# Jogo de Sobrevivência
# Última Fronteira

## INTRODUÇÃO

**"Última Fronteira"** é um jogo de sobrevivência baseado em eventos, onde os jogadores devem tomar decisões estratégicas para sobreviver em um ambiente hostil. Cada jogador assume o papel de um explorador perdido em uma terra desconhecida, enfrentando desafios como escassez de recursos, ataques de criaturas e condições climáticas extremas. O objetivo do jogo é **sobreviver pelo maior tempo possível**, gerenciando cuidadosamente alimentos, energia e equipamentos, enquanto enfrenta eventos imprevisíveis.

Aqui está o detalhamento dos **principais componentes do jogo** para o **Jogo de Sobrevivência – Última Fronteira**, com os **personagens flexíveis para diferentes classes e atributos**.

## PRINCIPAIS COMPONENTES DO JOGO

O jogo **"Última Fronteira"** é estruturado em torno de um sistema dinâmico de sobrevivência, onde **o jogador enfrenta desafios ambientais, toma decisões estratégicas e gerencia seus recursos** para sobreviver pelo maior tempo possível.

### Personagem

O jogador assume o papel de um sobrevivente em um ambiente hostil. **Diferentes classes de personagens podem ser criadas**, cada uma com atributos e habilidades distintas. Essas variações podem incluir exploradores, caçadores, cientistas, mercadores, entre outros, cada um trazendo vantagens e desvantagens para a sobrevivência.

**Atributos principais:**

- **Nome:** Identificação do personagem.
- **Vida:** Representa a resistência do personagem. Se chegar a zero, o jogo termina.
- **Fome:** Reduz a cada turno; se atingir um nível crítico, começa a afetar a vida.
- **Sede:** Similar à fome, mas com uma taxa de consumo mais rápida.
- **Energia:** Determina a capacidade do personagem de realizar ações; pode ser restaurada descansando.
- **Sanidade:** Algumas situações podem afetar a mente do personagem, levando a alucinações ou perda de controle.

- **Inventário:** Espaço onde o personagem armazena recursos como comida, ferramentas e armas.
- **Localização:** Define em qual parte do mapa o jogador se encontra.

**Habilidades especiais (dependentes da classe do personagem):**

Cada personagem pode ter habilidades únicas, como:

- **Rastreador:** Encontra comida e água com mais facilidade.
- **Mecânico:** Conserta ferramentas e cria novas armas.
- **Médico:** Pode tratar ferimentos sem necessidade de itens raros.
- **Sobrevivente Nato:** Menos impactado por fome e sede.

---

## Recursos e Itens

Os recursos e itens são fundamentais para a sobrevivência no jogo, sendo usados para restaurar atributos do personagem, fabricar novas ferramentas e interagir com o ambiente. Para estruturar esses elementos de forma eficiente dentro dos conceitos de **Orientação a Objetos (OO)**, foi adotado um sistema baseado em **herança e polimorfismo**, garantindo modularidade e reuso do código.

**Superclasse: Item**

Todos os itens do jogo compartilham características comuns, por isso são modelados como objetos que derivam da **superclasse abstrata** `Item`. Essa classe define **atributos essenciais** e métodos genéricos que podem ser estendidos por subclasses mais específicas.

**Atributos da superclasse Item:**

- **Nome:** Identificação do item.
- **Peso:** Influencia a quantidade de itens que o personagem pode carregar.
- **Durabilidade:** Alguns itens se desgastam com o uso e podem quebrar.

Além disso, **métodos como "usar()" são definidos na superclasse e podem ser sobrescritos** nas subclasses conforme o comportamento esperado de cada item.

**Subclasses de Item**

Cada tipo de item no jogo pertence a uma categoria específica e possui **atributos e comportamentos próprios**.

1. **Alimentos (`Alimento` – Subclasse de `Item`)**
    - **Atributos adicionais:**
        - Valor nutricional (pontos de fome restaurados).
        - Tipo (fruta, carne, enlatado, etc.).
        - Prazo de validade (alguns alimentos podem estragar).

