público com seu visual inspirado nas animações da década de 1930, além de ter entrado para o Guinness World Records [Guinness World Records 2019] com mais de 50.000 quadros feitos à mão — um marco para a indústria. A dedicação artesanal da equipe de desenvolvimento pode ser vista na Figura 6.1. À esquerda, Maja Moldehauer finalizando um dos 120.000 quadros do jogo. À direita, canetas consumidas em apenas duas semanas de traba- lho intenso. Mesmo após anos de seu lançamento, Cuphead continua sendo amplamente jogado e reconhecido, o que reforça o poder expressivo e a longevidade da estética 2D.

Figura 6.1. Produção de animação do jogo Cuphead [MDHR 2017]

No mercado profissional, a animação 2D está presente tanto em grandes estúdios e agências quanto nas mãos de profissionais autônomos. Empresas de tecnologia, publicidade e entretenimento seguem valorizando artistas que dominam essa linguagem visual para enriquecer suas experiências digitais. No entanto, é especialmente entre freelancers, pequenos empreendedores, professores, estudantes e desenvolvedores independentes que a animação 2D se destaca como um produto criativo e economicamente viável. Com recursos digitais cada vez mais disponíveis, esses profissionais conseguem transformar ideias em projetos — e projetos em oportunidades reais de atuação no mercado.

Com o passar dos anos, essa linguagem visual passou por transformações e técnicas profundas. Nas primeiras animações tradicionais, como no próprio Cuphead, cada movimento exigia dezenas — às vezes centenas — de quadros desenhados à mão. Era um trabalho exaustivo, mas necessário para dar vida aos personagens. Hoje, com a multiplicação de canais, estilos e formatos, o mercado demanda produções mais rápidas e eficientes, capazes de atender a prazos curtos e grande volume de conteúdos. Para equipes pequenas ou artistas independentes, replicar a metodologia tradicional pode se tornar um grande desafio, já que exige uma carga de trabalho maior e, muitas vezes, investimentos que nem sempre são viáveis. Por isso, surgiram técnicas que tornam o processo mais prático e acessível, preservando a expressividade que caracteriza a animação 2D.

Entre essas técnicas estão o *Rigging* e o *Normal Mapping*. O *Rigging* consiste em construir uma estrutura articulada dentro do personagem, como se fosse um esqueleto invisível, como mostra a Figura 6.2. Isso permite manipular o personagem com muito mais facilidade, ajustando poses e expressões de maneira rápida, sem a necessidade de desenhar cada quadro manualmente. Dentro da engine *Unity*, uma das plataformas de desenvolvimento multiplataforma mais utilizadas no mundo para jogos e aplicações interativas [Unity 2025], o *Rigging* se integra diretamente ao fluxo de produção, oferecendo uma forma prática de construir animações mais dinâmicas e detalhadas. Essa abordagem não apenas reduz o tempo de produção de animações, como também possibilita que profissionais criem projetos complexos com alto nível técnico e visual, mesmo trabalhando de maneira independente.

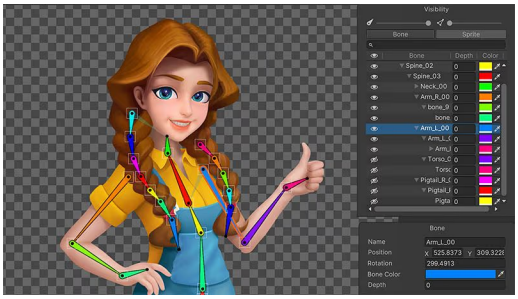


Figura 6.2. Aplicação de Rigging na Unity [Mango Animate 2023]

Outra técnica que contribui para ampliar as possibilidades visuais na animação 2D é o uso de *Normal mapping*. Embora tradicionalmente associados a gráficos 3D, os Normal Maps também podem ser aplicados em sprites 2D para criar efeitos de iluminação mais sofisticados. Eles funcionam armazenando informações sobre a orientação da superfície de um objeto em cada ponto da imagem, o que permite que a luz interaja de forma mais realista com os elementos visuais. Na prática, isso possibilita gerar sombras, brilhos e profundidade sem a necessidade de redesenhar manualmente cada variação de iluminação como visto na Figura 6.3. Para artistas independentes e pequenas equipes, o uso de Normal Maps pode representar uma grande economia de tempo e esforço, ao mesmo tempo em que eleva o nível de detalhamento e dinamismo das cenas.

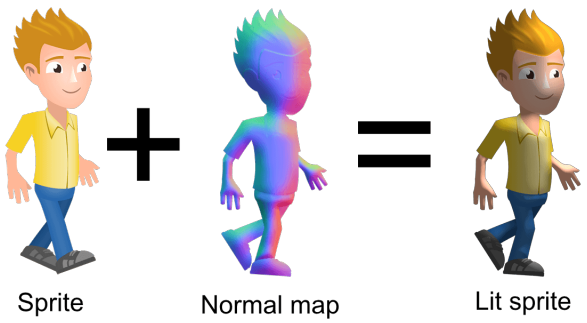


Figura 6.3. Aplicação de Normal Map no sprite

[GDQuest 2025]

Assim, ao longo deste minicurso, exploraremos as principais técnicas que tornam o desenvolvimento de animações 2D mais acessível e eficiente dentro do motor *Unity*. Através de conceitos como *Rigging* e *Normal Mapping*, será possível compreender como otimizar fluxos de trabalho, reduzir custos e, ainda assim, manter a expressividade e a qualidade estética nas produções. Nas próximas seções, vamos apresentar as bases fundamentais da animação digital, contextualizar a importância dessas práticas no mercado atual e, por fim, demonstrar na prática como implementar essas técnicas na criação de personagens e cenas interativas.

6.2. Rigging

A técnica de rigging (animação esquelética) tem raízes históricas na animação em recortes e fantoches. Lotte Reiniger, pioneira da animação de silhuetas, utilizou mecanismos de articulação física em seus famosos filmes de recortes em 1926, como Die Abenteuer des Prinzen Achmed, criando movimentos por meio de cavidades e rebites que funcionavam como juntas mecânicas [Moritz 1996].

Com a chegada da computação gráfica, esses princípios foram adaptados para o digital: a animação cutout evoluiu para o rigging virtual em softwares dedicados. Em 1988, Nadia Magnenat-Thalmann e colaboradores formalizaram o conceito de Joint-Dependent Local Deformations (JLD) para animar mãos e objetos articulados, introduzindo esqueletos hierárquicos que controlam a deformação de superfícies em tempo real [Magnenat-Thalmann et al. 1988]. Desde então, o rigging evoluiu com o surgimento de softwares especializados(como a própria Unity) que possibilitam pipelines digitais de animação 2D baseados em esqueletos.

6.2.1. Fundamentos

Na animação 2D, o rigging organiza um personagem em partes articuladas manipuláveis em um processo típico: primeiramente divide-se o desenho do personagem em camadas (corpo, membros, cabeça, olhos, boca etc.). Em seguida, define-se uma hierarquia de ossos interconectados (skeleton), onde cada osso corresponde a um segmento do personagem. Cada articulação (joint) estabelece um pivô de rotação – por exemplo, ombros e quadris são pontos de dobra naturais. A partir dos ossos, gera-se uma malha de vértices que cobre as partes do sprite; ferramentas modernas como Unity permitem autogerar essa geometria do sprite. Em seguida, atribuem-se pesos (weights) aos vértices: cada osso influencia certos vértices da malha, determinando como a imagem 2D se deforma durante o movimento. O resultado é um sistema de controle no qual o animador manipula "alavancas" em vez de redesenhar manualmente cada quadro. Isso agiliza a produção, permitindo movimentos fluídos e reutilização de animações complexas.

Exemplo de fluxo de rigging: (a ser ilustrado em figura) um personagem 2D é montado em partes; cria-se uma hierarquia de ossos e articulações; autorregenera-se a malha dos sprites; aplica-se skin weight para cada osso; então ajustam-se deformadores ou controladores inversos (IK) para animar o personagem.

