INSTITUTO FEDERAL
Sudeste de Minas Gerais
Campus Muriaé

MINISTÉRIO DA EDUCAÇÃO

PÁTRIA AMADA BRASIL
GOVERNO FEDERAL

Jogos Digitais

Apostila Didática

Publicado e distribuído por Paulo Vinícius Moreira Dutra

paulo.dutra@ifsudestemg.edu.br

Sumário

Parte Um

1. Introdução ao Jogos Digitais

1.1 O que é um jogo digital?

Um jogo digital é um tipo de entretenimento interativo que é executado em um dispositivo eletrônico, como um computador, console de videogame, smartphone ou tablet. Ele envolve a interação do jogador com elementos virtuais através de controles, como teclado, mouse, controle de videogame ou tela sensível ao toque.

Os jogos digitais podem ter uma ampla variedade de gêneros, como ação, aventura, estratégia, quebra-cabeças, RPG (role-playing game), esportes, simulação e muitos outros. Eles podem apresentar gráficos em 2D ou 3D, som e música imersivos, efeitos visuais e uma narrativa envolvente.

Os jogos digitais oferecem aos jogadores desafios, objetivos e recompensas, proporcionando uma experiência interativa e envolvente. Eles podem ser jogados individualmente ou em multiplayer, permitindo que os jogadores interajam com outras pessoas através da internet. Além disso, muitos jogos digitais oferecem recursos de personalização, como a criação de personagens, a escolha de habilidades e a customização de equipamentos.

Com o avanço da tecnologia, os jogos digitais evoluíram ao longo dos anos, tornando-se cada vez mais complexos e realistas. Eles são uma forma popular de entretenimento, com milhões de pessoas em todo o mundo desfrutando de jogos em diversas plataformas. Além do entretenimento, os jogos digitais também são usados em áreas como educação, treinamento e simulação.

1.2 Game Design

Game design é a disciplina que abrange o processo de criação e desenvolvimento de jogos. Envolve o planejamento, a concepção e a definição de todos os elementos que compõem um jogo, desde a jogabilidade e os sistemas de regras até a narrativa, os gráficos, o som e os níveis de dificuldade. O game design engloba uma variedade de aspectos importantes na criação de um jogo, incluindo:

- **Mecânicas de jogo:** São as regras e interações que definem como o jogo é jogado. Isso inclui movimento do personagem, combate, resolução de quebra-cabeças e qualquer outro elemento de jogabilidade.
  Jogabilidade: Refere-se à forma como os jogadores interagem com o jogo e como a experiência de jogo é estruturada. Isso inclui a curva de aprendizado, o equilíbrio entre desafio e recompensa, a fluidez do jogo e a sensação de controle do jogado
- **Narrativa:** É a história do jogo, incluindo personagens, cenários e eventos que ocorrem ao longo do jogo. A narrativa pode ser linear, ramificada ou até mesmo emergente, onde a história é influenciada pelas ações do

jogador.

- **Níveis e progressão:** Envolve a criação de desafios progressivamente mais difíceis ao longo do jogo, garantindo que os jogadores sejam incentivados a continuar jogando. Isso pode incluir a criação de fases, mundos ou níveis específicos.
- **Equilíbrio:** Envolve ajustar cuidadosamente as mecânicas do jogo, os desafios e as recompensas para garantir uma experiência equilibrada. O jogo deve ser desafiador o suficiente para manter os jogadores interessados, mas não tão difícil a ponto de se tornar frustrante.
- **Interface do usuário:** É a forma como os jogadores interagem com o jogo, incluindo menus, HUDs (exibição de informações) e controles. Uma interface intuitiva e fácil de usar é essencial para uma experiência de jogo satisfatória.
- **Estética visual e sonora:** Engloba a aparência visual e o estilo artístico do jogo, bem como a trilha sonora e os efeitos sonoros. Esses elementos contribuem para a atmosfera e a imersão do jogo.
- **Feedback:** É a resposta que o jogo fornece aos jogadores em relação às suas ações e decisões. O feedback pode ser visual, sonoro ou tátil e é importante para manter os jogadores engajados e informados sobre o que está acontecendo no jogo.
- Testes e ajustes: É importante realizar testes extensivos durante o desenvolvimento do jogo para identificar problemas, equilibrar a dificuldade e fazer ajustes nas mecânicas de jogo. Esses testes podem envolver uma equipe interna de desenvolvimento, bem como jogadores beta externos.

Os game designers geralmente trabalham em equipes multidisciplinares, colaborando com artistas, programadores, roteiristas e outros profissionais para criar um jogo completo e envolvente. Eles precisam ter uma compreensão profunda dos princípios do design de jogos, bem como um conhecimento sólido dos gostos e preferências do público-alvo.

1.3 Processo de Design de Jogo

O processo de design de jogos pode variar dependendo da equipe e do projeto específico, mas geralmente segue uma estrutura semelhante. Aqui está uma visão geral dos estágios típicos do processo de design de jogos:

- **Conceituação:** Nesta fase inicial, a equipe de design de jogos reúne-se para gerar ideias e conceitos para o jogo. Isso pode envolver brainstorming, pesquisas de mercado, identificação de público-alvo e definição dos principais objetivos e mecânicas do jogo.
- **Documentação:** Uma vez que uma ideia principal tenha sido estabelecida, é importante documentar os detalhes do jogo de forma mais abrangente. Isso pode incluir a criação de documentos como a visão geral do jogo, a descrição da história, os personagens, os níveis e as mecânicas de jogo. Essa documentação servirá como um guia durante o desenvolvimento do jogo.
- **Design de Níveis e Mecânicas:** Nesta etapa, a equipe de design de jogos trabalha na criação de níveis, que podem incluir layout de ambientes, quebra-cabeças, desafios e progressão de dificuldade. Além disso, são definidas as mecânicas de jogo, como movimento, combate, interações com o ambiente e sistemas de recompensa. É importante equilibrar a diversão, o desafio e a progressão do jogo.
- **Desenvolvimento:** Aqui, a equipe de desenvolvimento trabalha para criar o jogo completo, incluindo arte, música, design de níveis, programação e outros aspectos técnicos. É importante seguir um cronograma e manter uma comunicação eficiente entre os membros da equipe para garantir que o jogo seja desenvolvido de acordo com as especificações.
- **Prototipagem:** O próximo passo é criar um protótipo do jogo. Isso envolve a implementação de uma versão básica e funcional do jogo, geralmente com gráficos e recursos simplificados. O objetivo é testar as mecânicas do jogo, avaliar sua viabilidade e ajustar qualquer problema identificado.
- **Testes e Iteração:** Nesta fase, o jogo é testado por jogadores reais para identificar problemas, falhas, dificuldades e áreas de melhoria. Com base no feedback dos testes, a equipe faz iterações no design,

ajustando e refinando o jogo para torná-lo mais equilibrado, envolvente e divertido.

- **Arte e Produção:** Uma vez que o design básico do jogo esteja definido, a equipe de arte entra em ação para criar os visuais e os ativos do jogo. Isso inclui gráficos, animações, áudio, trilha sonora e qualquer outro elemento visual ou auditivo necessário para a experiência do jogo.
- **Implementação e Programação:** Nesta fase, os programadores trabalham na implementação do jogo, transformando os designs e ativos em código. Isso envolve a criação de sistemas de jogo, IA, mecânicas específicas e integração de arte e som. É importante garantir que o jogo funcione corretamente e que todos os elementos estejam em sincronia.
- **Polimento:** À medida que o jogo se aproxima da conclusão, é importante dedicar tempo para polir e refinar todos os aspectos. Isso envolve aprimorar a qualidade dos gráficos, aprimorar a trilha sonora, jogabilidade, corrigir bugs, ajustar o equilíbrio e a dificuldade, e garantir que a experiência geral do jogo seja a melhor possível.
- **Lançamento:** Após o polimento final, o jogo está pronto para ser lançado. Dependendo da escala do projeto, isso pode envolver o lançamento em plataformas específicas, distribuição online ou outras formas de disponibilização para o público.
- **Pós-lançamento:** Após o lançamento do jogo, a equipe de design continua a monitorar o feedback dos jogadores, corrigir bugs e lançar atualizações ou expansões, conteúdo adicional por meio de patches ou DLCs (conteúdo para download) se necessário. Esse estágio também pode envolver a análise de dados e o uso dessas informações para melhorar futuros projetos.

É importante lembrar que esse processo é uma visão geral e que cada estúdio ou equipe de desenvolvimento pode ter suas próprias variações e abordagens específicas. Além disso, o design de jogos é um processo iterativo, no qual os estágios podem se sobrepor e as decisões de design podem mudar ao longo do tempo, à medida que a equipe aprende com os testes e interações com o jogo em desenvolvimento.

1.4 Game Design Document (GDD)

O Game Design Document (GDD) é um documento usado no desenvolvimento de jogos para descrever e documentar todos os aspectos do design do jogo, desde a mecânica de jogo até a história, personagens, arte, som e muito mais. É um recurso essencial para garantir que todos os membros da equipe de desenvolvimento do jogo tenham uma compreensão clara e compartilhada do que está sendo criado.

Embora o conteúdo específico de um GDD possa variar de acordo com o projeto e a equipe, aqui estão alguns elementos comuns que podem ser encontrados em um GDD:

- **Visão geral do jogo:** Uma descrição geral do jogo, incluindo o conceito central, gênero, plataforma alvo e público-alvo.
- **Mecânica de jogo:** Uma explicação detalhada das regras, sistemas e interações do jogo, como movimento do personagem, combate, quebra-cabeças, etc.
- **História e personagens:** Uma sinopse da história do jogo, incluindo personagens principais, antagonistas e suas motivações.
- **Arte e estilo visual:** Descrições e exemplos de arte conceitual, estilo visual, gráficos, animações, efeitos especiais e design de níveis.
- **Áudio:** Descrições dos efeitos sonoros, trilha sonora, diálogos e narração que serão usados no jogo.
- **Fluxo de jogo:** Um diagrama ou descrição do fluxo geral do jogo, desde o início até o final, incluindo menus, fases, níveis, telas de carregamento, etc.
- **Progressão e recompensas:** Detalhes sobre a progressão do jogador, como desbloquear novas habilidades ou itens, e recompensas por alcançar objetivos no jogo.
- **Interface do usuário:** Descrição da interface do usuário, incluindo menus, HUD (Heads-Up Display),

controles e outras interações com o jogador.

Esses são apenas alguns exemplos de seções que podem ser incluídas em um GDD. Cada equipe de desenvolvimento pode personalizar o documento de acordo com suas necessidades e preferências. O GDD é uma ferramenta valiosa para manter todos os membros da equipe alinhados e fornecer uma referência centralizada durante todo o processo de desenvolvimento do jogo.

1.4.1 Level Design

Level design (ou design de níveis) é uma disciplina que faz parte do desenvolvimento de jogos de vídeo game e se refere à criação e organização dos diferentes níveis ou fases de um jogo. O level design envolve a concepção dos espaços virtuais, a disposição de obstáculos, inimigos, itens e desafios, bem como a definição da progressão e fluxo de gameplay em um jogo.

O objetivo do level design é criar experiências de jogo interessantes e envolventes, garantindo que os níveis sejam equilibrados, desafiadores e divertidos de jogar. Isso envolve a consideração de fatores como o equilíbrio de dificuldade, a fluidez da progressão, a exploração do espaço, o uso efetivo de mecânicas de jogo e a narrativa do jogo.

Os level designers geralmente trabalham em estreita colaboração com outros membros da equipe de desenvolvimento de jogos, como designers de jogabilidade, artistas, programadores e roteiristas, para criar os níveis. Eles podem usar ferramentas de design específicas para criar o layout dos níveis e testar o gameplay, fazendo ajustes conforme necessário para melhorar a experiência do jogador.

Em resumo, o level design é a arte de projetar e construir os níveis de um jogo, cuidando de todos os aspectos que influenciam a experiência de jogo, desde a estrutura espacial até os elementos de desafio e recompensa. É uma parte fundamental do desenvolvimento de jogos, pois contribui diretamente para a qualidade e diversão do jogo.

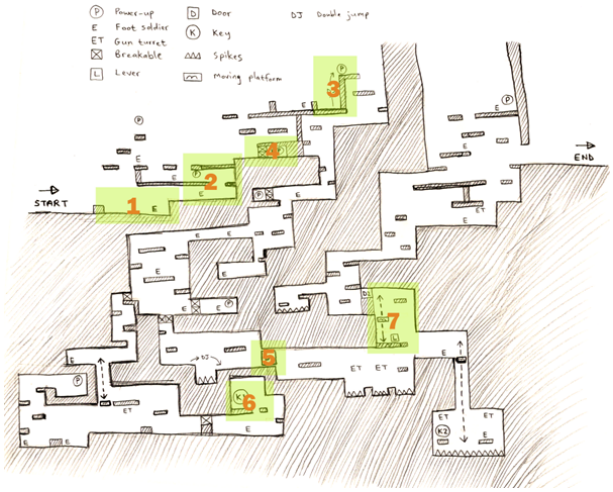


Figura 1.1: Exemplo level design. (Fonte Tilioi)

Exemplos de Level Design:

1. Super Mario Bros: Cada nível neste clássico da Nintendo é cuidadosamente projetado para introduzir novos elementos de jogabilidade e desafios enquanto mantém uma progressão suave.
2. Dark Souls: Os níveis neste jogo são notórios por sua complexidade e interconectividade. Cada área oferece diferentes desafios e segredos para os jogadores descobrirem.

3. The Legend of Zelda: Breath of the Wild: O vasto mundo aberto deste jogo apresenta um design de nível não linear, permitindo aos jogadores explorar livremente e descobrir novas áreas e desafios.