    - **Método sobrescrito:**
        - `consumir()`: Restaura fome e pode ter efeitos colaterais (como intoxicação alimentar).
2. **Água (`Agua` – Subclasse de `Item`)**
    - **Atributos adicionais:**
        - Pureza (potável, contaminada).
        - Volume (quantidade de consumo por unidade).
    - **Método sobrescrito:**
        - `beber()`: Restaura sede, mas pode causar doenças se for contaminada.
3. **Materiais (`Material` – Subclasse de `Item`)**
    - **Atributos adicionais:**
        - Tipo (madeira, pedra, metal).
        - Resistência (impacta a durabilidade de ferramentas fabricadas com ele).
    - **Método sobrescrito:**
        - `combinar(Material outroMaterial)`: Permite criar novos itens ao combinar materiais diferentes.
4. **Ferramentas (`Ferramenta` – Subclasse de `Item`)**
    - **Atributos adicionais:**
        - Tipo (machado, faca, isqueiro, lanterna).
        - Eficiência (impacta a rapidez ao coletar recursos).
    - **Método sobrescrito:**
        - `usar()`: Reduz durabilidade e realiza a ação correspondente, como cortar madeira ou acender fogo.
5. **Armas (`Arma` – Subclasse de `Item`)**
    - **Atributos adicionais:**
        - Tipo de arma (corpo a corpo ou à distância).
        - Dano (quantidade de dano causado ao alvo).
        - Alcance (distância efetiva da arma).
    - **Método sobrescrito:**
        - `atacar(Alvo inimigo)`: Causa dano ao inimigo e pode consumir munição (se aplicável).
6. **Remédios (`Remedio` – Subclasse de `Item`)**
    - **Atributos adicionais:**
        - Tipo (bandagem, antibiótico, analgésico).
        - Efeito (cura ferimentos, alivia dor, trata infecções).
    - **Método sobrescrito:**
        - `usar()`: Aplica o efeito medicinal no personagem.

**Relacionamento entre Itens e Personagens**

Os itens são **armazenados no inventário do personagem**, o que possibilita uma **relação de composição** entre as classes.

- A classe `Inventario` gerencia os itens carregados pelo personagem, permitindo adicionar, remover e usar itens de forma eficiente.

- A **quantidade de itens transportados** pode ser limitada pelo **peso total permitido**, exigindo que o jogador faça escolhas estratégicas sobre quais recursos carregar.

**Atributos da classe `Inventario`:**

- **Lista de Itens:** Contém todos os objetos que o personagem possui.
- **Peso Total:** Soma dos pesos dos itens carregados.
- **Espaço Disponível:** Capacidade máxima do inventário.

**Métodos da classe `Inventario`:**

- `adicionarItem(Item item)`: Insere um novo item no inventário, se houver espaço.
- `removerItem(String nomeItem)`: Retira um item do inventário.
- `usarItem(String nomeItem)`: Ativa o efeito do item no personagem.

---

## Eventos Aleatórios

Os **eventos aleatórios** desempenham um papel fundamental no jogo, adicionando imprevisibilidade e tornando cada sessão única. Esses eventos podem afetar diretamente o personagem, o ambiente ou os recursos disponíveis, forçando o jogador a adaptar sua estratégia.

Para estruturar os eventos no jogo de forma eficiente dentro da **programação orientada a objetos (OO)**, foi utilizada a abordagem de **herança, polimorfismo e interfaces**, garantindo que cada tipo de evento possua um comportamento específico e modular.

**Superclasse: Evento**

Todos os eventos compartilham algumas características básicas, por isso são modelados como objetos que derivam de uma **superclasse abstrata `Evento`**. Essa classe define **atributos essenciais** e um **método genérico `executar()`**, que será sobrescrito pelas subclasses de eventos específicos.

**Atributos da superclasse `Evento`:**

- **Nome:** Identificação do evento.
- **Descrição:** Texto explicativo sobre o evento.
- **Probabilidade de ocorrência:** Define a chance de um evento acontecer a cada turno.
- **Impacto:** Indica quais aspectos do jogo serão alterados (vida, fome, sede, energia, sanidade, inventário, etc.).
- **Condição de ativação:** Determina se o evento pode ocorrer (ex.: apenas em determinados ambientes).

**Método principal:**

- `executar(Personagem jogador, Ambiente local)`: Define a lógica do evento e aplica seus efeitos ao personagem e ao ambiente.

**Subclasses de Evento**

Cada tipo de evento no jogo possui **características e impactos distintos**, sendo modelado como uma **subclasse de `Evento`**.

**1. Eventos Climáticos (`EventoClimatico` – Subclasse de `Evento`)**

Os eventos climáticos alteram as condições ambientais, impactando a jogabilidade e as ações do jogador.