6.2.2. Exemplos de Produções que utilizam Rigging

O rigging 2D é amplamente utilizado para agilizar animações em diversas produções. Por exemplo, estúdios de jogos independentes usam técnicas de cutout rig em Flash ou engines modernas: o próprio estúdio *Klei Entertainment* descreve que títulos como *Shank*, *Mark of the Ninja* e *Don't Starve Together* são animados com personagens montados a partir de partes em Flash [Kevin (Klei Entertainment) 2012]. Nesses pipelines, cada personagem é definido por cerca de uma dúzia de partes móveis (e.g. Mark of the Ninja) até dezenas (e.g. Shank), o que permite criar múltiplas vistas de cada membro e gerar sprites animados via scripts.

Na indústria de animação 2D, muitas séries de TV e curtas também se beneficiam de rigs digitais. Por exemplo, personagens de desenhos modernos (como certas séries animadas em *Toon Boom Harmony* ou *Spine*) são criados como marionetes digitais: as articulações predefinidas tornam a produção mais rápida e facilitam ajustes finos de poses. Outros casos notáveis incluem produções cinematográficas com animação híbrida (e.g. *America: The Motion Picture* usa rig de bonecos 2D) e franquias de televisão (por exemplo, *O incrível Mundo de Gumball* e *South Park*), que incorporam rigs para movimentação consistente dos personagens.

6.2.3. Softwares e Plataformas de Rigging 2D

Atualmente, existem diversas plataformas que suportam a criação de animações com Rigging, cada uma delas se especificando em algum gênero de animação em específico ou caso de uso; alguns exemplos são:

- Unity [Unity 2025]: o Unity provê um sistema completo de rigging 2D integrado. Nele, o animador pode criar cadeias de ossos no Sprite Editor, autogerar a geometria da sprite (mesh) e definir pesos automaticamente. Essa estrutura converte o sprite em um ator skinned, permitindo não apenas poses articuladas mas também aplicação de mapas normais para iluminação dinâmica nas sprites. Em versões recentes, o Unity adicionou suporte a Secondary Textures (mapas de normais e altura) em sprites animadas por ossos, o que viabiliza luzes 2D realistas e senso de profundidade em jogos 2D.

- Spine [Software 2025b]: software dedicado a animação esquelética 2D. Permite importar imagens, criar ossos hierárquicos, pintar pesos e exportar a animação para várias engines (incluindo Unity) via runtimes. É amplamente usado em jogos por ser leve e flexível.

- Toon Boom Harmony [Inc. 2025]: padrão da indústria de animação 2D para TV e filmes. A versão Premium conta com um poderoso sistema de rigging: art-redecut (montagem de personagens em partes), ferramentas de deformação (bones, curvas, morphs) e gestão de cenas complexas. Riggers profissionais usam Harmony para fazer personagens articuláveis, o que acelera produções seriadas.

- Moho [LLC 2025]: ferramenta de animação 2D focada em cutout rigging. Usa esqueletos vetoriais e ossos rígidos, com editor de Smart Bones que previne distor-

ções. É popular em animação independente e educacional, oferecendo um workflow rápido de montagem.

Cada uma dessas plataformas aplica o conceito de ossos hierárquicos e pesos em sprites, tornando o rigging uma etapa fundamental de pipelines de animação 2D. Essas ferramentas consolidadas coexistem com outras (como DragonBones, Synfig e Adobe Animate) que também oferecem sistemas de rig e skin, todos visando acelerar a produção e possibilitar efeitos avançados, como iluminação dinâmica ou deformações sutis.

6.3. Normal Map

A técnica de *Normal Mapping* surgiu no final da década de 1990 como resposta à crescente demanda por realismo gráfico em aplicações tridimensionais [Cignoni et al. 1998]. O objetivo era representar detalhes visuais de relevo — como rugosidades, saliências e ranhuras — sem a necessidade de aumentar a quantidade de polígonos dos modelos 3D. Isso reduzia significativamente o custo de renderização em tempo real. Um marco importante dessa abordagem foi seu uso no jogo *Doom 3* (2004), que popularizou o efeito ao aplicar o *Normal Mapping* para gerar superfícies detalhadas em ambientes com iluminação dinâmica [id Software 2004]. Desde então, a técnica foi adotada amplamente em engines de jogos e softwares de renderização.

Com a evolução das engines 2D, o uso de *normal maps* passou a ser explorado também em ambientes bidimensionais, adaptando seus princípios à estética e aos desafios técnicos do 2D. Em essência, um *normal map* é uma textura que armazena, em cada pixel, informações sobre a orientação da superfície naquele ponto — codificadas como vetores tridimensionais representados pelas cores RGB. Ao aplicar esse mapa a um sprite, o sistema de renderização interpreta como a luz deve se comportar sobre ele, criando áreas de sombra e brilho como se houvesse relevo [GDQuest 2020].

6.3.1. Fundamentos

A técnica de *normal mapping* em ambientes 2D se baseia na utilização de uma textura complementar, conhecida como *normal map*, para simular relevo e profundidade em imagens planas. Cada pixel dessa textura codifica a orientação da superfície em **coordenadas tridimensionais X, Y e Z**, representadas pelas **cores RGB**. O canal R representa a direção horizontal, o G a direção vertical e o B a inclinação da superfície em relação ao observador. Essas informações permitem que a engine calcule como a luz interage com cada ponto da imagem, gerando efeitos de sombra e brilho.

O fluxo básico da aplicação envolve, inicialmente, a criação de um sprite base (imagem principal do personagem ou objeto). Em seguida, produz-se um *normal map* correspondente, seja manualmente, com pintura direta dos vetores de iluminação, ou automaticamente, por meio de softwares especializados. Esse *normal map* é então associado ao sprite dentro da engine, onde será interpretado por um sistema de iluminação 2D. Durante a execução da cena, o motor gráfico combina a direção da luz com os vetores presentes no *normal map* para alterar, em tempo real, a aparência do sprite. Isso simula uma iluminação volumétrica sem alterar a geometria da imagem. O resultado é uma ilusão convincente de profundidade que mantém alto desempenho, pois não exige múltiplas versões sombreadas da imagem.

Exemplo de fluxo do *normal mapping*: (a ser ilustrado em figura) cria-se o sprite base; gera-se um *normal map* equivalente que corresponda à imagem; ambos são importados para a engine com um sistema de luz 2D; a engine processa os vetores do mapa em tempo real conforme a direção e intensidade da fonte de luz. A imagem plana reage com sombras e brilhos, criando efeitos de relevo visual.

6.3.2. Exemplos de Produções que Utilizam *Normal Mapping*

O uso de *normal mapping* em ambientes 2D tem ganhado espaço em produções que buscam enriquecer a iluminação sem comprometer a leveza gráfica. Estúdios independentes e animadores digitais utilizam essa técnica para simular relevo e profundidade em sprites, principalmente quando desejam aplicar efeitos de luz dinâmicos sem multiplicar os quadros de animação.

Um exemplo prático pode ser observado no jogo *Crawl*, desenvolvido pela Powerhoof, que utiliza *normal maps* para criar atmosferas dramáticas com lanternas oscilantes e fontes de luz em constante movimento [Powerhoof nd]. Outro caso relevante é o jogo *Dead Cells*, da Motion Twin, onde o *normal mapping* é empregado para acentuar detalhes dos cenários e personagens, como texturas de pedra, metal ou tecido, sem comprometer o desempenho [Vasseur 2020].

Na área de animação, produções experimentais e curtas independentes também vêm incorporando essa técnica. Um exemplo notável é o curta "Sprite DLight Demo Reel" (2016), desenvolvido com o software SpriteIlluminator, que demonstra diferentes formas de aplicar iluminação dinâmica em sprites 2D utilizando *normal maps* [CodeAndWeb 2023]. Além disso, ferramentas como Unity [Unity 2025] e Godot [Godot Engine 2025] vêm oferecendo suporte nativo para sistemas de luz 2D baseados em *normal mapping*, o que facilita a adoção da técnica por animadores e desenvolvedores.