Dicas para Design de Níveis Efetivos:

1. Comece Simples: Introduza conceitos e mecânicas básicas antes de complicar o design dos níveis.
2. Equilíbrio de Desafios: Varie entre momentos de ação intensa e momentos mais calmos para manter o interesse do jogador.
3. Feedback Iterativo: Teste seus níveis regularmente com jogadores para identificar áreas problemáticas e fazer ajustes.
4. Recompensas Significativas: Certifique-se de que as recompensas oferecidas aos jogadores valem o esforço necessário para obtê-las.
5. Imersão na Narrativa: Use o design dos níveis para contar partes importantes da história do jogo de forma visual e interativa.
6. Acessibilidade: Certifique-se de que os níveis sejam acessíveis para jogadores de diferentes habilidades, oferecendo opções de dificuldade ajustáveis, por exemplo.
7. Consistência Visual: Mantenha um estilo visual consistente em todo o jogo para garantir uma experiência coesa.

Como planejar o level design do jogo

Planeje uma curva de dificuldade gradual, começando com fases mais fáceis e introduzindo gradualmente desafios mais complexos. Certifique-se de que cada fase tenha um objetivo claro e único para o jogador.

Layout dos Níveis:

Comece esboçando o layout dos níveis em papel ou usando ferramentas de design gráfico. Considere o equilíbrio entre os elementos de plataforma, inimigos, obstáculos e itens para criar desafios interessantes. Varie a altura das plataformas, crie caminhos alternativos e adicione segredos ou áreas ocultas para incentivar a exploração.

Planeje momentos de descanso entre os desafios mais intensos para evitar fadiga do jogador. Alterne entre seções de plataforma, combate e quebra-cabeças para manter a jogabilidade variada.

Design de Inimigos:

Introduza inimigos mais fracos no início do jogo e, gradualmente, apresente inimigos mais poderosos e estratégicos. Varie os padrões de movimento dos inimigos para evitar monotonia e criar desafios únicos em cada fase.

Objetos e Elementos Interativos:

Coloque itens e power-ups em locais estratégicos para incentivar a exploração e recompensar o jogador. Adicione alavancas, interruptores e outros elementos interativos para criar quebra-cabeças e desafios adicionais.

Teste e Ajuste:

Faça testes frequentes dos níveis para garantir que eles sejam desafiadores, mas não injustamente difíceis. Faça ajustes com base no feedback dos testadores, equilibrando a dificuldade e garantindo a diversão do jogo. Lembre-se de criar níveis interessantes, com uma combinação de desafios e recompensas para manter o jogador engajado. Experimente diferentes ideias, iterações e ajustes até encontrar o equilíbrio ideal.

Em resumo

Para planejar corretamente um level design, você pode seguir algumas etapas:

Defina o objetivo do nível: Antes de começar a criar o level design, é importante ter uma compreensão clara do propósito do nível. Pergunte-se o que você deseja que o jogador sinta, realize ou aprenda durante essa fase.

Identifique o público-alvo: Considere quem serão os jogadores do seu jogo e adapte o design do nível para atender às suas preferências e habilidades. Por exemplo, um jogo para crianças terá níveis mais simples e diretos do que um jogo para jogadores experientes.

Crie um fluxo de jogo: Planeje a progressão do jogador pelo nível, criando um fluxo de jogo que seja intuitivo e envolvente. Certifique-se de que existam desafios apropriados e que o nível esteja bem estruturado.

Estabeleça um tema ou estilo visual: Defina um tema ou estilo visual para o seu nível, que esteja alinhado com a narrativa ou estética geral do jogo. Isso ajudará a criar uma identidade visual única e coesa para o nível.

Projete os desafios: Crie desafios interessantes e variados para manter o jogador engajado. Considere a dificuldade gradual, proporcionando momentos de sucesso e momentos de maior dificuldade ao longo do nível.

Itere e teste: À medida que você projeta o level design, faça iterações e teste com jogadores reais para obter feedback. Isso ajudará a identificar problemas, ajustar o equilíbrio do jogo e refinar o design do nível.

Equilibre o nível: Certifique-se de equilibrar adequadamente o nível, ajustando a dificuldade, o ritmo, os itens e as recompensas para proporcionar uma experiência desafiadora, mas justa.

Oriente o jogador: Utilize elementos de design, como sinalização visual, iluminação, trilhas sonoras e outros recursos para guiar o jogador e garantir que ele compreenda a direção correta a seguir.

Considere a replayabilidade: Se possível, planeje o nível para ter fatores de replayabilidade, como objetivos opcionais, caminhos alternativos ou segredos escondidos. Isso incentivará os jogadores a revisitar o nível e descobrir novas coisas.

Polimento: Finalize o level design, realizando os ajustes finais, melhorando a estética, corrigindo bugs e garantindo que tudo esteja funcionando corretamente.

Lembre-se de que o level design é um processo criativo, e não existe uma fórmula única que funcione para todos os jogos. A prática, a iteração e a experiência ajudarão você a aprimorar suas habilidades de planejamento de level desig

1.4.2 Inimigos x Perigos

Em jogos digitais, os termos "inimigos"e "perigos"são frequentemente usados para descrever elementos que representam obstáculos e ameaças ao jogador. Embora esses termos possam ser usados de maneira intercambiável em alguns casos, eles têm significados distintos. Aqui está uma conceituação das diferenças entre inimigos e perigos:

Inimigos

Os inimigos em jogos digitais são personagens ou entidades controlados pelo computador que são projetados para interagir com o jogador de forma hostil. Eles são criados para criar desafios e conflitos no jogo. Os inimigos geralmente possuem inteligência artificial que lhes permite perseguir, atacar, esquivar-se ou adotar estratégias específicas para derrotar o jogador. Eles podem ser criaturas fictícias, como monstros, soldados inimigos ou até mesmo personagens não jogáveis controlados pelo computador.

Os inimigos podem ter diferentes habilidades, forças, fraquezas e padrões de comportamento, e muitas vezes o jogador precisa aprender a enfrentá-los e desenvolver estratégias para superá-los. Em muitos jogos, os inimigos são parte integrante da narrativa, e o jogador deve derrotá-los para progredir na história ou avançar para os próximos níveis. Em resumo, os inimigos representam uma ameaça ativa e direta ao jogador dentro do jogo.

Perigos

Os perigos, por outro lado, são elementos ambientais ou situações que representam ameaças ao jogador. Eles podem incluir obstáculos físicos, como buracos, plataformas instáveis, lava, espinhos ou armadilhas. Além disso, os perigos também podem incluir condições ambientais adversas, como tempestades, neblina densa, veneno, falta de oxigênio ou radiação.

Ao contrário dos inimigos, os perigos geralmente não possuem inteligência artificial e não estão ativamente tentando prejudicar o jogador. No entanto, eles podem ser igualmente desafiadores e requerem que o jogador utilize habilidades de movimento, reflexos rápidos e resolução de quebra-cabeças para evitá-los ou superá-los. Os perigos muitas vezes são colocados no jogo para adicionar variedade, tensão e imprevisibilidade ao ambiente, testando as habilidades do jogador em navegar com segurança pelo cenário.

Em resumo, enquanto os inimigos são personagens hostis controlados pelo computador que ativamente perseguem e atacam o jogador, os perigos são elementos ambientais ou situações que representam ameaças, mas não possuem uma inteligência artificial associada. Ambos desempenham papéis importantes na criação de desafios e na experiência geral do jogo.

Figura 1.2: Exemplo de Inimigos e Perigos (Castle in the Darkness)

1.5 Criação de personagens

A criação de personagens é uma parte essencial da construção de narrativas em jogos. Personagens bem desenvolvidos proporcionam profundidade e conexão emocional, tornando a história mais cativante para os leitores ou espectadores. Neste tópico, exploraremos os elementos fundamentais da criação de personagens, desde a introdução até a identificação do público com esses personagens.

1.5.1 Elaboração do personagens

Na construção de jogos digitais envolventes, a elaboração do personagem assume um papel central, contribuindo para a imersão dos jogadores e a ressonância emocional com a história do jogo. Esse processo vai além de simples atributos físicos e psicológicos; ele é a chave para criar personagens digitais que transcendem pixels e códigos, tornando-se entidades com as quais os jogadores podem se identificar profundamente.

A essência da elaboração do personagem em jogos digitais está na criação de avatares que não são apenas jogáveis, mas também convincentes. A aparência externa do personagem é habilmente integrada às suas motivações internas, objetivos e dilemas emocionais, proporcionando uma experiência mais rica e imersiva para o jogador.

O arco de desenvolvimento do personagem em um jogo não é apenas uma progressão linear, mas sim um mapeamento cuidadoso das mudanças na personalidade, habilidades e relações do personagem ao longo da jornada do jogo. Essa evolução não apenas impulsiona a narrativa, mas também influencia as escolhas dos jogadores, gerando um engajamento mais profundo.

O contexto e a história pessoal do personagem são fatores-chave na criação de uma experiência de jogo envolvente. Ao construir um passado significativo, os desenvolvedores não apenas fornecem um contexto para as ações do personagem, mas também criam uma conexão emocional entre o jogador e o mundo virtual.

Os relacionamentos e interações do personagem com outros elementos do jogo são essenciais para a construção de uma trama dinâmica. Relações bem elaboradas adicionam complexidade à narrativa, criando vínculos emocionais que impactam as escolhas dos jogadores e o curso da história.

Os desafios e obstáculos enfrentados pelo personagem digital não são apenas testes de habilidade, mas também oportunidades para os jogadores explorarem a profundidade da personalidade do personagem. Cada obstáculo superado contribui para a narrativa do jogo, revelando mais sobre a natureza do personagem e mantendo o interesse do jogador.

Em resumo, a elaboração do personagem em jogos digitais é um processo artístico e técnico que visa transformar linhas de código em seres virtuais cativantes. Quando feito com maestria, o personagem digital transcende sua natureza programada, tornando-se uma entidade memorável que não apenas conduz a história do jogo, mas também cria uma conexão duradoura com o jogador.

De forma resumida, segue alguns passos para elaborar um personagem.

1. **Características Físicas e Psicológicas: Físicas:** Descreva a aparência física do personagem, incluindo traços distintivos, roupas e expressões faciais. Psicológicas: Explore a mente do personagem, incluindo motivações, medos, desejos e conflitos internos.
2. **Arco de Desenvolvimento:** Planeje a trajetória do personagem ao longo da história. Isso inclui mudanças emocionais, aprendizados e desenvolvimentos pessoais.
3. **Contexto e História Pessoal:** Crie uma história de fundo para o personagem, explicando sua origem, experiências passadas e eventos que moldaram quem são.

4. **Relacionamentos e Conexões:** Desenvolva interações significativas com outros personagens na história. Relacionamentos bem construídos podem adicionar complexidade e realismo.
5. **Conflitos e Obstáculos:** Introduza desafios que o personagem enfrentará. O modo como eles lidam com conflitos revela muito sobre sua personalidade.

1.5.2 Identificação de personagens

A identificação do personagem é um aspecto fundamental no processo de criação de jogos digitais, estabelecendo uma ponte emocional entre o jogador e a experiência virtual. Esse elemento vai além do simples design visual ou habilidades do personagem; trata-se da capacidade de os jogadores se verem refletidos nos protagonistas digitais, conectando-se de forma significativa com suas histórias e jornadas.

Ao criar personagens identificáveis, os desenvolvedores buscam incorporar características universais que ressoem com a experiência humana. Aspectos como emoções, desafios pessoais, ambições e até mesmo falhas humanas são cuidadosamente entrelaçados na personalidade digital. Esse processo visa criar uma empatia instantânea, onde os jogadores podem se relacionar com as lutas e triunfos dos personagens como se fossem suas próprias.

A narrativa de um jogo muitas vezes se torna mais cativante quando os jogadores conseguem se projetar nos protagonistas. Por exemplo, em "The Last of Us", o jogador se conecta profundamente com Joel, um homem que enfrenta perdas significativas em um mundo pós-apocalíptico. Suas ações, motivações e dilemas ressoam com as emoções humanas, gerando uma identificação poderosa.

Outro exemplo notável é a personagem Ellie, também de "The Last of Us", cuja jornada de autoconhecimento, coragem e sobrevivência toca temas universais, criando uma ligação emocional com os jogadores. Sua identidade única, combinada com experiências compartilhadas, faz dela um personagem que transcende as limitações do meio digital.

Em "Life is Strange", Max Caulfield é uma estudante comum com poderes extraordinários. A narrativa se concentra em dilemas éticos e relações humanas, criando um vínculo de identificação à medida que os jogadores exploram as complexidades de suas escolhas.

Ao explorar exemplos emblemáticos, como Ryu e Ken de "Street Fighter", percebemos como a identificação do personagem pode moldar a experiência de jogo. Esses dois lutadores icônicos têm se destacado ao longo dos anos, não apenas por suas habilidades de combate, mas também por suas histórias e personalidades distintas.

Ryu, com sua busca incessante pela maestria e equilíbrio, personifica valores como disciplina e autoaperfeiçoamento. Sua jornada para se tornar um "lutador da rua"incide sobre temas universais de autodescoberta e superação de desafios. Os jogadores, ao controlarem Ryu, encontram uma figura que personifica a busca pela excelência pessoal, o que contribui para uma forte identificação.

Ken, por outro lado, representa uma abordagem mais extrovertida e competitiva. Sua amizade e rivalidade com Ryu são aspectos cruciais de sua caracterização. A história de Ken destaca a importância das relações interpessoais e da superação de diferenças, temas que muitos jogadores podem compreender e apreciar.