**Atributos adicionais:**

- **Tipo de clima:** (Nevasca, tempestade, calor extremo, etc.)
- **Duração:** Quantidade de turnos que o evento permanece ativo.
- **Efeito no ambiente:** Pode dificultar ou facilitar certas ações, como encontrar comida ou se deslocar.

**Exemplos de eventos climáticos:**

- **Nevasca:** Reduz a visibilidade e exige mais energia para se movimentar.
- **Chuva Forte:** Pode encharcar roupas e reduzir temperatura corporal.
- **Calor Extremo:** Aumenta o consumo de água do personagem.

**2. Eventos de Criaturas (`EventoCriatura` – Subclasse de `Evento`)**

Eventos relacionados a **encontros com animais selvagens**, que podem ser **hostis ou neutros**.

**Atributos adicionais:**

- **Tipo de criatura:** (Lobo, urso, cobra, corvo, etc.)
- **Nível de perigo:** Define o impacto potencial no personagem.
- **Opções de ação:** Algumas criaturas podem ser evitadas ou combatidas.

**Exemplos de eventos de criaturas:**

- **Ataque de Lobo:** O personagem perde pontos de vida e pode ter comida roubada.
- **Cobra Venenosa:** Se picado, o jogador perde vida progressivamente até encontrar um antídoto.
- **Corvos Furtivos:** Reduzem a sanidade do personagem, criando alucinações temporárias.

### 3. Eventos de Descoberta (`EventoDescoberta` – Subclasse de `Evento`)

Esses eventos **recompensam o jogador** com suprimentos, abrigo ou informações sobre o ambiente.

**Atributos adicionais:**

- **Tipo de descoberta:** (Caverna, abrigo, suprimentos abandonados, etc.)
- **Recursos encontrados:** Pode incluir comida, água, ferramentas ou armas.
- **Condição especial:** Algumas descobertas podem exigir habilidades específicas para serem exploradas.

**Exemplos de eventos de descoberta:**

- **Abrigo Abandonado:** O jogador pode encontrar alimentos, mas há o risco de ser ocupado por outra criatura.
- **Fonte de Água:** Pode fornecer água potável ou exigir filtragem antes do consumo.
- **Ruínas Misteriosas:** Possuem itens raros, mas podem estar protegidas por armadilhas.

### 4. Eventos de Doenças e Ferimentos (`EventoDoencaFerimento` – Subclasse de `Evento`)

Afetam diretamente o estado físico do personagem, tornando a sobrevivência mais desafiadora.

**Atributos adicionais:**

- **Tipo de condição:** (Infecção, febre, desidratação, fratura, etc.)
- **Impacto:** Reduz atributos como vida, energia ou sanidade.
- **Cura disponível:** Alguns eventos exigem itens específicos para serem tratados.

**Exemplos de eventos de doenças e ferimentos:**

- **Hipotermia:** Se o jogador não se aquecer, perderá vida gradativamente.
- **Infecção:** Se um ferimento não for tratado, a infecção pode piorar e reduzir drasticamente os atributos.
- **Desidratação:** Aumenta a fadiga e pode causar alucinações.

### Relacionamento entre Eventos, Personagens e Ambiente

Os eventos não ocorrem isoladamente; eles **interagem diretamente com os personagens e o ambiente**, criando desafios dinâmicos.

- **A classe `GerenciadorDeEventos`** é responsável por **sortear e aplicar eventos** com base nas condições do jogo.
- **Cada ambiente tem eventos específicos**, definidos em sua classe correspondente (`AmbienteFloresta`, `AmbienteMontanha`, etc.).
- **Os personagens reagem aos eventos** com base em seus atributos e habilidades especiais.

**Atributos da classe `GerenciadorDeEventos`:**

- **Lista de eventos possíveis:** Contém todos os eventos disponíveis.
- **Probabilidade de ocorrência:** Define a frequência dos eventos.
- **Histórico de eventos:** Evita repetições excessivas.

**Métodos da classe `GerenciadorDeEventos`:**

- `sortearEvento(Ambiente local)`: Escolhe aleatoriamente um evento compatível com o ambiente atual.
- `aplicarEvento(Personagem jogador)`: Executa os efeitos do evento no personagem.
- `removerEvento(Evento evento)`: Se um evento tiver duração limitada, ele pode ser encerrado após alguns turnos.