Esses exemplos mostram como o *normal mapping* amplia as possibilidades expressivas da arte 2D, permitindo controlar efeitos de sombra, brilho e ambientação com mais flexibilidade e impacto visual, sem sacrificar o desempenho.

6.3.3. Softwares e Plataformas com Suporte a *Normal Mapping* 2D

Com o avanço dos motores gráficos e ferramentas de animação, diversas plataformas passaram a oferecer suporte nativo a *normal mapping* em ambientes 2D. Essa técnica permite simular relevo e iluminação dinâmica em sprites planos, proporcionando profundidade visual sem exigir modelagem tridimensional. A seguir, destacam-se algumas das principais soluções utilizadas por desenvolvedores e animadores:

- **Unity**: a partir da versão 2023.2, o Unity integra sistemas de luz 2D que suportam diretamente texturas de *normal map*. O componente *Sprite-Lit* permite que artistas associem texturas secundárias aos sprites, resultando em iluminação volumétrica com performance otimizada [GDQuest 2025].
- **Godot Engine**: a engine open-source Godot oferece suporte a *normal mapping* em seu sistema 2D. Através do recurso *CanvasItemMaterial*, é possível aplicar mapas normais, mapas de altura e efeitos de luz localizados em sprites e tiles [Godot Engine 2025].

- **SpriteIlluminator**: ferramenta especializada na criação de *normal maps* 2D. Permite gerar os mapas automaticamente ou pintá-los manualmente, com suporte de visualização da luz em tempo real. Exporta arquivos compatíveis com Unity, Godot, entre outros [CodeAndWeb 2023].

- **Materialize**: ferramenta gratuita desenvolvida pela Bounding Box Software. Com ela, é possível gerar mapas normais, de altura e especular a partir de uma imagem comum. Ideal para artistas que querem refinar detalhes manuais com controle total [Software 2025a].

- **NormalMap-Online**: aplicação web gratuita que permite gerar *normal maps* a partir de imagens de altura diretamente no navegador, sem necessidade de instalação [Cpetry 2025].

- **Laigter**: ferramenta gratuita focada na criação de *normal maps* e mapas especulares para sprites 2D, com suporte a visualização em tempo real e exportação compatível com diversas engines [Azagaya 2025].

- **Krita**: software de pintura digital que, através do *Tangent Normal Brush Engine*, permite a pintura manual de *normal maps*, oferecendo controle artístico detalhado sobre a orientação das superfícies [Foundation 2025].

- **Medibang Paint** e **Clip Studio Paint**: embora não possuam suporte nativo a *normal mapping*, esses softwares permitem que artistas pintem mapas manualmente utilizando cores RGB correspondentes às direções de luz (**R** = X, **G** = Y, **B** = Z). Esses mapas podem ser exportados como texturas normais e integrados posteriormente nas engines.

Essas ferramentas cobrem desde pipelines profissionais até abordagens manuais mais experimentais. Assim, o *normal mapping* em 2D torna-se acessível tanto para grandes estúdios quanto para artistas independentes.

6.4. Unity

O motor gráfico Unity, desenvolvido pela Unity Technologies [Unity 2025], consolidou-se ao longo dos anos como uma das plataformas mais populares para criação de jogos e experiências interativas. Lançado oficialmente em 2005 com foco na democratização do desenvolvimento de jogos, o Unity propôs uma abordagem acessível: permitir que artistas, desenvolvedores e pequenos estúdios criassem conteúdos multimídia complexos com um fluxo de trabalho integrado.

Originalmente concebido para projetos em 3D, a engine expandiu-se com o tempo para incluir suporte robusto à produção em 2D — especialmente a partir da versão 4.3 (2013), que introduziu um pipeline dedicado para sprites. Com isso, a Unity tornou-se uma ferramenta relevante não apenas para desenvolvedores de jogos tridimensionais, mas também para animadores, ilustradores e criadores de narrativas interativas em duas dimensões.

6.4.1. Animação 2D

No contexto da animação 2D, como já discutido nas Seções 6.2 e 6.3, a Unity oferece um conjunto de ferramentas versátil que cobre desde a importação de assets até a animação e renderização avançada. A engine suporta a manipulação de imagens vetoriais e rasterizadas, organização hierárquica de elementos, e animações baseadas em esqueletos — uma técnica detalhada na Seção 6.2. Além disso, permite a aplicação de efeitos visuais sofisticados, como sombras e iluminação dinâmica, que se beneficiam de mapas de normais, como abordado na Seção 6.3.

Embora outras ferramentas e engines também ofereçam suporte a esses recursos, ao longo do minicurso será dada uma atenção especial à Unity como ambiente principal de desenvolvimento. Ainda assim, é importante reconhecer que, no universo da computação gráfica, existem diversas abordagens e plataformas que implementam essas mesmas técnicas com diferentes níveis de complexidade e flexibilidade.

6.5. Tutorial: como criar um protótipo de animações na Unity

Uma vez apresentados os conceitos base relacionados às técnicas de **Rigging** e **normal map**, nesta seção será feita uma demonstração das técnicas apresentadas por meio de um tutorial para a criação de um protótipo básico de animação 2D na *Unity*. Mostraremos como fazer a textura de mapeamento normal no sprite e a sua preparação para que o rigging seja feito adequadamente, além de como importamos esse sprite para a Unity. Após isso, mostraremos como criar os ossos no sprite e as animações do personagem. Então, ensinaremos como criar ativadores e gatilhos para animações. Por fim, mostraremos como implementar a luz dinâmica na cena. Ao final, teremos um pequeno cenário interativo onde será possível testar movimentos articulados e efeitos de luz dinâmica, servindo de base para projetos mais complexos.

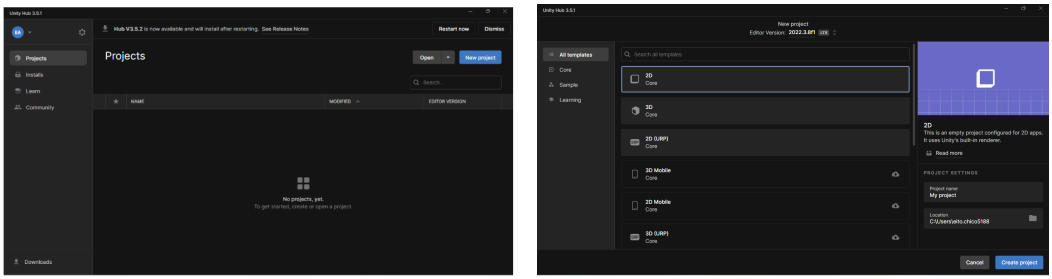
6.5.1. Preparação do Ambiente de Desenvolvimento

Antes de começarmos a desenvolver o sprite, devemos realizar alguns preparativos para a criação do protótipo. Inicialmente, é necessário fazer o download da plataforma de desenvolvimento Unity Hub, acessando o site oficial da plataforma [Unity 2025]. De acordo com o site oficial da Unity, os requisitos mínimos de sistema para utilização da plataforma são:

- OS: Windows 7 SP1+, 8, 10, 64-bit versions only; macOS 10.12+; Ubuntu 16.04, 18.04, and CentOS 7.
- GPU: Placa gráfica com recursos do DX10 (Shader Model 4.0).

Agora, criaremos o projeto do protótipo. Para isso, acesse o Unity Hub, localize o botão *New Project*, na aba de projetos, vista na Figura 6.4(a). Na aba New Project (Figura 6.4(b), selecione a opção *Universal 2D*, renomeie-o como preferir e clique no botão *Create Project*; assim, será criado seu projeto na Unity.

Ao fim deste processo de criação, aparecerá a interface de desenvolvimento básica da Unity, vista na Figura 6.5. Os elementos desta interface serão apresentados adiante.