A identificação com Ryu e Ken vai além das habilidades de luta; ela reside na construção de personagens que encapsulam aspirações e dilemas humanos. Os jogadores se conectam não apenas por meio dos movimentos especiais ou estratégias de combate, mas também através das narrativas pessoais e valores que esses personagens incorporam.

A identificação do personagem não é apenas uma ferramenta narrativa; é uma estratégia de engajamento. Quando

os jogadores conseguem se ver nos personagens, a imersão se aprofunda, tornando a experiência do jogo mais envolvente e memorável.

Portanto, a identificação do personagem desempenha um papel crucial no processo de criação de jogos digitais, transformando uma simples representação gráfica em uma experiência emocionalmente rica. Ao compreender e incorporar elementos que ressoam com a experiência humana, os desenvolvedores podem criar personagens que não apenas habitam mundos virtuais, mas que também deixam uma marca duradoura na mente e no coração dos jogadores.

1.5.3 Arquétipos de Personagem em Jogos

Os arquétipos de personagem são modelos ou padrões recorrentes de personalidades que são usados como base para criar personagens em jogos. Esses arquétipos ajudam a definir as características, comportamentos e papéis dos personagens de uma forma que os jogadores possam compreender facilmente. A criação de personagens em jogos é uma arte que combina elementos de narrativa, psicologia e design. Ao entender os arquétipos, você ganha uma estrutura sólida. Aqui estão alguns arquétipos comuns em jogos:

- **Herói:**
  Características: Coragem, altruísmo, habilidades excepcionais, senso de justiça. O herói muitas vezes enfrenta desafios, sacrificações e adversidades para salvar o mundo ou resgatar alguém.
  Exemplo: Link (The Legend of Zelda), Mario (Super Mario Bros.).
- **Anti-herói:**
  Características: Moral ambígua, métodos pouco convencionais, motivações egoístas. O anti-herói pode quebrar as regras tradicionais e questionar a moralidade. Exemplo: Kratos (God of War), Geralt of Rivia (The Witcher).
- **Mentor:**
  Características: Sabedoria, experiência, papel de guia para o protagonista. O mentor ajuda a desenvolver as habilidades do protagonista e fornece orientação. Exemplo: Dumbledore (Harry Potter), Master Yoda (Star Wars).
- **Cômico/Alívio Cômico:**
  Características: Humor, desajeitado, quebra a tensão. Este arquétipo proporciona momentos leves e divertidos na narrativa. Exemplo: Claptrap (Borderlands), Daxter (Jak and Daxter).
- **Donzela em Perigo:**
  Características: Vulnerabilidade, necessidade de resgate. Este arquétipo muitas vezes motiva o herói a agir. Exemplo: Princesa Peach (Super Mario Bros.), Princess Zelda (The Legend of Zelda).
- **Vilão:**
  Características: Motivações malignas, antagonista principal. O vilão cria conflito e desafios para o herói. Exemplo: Bowser (Super Mario Bros.), Ganondorf (The Legend of Zelda).
- **Arquétipo Relutante:**
  Características: Inicialmente relutante em assumir responsabilidades ou o papel de herói. Pode resistir ao chamado à aventura. Exemplo: Frodo (Senhor dos Anéis), Neo (Matrix).
- **Explorador/Aventureiro:**
  Características: Curiosidade, busca por novas terras e desafios. Este arquétipo está sempre em busca de descobertas. Exemplo: Lara Croft (Tomb Raider), Nathan Drake (Uncharted).
- **Guerreiro:**
  Características: Habilidade em combate, força física. O guerreiro geralmente é um especialista em lidar com ameaças físicas. Exemplo: Kratos (God of War), Marcus Fenix (Gears of War).
- **Ladrão/Assassino:**
  Características: Furtividade, habilidades de assassinato, motivações próprias. Este arquétipo opera nas sombras. Exemplo: Ezio Auditore (Assassin's Creed), Garrett (Thief).

1.5.4 Documentação de Personagens em Jogos Digitais: Elementos Essenciais

A documentação de personagens em jogos digitais desempenha um papel crítico no desenvolvimento de narrativas envolventes e experiências de jogos memoráveis. Esta prática não se limita apenas à descrição visual ou à atribuição de habilidades; ela serve como um guia abrangente que delineia cada faceta do personagem. Aqui, exploraremos os elementos importantes a serem documentados na criação de personagens para jogos digitais.

1. **Perfil do Personagem:** Nome: Identificação única do personagem. História Pessoal: Origens, eventos-chave e experiências que moldaram o personagem.
    - Arquétipo: Descreva o papel geral que o personagem desempenha, como "herói", "mentor", "anti-herói", "arquivilão", entre outros.
    - Traços do Arquétipo: Liste os traços de personalidade associados ao arquétipo do personagem. Por exemplo, se o personagem é um "herói", pode ser corajoso, altruísta e motivado por um senso de justiça.
    - Exemplos de Arquétipo: Ilustre como esse personagem se encaixa no contexto de arquétipos mais amplos, fornecendo exemplos específicos de outros personagens semelhantes em diferentes mídias.
    - Personalidade: Características comportamentais e traços psicológicos.
    - Motivações e Objetivos: Razões subjacentes às ações do personagem.
    - Arco de Desenvolvimento: Evolução do personagem ao longo da narrativa.
2. **Características Visuais:**
   Design Geral: Descrição visual do personagem, incluindo aparência, roupas e acessórios.
   Expressões Faciais e Postura: Detalhes sobre como o personagem expressa emoções e sua postura característica.
3. **Habilidades e Poderes:**
   Habilidades Básicas: Habilidades inerentes ao personagem.
   Evolução de Habilidades: Como as habilidades se desenvolvem ao longo do jogo.
   Fraquezas e Limitações: Restrições ou pontos fracos que o personagem possui.
4. **Relacionamentos e Conexões:**
   Ligações com Outros Personagens: Interações específicas com outros elementos da história.
   Desenvolvimento de Relacionamentos: Como os relacionamentos do personagem evoluem.
5. **Diálogos e Falas:** Fala Distinta: Estilo de linguagem e padrões de fala.
   Contexto para Diálogos: Situações ou eventos que acionam falas específicas.
6. Participação na Narrativa: Papel na História: Importância do personagem para o enredo global. Momentos-Chave: Participação em eventos cruciais da narrativa.
7. Comportamento em Jogabilidade: Inteligência Artificial (IA): Comportamento do personagem controlado pela IA. Reações a Decisões do Jogador: Como o personagem responde às escolhas dos jogadores.
8. Estilo de Luta (se aplicável): Técnicas e Movimentos: Descrição das habilidades de combate. Estratégias e Fraquezas em Combate: Como o personagem lida em situações de luta.

A documentação abrangente desses elementos cria uma base sólida para o desenvolvimento do personagem, garantindo consistência, coesão e autenticidade ao longo da experiência de jogo. Essa prática não apenas orienta os desenvolvedores, mas também fornece uma referência valiosa para garantir que a visão inicial do personagem seja preservada durante todo o processo de criação.

1.6 Resolução do meu jogo - Qual escolher?

A resolução ideal para desenvolver jogos digitais pode variar dependendo de vários fatores, incluindo o público-alvo, a plataforma de destino e os recursos técnicos disponíveis. No entanto, existem algumas diretrizes gerais que podem ajudar na escolha da resolução adequada.

Além disso, é importante considerar que os jogos podem precisar ser otimizados para diferentes resoluções, dependendo das plataformas de destino. Por exemplo, jogos para dispositivos móveis podem ser desenvolvidos com resoluções mais baixas, como 720p, para garantir um desempenho suave e economia de energia.

- **Público-alvo:** É importante considerar as preferências e os dispositivos usados pelo público-alvo do jogo. Se o jogo for direcionado para dispositivos móveis, por exemplo, é necessário levar em conta as diferentes resoluções de tela dos smartphones e tablets mais populares.
- **Plataforma de destino:** Se você está desenvolvendo um jogo para uma plataforma específica, como PC, consoles ou dispositivos móveis, é recomendado pesquisar as resoluções de tela comuns para essa plataforma. Isso pode ajudar a otimizar o jogo para uma experiência visual consistente.
- **Escalabilidade:** É sempre bom projetar o jogo de forma escalável, permitindo que ele se adapte a diferentes resoluções de tela. Isso pode ser feito usando técnicas como design responsivo ou permitindo que os jogadores ajustem as configurações de resolução no jogo.
- **Recursos técnicos:** Leve em consideração a capacidade do seu hardware e o desempenho que você deseja alcançar. Quanto maior a resolução, mais recursos de processamento e gráficos serão necessários. Certifique-se de que o hardware alvo seja capaz de lidar com a resolução escolhida sem sacrificar a jogabilidade ou o desempenho. Portanto, se você está desenvolvendo um jogo para dispositivos móveis ou PCs com especificações mais modestas, pode ser necessário optar por uma resolução mais baixa para garantir um desempenho suave e uma boa taxa de quadros (FPS).
- **Escolha uma resolução que seja adequada para o dispositivo de destino:** Se você está desenvolvendo um jogo para um dispositivo específico, como um console de videogame, é importante considerar a resolução suportada por esse dispositivo. Por exemplo, muitos consoles modernos suportam resoluções de 1080p (Full HD) ou até mesmo 4K. Certifique-se de que seu jogo seja capaz de renderizar adequadamente nessas resoluções almejadas.
- **Equilibre a qualidade visual com a acessibilidade:** Uma resolução mais alta geralmente resulta em gráficos mais nítidos e detalhados, mas também requer atenção especial para garantir que o jogo seja acessível em uma variedade de dispositivos e tamanhos de tela. Certifique-se de testar e ajustar seu jogo em diferentes resoluções para garantir que ele seja jogável e visualmente agradável em uma ampla gama de configurações.
- **Considere o design da interface do usuário:** A resolução escolhida também afetará o design da interface do usuário (UI). Resoluções mais altas podem permitir layouts mais complexos e exibição de mais informações, enquanto resoluções mais baixas podem exigir designs mais simplificados e ícones maiores para garantir a legibilidade.
- **Adapte-se às diferentes plataformas:** Se você planeja lançar seu jogo em várias plataformas, como consoles, PCs e dispositivos móveis, é importante considerar as diferentes resoluções suportadas por cada plataforma e otimizar seu jogo para se adequar a cada uma delas. Isso pode envolver ajustes nos gráficos, UI e recursos visuais para garantir que seu jogo seja exibido corretamente em cada plataforma.

Em geral, as resoluções mais comuns para jogos atualmente são 1920x1080 pixels (Full HD) e 2560x1440 pixels (Quad HD). No entanto, é importante adaptar a resolução com base nas considerações acima mencionadas. Alguns jogos podem até ter suporte para resoluções mais altas, como 3840x2160 pixels (4K), para proporcionar uma experiência visual ainda mais imersiva em dispositivos compatíveis. No entanto, com o avanço da tecnologia, resoluções mais altas estão se tornando cada vez mais comuns. Por exemplo, a resolução 4K (3840x2160 pixels) está se tornando mais popular e pode oferecer uma experiência visual ainda mais imersiva para jogadores com monitores ou TVs compatíveis.

No entanto, aqui estão algumas sugestões comuns para resoluções de jogos em 2D:

- **320x160, 640x360 ou 854x480:** Essas resoluções são adequadas para jogos com uma estética mais retrô ou pixel art, onde os gráficos são mais simplificados.
- **1280x720:** Essa é uma resolução mais padrão para jogos 2D, oferecendo uma boa combinação de detalhes visuais e desempenho.

- **1920x1080:** Essa é uma resolução mais alta e é adequada se você quiser que o jogo tenha uma aparência mais nítida e detalhada. No entanto, tenha em mente que pode exigir mais recursos do sistema e afetar o desempenho em dispositivos mais fracos.

Lembre-se de levar em consideração as limitações de desempenho do hardware e do público-alvo do seu jogo ao decidir a resolução. Além disso, certifique-se de que os elementos do jogo, como plataformas e inimigos, sejam adequadamente dimensionados e visíveis na resolução escolhida. Experimente diferentes resoluções durante o desenvolvimento para encontrar a que melhor se adequada ao seu jogo e ao estilo visual que você deseja alcançar.

1.6.1 Escolhendo Resoluções e Dimensões de Sprites em Jogos Retro

Em jogos retro, a escolha da resolução e das dimensões dos sprites (imagens dos personagens, objetos, entre outros) é crucial para alcançar o estilo visual desejado. As resoluções 640x360 e 854x480 são comumente escolhidas por serem escalas 16:9, o que se encaixa bem com os monitores widescreen modernos. A relação de aspecto 16:9 significa que a largura é 16 unidades para cada 9 unidades de altura. Neste tópico, exploraremos as considerações essenciais ao trabalhar com resoluções populares e discutiremos as melhores práticas para dimensionar sprites em proporções clássicas, como 16x16, 32x32 e 64x64.

