

**ESCOLA SESC, CURSO SENAC DE JOGOS DIGITAIS DE MATO GROSSO**

**Davi Azevedo Danelichen de Oliveira**

**Integração dos elementos multimídia ao jogo digital**

Funções, atividade e trabalho

Orientador: Wanderson Timóteo

**Mato Grosso, Cuiabá**

**2024**

**Davi Azevedo Danelichen de Oliveira**

**Integração dos elementos multimídia ao jogo digital**

Funções, atividade e trabalho

**Integração dos elementos**

**multimídia ao jogo digital:**

Projeto de pesquisa apresentado ao

curso Senac , a ser utilizados

como diretrizes de produção do

trabalho de conclusão de curso.

Orientador: Wanderson Timóteo

**Mato Grosso ,Cuiabá**

**2024**

**SUMÁRIO:**

* 1. **IMPORTAÇÃO DE ASSETS.....................................................................1**
  2. **FORMATOS............................................................................................ 2**
  3. **COMPATIBILIDADE................................................................................ 3**
  4. **FUNCIONALIDADES DE IMPORTAÇÃO DE ENGINE.................................4**
  5. **RECONFIGURAÇÃO DE MODELOS E SPRITES..........................................5**

**2.0 REPOSITÓRIOS ONLINE DE ARTES.........................................................6**

**2.1 OBTENÇÃO DE ASSETS...........................................................................7**

**2.2 IMPORTAÇÃO.........................................................................................8**

**2.3 UTILIZAÇÃO............................................................................................9**

**3 ANIMAÇÃO EM ENGINE........................................................................10**

**3.1 CRIAÇÃO..............................................................................................11**

**3.2 MANIPULAÇÃO....................................................................................12**

**3.3 CONFIGURAÇÃO..................................................................................13**

**3.4 CONTROLE VIA CÓDIGO.......................................................................14**

**4 MANIPULAÇÃO DE MAPAS DE IMAGENS..............................................15**

**4.1 CONFIGURAÇÕES DE SPRITESHETSE DE TILEMAPS..............................16**

**5 ÁUDIO EM ENGINE.................................................................................17**

**5.1 MANIPULAÇÃO....................................................................................18**

**5.2 CONFIGURAÇÃO..................................................................................19**

**5.3 COMPONENTES DO ENGINE PARA ÁUDIO..........................................20**

**5.4 CONTROLE VIA CÓDIGO......................................................................21**

**5.5 PADRÃO SINGLETON...........................................................................22**

**5.6 MELHORES PRÁTICAS NO GERENCIAMENTO DE ÁUDIO.....................23**

**6 MATERIAIS .............................................................................................24**

**6.1 TEXTURAS.............................................................................................25**

**6.2 CONCEITO.............................................................................................26**

**6.3 CONFIGURAÇÃO...................................................................................27**

**6.4 APLICAÇÃO...........................................................................................28**

**7 SISTEMA DE PARTICULAS.......................................................................29**

**7.1 CONCEITO............................................................................................30**

**7.2 UTILIZAÇÃO..........................................................................................31**

**8 CONTROLE DE VERSÃO DO PROJETODE JOGO DIGITAL........................32**

**8.1 RECURSOS DE NUVEM.........................................................................33**

**8.2 INTEGRAÇÃO COM ENGINE.................................................................34**

**8.3 CONFIGURAÇÃO DE REPOSITÓRIO......................................................35**

**9 VALIDAÇÃO DA INTEGRAÇÃO................................................................36**

**9.1 PROCEDIMENTOS.................................................................................37**

**9.2 TÉCNICAS DE TESTE..............................................................................38**

**10 PROGRAMAÇÃO ORIENTADA A OBJETOS............................................39**

**10.1 ESTRUTURAS DE DADOS.....................................................................40**

**10.2 TIPOS GENÉRICOS...............................................................................41**

**RESUMO:**

A integração dos elementos multimídia ao jogo digital refere-se à incorporação de diferentes tipos de mídia, como gráficos, áudio, vídeo e interatividade, para enriquecer a experiência do jogador.

Em resumo, a integração eficaz de elementos multimídia ao jogo digital é fundamental para criar experiências envolventes e imersivas, proporcionando aos jogadores uma variedade de estímulos visuais, auditivos e interativos que contribuem para uma experiência de jogo rica e memorável.