(a) Aba Projects na Unity (b) Criação de um novo Projeto

Figura 6.4. Abas Project e New Project na Unity.

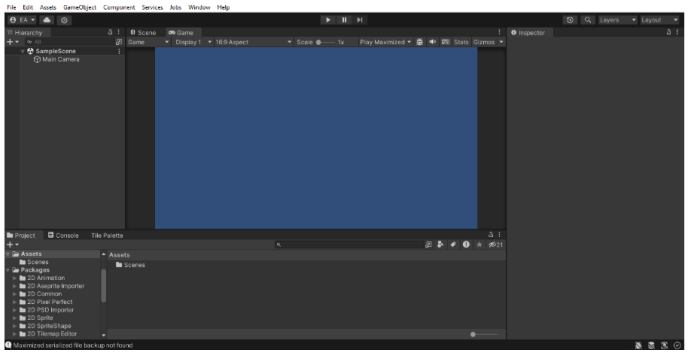


Figura 6.5. Interface Básica de Desenvolvimento

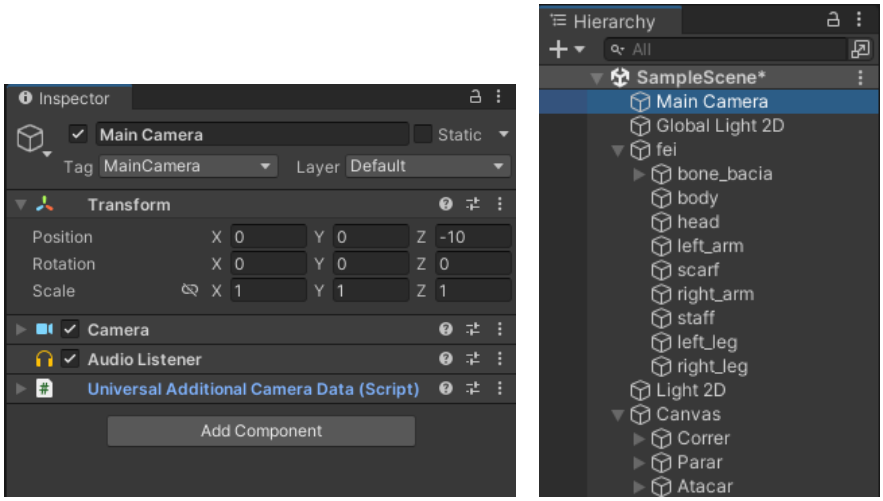
Na Unity, um projeto é representado por uma cena contendo objetos. Estes objetos contêm componentes que representam características do objeto, como: posicionamento, detector de colisões, corpo físico, elemento gráfico, elemento sonoro, etc. Além disso, é possível adicionar scripts a esses objetos para acessar as propriedades e manipulá-los.

6.5.1.1. Inspector

Localizada no lado direito do monitor, a janela inspector aborda as características e os atributos dos objetos. Nela são apresentadas as opções de configuração de um objeto dentro da Unity. Como pode-se ver na Figura 6.6(a), o objeto “Main Camera” possui o script "Universal Additional Camera Data"e os componentes “Transform”, “Camera” e “Audio Listener”.

6.5.1.2. Hierarchy

Nesta janela, localizada no lado esquerdo do visor, são organizados os objetos presentes na “cena”, como a câmera, o ponto de luz, os sprites do personagem e os seus ossos. Os objetos podem possuir sub-objetos, que são organizados e separados nesta janela, como é visto na Figura 6.6(b).

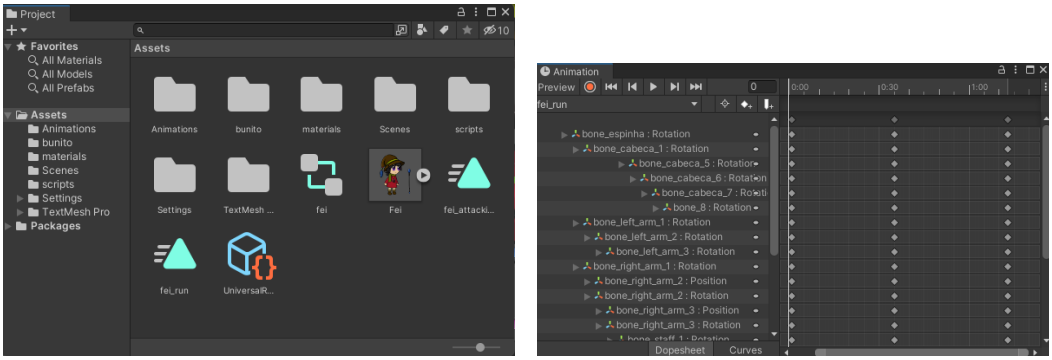


(a) Inspector (b) Hierarchy

Figura 6.6. Janelas Inspector e Hierarchy da Unity.

6.5.1.3. Project e Animation

Localizadas na região interior do editor estão as janelas Project e Animations. Na janela Project estão localizados os arquivos do protótipo, sendo possível acessar/escolher arquivos. Pode-se observar na Figura 6.7(a) que os arquivos ficam organizados em ficheiros. Na janela Animations (Figura 6.7(b) é onde as animações são desenvolvidas. Ela também possui uma aparência semelhante a um editor de vídeo, com botões de gravação e de reprodução e uma linha do tempo. Inicialmente, essa aba não está visível; para exibi-la deve-se utilizar o comando *Cntrl + 6* e, após isso, arrastar a janela para o lado da aba Project.



(a) Project (b) Console

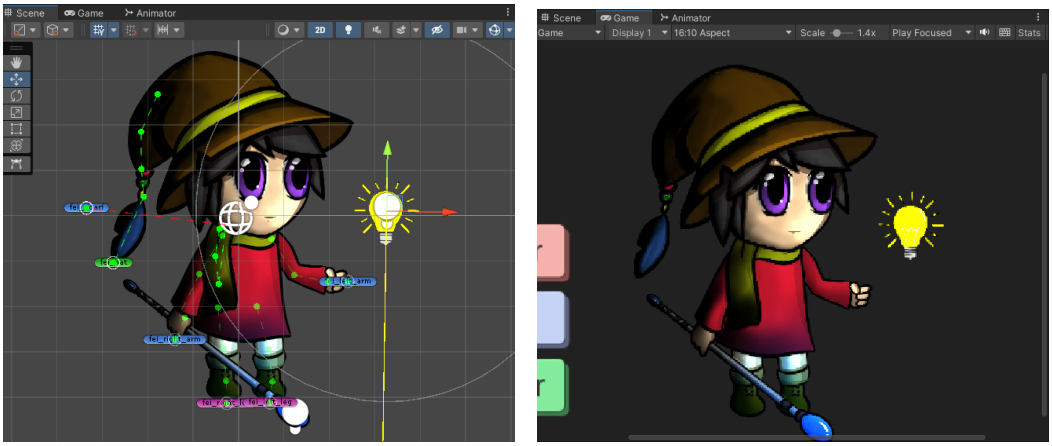
Figura 6.7. Janelas Project e Console da Unity.

6.5.1.4. Scene e Game

Em todo projeto, é necessário um local para organizar, posicionar e visualizar os objetos da cena. Na Unity, esta janela é chamada de Scene (Figura 6.8(a)). Nela, podemos alterar

características dos objetos, como suas posições, tamanho e rotação.

A janela Game possui a função de simular a visualização da animação quando ela é executada durante o processo de desenvolvimento(Figura 6.8(b)).



(a) Scene (b) Game

Figura 6.8. Janelas Scene e Game da Unity.

6.5.1.5. Animator

Essa janela é responsável por gerenciar a reprodução das animações na cena. Ela será descrita com mais detalhes posteriormente (Figura 6.9). Assim como a janela Animation, inicialmente ela não está visível, mas é exibida assim que um *Animation Controller* é criado.