---

# Ambientes

Os **ambientes** representam os diferentes locais onde o personagem pode se mover, explorar e interagir. Cada **bioma** influencia os **recursos disponíveis, os eventos que podem ocorrer e as dificuldades encontradas**. O design modular dos ambientes permite que novos biomas sejam adicionados sem afetar a estrutura principal do jogo.

Para a implementação dentro dos conceitos de **programação orientada a objetos (POO)**, foi utilizada a abordagem de **herança, polimorfismo e composição**, garantindo que cada ambiente possua características únicas e interaja de maneira dinâmica com o personagem e os eventos do jogo.

**Superclasse: Ambiente**

Todos os ambientes compartilham atributos e comportamentos comuns, sendo modelados por uma **superclasse abstrata `Ambiente`**. Essa classe define **atributos essenciais** e métodos que podem ser sobrescritos pelas subclasses que representam ambientes específicos.

**Atributos da superclasse `Ambiente`:**

- **Nome:** Identificação do ambiente.
- **Descrição:** Texto explicativo sobre as características gerais do local.

- **Dificuldade de exploração:** Define se o ambiente consome mais energia ao ser percorrido.
- **Recursos disponíveis:** Lista de itens que podem ser coletados na área.
- **Probabilidade de eventos:** Define a frequência e o tipo de eventos que ocorrem no ambiente.
- **Condições climáticas predominantes:** Influencia a jogabilidade (exemplo: florestas são úmidas, montanhas podem ser frias, desertos podem ter tempestades de areia).

**Métodos principais:**

- `explorar(Personagem jogador)`: O personagem pode tentar encontrar recursos ou enfrentar desafios no ambiente.
- `gerarEvento()`: Sorteia um evento compatível com o bioma.
- `modificarClima()`: Simula mudanças climáticas no ambiente, impactando a jogabilidade.

**Subclasses de Ambiente**

Cada ambiente possui **características únicas**, influenciando a **sobrevivência do personagem e a disponibilidade de recursos**.

**1. Floresta (`AmbienteFloresta` – Subclasse de `Ambiente`)**

Uma área rica em recursos naturais, mas também habitada por predadores.

**Atributos adicionais:**

- **Vegetação densa:** Reduz visibilidade e dificulta a movimentação.
- **Fauna abundante:** Possibilidade de caça, mas também de ataques de criaturas.
- **Clima úmido:** A umidade dificulta o acendimento de fogueiras.

**Recursos disponíveis:**

- Frutas, raízes e cogumelos (alguns venenosos).
- Madeira para fogueiras e ferramentas.
- Pequenos animais para caça.

**Eventos comuns:**

- Ataque de lobo ou urso.
- Encontro com um explorador perdido.
- Chuva intensa, dificultando a exploração.

**2. Montanha (`AmbienteMontanha` – Subclasse de `Ambiente`)**

Uma região de difícil acesso, mas rica em minérios e pedras preciosas.

**Atributos adicionais:**

- **Terreno acidentado:** Exige mais energia para ser explorado.
- **Clima instável:** Nevascas e ventos fortes podem ocorrer repentinamente.
- **Baixa vegetação:** Pouca disponibilidade de alimentos naturais.

**Recursos disponíveis:**

- Minérios e pedras preciosas.
- Água de degelo, mas precisa ser purificada.
- Refúgios naturais em cavernas.

**Eventos comuns:**

- Nevasca repentina, reduzindo drasticamente a temperatura.
- Deslizamento de pedras, causando ferimentos.
- Descoberta de uma caverna segura.

**3. Caverna (`AmbienteCaverna` – Subclasse de `Ambiente`)**

Um ambiente subterrâneo que pode oferecer abrigo contra o clima, mas esconde perigos desconhecidos.

**Atributos adicionais:**

- **Pouca luz:** Exige lanterna ou tochas para exploração eficiente.
- **Presença de criaturas desconhecidas:** Pode ser um refúgio seguro ou um local perigoso.
- **Água de gotejamento:** Possível fonte de hidratação.

**Recursos disponíveis:**

- Rochas e minérios raros.
- Pequenos lagos subterrâneos (algumas vezes contaminados).
- Ossos e vestígios de exploradores antigos.

**Eventos comuns:**

- Encontro com uma criatura hostil.
- Descoberta de um túnel oculto.
- Desmoronamento parcial, bloqueando saídas.

**4. Lago e Rio (`AmbienteLagoRio` – Subclasse de `Ambiente`)**

Regiões ricas em água, mas que podem esconder riscos como afogamento ou criaturas aquáticas.