Antes de continuar, as proporções de sprites 16x16, 32x32 e 64x64 tornaram-se clássicas em jogos retro por uma série de razões históricas, práticas e estéticas. Aqui estão alguns motivos para a popularidade dessas dimensões:

- **Restrições Técnicas:** Em muitos sistemas e consoles antigos, os recursos de hardware eram limitados, incluindo a capacidade de armazenar e processar gráficos. Dimensões menores, como 16x16, eram eficientes em termos de memória e poder de processamento.
- **Grade Regular:** Essas dimensões se encaixam bem em grades regulares, facilitando o design de níveis e ambientes. Uma grade regular simplifica a criação e posicionamento de elementos no jogo.
- **Facilidade de Design:** Dimensões como 16x16 oferecem uma escala compacta que é fácil de trabalhar e permite a representação clara de personagens e objetos em um espaço relativamente pequeno.
- **Estilo Pixel Art:** O estilo visual de pixel art, com seus detalhes limitados e ênfase na simplicidade, é comumente associado a jogos retro. Essas dimensões são ideais para criar arte pixelada devido à sua simplicidade e facilidade de manipulação.
- **Compatibilidade com Múltiplos de 8(8 bits):** Múltiplos de 8 são frequentemente preferidos em design de jogos porque são compatíveis com estruturas de dados e operações de renderização eficientes em termos de desempenho.
- **Herança Cultural:** Muitos dos jogos clássicos que definiram a era retro, como os jogos de 8 bits e 16 bits, adotaram essas dimensões. Como resultado, elas se tornaram parte integrante da estética associada a jogos retro.
- **Padrões de Indústria:** Ao longo do tempo, essas dimensões tornaram-se padrões de fato na indústria de jogos retro. Desenvolvedores e artistas adotaram essas proporções para criar uma experiência visual consistente e familiar para os jogadores.

Embora essas dimensões sejam consideradas clássicas, é importante observar que não há regras restritas. Os desenvolvedores têm liberdade para ajustar as dimensões conforme necessário para atender às demandas específicas de seus jogos e à evolução das tecnologias modernas. A tabela 1.1 lista exemplos de várias resoluções.

Na tabela 1.1 temos a coluna proporção. As proporções de tela, também conhecidas como relações de aspecto, representam a relação entre a largura e a altura de uma tela ou imagem. Elas são expressas como uma razão entre essas duas dimensões. Por exemplo, uma proporção de tela de 4:3 significa que a largura é 4 unidades e a altura é 3 unidades.

Tabela 1.1: Informações de Resoluções

Resolução	Dimensões (px)	Proporção	Total de Pixels
QVGA	320x240	4:3	76,800
VGA	640x480	4:3	307,200
WVGA	800x480	5:3	384,000
SVGA	800x600	4:3	480,000
FWVGA	854x480	16:9	409,920
XGA	1024x768	4:3	786,432
HD	640x360	16:9	230,400
HD 720p	1280x720	16:9	921,600
HD+ 900p	1600x900	16:9	1,440,000
Full HD 1080p	1920x1080	16:9	2,073,600
WUXGA	1920x1200	16:10	2,304,000
WXGA	1280x800	16:10	1,024,000
SXGA	1280x1024	5:4	1,310,720
WXGA+	1440x900	16:10	1,296,000
UXGA	1600x1200	4:3	1,920,000
Full HD 1080p	1920x1080	16:9	2,073,600
WUXGA	1920x1200	16:10	2,304,000
QWXGA	2048x1152	16:9	2,359,296
2K	2048x1080	17:9	2,211,840
WQXGA	2560x1600	16:10	4,096,000
2.5K	2560x1440	16:9	3,686,400
UHD 4K	3840x2160	16:9	8,294,400

As proporções de tela são importantes porque influenciam a aparência visual de uma imagem ou vídeo em um dispositivo. Diferentes proporções de tela são adequadas para diferentes finalidades e tipos de conteúdo.

A proporção de tela é calculada pela relação entre a largura e a altura de uma tela ou imagem. A fórmula básica para calcular a proporção é:

$$\text{proporcao} = \frac{largura}{altura} \tag{1.1}$$

Essa fórmula fornece a razão entre a largura e a altura da tela. O resultado é frequentemente expresso como uma proporção simplificada, por exemplo, 4:3, 16:9, entre outras. Suponha que você tenha uma tela com largura de 1920 pixels e altura de 1080 pixels:

$$\text{proporcao} = \frac{1920}{1080} = 1.7778 \tag{1.2}$$

O arredondamento para a proporção de tela 16:9 geralmente envolve a simplificação da fração resultante após a divisão da largura pela altura. Neste caso, devemos encontrar uma fração cujo valor decimal seja próximo ao valor decimal da proporção original. Por exemplo:

- Para arredondar para 16:9, procure uma fração equivalente cujo valor decimal seja próximo a 1.7778. O resultado final da fração 1.3 é uma representação aproximada da proporção 1.7778, o que é comumente considerado como a proporção de tela 16:9. Isso é feito para simplificar a representação da proporção sem perder muita precisão.

$$\frac{16}{9} = 1.7778 \tag{1.3}$$

Grade Regular

A "Grade Regular"refere-se a uma estrutura de organização espacial em que elementos gráficos, como sprites e tiles, são posicionados em uma grade consistente e regular. Essa abordagem oferece vários benefícios no desenvolvimento de jogos, especialmente em termos de design de níveis, eficiência e coesão visual. A figura 1.8 ilustra um exemplo na engine GDevelop 5 utilizando o recurso de grade para alinhar os objetos na cena.

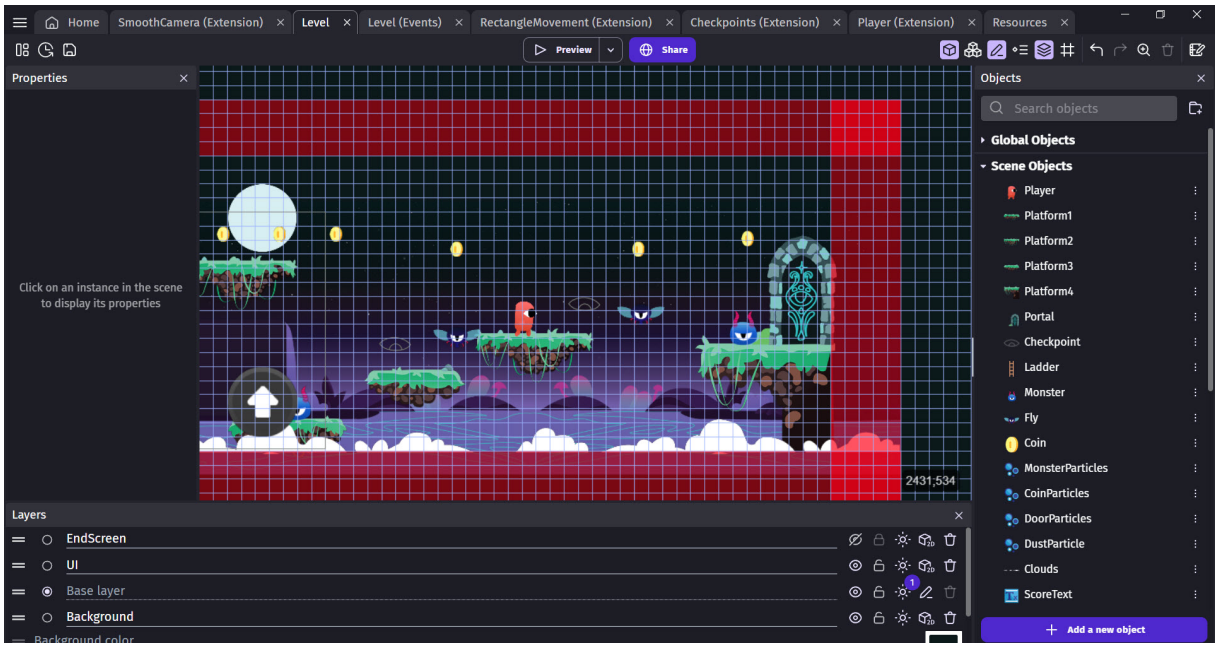


Figura 1.3: Uso de grade na engine GDevelop 5.

Aqui estão alguns detalhes sobre a grade regular:

- **Organização Sistemática:** Em uma grade regular, o espaço do jogo é dividido em células retangulares ou quadradas de tamanho uniforme. Cada célula representa uma unidade de medida na grade, tornando a organização do mundo do jogo mais sistemática e previsível.
- **Facilita o Design de Níveis:** Uma grade regular simplifica o processo de design de níveis, pois os desenvolvedores podem posicionar facilmente elementos, como plataformas, obstáculos e itens, em uma grade que segue um padrão consistente. Isso facilita a criação de ambientes jogáveis e a construção de níveis coesos.
- **Alinhamento Preciso:** A grade regular facilita o alinhamento preciso de elementos gráficos. Isso é particularmente útil ao trabalhar com sprites e tiles, pois ajuda a evitar desalinhamentos visuais que podem ocorrer quando os elementos não seguem um padrão regular.
- **Simplicidade e Clareza:** Uma grade regular contribui para a simplicidade visual do jogo, o que é especialmente importante em jogos retro. A clareza na disposição dos elementos na grade pode melhorar a compreensão do jogador sobre o layout do mundo do jogo.
- **Coerência Visual:** Ao seguir uma grade regular, os elementos visuais mantêm uma coerência visual em relação uns aos outros. Isso é essencial para criar uma estética consistente em todo o jogo.
- **Facilidade no Design:** Ao dividir o espaço de jogo em células regulares, os desenvolvedores podem facilmente calcular distâncias, definir limites e criar padrões de movimento para personagens. Isso contribui para um processo de design mais eficiente.
- **Adaptação a Estilos Visuais Retro:** Muitos jogos retro adotam a grade regular como parte integrante de seu estilo visual. O retorno a esse método de design é influenciado pela nostalgia associada aos jogos clássicos que usavam esse tipo de abordagem.

1.6.2 Exemplos de Braingstorming

O brainstorming, ou tempestade de ideias, é uma prática fundamental no desenvolvimento de jogos digitais. Trata-se de uma abordagem colaborativa e criativa para gerar conceitos, soluções e ideias inovadoras que impulsionarão o processo de design do jogo.

Nesta seção veremos alguns exemplos de braingstorming.

Escape from the Lost Temple

"Escape from the Lost Temple"é um jogo de plataforma 2D no estilo de aventura e ação, onde o jogador assume o papel de um explorador que deve escapar de um antigo templo cheio de armadilhas mortais, inimigos e enigmas desafiadores. O objetivo é encontrar o tesouro escondido no coração do templo e escapar com vida.

Mecânicas de jogo:

- Plataformas e movimento: O jogador pode pular, correr e se agachar para navegar pelo templo, superar obstáculos e evitar armadilhas.
- Combate: O jogador pode enfrentar inimigos com ataques corpo a corpo ou utilizando armas que encontrar pelo caminho.
- Quebra-cabeças: O templo está repleto de quebra-cabeças, como blocos que precisam ser empurrados, alavancas que ativam mecanismos e portas trancadas que requerem chaves.
- Coleta de itens: O jogador pode coletar moedas e outros itens espalhados pelo templo, que podem ser usados para comprar melhorias e itens especiais em lojas dentro do jogo.
- Armadilhas: O templo está cheio de armadilhas mortais, como lanças, buracos, flechas e espinhos. O jogador deve evitá-las habilmente para sobreviver.

Recursos necessários:

- Assets gráficos: Para criar o jogo, você precisará de sprites para o personagem principal, inimigos, armadilhas, tesouro, plataformas, objetos interativos, fundos e outros elementos visuais. Você pode encontrar assets gratuitos em sites como o OpenGameArt (https://opengameart.org/) e o Kenney.nl (https://kenney.nl/assets).
- Música e efeitos sonoros: Para adicionar áudio ao jogo, você pode procurar música e efeitos sonoros gratuitos em sites como o Freesound (https://freesound.org/) e o Incompetech (https://incompetech.com/).
- Interface do usuário: Será necessário criar uma interface de usuário simples e intuitiva, incluindo menus, barras de vida, pontuação e outros elementos de HUD (Heads-Up Display).

Lembre-se de verificar as licenças dos assets que você escolher para garantir que sejam compatíveis com o seu projeto.

Pixel Quest

Gênero: Plataforma / Aventura

Resolução Ideal: 640x480 pixels

Tamanho dos Cenários: Os cenários podem ser divididos em telas individuais, com tamanho de 320x240 pixels cada, para criar um estilo retrô e simplificar o desenvolvimento.

Mecânica do Jogo:

O jogador controla um personagem em um mundo de fantasia em 2D. O objetivo principal é coletar itens mágicos perdidos e restaurar o equilíbrio ao reinos encantados. O jogo será baseado em fases, cada uma com um desafio único. O personagem pode pular, correr, atacar com espada e usar habilidades mágicas. Haverá inimigos e obstáculos que o jogador deve evitar ou derrotar para progredir. O jogo pode apresentar quebra-cabeças simples para adicionar variedade.

Elementos do Jogo:

- Personagem principal: Um herói com habilidades de espada e magia.
- Inimigos: Monstros, criaturas mágicas e outros adversários.
- Itens: Poções, pergaminhos, chaves e outros itens mágicos.
- Ambientes: Florestas encantadas, cavernas escuras, montanhas nevadas e castelos misteriosos.
- Chefes: Chefes desafiadores em cada fase para aumentar a dificuldade.
- NPCs: Personagens não jogáveis que podem fornecer informações ou missões secundárias.
- Assets e Recursos: Você pode encontrar assets para criar o jogo em pixel art em diferentes sites, como: OpenGameArt (https://opengameart.org/): Uma plataforma com uma vasta coleção de assets gratuitos. Kenney (https://kenney.nl/assets): Oferece uma variedade de pacotes de assets gratuitos para jogos em pixel art. Pixel Game Art (https://pixelgameart.org/web/index.html): Um recurso com muitos assets pagos, mas também oferece alguns gratuitos. Lembre-se de verificar as licenças dos assets e garantir que estejam em conformidade com o uso comercial.