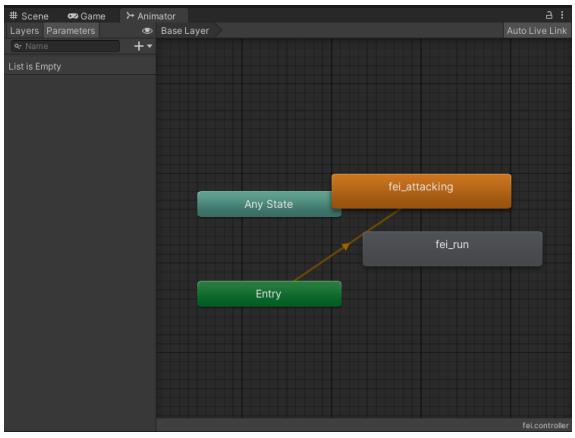


Figura 6.9. Janela Animator da Unity

6.5.2. Preparação do Sprite

Neste tutorial, vamos aplicar as animações e o mapeamento normal em um sprite obtido através da loja de *assets* da Unity, no entanto essas técnicas podem ser aplicadas em

qualquer desenho de personagem 2D. Para que a técnica de rigging seja aplicada com uma maior eficiência, é comum que seja feita a separação dos membros do personagem em camadas diferentes (Figura 6.10). Para a fabricação desse tipo de sprite, apenas é necessária a utilização de um software de desenho que possua um recurso de camadas. Neste tutorial, utilizaremos o software **Medibang Paint Pro** [MediBang Inc. 2025] para a separação do sprite e também, posteriormente, para o mapeamento normal. Após isso, cada camada deve ser exportada em um arquivo png próprio. No Medibang, isso é feito por meio do atalho *Cntrl* + *Shift* + *S* e selecionando o tipo de arquivo .png para salvar.

Figura 6.10. Separação do Sprite do personagem em camadas

Uma vez que a separação foi feita, deverá ser feita uma textura complementar de normal map para cada uma das camadas. As *coordenadas tridimencionais* - citadas na Seção 6.3 - são aplicadas utilizando uma *roda de cor normal* como base para a texturização. Exemplos de roda de cor de normal map podem ser observados na Figura 6.11(a), elas servem para mostrar quais cores indicam ângulos específicos de uma "esfera"com mapeamento normal, por exemplo: Se uma fonte de luz for colocada à direita de uma das esferas da Figura 6.11(a), as regiões com cores frias (azul, verde e roxo) serão mais iluminadas do que as regiões com cores quentes (rosa, vermelho e amarelo).

(a) Roda de Cor (b) Aplicação da Roda de Cor

Figura 6.11. Demonstração do efeito do mapeamento normal

Para criar a textura de um sprite, este raciocínio deve ser seguido: a forma como a luz deve se comportar no objeto deve ser similar à da roda de cor. Para uma melhor vi-

sualização, veja a Figura 6.11(b), perceba que as regiões pintadas com as cores que ficam mais à direita na roda de cor (regiões vermelhas e amareladas) estão mais iluminadas do que o resto da cabeça da personagem. Um pensamento simples que pode ser seguido no processo de texturização é: "Se eu colocar um ponto de luz em uma certa posição, quais partes da figura estarão mais iluminadas?". As regiões mais iluminadas serão aquelas mais próximas do ponto de luz na orientação da roda de cor, enquanto as mais escuras estarão do lado oposto à luz na roda de cor.

A texturização pode tanto ser feita em softwares especializados para criação de texturas, que dispõem de ferramentas para facilitar esse processo; porém, ela também pode ser feita em softwares de desenho com ferramentas de pintura e conta-gotas. As texturas complementares, assim como cada parte do sprite, devem ser salvas em formato .png transparente, e preferencialmente, em um diretório reservado junto com os arquivos do sprite principal. Uma boa prática para a criação dessas texturas é nomear o seu arquivo com o mesmo nome do sprite em que será aplicado juntamente com o sufixo "*_n*", por exemplo, se o arquivo do sprite da cabeça do personagem da Figura 6.11(b) for nomeado como "*head*", o nome do arquivo da textura deverá ser "*head_n*".

6.5.3. Criando os rigs de animação

Uma vez que o sprite esteja preparado, é preciso importar os arquivos para o projeto na Unity para iniciarmos o processo de **Rigging**. Para isso, no explorador de arquivos, arraste a pasta contendo os sprites e normal maps para dentro da janela *Project*, que está na interface do projeto Unity. Feito isso, todos os arquivos .png estarão dentro dos diretórios do projeto, classificados como "Sprites 2D".

Agora, para a criação dos rigs, devemos acessar um ambiente chamado *Skinning Editor*. Clique em um dos arquivos de sprite do personagem, assim a janela *Inspector* irá mostrar suas propriedades; nessa janela, clique no botão *Sprite Editor*, isso abrirá uma nova janela do sprite editor; no canto superior esquerdo, clique no menu dropdown e selecione *Skinning Editor*, isso levará a um ambiente semelhante ao da Figura 6.12.

Figura 6.12. Janela Skinning Editor Unity

Esse ambiente possui as seguintes abas que serão utilizadas no tutorial:

- **Pose** - Nessa aba você pode interagir com os ossos criados e ver a sua influência no sprite e também restaurar para o formato original.

- **Bones** - Aqui são criados os ossos por meio do *Create Bone*, quando o usuário clica nessa ferramenta ele pode gerar os ossos do sprite ao clicar e arrastar para definir tamanho e direção.
- **Geometry** - A geometria é responsável por delimitar qual serão as partes afetadas pelos ossos, criando arestas e vértices ao redor do Sprite.
- **Weights** - Os pesos definem quais arestas e vértices serão deformados pelos ossos e a intensidade da influência. As áreas de influência de cada osso são representadas pelas manchas coloridas

Para criar e configurar os ossos, deve-se primeiro clicar duas vezes no sprite para selecioná-lo, logo após ir para a aba de geometria e selecionar a ferramenta *Auto Geometry* e depois clicar no botão *Generate for Selected* - que aparecerá no canto inferior direito - e assim, automaticamente, a geometria do sprite será criada. Após isso, na aba *Bones*, selecione a ferramenta *Create Bone* e posicione os ossos conforme o formato do sprite, considerando a forma como aquele objeto deve se comportar. Por fim, na aba *Weights*, selecione a ferramenta *Auto Weights* e, de forma semelhante à geometria, clique no botão *Generate*. Com esses passos realizados, você vai ter um resultado semelhante à Figura 6.13, teste como os ossos estão afetando o sprite com o *Preview Pose*, se for necessário, faça ajustes utilizando *Edit Bone*, *Edit Geometry* e *Weight Slider*.

Visualizando a Figura 6.13, observa-se que o rigging na Unity é constituído de 3 principais elementos: os ossos, representados pelos ponteiros coloridos; a geometria, representada pelas linhas e pontos azuis que contornam o sprite e as linhas brancas que atravessam o sprite; e os pesos, representados pelas manchas coloridas, definindo a área de influência de cada osso.