**The integration of multimedia elements into the digital game refers to the incorporation of different types of media, such as graphics, audio, video and interactivity, to enrich the player's experience.**

**In summary, effective integration of multimedia elements into digital gaming is critical to creating engaging and immersive experiences, providing players with a variety of visual, auditory and interactive stimuli that contribute to a rich and memorable gaming experience.**

1. **IMPORTAÇÃO DE ASSETS:**

A importação de ativos é o processo de trazer elementos digitais, como modelos 3D, texturas, animações, áudio e outros recursos, para dentro de um ambiente de desenvolvimento de jogos ou de software.

**1.1 FORMATOS:**

Escolher os formatos de arquivo adequados para os diferentes tipos de ativos, considerando a compatibilidade com o software ou mecanismo de desenvolvimento utilizado.

**1.2COMPATIBILIDADE:** Verificar se os formatos de arquivo selecionados são compatíveis com o software ou mecanismo de desenvolvimento em uso, garantindo que os ativos possam ser corretamente interpretados e utilizados.

**1.3 FUNCIONALIDADES DE IMPORTAÇÃO DE ENGINE:**

Utilizar as ferramentas e funcionalidades disponíveis no software ou mecanismo de desenvolvimento para importar os ativos, o que pode incluir opções para visualização, configuração de propriedades e ajustes de escala e rotação.

* 1. **RECONFIGURAÇÃO DE MODELOS E SPRITES:**

Dependendo do mecanismo de jogo, você pode precisar reconfigurar seus modelos e sprites para garantir que eles funcionem corretamente no jogo. Isso pode incluir ajustes na escala, rotação, configuração de colisões, otimização de desempenho, etc.

**2.0** **REPOSITÓRIOS ONLINE DE ARTES**

Repositórios online de arte são plataformas onde artistas compartilham e disponibilizam seus trabalhos para uso por outros criadores, desenvolvedores de jogos e designers.

**2.1 OBTENÇÃO DE ASSETS:**

**PESQUISA E NAVEGAÇÃO:** Os repositórios geralmente oferecem uma ampla variedade de categorias e tags para ajudar os usuários a encontrar o que estão procurando.

**LICENCIAMENTO:** É importante observar as licenças associadas aos assets. Alguns podem ser gratuitos para uso comercial ou não comercial, enquanto outros podem ter restrições ou exigir pagamento.

**2.2 IMPORTAÇÃO:**

**DOWNLOAD:** Os assets desejados são baixados do repositório para o computador local.

**INTEGRAÇÃO NO PROJETO:** Os assets são então importados para a ferramenta de desenvolvimento ou mecanismo de jogo sendo utilizado. Isso pode envolver a cópia dos arquivos para os diretórios corretos do projeto.

**2.3 UTILIZAÇÃO:**

**INTEGRAÇÃO NO PROJETO:** Os assets importados são utilizados dentro do projeto de acordo com suas necessidades. Isso pode incluir a criação de cenas, personagens, animações, efeitos visuais, etc.

**AJUSTES E ADAPTAÇÕES:** Dependendo do caso, pode ser necessário fazer ajustes nos assets importados para se adequarem ao estilo artístico do projeto ou às especificações técnicas.

1. **ANIMAÇÃO EM ENGINE:**

A criação, manipulação, configuração e controle de animações em um mecanismo de jogo envolvem uma combinação de habilidades artísticas e técnicas de programação para criar uma experiência visualmente atraente e interativa para os jogadores.

**3.1 CRIAÇÃO:**

A criação de animações geralmente envolve o uso de software de modelagem e animação 3D, onde os animadores criam os movimentos dos personagens, objetos ou elementos do jogo.

**3.2 MANIPULAÇÃO:**

Após a criação das animações, elas precisam ser importadas para o mecanismo de jogo. Isso pode ser feito por meio de formatos de arquivo compatíveis com o mecanismo, como FBX, COLLADA, entre outros.

**3.3 CONFIGURAÇÃO:**

No mecanismo de jogo, as animações são configuradas para serem reproduzidas nos objetos corretos, com as configurações adequadas de reprodução. Isso pode incluir a definição de loops, velocidade de reprodução, transições entre animações, entre outros aspectos.

**3.4 CONTROLE VIA CÓDIGO:**

O controle das animações geralmente é realizado via código. Os desenvolvedores escrevem scripts para acionar as animações em resposta a eventos do jogo, como movimento do personagem, interações com o ambiente, combate, etc. Isso pode ser feito por meio de bibliotecas de animação ou funcionalidades específicas do mecanismo de jogo.