Exploração

- Gênero: Plataforma de ação em 2D.
- Tema: Exploração de uma masmorra misteriosa.
- Objetivo: Encontrar tesouros escondidos e derrotar chefes para progredir para níveis mais profundos da masmorra.
- Resolução ideal: Recomenda-se uma resolução de 320x240 pixels para uma estética retrô de pixel art. No entanto, você pode ajustar essa resolução conforme sua preferência.
- Tamanho dos cenários: Cada cenário pode ter uma largura de 20-25 telas de largura (6400-8000 pixels) e uma altura de 15-20 telas (4800-6400 pixels). Isso permitirá uma exploração significativa e níveis desafiadores. Elementos que podem ser incluídos:
- Personagem jogável: Um aventureiro ágil com habilidades de salto, ataque básico e habilidade especial.
- Inimigos: Criaturas perigosas, como esqueletos, aranhas venenosas, morcegos, etc.
- Chefes: Inimigos poderosos que requerem estratégia para derrotar.
- Armadilhas: Espinhos, buracos, flechas, entre outros, que podem causar dano ao jogador.
- Tesouros e power-ups: Itens colecionáveis que podem aumentar a vida, dar habilidades especiais temporárias ou melhorar as habilidades do jogador.
- Quebra-cabeças: Mecanismos e enigmas simples que o jogador precisa resolver para avançar.
- NPCs: Personagens não jogáveis que podem fornecer dicas, missões secundárias ou itens. Portais e checkpoints: Pontos de salvamento e portais de teletransporte para facilitar a progressão do jogador.
- Assets: Existem vários recursos disponíveis online para criar jogos em pixel art. Alguns sites populares para encontrar assets de pixel art incluem: itch.io: https://itch.io/game-assets/tag-pixel-art OpenGameArt: https://opengameart.org/art-search-advanced?keys=pixel Kenney.nl: https://kenney.nl/assets?q=pixel Level design:

Planeje a progressão dos níveis, aumentando gradualmente a dificuldade e introduzindo novos elementos de jogabilidade. Crie níveis com diferentes layouts, com desafios como plataformas móveis, áreas escuras, labirintos,

etc. Certifique-se de balancear a distribuição de inimigos, armadilhas e tesouros para manter o jogo desafiador, mas justo. Considere a adição de áreas secretas ou rotas alternativas para recompensar jogadores exploradores. Faça iterações e ajustes no level design para garantir uma curva de dificuldade adequada e um fluxo de jogo satisfatório. Lembre-se de que este é apenas um brainstorming inicial e você pode adaptar e expandir essas ideias de acordo com sua visão e habilidades de desenvolvimento.

1.7 Pixel Art e Imagens Vetorizadas em Jogos Digitais

1.7.1 Pixel Art

Pixel art é uma forma de arte digital que utiliza pixels, os pontos individuais que compõem uma imagem, como blocos fundamentais. Cada pixel é cuidadosamente colocado para criar uma composição visual. Esse estilo artístico é frequentemente associado a jogos retro e indie, e destaca-se por sua estética nostálgica e limitações deliberadas de resolução.

Técnicas de Criação

- **Resolução Limitada:** Em pixel art, a resolução é geralmente baixa, e os artistas trabalham com uma paleta de cores restrita para enfatizar a simplicidade e o estilo distintivo.
- **Trabalho Manual:** Cada pixel é desenhado manualmente, permitindo um controle preciso sobre cada detalhe da imagem.
- **Animação Simplificada:** Muitos jogos retro utilizam animações em pixel art, aproveitando a facilidade de criar sequências de imagens pequenas e detalhadas.

(a) Super Mario World

(b) Dead Cells

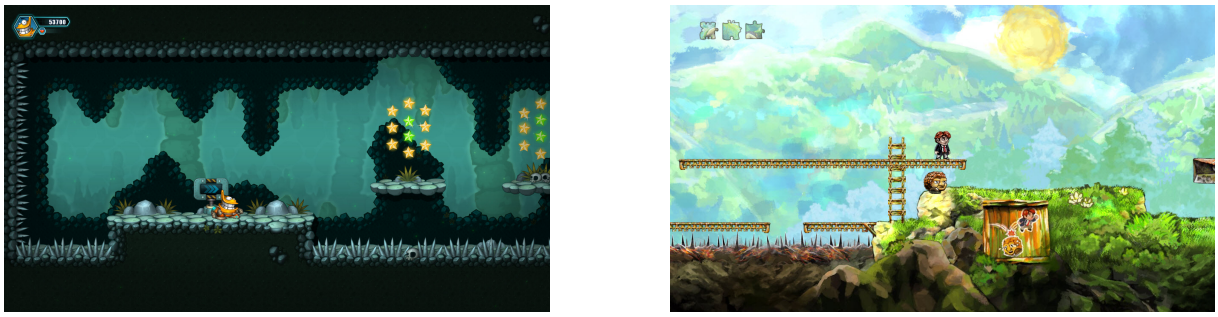
Figura 1.4: Exemplos de jogos em pixel art.

1.7.2 Imagens Vetorizadas

Imagens vetorizadas são criadas usando gráficos vetoriais, onde as formas são representadas por equações matemáticas. Ao contrário de pixel art, as imagens vetoriais são independentes de resolução, escalando perfeitamente sem perder qualidade. Este estilo é comumente usado em design gráfico e ilustrações.

Técnicas de Criação

- **Escala Infinita:** Imagens vetoriais podem ser escaladas indefinidamente sem perder detalhes, tornando-as ideais para logotipos, ícones e outros elementos que precisam ser redimensionados.
- **Edição Flexível:** Os artistas podem facilmente editar e manipular formas, cores e tamanhos sem perder qualidade.



(a) Oozi: Earth Adventure (b) Braid

Figura 1.5: Exemplos de jogos que utilizam imagens vetoriais

Jogos em 3D

Jogos em 3D representam um avanço significativo na indústria de jogos digitais, trazendo ambientes tridimensionais que oferecem uma experiência mais imersiva. Ao contrário de pixel art e imagens vetorizadas, os jogos em 3D buscam simular a realidade, proporcionando ao jogador a sensação de profundidade e perspectiva.

Técnicas de Criação

- **Modelagem Tridimensional:** Os objetos e ambientes são criados em três dimensões, utilizando modelos 3D para representar personagens, cenários e objetos.
- **Texturização e Iluminação:** Texturas realistas e técnicas avançadas de iluminação são aplicadas para criar ambientes visualmente ricos e detalhados.
- **Animação Complexa:** A animação em jogos 3D permite movimentos mais fluidos e realistas dos personagens, contribuindo para uma experiência de jogo mais convincente.

(a) Forza Horizon 4 (b) GTA V

Figura 1.6: Exemplos de jogos em 3D

Utilização em Jogos Digitais

Variedade de Gêneros

Jogos em 3D são amplamente utilizados em uma variedade de gêneros, desde aventuras épicas até simulações realistas. A capacidade de explorar ambientes tridimensionais abre possibilidades criativas para desenvolvedores de jogos.

Experiência Imersiva

A capacidade de oferecer uma experiência mais imersiva faz dos jogos em 3D uma escolha popular para títulos de última geração. A sensação de presença e interação direta com o ambiente contribui para a narrativa e a jogabilidade.

Integração com Pixel Art e Imagens Vetorizadas

Embora jogos em 3D representem uma evolução técnica, é possível integrar elementos de pixel art e imagens vetorizadas em certos contextos, como interfaces de usuário, ícones e elementos estilizados. Contudo, é crucial manter a coesão visual para garantir uma experiência de jogo consistente.

(a) Fez (b) Minecraft

Figura 1.7: Exemplos de jogos em 3D com pixel art

1.8 Aperfeiçoando a Jogabilidade com Técnicas Avançadas

1.8.1 Better Jump: Melhorando a Responsividade dos Saltos

A técnica do "Better Jump"visa tornar os saltos dos personagens mais responsivos e precisos. Isso é alcançado ao ajustar a física do salto para que ele tenha um controle mais suave no ar, permitindo ao jogador maior controle sobre a altura e a distância do salto.

Para implementar isso, é preciso ajustar parâmetros como a força aplicada ao salto, a gravidade enquanto o personagem está no ar e a velocidade máxima vertical que o personagem pode atingir.

1.8.2 Coyote Time: Ampliando a Oportunidade para Saltos

O "Coyote Time"é uma homenagem ao famoso desenho animado dos Looney Tunes, onde o coiote, conhecido por suas tentativas malucas de capturar o Papa-Léguas, muitas vezes desafia temporariamente a gravidade. Esta mecânica é sobre dar ao jogador uma pequena margem de erro ao cronometrar seus pulos ao correr na borda de uma plataforma. Muitas vezes, o jogador espera até o último momento possível para pular, para poder cobrir a maior distância se possível, mas frequentemente eles acabam sendo pulando alguns frames atrasados. Quando o personagem deixa de tocar o chão e entra no estado de queda, ele não pode mais realizar um pulo imediato e corre o risco de desperdiçar o segundo pulo caso o jogador tenha pressionado o botão de salto muito tarde. Em vez de penalizar os jogadores por esse timing menor, podemos configurar nosso sistema para detectar um comando de salto logo após a transição para o estado de queda e permitir que o jogador execute o salto mesmo que execute o comando de pulo no momento errado.

Podemos fazer isso simplesmente definindo um temporizador quando eles entram no estado de queda e ouvindo por um evento de pulo. Desde que o pulo ocorra antes que este temporizador acabe, permitimos que o jogador acione o estado de pulo. Caso contrário, os fazemos cair.

Em resumo, O "Coyote Time"é uma técnica que concede um pequeno período de tempo após o jogador sair de uma plataforma para realizar um salto. Isso ajuda a evitar a frustração de quedas rápidas e imprecisas, dando uma margem de erro mais generosa ao jogador.

A implementação dessa técnica envolve adicionar um temporizador que permite que o jogador execute um salto mesmo após ter saído da plataforma por um curto período de tempo.

1.8.3 Jump Buffer: Registrando Saltos com Precisão

Um problema comum em jogos de plataforma ocorre quando o jogador está no ar, prestes a cair no chão, e imediatamente quer pular novamente. Eles pressionam o botão de pulo, mas estão realmente um ou dois frames antes de passar por qualquer verificação de colisão com o chão e acabam apenas pousando no chão em vez de pular. Com o "jump buffering"(bufferização de pulo), mantemos o comando de pulo por alguns frames e pulamos automaticamente para eles se eles caiam no chão dentro do tempo permitido, ajudando a preencher a espaço de tempo entre a ação de pulo do jogador e a precisão milimétrica necessária para realmente executar esse movimento.

Podemos realizar isso simplesmente ouvindo os comandos de pulo enquanto estivermos caindo e iniciando um temporizador. Se o jogador cair no chão antes que o temporizador acabe, iniciamos o comando de pulo automaticamente.

O "Jump Buffer"é uma técnica que permite que o jogo registre um salto pressionado pelo jogador antes que o personagem realmente toque o chão. Isso é útil para garantir que os saltos sejam realizados com precisão, mesmo que o jogador pressione o botão de salto um pouco antes de atingir o solo.

Para implementar isso, é necessário criar um buffer que armazene o comando de salto por um curto período de tempo e o execute assim que o personagem tocar o chão.

1.8.4 Custom Jump: Criando Saltos Personalizados

A mecânica de "Custom Jump"oferece ao jogador a capacidade de variar a altura ou a potência do salto com base em diferentes inputs ou condições no jogo. Isso pode incluir a habilidade de executar um salto duplo, triplo ou até mesmo ajustar a direção do salto.

Figura 1.8: Pressionando o botão de pulo("A") antes de cair no chão.[1]

Para implementar um "Custom Jump", é preciso criar lógicas específicas que permitam ao jogador modificar o comportamento padrão do salto com base em ações como pressionar o botão de salto novamente durante o salto, coletar itens no jogo que afetam a habilidade de salto, entre outras possibilidades.

1.8.5 Double Jump: Realizando Pulos Duplos

O Double Jump (pulo duplo) é uma mecânica comum em jogos de plataforma que permite aos jogadores realizar um segundo salto no ar após o primeiro salto. Essa mecânica adiciona uma camada extra de mobilidade e estratégia ao jogo, permitindo que os jogadores alcancem áreas mais altas, evitem obstáculos ou até mesmo ataquem inimigos de maneiras diferentes.

A implementação do Double Jump envolve o seguinte:

1. Condição de Ativação: O Double Jump geralmente é ativado quando o jogador está no ar e pressiona o botão de salto pela segunda vez. A mecânica requer uma verificação para garantir que o jogador esteja no ar e que já tenha realizado o primeiro salto antes de permitir o segundo.
2. Consumo de Recurso: Em alguns jogos, o Double Jump pode consumir um recurso, como uma quantidade de energia, que precisa ser regenerada ou coletada antes que o jogador possa realizar outro Double Jump. Isso adiciona um elemento estratégico ao uso do Double Jump, pois os jogadores precisam gerenciar seus recursos para aproveitá-lo ao máximo.
3. Limite de Uso: Alguns jogos impõem um limite no número de Double Jumps que um jogador pode realizar consecutivamente. Por exemplo, um jogador pode ter um Double Jump inicial e, ao coletar certos itens ou alcançar certos marcos no jogo, pode desbloquear Double Jumps adicionais.
4. Aprimoramentos e Variações: Além do Double Jump básico, existem variações e aprimoramentos dessa mecânica, como o Triple Jump (pulo triplo), que permite três saltos consecutivos no ar, ou a capacidade de realizar manobras acrobáticas durante o Double Jump, como saltar na diagonal ou realizar um ataque especial no ar.

O Double Jump é uma mecânica versátil que pode ser implementada de várias maneiras para se adequar ao estilo e à dinâmica de um jogo específico. Ela adiciona profundidade à jogabilidade e permite aos jogadores explorar o ambiente do jogo de maneira mais criativa e dinâmica.