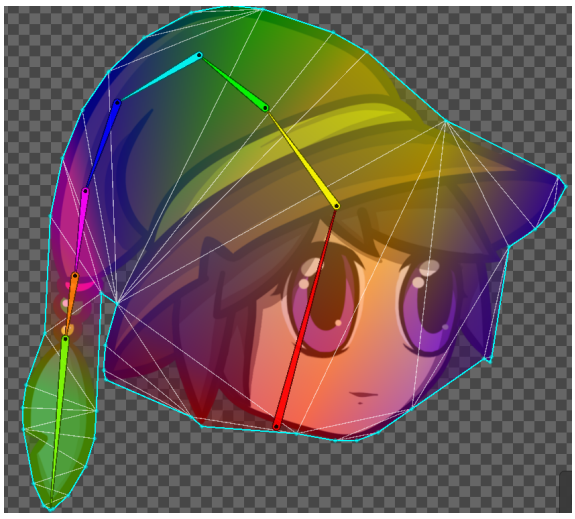
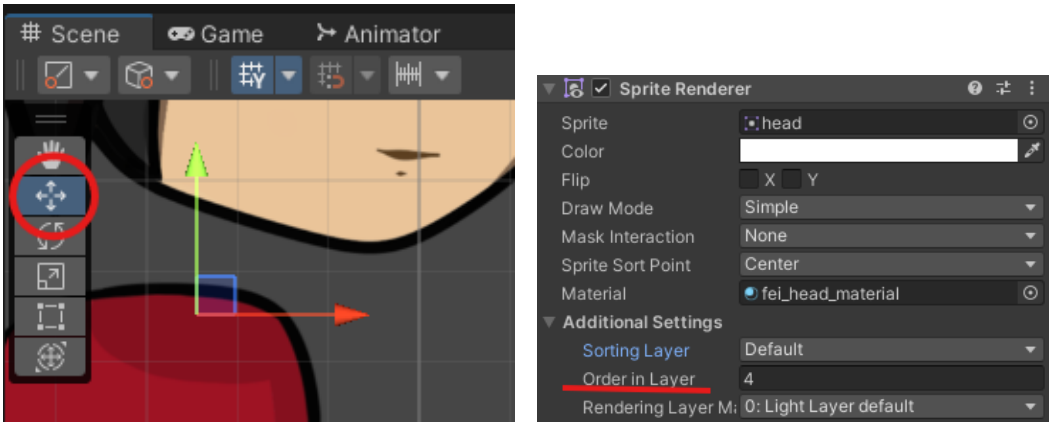


Figura 6.13. Rigging na Unity

6.5.4. Colocando o sprite no cenário e aplicando os ossos

Uma vez que o rigging for realizado em todas as partes do personagem, podemos montá-lo novamente dentro do cenário. Primeiro, vamos criar um objeto para o nosso player: na

janela *Hierarchy*, acesse o menu de contexto (pressionando o botão direito do mouse) e selecione a opção *Create Empty*, isso irá criar um objeto vazio onde iremos armazenar os ossos e os sprites do personagem, e então, renomeie o objeto para "Character". Logo após isso, crie mais dois objetos dentro de Character (faça o mesmo processo só que abrindo o menu de contexto clicando no objeto Character), e os renomeie para *Character_Sprites* e *Character_Bones*. Após isso, arraste e solte os sprites da janela *Project* para a cena, isso criará vários objetos com um componente chamado *Sprite Renderer* na hierarquia, arraste esses objetos para dentro de *Character_Sprite*. Feito isso, ajuste a posição dos sprites na cena utilizando a ferramenta de mover (Figura 6.14(a)) e também defina uma ordem de camada no componente *Sprite Renderer* - como na Figura 6.14(b) - para que as sobreposições estejam corretas.



(a) Ferramenta de Mover da cena (b) Order in Layer

Figura 6.14. Ferramenta de mover e Atributo de ordem de camada

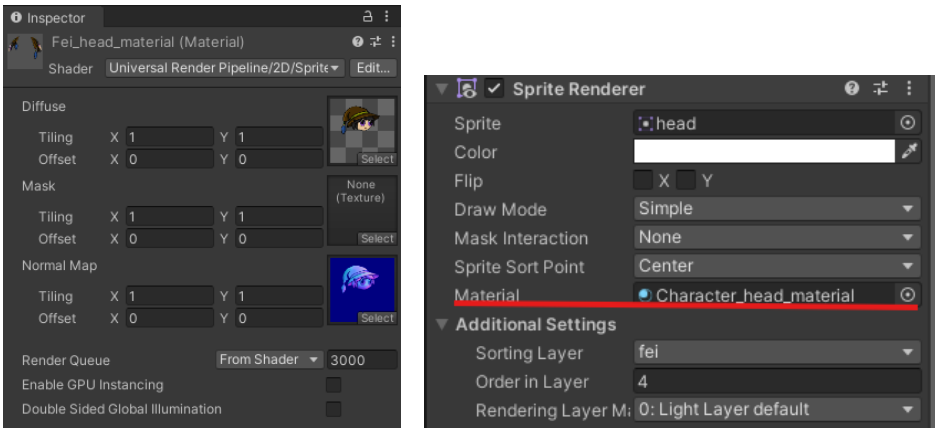
Para adicionarmos os rigs na cena, devemos adicionar um componente chamado *Sprite Skin* em todos os sprites da cena, isso é feito acionando o botão *Add Component* no inspetor do sprite e pesquisando pelo nome *Sprite Skin* na barra de pesquisa. Seguindo, pressione o botão *Create Bones* (Figura), isso criará os ossos como filhos do sprite selecionado na janela *Hierarchy*. Os ossos também têm uma hierarquia própria entre si, quando um osso pai é movimentado, todos os seus filhos são movimentados com ele, enquanto o contrário não ocorre, essa característica serve para criarmos ossos centrais, que funcionam como a espinha de todo o esqueleto do personagem. Por isso, mova os ossos para que estejam dentro do objeto *Character Bones* na hierarquia, e os organize para que componham um "esqueleto"unificado do seu personagem - esse passo é importante para facilitar a animação, e dar nomes identificativos para cada osso ajuda nesse processo. No final dessa etapa, a sua cena e hierarquia deverá estar parecida com a Figura 6.15.

6.5.5. Implementação do Normal Map no Sprite

O único passo restante antes de começarmos as animações é aplicarmos as texturas complementares no cenário. Primeiro, no diretório em que estão localizadas as texturas normais, selecione uma textura, e no *Inspector*, mude o parâmetro *Texture Type* para *normal map*, faça isso para todas as texturas. Então, crie uma pasta chamada "Materials"no diretório de assets do projeto, fazemos isso abrindo o menu de contexto na janela *Project*, se-

Figura 6.15. Rigging do Personagem na Cena

lecionando "*Create*"e então "*Folder*". De forma semelhante, crie **materiais** dentro desse diretório, seguindo os passos: Menu de contexto > Create > Material. Ao clicar em um material, o *Inspector* exibirá a interface mostrada na Figura 6.16(a). No espaço chamado *Diffuse*, clique em *Select* e pesquise por um dos sprites e selecione-o. No espaço *normal map*, adicione a textura complementar correspondente ao sprite selecionado. Tendo feito isso, só resta aplicarmos este material no *Sprite Renderer* do sprite na cena, como na Figura 6.16(b), e repetirmos esse processo para todos os membros do personagem.



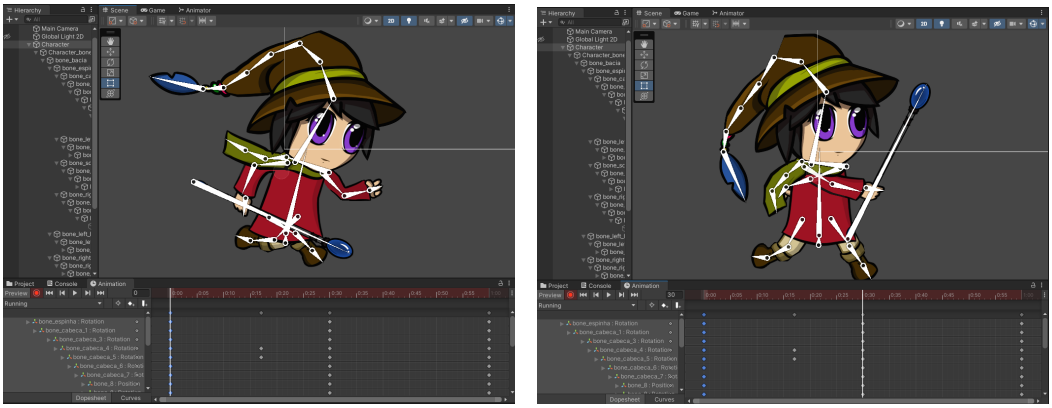
(a) Inspector de um Material (b) Material Adicionado no Sprite

Figura 6.16. Inspector de um Material

6.5.6. Montagem das animações

Com a preparação do personagem completa, crie uma nova pasta e a renomeie com o nome "*Animations*", nela armazenaremos os arquivos de animação e o controlador. Feito isso, por meio do menu de contexto, crie um arquivo do tipo *Animator Controller* e outro do tipo *Animation* e renomeie-os para "Character Controller"e "Running". Após isso, selecione o objeto "Character"na hierarquia e adicione o componente "*Animator*", então, coloque o "Character Controller"no campo *Controller* desse componente, de forma semelhante à aplicação do material no Sprite Renderer feita na seção passada.