**Atributos adicionais:**

- **Água abundante:** Pode ser potável ou precisar de purificação.
- **Possibilidade de pesca:** Peixes podem ser uma excelente fonte de alimento.
- **Terreno lamacento:** Pode dificultar a movimentação.

**Recursos disponíveis:**

- Peixes e algas comestíveis.
- Água doce (algumas vezes contaminada).
- Vegetação ribeirinha útil para fabricação de cordas e armadilhas.

**Eventos comuns:**

- Ataque de criatura aquática (como piranhas ou jacarés).
- Tempestade, aumentando o nível da água.
- Encontro de um barco abandonado.

**5. Ruínas Abandonadas (`AmbienteRuinas` – Subclasse de `Ambiente`)**

Restos de antigas construções que podem conter suprimentos valiosos ou armadilhas.

**Atributos adicionais:**

- **Estruturas instáveis:** O local pode desmoronar a qualquer momento.
- **Presença de outros sobreviventes:** Algumas ruínas podem estar ocupadas.
- **Baixo risco climático:** Normalmente oferecem abrigo contra o clima.

**Recursos disponíveis:**

- Ferramentas antigas e munição.
- Alimentos enlatados ainda comestíveis.
- Mapas e pistas sobre o ambiente ao redor.

**Eventos comuns:**

- Encontrar um grupo de sobreviventes (podem ser aliados ou hostis).
- Armadilhas deixadas por antigos ocupantes.
- Descoberta de uma passagem secreta para outra área.

**Relacionamento entre Ambientes, Personagens e Eventos**

Os ambientes interagem diretamente com **personagens e eventos**, criando um ecossistema dinâmico.

- **Cada ambiente tem eventos específicos** que podem ocorrer apenas em determinadas áreas.
- **Personagens podem ter vantagens ou desvantagens em certos ambientes**, dependendo de suas habilidades.
- **Mudanças climáticas afetam a jogabilidade**, tornando alguns ambientes mais ou menos perigosos com o tempo.

**Atributos da classe `GerenciadorDeAmbientes`:**

- **Lista de ambientes disponíveis:** Define as áreas do jogo.

- **Clima global:** Pode influenciar vários ambientes ao mesmo tempo.
- **Histórico de movimentação:** Registra onde o jogador já esteve.

**Métodos da classe `GerenciadorDeAmbientes`:**

- `mudarAmbiente(Personagem jogador, Ambiente novoAmbiente)`: Move o personagem para uma nova área.
- `gerarEvento(Ambiente local)`: Ativa um evento aleatório com base no ambiente atual.
- `modificarRecursos(Ambiente local)`: Atualiza a quantidade de recursos disponíveis conforme são coletados.

---

## Fluxo do Turno

Cada turno segue um **ciclo de fases**, garantindo que todas as mecânicas do jogo sejam acionadas corretamente.

1. **Fase de Início:**
   - O jogo **exibe o status do personagem** (vida, fome, sede, energia, sanidade, inventário).
   - O ambiente e as condições climáticas são atualizados.
   - O jogador recebe um resumo do que aconteceu no turno anterior.
2. **Fase de Ação:**
   - O jogador escolhe **uma ação principal** para realizar no turno.
   - Algumas ações consomem **energia**, enquanto outras podem recuperar atributos.
3. **Fase de Evento Aleatório:**
   - O sistema verifica se um **evento aleatório** será acionado.
   - Caso ocorra, o evento é **executado e seus efeitos aplicados ao personagem e ao ambiente**.
4. **Fase de Manutenção:**
   - Atributos como **fome, sede e sanidade** são ajustados.
   - Recursos do ambiente podem **se esgotar ou se regenerar**.
   - O turno **avança para o próximo ciclo**.

---

## Condições de Vitória e Derrota

**Condições de Vitória**

1. **Sobrevivência por Tempo Determinado** – O jogador deve sobreviver por um número específico de turnos.
2. **Descoberta de um Refúgio Seguro** – O jogador encontra um abrigo protegido que encerra o jogo com sucesso.

3. **Construção de um Abrigo Permanente** – O jogador acumula recursos suficientes para construir um abrigo estável.
4. **Resgate Bem-Sucedido** – O jogador ativa um pedido de resgate e sobrevive até ser salvo.