1.8.6 Wall Sliding: Deslizando pela Parede

A mecânica de Wall Sliding (deslize na parede) é uma técnica comum em jogos de plataforma que permite aos jogadores deslizarem verticalmente pelas paredes. Isso não apenas adiciona dinamismo e fluidez ao movimento do personagem, mas também abre oportunidades para navegação estratégica e evasão de obstáculos.

Aqui estão alguns pontos-chave sobre a mecânica de Wall Sliding:

1. Ativação: A Wall Sliding geralmente é ativada quando o jogador pressiona o botão de movimento na direção da parede enquanto está em contato com ela. Isso permite que o jogador "grude"na parede e comece a deslizar para cima ou para baixo.
2. Controle do Deslizamento: Durante o deslizamento na parede, os jogadores podem ter controle sobre a velocidade e a direção, permitindo-lhes ajustar sua posição verticalmente conforme necessário. Isso pode ser alcançado por meio da entrada do jogador (por exemplo, pressionando para cima ou para baixo) ou por meio de mecânicas específicas de deslizamento.
3. Integração com Outras Habilidades: A Wall Sliding frequentemente é combinada com outras habilidades de movimentação, como o Wall Jump (pulo na parede). Isso permite que os jogadores realizem movimentos acrobáticos avançados, como deslizar pela parede, pular para outra parede e continuar deslizando, proporcionando uma experiência de jogo mais dinâmica e envolvente.
4. Tempo Limite: Em alguns jogos, a Wall Sliding pode ter um tempo limite associado. Isso significa que os jogadores só podem deslizar pela parede por um período específico antes de serem forçados a soltar e cair. Essa limitação adiciona um elemento estratégico ao uso da mecânica, incentivando os jogadores a tomar decisões rápidas e eficientes.
5. Design de Níveis: A implementação eficaz da Wall Sliding requer um design de níveis que inclua superfícies verticais adequadas para deslizar. É importante garantir que as paredes tenham uma inclinação e textura adequadas para permitir um deslizamento suave e controlável.

A Wall Sliding é uma mecânica versátil que pode ser usada para criar desafios de plataforma interessantes, permitindo aos jogadores explorar e navegar pelos níveis de maneiras criativas e estratégicas. Ela adiciona profundidade e diversidade ao movimento do personagem, enriquecendo a experiência de jogo como um todo.

1.8.7 Cooldown - Tempo de Espera

Cooldown é um termo comumente utilizado em jogos para se referir ao período de espera entre usos de uma habilidade, item ou ação específica. Quando uma habilidade ou ação é usada, geralmente ela entra em um estado de cooldown, durante o qual não pode ser usada novamente até que o tempo de cooldown seja concluído.

Aqui estão algumas características do cooldown em jogos:

1. Limitação de uso: O cooldown é usado para limitar a frequência com que uma habilidade ou ação pode ser usada. Isso impede que os jogadores spamem habilidades poderosas ou itens sem consequências.
2. Estratégia e planejamento: O cooldown adiciona uma camada de estratégia ao jogo, pois os jogadores precisam considerar o tempo de espera antes de poderem usar uma habilidade novamente. Isso incentiva a tomada de decisões táticas e o planejamento de ataques e defesas.
3. Balanceamento do jogo: O cooldown é uma ferramenta de balanceamento usada pelos desenvolvedores para garantir que habilidades poderosas não sejam abusadas e para promover uma experiência de jogo equilibrada para todos os jogadores.
4. Variação de cooldowns: Diferentes habilidades, itens ou ações podem ter cooldowns diferentes, dependendo de sua natureza e impacto no jogo. Habilidades mais poderosas geralmente têm cooldowns mais longos, enquanto habilidades básicas podem ter cooldowns mais curtos.

Tipos de Cooldown

Existem vários tipos de cooldowns que são comumente usados em jogos para regular o uso de habilidades, itens ou ações. Aqui estão os principais tipos de cooldowns:

1. Cooldown Global: Um cooldown global afeta todas as habilidades ou ações do jogador simultaneamente. Isso significa que, após usar qualquer habilidade ou ação, todas as outras habilidades também entram em cooldown por um período de tempo antes de poderem ser usadas novamente.
2. Cooldown Individual: Um cooldown individual é específico para cada habilidade ou ação. Cada habilidade tem seu próprio cooldown separado, o que significa que usar uma habilidade não afeta o cooldown de outras habilidades. Isso permite que os jogadores usem diferentes habilidades de forma independente.
3. Cooldown de Categoria: Nesse tipo de cooldown, habilidades são agrupadas em categorias e compartilham um cooldown comum dentro da mesma categoria. Por exemplo, todas as habilidades de cura podem ter um cooldown compartilhado, enquanto as habilidades de dano têm um cooldown separado.
4. Cooldown Progressivo: Um cooldown progressivo aumenta a duração do cooldown cada vez que a habilidade é usada. Por exemplo, se uma habilidade tem um cooldown progressivo de 10 segundos, o primeiro uso pode resultar em um cooldown de 10 segundos, o segundo uso em 20 segundos e assim por diante.
5. Cooldown Reduzido por Ação: Alguns jogos têm cooldowns que são reduzidos quando certas ações são realizadas. Por exemplo, realizar uma ação específica pode reduzir o cooldown de uma habilidade em uma certa quantidade de tempo.
6. Cooldown Fixo: Um cooldown fixo é uma duração constante de espera entre usos de uma habilidade. Independentemente de outras variáveis, como nível do personagem ou equipamento, o cooldown permanece o mesmo.
7. Cooldown de Carga: Nesse tipo de cooldown, habilidades têm cargas que são recarregadas ao longo do tempo. Cada carga representa a capacidade de usar a habilidade uma vez. Por exemplo, uma habilidade pode ter três cargas que são recarregadas a cada 10 segundos.

Tempo de espera para deslizar pela parede

O "Wall Sliding Cooldown"refere-se ao tempo de espera entre os momentos em que um personagem pode usar a mecânica de "escorregar na parede"(wall sliding) em um jogo. Essa mecânica é comum em jogos de plataforma e ação, onde permite que os jogadores deslizem pelas paredes para alcançar áreas altas, evitar obstáculos ou realizar movimentos acrobáticos.

Aqui estão algumas características e usos do "Wall Sliding Cooldown":

1. Limitação de uso: O "Wall Sliding Cooldown"é usado para limitar a frequência com que os jogadores podem escorregar pelas paredes. Isso impede que os jogadores abusem da mecânica ao deslizar constantemente pelas paredes sem interrupções.
2. Tempo estratégico: O cooldown do escorregamento na parede adiciona uma camada estratégica ao jogo. Os jogadores precisam escolher o momento certo para usar essa habilidade, considerando o tempo de espera antes de poderem escorregar novamente.
3. Equilíbrio de jogo: O cooldown é uma ferramenta de balanceamento que os desenvolvedores podem usar para garantir que a mecânica de escorregamento na parede não se torne excessivamente dominante ou disruptiva na jogabilidade. Um cooldown adequado ajuda a manter um equilíbrio entre habilidades e movimentos no jogo.
4. Progressão e habilidades: Em alguns casos, o cooldown do escorregamento na parede pode ser afetado por habilidades ou melhorias do personagem. Por exemplo, um upgrade pode reduzir o tempo de cooldown, permitindo que os jogadores usem a mecânica com mais frequência ou de maneira mais eficiente à medida que progridem no jogo.

5. Experiência de jogo dinâmica: Ao usar o "Wall Sliding Cooldown"de forma eficaz, os desenvolvedores podem criar uma experiência de jogo mais dinâmica e envolvente. Os jogadores precisam considerar o tempo de espera e planejar seus movimentos de forma estratégica, o que adiciona profundidade ao gameplay.

Tempo de recarga de dano

A mecânica de "Damage Cooldown"(tempo de recarga de dano) é um elemento comum em jogos de ação e aventura, especialmente em jogos de combate. Ela se refere ao período de tempo que um personagem ou jogador deve esperar após receber dano antes de poder sofrer outro dano.

Aqui estão alguns aspectos importantes sobre a mecânica de Damage Cooldown:

1. Prevenção de Dano Excessivo: O Damage Cooldown é projetado para evitar que um personagem seja prejudicado repetidamente em um curto período de tempo. Ele impõe um intervalo entre os ataques ou fontes de dano, permitindo que o personagem se recupere, reaja e planeje estratégias de defesa ou evasão.
2. Tempo de Vulnerabilidade: Durante o Damage Cooldown, o personagem geralmente está em um estado de vulnerabilidade aumentada, pois não pode sofrer mais danos. Isso pode ser representado visualmente por animações de recuo, invencibilidade temporária ou outras indicações visuais e sonoras.
3. Balanceamento de Jogabilidade: A mecânica de Damage Cooldown é crucial para o balanceamento da jogabilidade, especialmente em jogos de combate multiplayer. Ela evita situações em que um jogador pode abusar de ataques rápidos e contínuos para dominar o combate de forma desequilibrada.
4. Estratégias de Combate: Os jogadores podem usar o conhecimento do Damage Cooldown ao seu favor, planejando ataques e combos estratégicos que tiram proveito dos momentos em que o oponente está vulnerável devido ao tempo de recarga de dano.
5. Personalização e Ajuste: Em muitos jogos, os desenvolvedores oferecem a capacidade de ajustar o tempo de recarga de dano para diferentes personagens, habilidades ou situações. Isso permite uma personalização mais profunda da jogabilidade e uma melhor adaptação ao estilo de jogo dos jogadores.
6. Estratégia e gestão de recursos: Os jogadores precisam gerenciar seus recursos, como tempo de recarga e mana/energia, de forma estratégica. Saber quando usar uma habilidade poderosa ou esperar até que o tempo de recarga termine pode ser crucial para vencer confrontos difíceis.
7. Balanceamento de habilidades: O Damage Cool Down é uma ferramenta vital para equilibrar habilidades poderosas. Habilidades que causam muito dano geralmente têm um tempo de recarga mais longo, enquanto habilidades mais fracas podem ter um tempo de recarga mais curto. Isso incentiva os jogadores a usar uma variedade de habilidades em vez de depender apenas de uma ou duas.

II Parte Dois

2. Introdução à Matemática nos Jogos 2D

2.1 Objetivo

Nesta seção, você aprenderá como a matemática é fundamental para o desenvolvimento de jogos digitais, especialmente jogos 2D. A matemática está presente em diversas mecânicas, como o movimento dos personagens, a física aplicada ao jogo, a detecção de colisões e muitos outros aspectos que garantem a funcionalidade e a diversão dos jogos.

2.1.1 Importância da Matemática nos Jogos

A matemática oferece uma base sólida para a criação de lógicas e regras que governam o comportamento de personagens, objetos e ambientes em um jogo. A precisão nos cálculos matemáticos é crucial para garantir que as mecânicas funcionem corretamente, proporcionando uma experiência fluida para o jogador.

2.2 O Papel da Matemática no Comportamento dos Jogos

2.2.1 Movimentação de Personagens

Um dos exemplos mais básicos da aplicação da matemática nos jogos 2D é o movimento dos personagens. Para mover um objeto na tela, utilizamos coordenadas (x, y) para definir sua posição no espaço 2D.

Exemplo simples: Se um personagem se move em linha reta para a direita, sua posição no eixo x aumenta. Se ele se move para cima, sua posição no eixo y diminui.

A fórmula para movimentação básica pode ser expressa como:

$$\text{nova\_posição} = \text{posição\_atual} + \text{velocidade} \times \text{tempo}$$

Onde:

- **posição_atual**: posição inicial do objeto no eixo x ou y.
- **velocidade**: a taxa de mudança da posição (em pixels por segundo).

- **tempo**: a quantidade de tempo decorrido.

Exemplo Prático: Imagine que você está criando um jogo de plataforma. O jogador pressiona uma tecla para mover o personagem para a direita. Nesse momento, a matemática entra em ação para calcular a nova posição do personagem a cada atualização do quadro (frame).

$$x_{\text{novo}} = x_{\text{atual}} + v \times t$$

Se a velocidade (v) do personagem é 100 pixels por segundo e o tempo (t) entre os quadros é de 0,016 segundos (60 FPS), então o personagem se moverá 1,6 pixels a cada frame.

2.2.2 Rotação e Direção

A rotação de objetos no espaço 2D também é governada por matemática, especificamente por trigonometria. Para rotacionar um objeto em torno de um ponto (normalmente o centro do objeto), usamos as funções seno e cosseno para calcular as novas coordenadas.

A fórmula para rotacionar um ponto (x, y) em torno da origem (0, 0) por um ângulo θ é:

$$x_{\text{novo}} = x \cdot \cos(\theta) - y \cdot \sin(\theta)$$

$$y_{\text{novo}} = x \cdot \sin(\theta) + y \cdot \cos(\theta)$$

Exemplo Prático: Se o jogador mover o mouse para girar um personagem, você pode usar essa fórmula para ajustar o ângulo do personagem em direção ao ponteiro do mouse.

2.2.3 Movimento Circular

O movimento circular de objetos em um espaço 2D é governado por funções trigonométricas, como seno e cosseno. Esse tipo de movimento é útil quando queremos que um objeto se mova em um caminho circular ao redor de um ponto fixo, mantendo um raio constante e uma velocidade controlada.

Para calcular a posição de um objeto ao longo de um círculo, utilizamos a seguinte fórmula, onde $(x_{\text{origem}}, y_{\text{origem}})$ representa o ponto fixo (centro da órbita), r é o raio do círculo, e θ é o ângulo atual de movimento:

$$x_{\text{novo}} = x_{\text{origem}} + r \cdot \cos(\theta)$$

$$y_{\text{novo}} = y_{\text{origem}} + r \cdot \sin(\theta)$$

Para criar o movimento circular contínuo, o ângulo θ deve ser incrementado ao longo do tempo. Isso pode ser feito com base no tempo decorrido, o que garantirá que o objeto se mova de maneira uniforme.