Com esses passos, agora podemos gravar animações na janela *Animation* (Figura



(a) Keyframe 1 (b) KeyFrame 2

Figura 6.17. Gravação da Animação

6.17(a)). Vamos agora demonstrar como é feito um clipe de animação. Primeiramente, clique no botão de iniciar a gravação de keyframes - representado pelo círculo vermelho - isso fará com que o editor entre no modo de gravação - isso significa que todas as alterações feitas na cena existirão apenas como registros no arquivo Animation e serão aplicadas quando a animação for tocada. Logo, nesse estado, clique no objeto Character na hierarquia (isso tornará visíveis os ossos do personagem) e, clicando e girando os rigs, faça uma pose de corrida no personagem de forma semelhante à Figura 6.17(a). Com isso, você terá criado o primeiro *keyframe* da animação, representado pelo conjunto de pontos na linha do tempo; esses pontos são chamados de propriedades e eles são as alterações da cena feitas naquele frame, que no caso atual são apenas rotações feitas nos componentes *Transform* dos ossos movidos.

Em seguida, mova o indicador da linha do tempo para 0:30 (clique e arraste nos indicadores de milissegundos) e faça o segundo keyframe da animação, como na Figura 6.17(b). Clicando no botão de reprodução, veja que a Unity fará automaticamente a transição entre esses dois frames-chave, tirando a necessidade de cada frame ser construído como nos métodos de animação tradicional. Para terminarmos o loop de corrida, clique no primeiro keyframe (ponto mais acima, que representa todas as transformações), copie com o atalho *Cntrl + C*, mova o indicador para 1:00 e cole. Assim, clique novamente no botão de gravar keyframes para encerrar a gravação.

Com a animação feita, precisamos agora colocá-la no *Animator*, acessamos essa janela clicando duas vezes no *Animation Controller* na aba do projeto. Isso abrirá a interface chamada *Animator*, exibida na Figura 6.18. Nela, toda a animação é organizada em *States*, que representam clipes de animação reproduzidos conforme a máquina de estados avança; além dos states comuns (por exemplo a animação que criamos), existem três especiais: o *Entry*, que determina por qual animação a máquina inicia; o *Any State*, que permite disparar transições a partir de qualquer ponto; e o *Exit*, que encerra o fluxo de animação quando alcançado. Para adicionarmos a animação *Running* nesse ambiente, podemos arrastá-la da nossa janela project para dentro do controlador.

Entre esses estados, desenham-se transições, que conectam um estado a outro e podemos definir as regras de quando a máquina deve sair de um estado e entrar em outro.

Como a animação *Running* é a única que incluímos no ambiente, ela automaticamente virá com uma transição entre ela e o estado *Entry*, assim como na Figura 6.18. Feito isso, já é possível simular o protótipo (Botão de reprodução na parte superior do editor ou o atalho *Cntrl + C*), e vermos a animação que fizemos em ação na visão do usuário final.

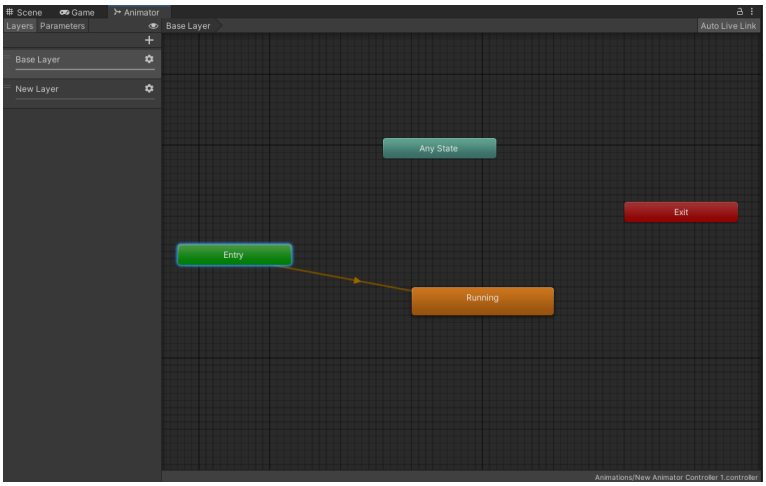


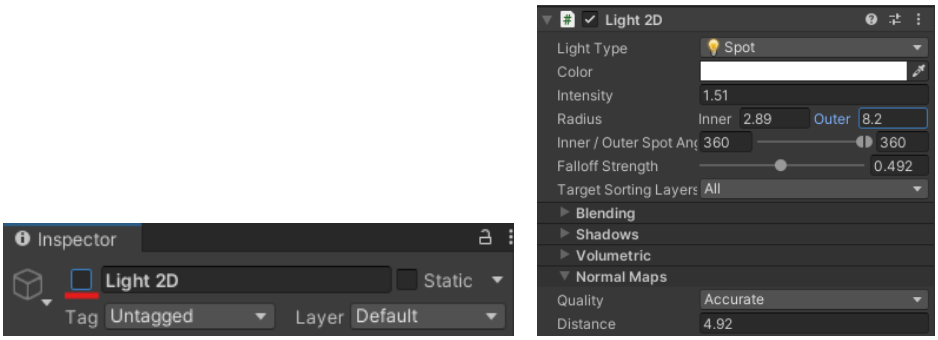
Figura 6.18. Caption

Se quiser adicionar outros estados, para criar uma nova transição, deve-se abrir o menu de contexto em um estado e selecionar *Make Transition*, e selecionar o novo estado para que a máquina irá se direcionar. Por padrão, a condição para a mudança de estados é o fim da reprodução do estado atual, se ele não for um loop, mas podemos adicionar regras para fazermos transições. Essas regras são avaliadas com base em **parâmetros** — variáveis do tipo float, int, bool ou trigger — que podem ser alteradas em tempo real pelo código ou pelo próprio Animator, guiando a evolução das animações conforme valores como velocidade ou condições lógicas. Para criar um novo parâmetro, clique no botão "+"(Figura 6.18) e selecione o tipo de parâmetro que definirá a transição; então, clique na transição, e no *Inspector* dela, adicione o parâmetro no segmento *Transitions*.

6.5.7. Criando a luz dinâmica na cena

Para podermos visualizar o efeito do mapeamento normal no personagem, devemos também utilizar um sistema de iluminação que explore este recurso. No momento, a cena está iluminada pelo objeto chamado *Global Light*, essa luz ilumina igualmente todos os objetos da cena, sem considerar o mapeamento normal. Então, para vermos o normal map em ação, devemos desabilitar a luz atual e adicionar uma luz que considere essa textura para aplicar a animação. Para isso, primeiro desative as propriedades da luz global na cena. Com esse objetivo, clique no objeto *Global Light 2D* na hierarquia - e no seu Inspector, desative o campo localizado ao lado do nome do objeto, como na Figura 6.19(a). Em seguida, na hierarquia, abra o menu de contexto, selecione a opção *Light* e depois *SpotLight 2D*, isso criará um novo objeto de luz, dessa vez uma luz limitada, que podemos mover pela cena, porém perceba que o mapeamento normal ainda não está em efeito. Para resolvermos esse problema, vá até o componente *Light 2D*, no campo *normal map*, mude o valor *Quality* para "**Accurate**", como na Figura 6.19(b). Por fim, faça ajustes nos valores *Inner* e *Outer* para deixar o raio em um tamanho favorável à visualização do