**4.0 MANIPULAÇÃO DE MAPAS DE IMAGENS:**

Preparação de Imagens: Os mapas de imagens são geralmente criados em softwares de edição de imagens, como Adobe Photoshop, GIMP ou software similar. Eles podem consistir em vários elementos gráficos, como cenários, objetos, personagens, etc.

Divisão em Layers: Muitas vezes, é útil dividir o mapa de imagem em camadas separadas para diferentes elementos, como fundo, plataformas, objetos interativos, etc. Isso facilita a manipulação e a interação com esses elementos no jogo.

Exportação: Depois que o mapa de imagem é criado e organizado, ele é exportado em um formato adequado para ser utilizado no mecanismo de jogo. Formatos comuns incluem PNG, JPEG ou formatos específicos do mecanismo de jogo.

**4.1 MANIPULAÇÃO DE MAPAS DE IMAGENS:**

**Criação de Spritesheets**: Spritesheets são imagens que contêm vários sprites (imagens individuais) organizados em uma grade. Eles são usados para armazenar várias animações ou estados de um personagem, objeto ou efeito visual em um único arquivo.

**Definição de Regiões de Sprites**: Cada sprite na spritesheet é definido por uma região retangular específica. É importante definir corretamente essas regiões para que o mecanismo de jogo possa renderizar os sprites corretamente durante o jogo.

**Configuração de Tilemaps**: Tilemaps são grades de células que representam diferentes partes do ambiente do jogo, como terreno, paredes, obstáculos, etc. Eles são usados para construir cenários e níveis de forma eficiente e modular.

**Definição de Tiles e Conjuntos de Tiles**: Cada célula no tilemap é preenchida com um tile, que pode ser uma imagem individual ou parte de um conjunto de tiles. Os conjuntos de tiles são coleções de imagens que representam diferentes texturas e elementos do ambiente do jogo.

**Colisões e Propriedades**: Além de renderizar o ambiente do jogo, os tilemaps também podem ser usados para definir colisões e outras propriedades físicas dos elementos do jogo, facilitando a interação dos personagens e objetos com o ambiente.

**5.0 ÁUDIO EM ENGINE**:

**Importação de Arquivos de Áudio**: Os arquivos de áudio, como WAV, MP3, ou outros formatos suportados, devem ser importados para o mecanismo de jogo.

**5.1 CONFIGURAÇÃO**:

Isso inclui ajustar o volume, a frequência de reprodução, os efeitos de áudio (como eco ou reverb), entre outros.

**Associação de Áudio a Eventos:** Os sons devem ser associados a eventos específicos do jogo, como ações do jogador, eventos de jogo, ou outras interações.

**Implementação de Música de Fundo e Efeitos Sonoros**: Além da música de fundo, os efeitos sonoros devem ser implementados de forma a complementar a jogabilidade e a experiência do usuário.

**5.2 COMPONENTES DO ENGINE PARA ÁUDIO:**

**Módulo de Áudio**: O mecanismo de jogo deve ter um módulo dedicado ao processamento e reprodução de áudio.

**Controles de Reprodução de Áudio**: O mecanismo deve fornecer controles para iniciar, pausar, parar e ajustar o volume dos sons.

**5.3 CONTROLE VIA CÓDIGO:**

**APIs de Áudio**: APIs ou bibliotecas para manipulação avançada de áudio, como posicionamento espacial, filtragem, efeitos especiais, entre outros.

**5.4 PADRÃO SINGLETON:**

Um Singleton é um padrão de design que garante a existência de apenas uma instância de uma classe em um programa. No contexto de áudio em um mecanismo de jogo, um Singleton pode ser usado para gerenciar o sistema de áudio de forma centralizada, garantindo que haja apenas uma instância do sistema de áudio em execução no jogo.

**5.5 MELHORES PRÁTICAS NO GERENCIAMENTO DE ÁUDIO:**

**Otimização de Recursos**: Utilize formatos de áudio comprimidos para reduzir o tamanho dos arquivos e minimizar o uso de memória.

**Carregamento Antecipado**: Carregue os arquivos de áudio antecipadamente, preferencialmente durante telas de carregamento, para evitar atrasos durante o jogo.

**Pooling de Áudio**: Utilize pooling de áudio para reutilizar instâncias de áudio, reduzindo a sobrecarga de criar e destruir instâncias repetidamente.

**Gerenciamento de Volume**: Permita aos jogadores ajustar o volume de áudio no jogo e respeite as configurações de áudio do sistema do usuário.