A cada instante de tempo t, o ângulo θ é atualizado de acordo com a fórmula:

$$\theta_{\text{novo}} = \theta_{\text{anterior}} + \omega \cdot \Delta t$$

Onde: - θ_{anterior} é o ângulo anterior, - ω é a **velocidade angular** do objeto, - Δt é o intervalo de tempo entre as atualizações (ou frames, no caso de um jogo).

A velocidade angular ω define a rapidez com que o objeto percorre o círculo. Ela pode ser calculada em termos de radianos por segundo usando a fórmula:

$$\omega = \frac{2\pi}{T}$$

Onde: - T é o período, ou seja, o tempo necessário para o objeto completar uma volta completa ao redor do círculo, - 2π é o número de radianos em uma volta completa.

Dessa forma, quanto menor o valor de T, mais rápida será a velocidade angular, fazendo o objeto completar o círculo em menos tempo.

Exemplo Prático

Vamos considerar um exemplo prático: queremos que um objeto orbite em torno de um ponto central com raio de 100 pixels, completando uma volta a cada 5 segundos. Podemos calcular a velocidade angular e, em cada frame, atualizar a posição do objeto.

Passos: 1. Definimos o raio $r = 100$ pixels. 2. O período $T = 5$ segundos. 3. Calculamos a velocidade angular:

$$\omega = \frac{2\pi}{T} = \frac{2\pi}{5} \approx 1.2566 \,\text{radianos por segundo}$$

4. A cada frame, incrementamos o ângulo:

$$\theta_{\text{novo}} = \theta_{\text{anterior}} + \omega \cdot \Delta t$$

5. Com o novo valor de θ, calculamos a nova posição do objeto usando as funções seno e cosseno:

$$x_{\text{novo}} = x_{\text{origem}} + 100 \cdot \cos(\theta)$$

$$y_{\text{novo}} = y_{\text{origem}} + 100 \cdot \sin(\theta)$$

Dessa forma, o objeto se moverá de forma suave ao longo do círculo, completando uma volta completa a cada 5 segundos.

Exemplo Prático: Se um objeto deve orbitar em torno de um ponto central com raio de 100 pixels e completar uma volta em 5 segundos, podemos usar a fórmula acima para calcular sua posição em cada frame, onde o valor de θ varia com o tempo de acordo com a velocidade angular calculada.

2.3 Exemplos Simples de Como a Matemática Afeta Mecânicas de Jogo

2.3.1 Movimento do Personagem em Plataformas

Em um jogo de plataforma, quando o jogador pressiona uma tecla para mover o personagem, você usa a matemática para calcular como o personagem se move.

Equação do Movimento Horizontal:

$$x_{\text{novo}} = x_{\text{atual}} + v \times t$$

Isso significa que, ao segurar a tecla "direita", o valor de x aumenta constantemente, movendo o personagem para a direita.

2.3.2 Gravidade e Física no Salto

a física do movimento parabólico é aplicada. O movimento vertical é governado pela equação:

2.3.3 Gravidade e Física no Salto em Jogos 2D

Em jogos de plataforma, como *Super Mario Bros*, quando o personagem salta, é aplicada uma equação para simular o salto e a gravidade. A fórmula usada para calcular a nova posição vertical de um personagem ao saltar é baseada na equação do movimento uniformemente variado. O movimento do salto é dado pela equação:

$$y_{\text{novo}} = y_{\text{atual}} + v_y \times t - \frac{1}{2} \times g \times t^2$$

2.3.4 Componentes da Fórmula

- y_{novo}: A nova posição vertical do personagem em pixels após um intervalo de tempo t.
- y_{atual}: A posição vertical inicial do personagem em pixels antes do salto ou da atualização da posição.
- $v_y \times t$: O quanto a velocidade vertical inicial v_y (em pixels por segundo) move o personagem durante o tempo t.
- $-\frac{1}{2} \times g \times t^2$: A desaceleração causada pela gravidade g, medida em pixels por segundo ao quadrado (pixels/seg^2). Esta parte descreve o quanto a gravidade desacelera o personagem conforme o tempo t passa.

2.3.5 Ajustes de Gravidade e Velocidade para Jogos 2D

Nos jogos, os valores de v_y (velocidade inicial do salto) e g (gravidade) são ajustados para criar uma sensação de física apropriada para o jogo. Por exemplo:

- **Velocidade inicial do salto**: Um valor comum pode ser $v_y = -300\,\text{pixels/seg}$, onde o sinal negativo indica o movimento para cima.
- **Gravidade**: Um valor típico para gravidade pode ser $g = 800\,\text{pixels/seg}^2$. Esse valor faz com que o personagem desacelere durante o salto e acelere ao cair.

2.3.6 Exemplo de Aplicação

Considere um personagem com uma velocidade inicial de salto $v_y = -300\,\text{pixels/seg}$ e gravidade $g = 800\,\text{pixels/seg}^2$. Se o tempo decorrido for $t = 0,1$ segundos, a nova posição vertical pode ser calculada como:

$$y_{\text{novo}} = y_{\text{atual}} + (-300) \times 0,1 - \frac{1}{2} \times 800 \times (0,1)^2$$

$$y_{\text{novo}} = y_{\text{atual}} - 30 - \frac{1}{2} \times 800 \times 0,01$$

$$y_{\text{novo}} = y_{\text{atual}} - 30 - 4$$

$$y_{\text{novo}} = y_{\text{atual}} - 34\,\text{pixels}$$

Após $0,1$ segundos, o personagem terá subido 34 pixels a partir de sua posição inicial.

Os valores de v_y e g podem ser ajustados conforme o estilo de jogo. Jogos de plataforma, por exemplo, podem ter saltos mais altos e lentos, enquanto jogos de ação podem ter saltos mais curtos e rápidos.

- **Velocidade do salto** (v_y): Afeta o quão alto o personagem vai pular.
- **Gravidade** (g): Afeta a velocidade com que o personagem cai após atingir o pico do salto.

2.4 Vetores e Espaço 2D

O objetivo dessa seção é introduzir os conceitos de vetores e sua aplicação no desenvolvimento de jogos, com foco em movimentação e direção de personagens.

2.4.1 Definição de Vetores

Um vetor é uma entidade matemática que possui duas características principais: **magnitude** e **direção**. Em um espaço 2D, um vetor é frequentemente representado como uma seta que parte de um ponto (origem) e aponta para outro ponto (destino).

- **Magnitude**: É o comprimento do vetor, representando a quantidade de movimento. Pode ser calculada usando a fórmula:
$$\text{Magnitude} = \sqrt{x^2 + y^2}$$
- **Direção**: Indica para onde o vetor está apontando. Pode ser expressa em graus.

Um vetor no espaço 2D pode ser representado como $\mathbf{v} = (x, y)$, onde:

- x é a componente horizontal.
- y é a componente vertical.

Para calcular a magnitude de um vetor que vai de um ponto de origem (x_1, y_1) a um ponto de destino (x_2, y_2), você pode usar a fórmula da distância euclidiana. A magnitude é dada pela fórmula:

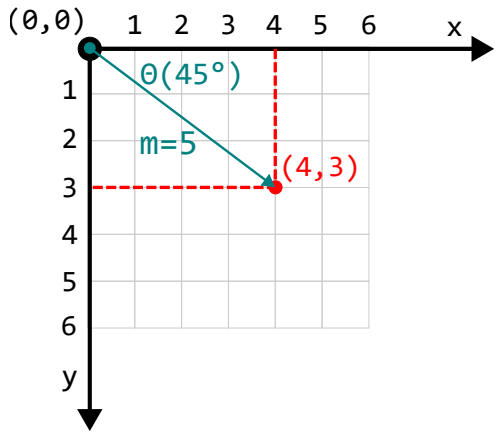


Figura 2.1: Exemplo de um Vetor)

$$\text{Magnitude} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Passos para calcular a magnitude:

1. **Determine as coordenadas:**
   - Ponto de origem: (x_1, y_1)
   - Ponto de destino: (x_2, y_2)
2. **Subtraia as coordenadas:**
   - Calcule $dx = x_2 - x_1$
   - Calcule $dy = y_2 - y_1$
3. **Aplique a fórmula:**
   - Use a fórmula acima para calcular a magnitude.

Exemplo

Se a origem é $(1,2)$ e o destino é $(4,6)$:

1. $dx = 4 - 1 = 3$
2. $dy = 6 - 2 = 4$
3. Magnitude $= \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$

Assim, a magnitude do vetor entre esses dois pontos é 5.

2.5 Normalização de Vetores

Objetivo

O objetivo desta seção é explicar o conceito de normalização de vetores, como e por que normalizar um vetor e suas aplicações práticas no desenvolvimento de jogos. A normalização é um processo crucial quando trabalhamos com vetores direcionais, pois garante que o vetor mantenha sua direção, mas com magnitude igual a 1.

2.5.1 O que é a Normalização de Vetores?

Normalizar um vetor significa ajustar sua magnitude para que seja igual a 1, sem alterar sua direção. Um vetor normalizado é frequentemente chamado de *vetor unitário*. A normalização é amplamente usada em gráficos 2D e 3D, principalmente para garantir que direções e movimentações sejam consistentes, independentemente da magnitude original do vetor.

Fórmula de Normalização: Para normalizar um vetor $\mathbf{v}$, com componentes x e y em um espaço 2D, seguimos o seguinte procedimento:

- Primeiro, calculamos a magnitude (ou comprimento) do vetor, dada pela fórmula:

$$\|\mathbf{v}\| = \sqrt{x^2 + y^2}$$

- Em seguida, dividimos cada componente do vetor pela sua magnitude:

$$\mathbf{v}_{normalizado} = \left(\frac{x}{\|\mathbf{v}\|}, \frac{y}{\|\mathbf{v}\|} \right)$$

2.5.2 Por que Normalizar um Vetor?

A normalização de vetores é fundamental quando queremos apenas a direção de um vetor, sem nos preocuparmos com sua magnitude. No desenvolvimento de jogos, isso é essencial para várias tarefas:

- **Movimentação uniforme:** Quando personagens ou objetos se movem em uma direção específica, podemos usar vetores normalizados para garantir que a velocidade seja uniforme, independentemente da direção.
- **Cálculos com direções:** Operações vetoriais, como a soma ou multiplicação de vetores, geralmente são mais fáceis de controlar se os vetores estiverem normalizados.
- **Controle de força:** Em simulações físicas, a força aplicada em um objeto pode ser representada por um vetor. Ao normalizá-lo, podemos definir a intensidade dessa força separadamente da sua direção.

2.5.3 Exemplo Prático de Normalização de Vetores

Vamos supor que um objeto em um jogo deve se mover na direção de um ponto alvo. O vetor que representa essa direção é chamado de *vetor direção*. Para garantir que o objeto se mova a uma velocidade constante, independentemente da distância, precisamos normalizar esse vetor direção.

Passo a passo:

1. **Calcular o vetor direção:** Para calcular o vetor direção entre um ponto de origem (x_0, y_0) e um ponto de destino (x_1, y_1), subtraímos as coordenadas do ponto de origem das coordenadas do ponto de destino:

$$\mathbf{v} = (x_1 - x_0, y_1 - y_0)$$

2. **Calcular a magnitude do vetor:**

$$\|\mathbf{v}\| = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

3. **Normalizar o vetor:** Dividimos cada componente do vetor pela sua magnitude:

$$\mathbf{v}_{normalizado} = \left(\frac{x_1 - x_0}{\|\mathbf{v}\|}, \frac{y_1 - y_0}{\|\mathbf{v}\|} \right)$$

4. **Multiplicar pela velocidade desejada:** Após normalizar o vetor, podemos multiplicá-lo pela velocidade v desejada do objeto:

$$\mathbf{v}_{final} = v \cdot \mathbf{v}_{normalizado}$$

   Isso garante que o objeto se moverá a uma velocidade constante v, na direção correta.

Exemplo numérico: Suponha que um objeto se mova do ponto $(2,3)$ para $(5,7)$. Seguindo os passos acima:

1. Calculamos o vetor direção:

$$\mathbf{v} = (5-2, 7-3) = (3,4)$$

2. Calculamos a magnitude do vetor:

$$\|\mathbf{v}\| = \sqrt{3^2 + 4^2} = \sqrt{9+16} = \sqrt{25} = 5$$

3. Normalizamos o vetor:

$$\mathbf{v}_{normalizado} = \left(\frac{3}{5}, \frac{4}{5} \right) = (0.6, 0.8)$$

4. Se desejamos que o objeto se mova a uma velocidade de $v = 2$ unidades por segundo, multiplicamos o vetor normalizado pela velocidade:

$$\mathbf{v}_{final} = 2 \cdot (0.6, 0.8) = (1.2, 1.6)$$

Neste exemplo, o objeto se moverá 1.2 unidades no eixo x e 1.6 unidades no eixo y a cada segundo, mantendo a direção original, mas com velocidade constante.

2.5.4 Aplicações em Jogos

A normalização de vetores tem várias aplicações práticas no desenvolvimento de jogos:

- **Movimentação de personagens:** Em jogos de tiro ou de plataforma, a movimentação de personagens pode ser controlada através de vetores normalizados, permitindo que eles se movam de forma uniforme, independentemente da direção.
- **Cálculo de colisões:** Vetores normalizados são usados para calcular a direção de colisões e rebotes de objetos. Quando um objeto colide com uma superfície, a normal da superfície é muitas vezes um vetor unitário, e isso facilita o cálculo da nova direção do objeto.
- **Direção de projéteis:** Para disparar projéteis em uma direção específica, os vetores normalizados garantem que todos os projéteis se movam na mesma velocidade, independentemente da direção em que são disparados.
- **Câmera e Iluminação:** Em jogos 3D, vetores normalizados são usados para controlar a direção da câmera e fontes de luz, garantindo que o movimento e a iluminação sejam aplicados de maneira uniforme e previsível.