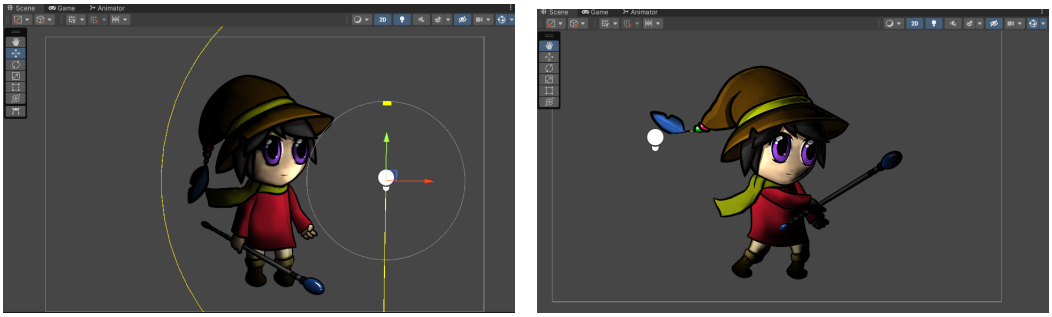
personagem. Assim, você terá uma luz que irá considerar a textura complementar dos sprites no momento de iluminá-los, dando a impressão de relevo.



(a) Desativando a Luz Global na cena (b) Configurando a nova luz Spot

Figura 6.19. Implementando a Nova Luz

Com todos esses passos concluídos, você terá um personagem totalmente articulado, com textura que reage dinamicamente à luz, permitindo a criação de diversos tipos de animações para diferentes gestos e ações. Além disso, um objeto de luz poderá ser movimentado ao redor do personagem, possibilitando a simulação de diferentes situações de iluminação, como ilustrado na Figura 6.20.



(a) Exemplo 1 (b) Exemplo 2

Figura 6.20. Resultado do Tutorial

6.6. Conclusão

Neste capítulo, apresentamos o rigging como a criação de um “esqueleto invisível” em cada sprite: uma estrutura hierárquica de ossos e articulações que permite deformar e posicionar personagens de forma fluida, sem a necessidade de desenhar todos os quadros manualmente. Mostramos que essa técnica integra conceitos tradicionais de animação e agiliza a produção ao oferecer controles diretos sobre poses e gestos. Posteriormente, introduzimos o normal mapping 2D, explicando sua origem no 3D e a adaptação para sprites: uma textura adicional cujos pixels codificam vetores de orientação de superfície em RGB, permitindo que sistemas de luz 2D gerem sombras e brilhos simulando relevo, sem aumentar a geometria ou comprometer o desempenho.

Em seguida, destacamos como Unity integra essas técnicas em um pipeline acessível e visual, sem necessidade de programação. Ao usar o 2D Animation Package, o artista pode trabalhar no Sprite Editor e no Skinning Editor para gerar automaticamente geometria, ossos e pesos, em conjunto com o sistema nativo de luzes 2D que suporta mapas normais e texturas secundárias. Esse fluxo oferece previews em tempo real — desde a montagem do rig até a aplicação de materiais com mapas normais e ajustes de iluminação.

Por fim, percorremos todo o processo prático, desenvolvendo um protótipo que aborda os conceitos principais das duas técnicas e que está pronto para servir de base a projetos mais complexos.

Portanto, espera-se que os conhecimentos adquiridos ao longo deste capítulo sejam não apenas valiosos para a compreensão do desenvolvimento de animações e texturas, mas também funcionem como uma porta de entrada para aqueles que desejam ingressar nessa área profissionalmente.

Referências

[Azagaya 2025] Azagaya (2025). Laigter - normal maps for 2d sprites. `https://azagaya.itch.io/laigter`. Acesso em: 12 maio 2025.

[Cignoni et al. 1998] Cignoni, P., Montani, C., and Scopigno, R. (1998). A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54.

[CodeAndWeb 2023] CodeAndWeb (2023). Spriteilluminator – normal map editor. `https://www.codeandweb.com/spriteilluminator`.

[Cpetry 2025] Cpetry (2025). Normalmap-online. `https://cpetry.github.io/NormalMap-Online/`. Acesso em: 12 maio 2025.

[Foundation 2025] Foundation, K. (2025). Tangent normal brush engine - krita manual. `https://docs.krita.org/en/reference_manual/brushes/brush_engines/tangen_normal_brush_engine.html`. Acesso em: 12 maio 2025.

[GDQuest 2020] GDQuest (2020). Lighting with 2d normal maps. `https://gdquest.com/tutorial/godot/2d/lighting-with-normal-maps/`.

[GDQuest 2025] GDQuest (2025). Lighting with 2d normal maps - gdquest. `https://gdquest.com/tutorial/godot/2d/lighting-with-normal-maps/`. Acesso em: 12 maio 2025.

[Godot Engine 2025] Godot Engine (2025). 2d lights and shadows. Acesso em: 14 maio 2025.

[Guinness World Records 2019] Guinness World Records (2019). Most hand-drawn frames in a video game. Acesso em 28 abr. 2025.

[id Software 2004] id Software (2004). Doom 3. Game, id Tech 4 Engine.

[Inc. 2025] Inc., T. B. A. (2025). Toon boom harmony. Disponível em: https://www.toonboom.com/products/harmony Acesso em: 2 maio 2025.

[Kevin (Klei Entertainment) 2012] Kevin (Klei Entertainment) (2012). What animation technique is used in 'Don2019t Starve'? https://gamedev.stackexchange.com/questions/44319/what-animation-technique-is-used-in-dont-starve. Answer on GameDev StackExchange.

[LLC 2025] LLC, L. M. (2025). Moho animation software. Disponível em: https://moho.lostmarble.com/ Acesso em: 2 maio 2025.

[Magnenat-Thalmann et al. 1988] Magnenat-Thalmann, N., Laperrière, R., and Thalmann, D. (1988). Joint-dependent local deformations for hand animation and object grasping. In *Proc. Graphics Interface '88*, pages 25–34.

[Mango Animate 2023] Mango Animate (2023). Animação e rigging de personagem 2d: Faça rigging 2d em 5 minutos. Disponível em: https://school.mangoanimate.com/pt/2d-character-rigging-and-animation-make-2d-rigging-in-5-minutes/. Acesso em: 31 de maio de 2024.

[MDHR 2017] MDHR, S. (2017). Cuphead. Jogo eletrônico.

[MediBang Inc. 2025] MediBang Inc. (2025). MediBang Paint Pro. https://medibangpaint.com/. Acesso em: 17 maio 2025.

[Mencari 2007] Mencari, M. A. (2007). O mundo animado nos comerciais: dos cobertores parahyba ao assolan. *Unknown Journal*.

[Moritz 1996] Moritz, W. (1996). Some critical perspectives on lotte reiniger. *Animation Journal*. Discusses early silhouette animation techniques and historical context.

[Patel and Desai 2022] Patel, R. and Desai, A. (2022). Animation techniques and trends in digital media. *Journal of Engineering Design and Analysis*, 4(2):15–22.

[Powerhoof nd] Powerhoof (n.d.). Crawl art process vid 1. https://www.powerhoof.com/crawl-art-process-vid-1/. Acesso em: 12 maio 2025.

[Software 2025a] Software, B. B. (2025a). Materialize - free texture map generator. https://www.boundingboxsoftware.com/materialize/. Acesso em: 12 maio 2025.

[Software 2025b] Software, E. (2025b). Spine - animação 2d para jogos. Disponível em: https://esotericsoftware.com/ Acesso em: 2 maio 2025.

[Unity 2025] Unity (2025). Plataforma de desenvolvimento em tempo real do unity. Disponível em: https://unity.com/pt. Acesso em: 04 de outubro 2023.

[Vasseur 2020] Vasseur, T. (2020). Art design deep dive: Using a 3d pipeline for 2d animation in dead cells. *Game Developer*. Acesso em: 12 maio 2025.