**Condições de Derrota**

1. **Vida Chegando a Zero** – O jogador morre devido a ferimentos, ataques ou outros danos.
2. **Morte por Fome ou Sede** – O jogador não consome comida ou água a tempo e morre.
3. **Perda Total de Sanidade** – A sanidade atinge zero, levando a alucinações e decisões fatais.
4. **Esgotamento de Recursos Essenciais** – Sem comida, água ou ferramentas, a sobrevivência se torna impossível.
5. **Captura ou Morte por Outros Sobreviventes** – O jogador é derrotado por grupos hostis ou sobreviventes rivais.

---

# Desafio Opcional: Sistema de Facções e Diplomacia

Adicione **facções de sobreviventes** ao jogo, cada uma com suas próprias regras, objetivos e níveis de hostilidade. O jogador pode **interagir, negociar ou lutar** contra essas facções, impactando diretamente sua sobrevivência.

**Objetivos do Desafio:**

- Criar **diferentes facções** com características únicas.
- Implementar um **sistema de reputação** baseado nas escolhas do jogador.
- Permitir interações como **trocas, alianças ou conflitos** com as facções.
- Adicionar eventos exclusivos que dependam da **relação do jogador com cada grupo**.

**Elementos do Desafio:**

**1. Facções no Jogo**

Cada facção possui características distintas, podendo ser amigável, neutra ou hostil. Exemplos de facções:

- **Nômades Pacíficos** – Oferecem trocas justas, mas evitam combate.
- **Mercadores de Recursos** – Vendem suprimentos raros, mas cobram caro.
- **Caçadores Brutais** – Atacam o jogador se ele estiver fraco.
- **Sobreviventes Desesperados** – Tentam roubar recursos em situações extremas.

**2. Sistema de Reputação**

As ações do jogador afetam a relação com as facções.

- Ajudar uma facção **melhora a reputação** (trocas melhores, acesso a áreas seguras).
- Atacar ou roubar deles **torna a facção hostil** (ataques frequentes, emboscadas).
- As facções **podem se aliar entre si** para caçar o jogador, caso ele seja muito agressivo.

**3. Novos Eventos Baseados em Facções**

- **Pedido de Ajuda:** O jogador pode ajudar uma facção, melhorando sua relação.
- **Emboscada:** Facções hostis podem tentar roubar o inventário do jogador.
- **Oferta de Aliança:** Algumas facções oferecem abrigo ou suporte em troca de lealdade.
- **Traição:** O jogador pode ser enganado por um grupo aparentemente pacífico.

**4. Decisões Estratégicas**

- O jogador pode escolher **se infiltrar ou atacar** certas facções.
- A relação com uma facção pode afetar **como outras facções reagem** ao jogador.
- O sistema pode influenciar **o final do jogo**, permitindo **diferentes desfechos** dependendo da diplomacia escolhida.

---

# Conceitos de Orientação a Objetos

O desenvolvimento do jogo de **Sobrevivência Baseada em Eventos** é estruturado com base nos pilares da **Programação Orientada a Objetos (POO)**, garantindo um **design modular, expansível e eficiente**. O uso de **herança, polimorfismo, encapsulamento e composição** permite criar um sistema dinâmico, onde cada elemento do jogo interage de maneira organizada e previsível.

**Classes e Objetos**

Cada elemento do jogo, como **personagem, ambiente, eventos e itens**, é representado como um **objeto** instanciado a partir de suas respectivas **classes**. Por exemplo, um **lobo selvagem** é uma instância da classe **Criatura**, enquanto um **rio** é uma instância da classe **AmbienteLagoRio**. Essa abordagem modular facilita a organização do código e permite a reutilização de estruturas já criadas.

**Herança**

A estrutura do jogo utiliza **herança** para especializar elementos dentro de categorias principais. Por exemplo, todos os ambientes herdam de uma classe-base `Ambiente`, que define atributos comuns como **nome, dificuldade e recursos disponíveis**. Subclasses como `AmbienteFloresta`,

`AmbienteMontanha` e `AmbienteCaverna` especializam essa classe base, adicionando **características únicas** que impactam a jogabilidade.

Da mesma forma, a classe `Evento` serve como base para eventos específicos como **Eventos Climáticos, Ataques de Criaturas e Descobertas**, permitindo uma implementação organizada e escalável.