A normalização de vetores é um conceito fundamental no desenvolvimento de jogos, pois permite que os desenvolvedores trabalhem com direções de maneira eficiente e previsível, sem se preocupar com a magnitude do vetor. Seja para movimentação de personagens, disparos de projéteis, ou cálculos de colisão, a normalização de vetores garante uma uniformidade crucial em muitas operações de jogos.

2.6 Sistema de Coordenadas Cartesianas

Nesta seção veremos como funciona o sistema de coordenadas no espaço 2D, com ênfase na utilização dos eixos X e Y, posicionamento de objetos e a diferença entre coordenadas globais e locais.

2.6.1 Eixo X e Y

No sistema de coordenadas cartesianas 2D, o plano é dividido por dois eixos principais:

- **Eixo X**: O eixo horizontal, que aumenta da esquerda para a direita. No lado positivo, o valor de x aumenta; no lado negativo, o valor de x diminui.
- **Eixo Y**: O eixo vertical, que aumenta de baixo para cima. No lado positivo, o valor de y aumenta; no lado negativo, o valor de y diminui.

Esses dois eixos se cruzam no ponto $(0,0)$, conhecido como a **origem**. Veja a Figura **??** a seguir.

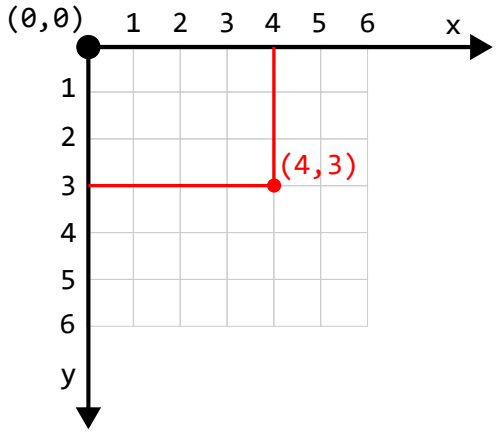


Figura 2.2: Sistemas de Coordenadas 2D)

2.6.2 Posicionamento de Objetos no Espaço 2D

Cada objeto no espaço 2D é definido por uma posição em relação aos eixos X e Y. A posição de um objeto é representada por um par de coordenadas (x,y), onde:

- x representa a posição ao longo do eixo horizontal (Eixo X).
- y representa a posição ao longo do eixo vertical (Eixo Y).

Por exemplo, se um objeto está na posição $(3,5)$, isso significa que ele está 3 unidades à direita da origem e 5 unidades acima.

2.6.3 Diferença entre Coordenadas Globais e Locais

Coordenadas Globais

As coordenadas globais se referem à posição de um objeto no espaço 2D em relação ao sistema de coordenadas geral (a origem $(0,0)$ do mundo). Ou seja, elas indicam a posição de um objeto no mundo do jogo ou na cena.

Exemplo: Se um objeto está na posição global $(10, 15)$, isso significa que ele está 10 unidades à direita e 15 unidades acima da origem do mundo.

Coordenadas Locais

As coordenadas locais, por outro lado, são relativas a um ponto de referência, que normalmente é outro objeto ou uma "origem local". Elas indicam a posição do objeto em relação ao seu "pai"ou ponto de referência local.

Exemplo: Se um objeto tem coordenadas locais $(2, 3)$ em relação a um ponto de referência que está em $(5, 5)$, então sua posição global será $(7, 8)$.

2.6.4 Visualização da Área do Jogo na Tela do Computador

Em jogos 2D, o sistema de coordenadas cartesianas não apenas define a posição dos objetos no mundo do jogo, mas também controla como essa parte do mundo será exibida na tela do jogador. Mesmo que o mundo do jogo seja maior do que a resolução da tela, apenas uma parte específica do mundo será visível de acordo com a configuração da resolução do jogo.

Por exemplo, se um jogo foi desenvolvido para rodar em uma resolução de **960 x 540**, isso significa que a área visualizada pelo jogador terá 960 pixels de largura e 540 pixels de altura. Essa área é conhecida como a **janela de visualização** ou **viewport**. Mesmo que o mundo do jogo seja maior, como **1920 x 1080**, apenas uma parte correspondente à resolução **960 x 540** será mostrada na tela em um dado momento.

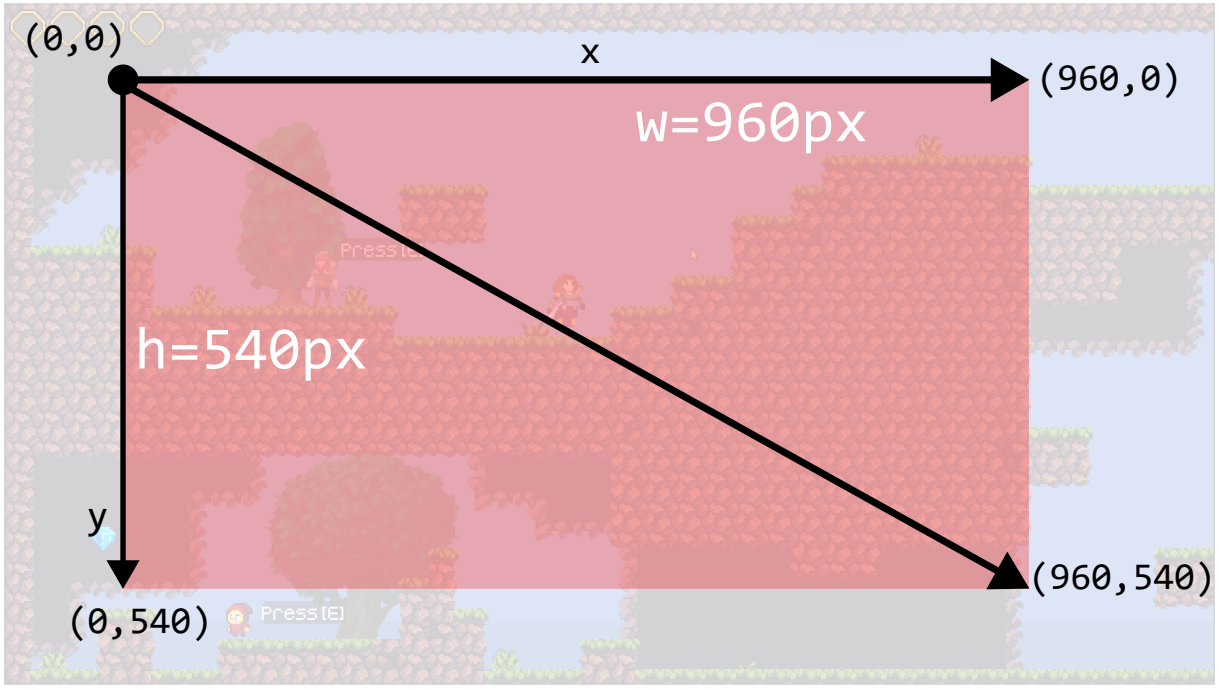


Figura 2.3: Sistemas de Coordenadas 2D - Viewport

Exemplo:

Suponha que o mundo do jogo tenha dimensões de **1920 x 1080**, mas o jogo foi projetado para ser exibido em **960 x 540**. Nesse caso:

- A tela do jogador mostrará apenas um quarto do mundo total a cada momento.
- Conforme o jogador se move pelo mundo, a **janela de visualização** se desloca, revelando outras partes do mundo.
- Se o jogador estiver no canto superior esquerdo do mundo, a posição visualizada seria próxima de $(0,0)$.
- Se o jogador se deslocar para o canto inferior direito, a janela se moverá, mostrando uma área próxima a $(960,540)$ dentro do mundo.

Impacto da Resolução e Escalonamento

É importante notar que a resolução da tela do jogo pode afetar a qualidade da exibição dos gráficos, especialmente quando o jogo é redimensionado. Se o jogador aumentar a resolução da tela para **1920 x 1080**, o mundo do jogo poderá ser visualizado por completo, sem necessidade de deslocamento. No entanto, em resoluções menores, como **640 x 360**, a janela de visualização será menor e mostrará uma porção ainda menor do mundo, fazendo com que mais movimento seja necessário para explorar o ambiente completo.

A resolução da tela e o sistema de visualização de um jogo são aspectos fundamentais para controlar o que o jogador pode ver em cada momento. Mesmo que o mundo do jogo seja maior, a área exibida dependerá da resolução configurada para a tela. Compreender como a janela de visualização funciona é crucial para o desenvolvimento de jogos que exigem exploração e movimentação em grandes ambientes.

2.7 Transformações: Translação, Rotação e Escala

Objetivo

O objetivo desta seção é ensinar como aplicar transformações geométricas básicas, como translação, rotação e escala, que são fundamentais para manipular objetos no espaço 2D no desenvolvimento de jogos. Além disso, aprenderemos como essas transformações podem ser aplicadas para criar animações básicas.

2.7.1 Translação: Movendo Objetos no Espaço 2D

A **translação** é a operação que move um objeto de uma posição para outra no espaço 2D. No sistema de coordenadas cartesianas, isso é feito somando (ou subtraindo) um valor ao par de coordenadas (x,y) do objeto.

- Para mover um objeto para a direita, aumentamos o valor de x.
- Para mover o objeto para a esquerda, diminuímos o valor de x.
- Para mover o objeto para cima, aumentamos o valor de y.
- Para mover o objeto para baixo, diminuímos o valor de y.

Exemplo de Fórmula: Se o objeto está na posição (x_0,y_0), após ser movido (transladado) por uma distância dx no eixo x e dy no eixo y, a nova posição será:

$$(x,y) = (x_0 + dx, y_0 + dy)$$

Aplicação em jogos: A translação é usada para movimentar personagens, projetar animações e mover elementos do cenário.

2.7.2 Rotação: Rotacionando Sprites e Elementos de Jogo

A **rotação** é a transformação que altera a orientação de um objeto em torno de um ponto de referência, geralmente a sua origem (ponto $(0,0)$). Em um jogo 2D, isso significa rotacionar sprites ou objetos ao redor do eixo z perpendicular ao plano.

Fórmula de Rotação: A posição de um ponto (x,y) após ser rotacionado em um ângulo θ em torno da origem é dada pelas fórmulas:

$$x' = x \cdot \cos(\theta) - y \cdot \sin(\theta)$$

$$y' = x \cdot \sin(\theta) + y \cdot \cos(\theta)$$

- Um ângulo positivo ($\theta > 0$) rotaciona o objeto no sentido anti-horário.
- Um ângulo negativo ($\theta < 0$) rotaciona o objeto no sentido horário.

Aplicação em jogos: A rotação é utilizada para girar objetos como veículos, rodas, armas, ou até mesmo personagens em jogos de plataforma ou aventura.

2.7.3 Escala: Alterando o Tamanho dos Objetos

A **escala** é a transformação que modifica o tamanho de um objeto, aumentando ou diminuindo suas dimensões em uma certa proporção. Quando escalamos um objeto, alteramos suas coordenadas multiplicando-as por um fator de escala s_x no eixo x e s_y no eixo y.

Fórmula de Escala: Se o objeto tem coordenadas (x,y), após aplicar uma escala s_x e s_y nos eixos x e y, a nova posição será:

$$x' = x \cdot s_x$$

$$y' = y \cdot s_y$$

- Se $s_x > 1$, o objeto é alongado no eixo x.
- Se $s_y > 1$, o objeto é alongado no eixo y.
- Se $0 < s_x < 1$, o objeto é comprimido no eixo x.
- Se $0 < s_y < 1$, o objeto é comprimido no eixo y.

Aplicação em jogos: A escala é usada para alterar o tamanho de personagens, ajustar elementos gráficos e criar efeitos de zoom in e zoom out.

2.7.4 Aplicação Prática: Criando Animações Básicas

Transformações como translação, rotação e escala são frequentemente combinadas para criar animações em jogos. A seguir, apresentamos um exemplo básico de como usar essas transformações para animar um objeto.

Exemplo de Animação: Vamos criar uma animação em que um personagem se move da esquerda para a direita (translação), enquanto gira lentamente (rotação) e aumenta de tamanho (escala).

- A cada quadro (frame) da animação, a posição do personagem será atualizada usando a translação, movendo-o para a direita.
- Simultaneamente, aplicamos uma rotação em torno de sua origem para fazê-lo girar.
- Também aplicamos uma escala crescente para que o personagem aumente de tamanho à medida que se move.

Pseudo-código para Animação:

```
while (jogo em execução):
    personagem.x += velocidade_x
    personagem.angulo += velocidade_rotacao
    personagem.escala_x *= fator_crescimento
    personagem.escala_y *= fator_crescimento
```

Neste pseudo-código, a cada iteração do loop de jogo:

- `personagem.x` é atualizado para mover o personagem horizontalmente.
- `personagem.angulo` é incrementado para rotacionar o personagem.
- `personagem.escala_x` e `personagem.escala_y` são multiplicados por um fator de crescimento para aumentar o tamanho do personagem.

Entender e aplicar transformações geométricas como translação, rotação e escala é fundamental para a criação de jogos dinâmicos e interativos. Essas transformações são essenciais para o movimento de personagens, a manipulação de sprites e a criação de animações fluidas. Ao dominar esses conceitos, você poderá criar movimentações e efeitos visuais que tornam seu jogo mais envolvente e visualmente atrativo.