**Teste em Diferentes Dispositivos**: Teste o áudio em uma variedade de dispositivos e sistemas operacionais para garantir compatibilidade e qualidade sonora consistente.

**6.0 MATERIAIS:**

Em jogos, um material é uma combinação de propriedades visuais que define como a luz interage com uma superfície. Isso inclui características como cor, reflexão, brilho, opacidade e textura.

**6.1 TEXTURAS:**

Texturas são imagens aplicadas a superfícies 3D para adicionar detalhes visuais. Elas podem ser simplesmente uma cor sólida (textura plana) ou uma imagem complexa que define características como superfícies rugosas, metálicas, etc.

**6.2 CONCEITO:**

**Gráficos e Animações**: Utilize gráficos e animações de alta qualidade para representar diferentes tipos de materiais, suas propriedades e comportamentos. Isso pode incluir texturas realistas, efeitos de luz e sombra, bem como animações que mostram processos de fabricação ou mudanças de estado dos materiais.

**Vídeos Explicativos**: Integre vídeos curtos explicativos que abordem conceitos-chave sobre materiais, como resistência, condutividade, durabilidade, entre outros. Esses vídeos podem ser inseridos em momentos estratégicos do jogo para fornecer informações adicionais aos jogadores.

**Áudio Imersivo**: Utilize efeitos sonoros realistas para acompanhar as interações dos jogadores com diferentes materiais. Por exemplo, ao selecionar um material metálico, o jogador pode ouvir o som de metal batendo, enquanto um material de plástico pode produzir um som diferente ao ser manipulado.

**6.3 CONFIGURAÇÃO:**

**Atributos do Material**: Normalmente, um material possui diversos atributos configuráveis, como cor base, textura, brilho, opacidade, entre outros. Cada mecanismo de jogo tem sua própria forma de configurar esses atributos, que pode ser através de interfaces gráficas ou scripts.

**Mapeamento de Texturas**: O mapeamento de texturas define como uma textura é aplicada a uma malha 3D. Isso inclui técnicas como mapeamento UV, onde coordenadas de textura são atribuídas a cada vértice da malha.

**6.4 APLICAÇÃO:**

**Atribuição de Materiais**: Os materiais são atribuídos a objetos 3D no mecanismo de jogo. Isso pode ser feito manualmente através de interfaces gráficas ou programaticamente via código.

**Ajustes e Testes**: Após aplicar um material a um objeto, é importante ajustar seus parâmetros e testá-lo em diferentes condições de iluminação e visualização para garantir o resultado desejado.

**Otimização**: Texturas e materiais devem ser otimizados para melhorar o desempenho do jogo. Isso pode incluir a redução do tamanho das texturas, o uso de técnicas de compressão, entre outras práticas.

**Iteração e Feedback**: A aplicação de materiais e texturas é muitas vezes um processo iterativo, onde o desenvolvedor faz ajustes com base no feedback visual até alcançar o resultado desejado.

**7 SISTEMA DE PARTICULAS:**

Um sistema de partículas é uma técnica de computação gráfica que simula o comportamento de um grande número de pequenas partículas individuais em um ambiente virtual. Essas partículas podem representar uma variedade de fenômenos naturais, como fogo, fumaça, água, poeira, explosões, entre outros. Aqui está uma explicação sobre o conceito e algumas aplicações desse tipo de sistema.

**7.1 CONCEITO:**

Um sistema de partículas é composto por três componentes principais:

**Emitter (Emissor)**: Define a origem das partículas e as propriedades iniciais, como posição, velocidade, cor e vida útil.

**Forces (Forças):** Aplicam influências externas às partículas, como gravidade, vento, turbulência, etc.

**Particles (Partículas):** São as entidades individuais que se movem e interagem de acordo com as propriedades definidas pelo emissor e pelas forças.

**7.2 UTILIZAÇÃO:**

**Efeitos Visuais em Jogos e Filmes:** Os sistemas de partículas são amplamente utilizados na indústria de entretenimento para criar efeitos visuais impressionantes. Eles podem ser empregados para simular fogo, explosões, água, fumaça, magia, entre outros, adicionando realismo e imersão aos jogos de vídeo e filmes.

**Simulações Científicas e Engenharia:** Além de fins estéticos, os sistemas de partículas também são usados em simulações científicas e engenharia. Eles podem ser empregados para simular fenômenos físicos complexos, como o comportamento de fluidos, reações químicas, colisões de partículas, entre outros.

**Ambientes Interativos:** Em ambientes interativos, os sistemas de partículas podem ser utilizados para criar elementos interativos, como chuva, neve, folhas caindo, entre outros. Esses elementos podem contribuir para a atmosfera do ambiente virtual e melhorar a experiência do usuário.

**8 CONTROLE DE VERSÃO DO PROJETODE JOGO DIGITAL:**

Existem várias opções populares para plataformas de controle de versão, como Git (com GitHub, GitLab, Bitbucket), Subversion (SVN), Mercurial, entre outros. Para projetos de jogo digital, o Git é frequentemente preferido devido à sua flexibilidade, eficiência e ampla adoção na comunidade de desenvolvimento de software.

**8.1 RECURSOS DE NUVEM** :

Certifique-se de que o repositório remoto na plataforma de controle de versão (GitHub, GitLab, Bitbucket, etc.) esteja configurado para armazenar os arquivos na nuvem.

Isso permite acesso fácil aos arquivos do projeto de qualquer lugar e evita perda de dados em caso de problemas locais.

**8.2 INTEGRAÇÃO COM ENGINE** :

Se você estiver usando uma engine popular, como Unity, Unreal Engine ou Godot, essas engines geralmente têm integração direta com sistemas de controle de versão.

No Unity, por exemplo, você pode conectar seu projeto ao Git diretamente do Unity Hub.

Para a Unreal Engine, você pode configurar o Git diretamente no editor da engine.

Em outras engines ou ferramentas de desenvolvimento, você pode usar a linha de comando ou plugins específicos para integrar o controle de versão.

**8.3 CONFIGURAÇÃO DE REPOSITÓRIO**:

Clone o repositório criado na plataforma de controle de versão para o seu ambiente de desenvolvimento local.

Certifique-se de ter o Git instalado em sua máquina local.

**9 VALIDAÇÃO DA INTEGRAÇÃO** :

Para validar a integração do controle de versão com sua engine de desenvolvimento e garantir que tudo esteja funcionando corretamente, é importante realizar procedimentos de teste adequados.

**9.1 PROCEDIMENTOS:**

**1. Clonagem do Repositório:**

Objetivo: Verificar se é possível clonar o repositório remoto para o ambiente local.

Procedimento:

Utilize o comando git clone (ou a opção equivalente na sua plataforma de controle de versão) para clonar o repositório remoto.

Verifique se todos os arquivos e pastas foram baixados corretamente e se a estrutura do projeto está intacta.

**2. Adição e Comissão de Alterações**:

Objetivo: Garantir que alterações feitas no ambiente local possam ser adicionadas ao controle de versão.

Procedimento:

Faça alterações em arquivos no projeto local.

Utilize os comandos git add e git commit (ou ferramentas integradas da sua engine) para adicionar e confirmar as alterações.

Verifique se as alterações foram registradas corretamente no histórico de revisões.

**3. Recuperação de Versões Anteriores:**

Objetivo: Testar a capacidade de recuperar versões anteriores do projeto.

Procedimento:

Utilize o comando git checkout (ou opção equivalente) para mudar para uma versão específica do projeto.

Verifique se todos os arquivos e pastas são recuperados corretamente e se o projeto é revertido para o estado desejado.

**4. Ramificação e Mesclagem:**

Objetivo: Testar a capacidade de trabalhar em paralelo em diferentes branches e mesclar alterações de volta para o branch principal.

Procedimento:

Crie um novo branch para uma funcionalidade específica ou correção de bug.

Faça alterações nesse branch e teste-as localmente.

Utilize o comando git merge (ou ferramentas equivalentes) para mesclar as alterações de volta para o branch principal.

Verifique se a mesclagem ocorre sem conflitos e se as alterações são refletidas corretamente no projeto principal.

**5. Histórico de Revisões:**

Objetivo: Verificar se é possível visualizar o histórico completo de revisões do projeto.

Procedimento:

Utilize o comando git log (ou opção equivalente) para visualizar o histórico de commits.

Verifique se todas as mensagens de commit são claras e descritivas, e se todas as alterações são devidamente registradas.

**9.2 TÉCNICAS DE TESTE:**

**1. Teste de Clonagem do Repositório:**

**Verifique se é possível clonar o repositório remoto para um ambiente local.**

**Garanta que todos os arquivos e pastas estejam corretamente sincronizados.**

**2. Teste de Adição e Comissão de Alterações:**

**Faça alterações em arquivos no seu projeto local.**

**Adicione essas alterações ao controle de versão usando comandos Git (ou ferramentas integradas da sua engine).**

**Verifique se as alterações foram corretamente registradas e podem ser visualizadas no repositório remoto.**

**3. Teste de Recuperação de Versões Anteriores:**

**Volte para uma versão anterior do seu projeto clonando uma versão específica do repositório.**

**Verifique se todos os arquivos e pastas são recuperados corretamente.**

**4. Teste de Ramificação (Branching) e Mesclagem (Merging):**

**Crie um novo branch para desenvolver uma nova funcionalidade ou correção de bug.**

**Faça alterações nesse branch e teste-as localmente.**

**Mescla essas alterações de volta para o branch principal (ou outro branch de destino) e verifique se a mesclagem ocorre sem problemas.**

**5. Teste de Colaboração:**

**Trabalhe com outros membros da equipe em diferentes branches.**

**Teste a capacidade de compartilhamento de código e colaboração em tempo real.**

**6. Teste de Histórico de Revisões:**

**Verifique se é possível visualizar o histórico completo de revisões e alterações feitas no projeto ao longo do tempo.**

**Garanta que todas as mensagens de commit estejam claras e descritivas.**

**7. Teste de Conflitos de Mesclagem:**

**Faça alterações simultâneas em um mesmo arquivo por diferentes membros da equipe.**

**Tente mesclar essas alterações e resolva quaisquer conflitos que surgirem durante o processo.**

**8. Teste de Integração com a Engine de Desenvolvimento:**

**Abra o projeto na sua engine de desenvolvimento.**

**Verifique se todas as referências de arquivos e recursos estão corretamente vinculadas.**

**Garanta que as alterações feitas no ambiente de desenvolvimento sejam refletidas no controle de versão e vice-versa.**

**9. Teste de Implantação e Compilação:**

**Implante o projeto em diferentes plataformas de destino (PC, consoles, dispositivos móveis, etc.).**

**Verifique se todas as dependências e arquivos necessários estão corretamente incluídos na compilação final.**

**10 PROGRAMAÇÃO ORIENTADA A OBJETOS:**

A Programação Orientada a Objetos é um paradigma de programação que organiza o código em torno de "objetos", que podem conter dados (atributos) e métodos (funções). Os principais conceitos da POO incluem:

**Classes e Objetos**: As classes são modelos para criar objetos, enquanto os objetos são instâncias de classes que representam entidades do mundo real.

Encapsulamento: É o conceito de agrupar dados (atributos) e métodos em um único objeto, e restringir o acesso a eles por meio de modificadores de acesso (public, private, protected).

Herança: Permite que uma classe herde características (atributos e métodos) de outra classe, estendendo seu comportamento.

Polimorfismo: Permite que objetos de diferentes classes sejam tratados de maneira uniforme, usando métodos com o mesmo nome, mas com implementações diferentes.

**10.1 ESTRUTURAS DE DADOS:**

Estruturas de Dados:

Estruturas de dados são formas de organizar e armazenar dados em um computador para facilitar a manipulação e recuperação eficiente desses dados. Algumas estruturas de dados comuns incluem:

**Listas:** Sequências ordenadas de elementos, onde cada elemento tem uma posição específica.

**Pilhas (Stacks):** Uma estrutura de dados em que os elementos são inseridos e removidos de forma "Last-In, First-Out" (LIFO).

**Filas (Queues):** Uma estrutura de dados em que os elementos são inseridos no final e removidos do início, seguindo o princípio "First-In, First-Out" (FIFO).

**Árvores:** Estruturas de dados hierárquicas que consistem em nós conectados por arestas, com um nó especial chamado raiz e nodos secundários chamados filhos.

**Grafos:** Estruturas de dados compostas por vértices (ou nós) conectados por arestas.

**10.2 TIPOS GENÉRICOS:**

**Tipos Genéricos:**

Os tipos genéricos, também conhecidos como "parametrizados", são uma maneira de escrever algoritmos e estruturas de dados que podem funcionar com qualquer tipo de dado, sem especificar os tipos de dados antecipadamente. Isso promove a reutilização de código e a flexibilidade no design de programas. Em linguagens como C++, Java, C# e Python, os tipos genéricos são usados para criar classes, funções ou estruturas de dados que podem trabalhar com qualquer tipo de dados, enquanto ainda fornecem segurança de tipo durante a compilação.