**Polimorfismo**

O polimorfismo é amplamente utilizado para **personalizar o comportamento de diferentes elementos do jogo** sem alterar a estrutura principal. Métodos como `executar()` são implementados de forma diferente nas subclasses de eventos, garantindo que um **Ataque de Lobo** tenha efeitos distintos de um **Deslizamento de Rochas**.

Da mesma forma, o método `usar()` pode ser aplicado a diferentes tipos de itens, permitindo que um **alimento recupere fome**, enquanto uma **arma cause dano a um inimigo**, sem que seja necessário reescrever a lógica principal do jogo.

**Encapsulamento**

Atributos e métodos são protegidos para garantir que **o estado interno dos objetos seja modificado apenas por meio de interações controladas**.

- Os atributos de `Personagem`, como **vida, fome, sede e energia**, são privados e só podem ser alterados através de métodos públicos bem definidos, como `comer()`, `beber()` e `descansar()`.
- A classe `Inventario` encapsula a manipulação de itens, garantindo que o jogador **não possa acessar ou modificar diretamente os objetos**, evitando inconsistências na lógica do jogo.

Essa estrutura impede que o jogador manipule **valores críticos de forma indevida**, assegurando que a experiência de jogo siga as regras definidas.

# Desenvolvimento

Os projetos poderão ser realizados em dupla e cada dupla semanalmente determinará a velocidade com a qual irá evoluir os projetos determinando quais entregas fará na semana seguinte. Para isso, será utilizada a plataforma do Trello (veja o Anexo I).

O roteiro de execução do projeto é dividido em vários passos que envolvem assistir aulas, consolidar o aprendizado e desenvolver o projeto. Todo o desenvolvimento do projeto deverá possuir um diagrama que explique o que foi feito com o código salvo num espaço no Github (veja o Anexo II).

Ao assistir as aulas, um relatório individual deve sempre ser gerado. O relatório consiste em um resumo do conteúdo da aula - preferencialmente na estrutura de mapa mental.

A consolidação do aprendizado pode considerar classes criadas em etapas anteriores da própria consolidação.

**IMPORTANTE 1:** As duplas **não** devem fazer o projeto juntas para depois submeter no GitHub. As atividades do projeto devem ser divididas entre cada membro da equipe, de forma que cada um faça sua parte e coloque no repositório para que um outro integrante pegue e evolua. As duplas serão avaliadas pelos *commits* realizados no repositório.

**IMPORTANTE 2:** Existem ao longo do roteiro de execução vários *checkpoints* relacionados ao projeto. A cada checkpoint, o projeto deve ser extraído do Github e ter seu *upload* feito no Google Classroom. Só é possível considerar o avanço na próxima etapa quando os *checkpoints* anteriores tiverem sido realizados.

**IMPORTANTE 3:** as implementações da consolidação do aprendizado devem ser realizadas individualmente. Para isso, deve ser feito o *upload* da consolidação na atividade correspondente do Google Classroom.

**IMPORTANTE 4:** os relatórios de aprendizado requisitados a cada aula devem ser escritos individualmente. Os mesmos deverão ter o *upload* feito na atividade correspondente no Google Classroom.

**IMPORTANTE 5**: A cada semana, deve ser registrado as tarefas e avanços no Trello incluindo a atualização do diagrama de classes. Logo depois deve tirar um print do Trello e armazenado no classroom.

**IMPORTANTE 6**: o aprendizado de um paradigma de programação é não linear. Pense a respeito!

# Roteiro de Desenvolvimento - Jogo de Sobrevivência Baseado em Eventos

### Módulo 1: Introdução à Programação Orientada a Objetos (OO)

1. **Assista à Aula 01 - Introdução a OO:**
   - Escreva sobre o que aprendeu sobre os conceitos fundamentais de POO, incluindo **classes, objetos, encapsulamento, herança e polimorfismo**.
2. **Assista à Aula 02 - Conceitos Básicos de OO:**
   - Escreva sobre a **criação de classes e objetos**, a importância da modelagem de dados e o conceito de estado e comportamento.
3. **Consolidação do Aprendizado 1:**
   - Revise os conceitos aprendidos e aplique-os em **exercícios práticos relacionados ao jogo**.
4. **Consolidação do Aprendizado 2:**
   - Continue praticando conceitos básicos de **OO, classes e métodos** com desafios adicionais.

**Módulo 2: Estruturas de Dados e Modelagem**

5. **Assista à Aula 03 - Strings e Arrays:**
    - Escreva sobre o que aprendeu sobre o uso de **strings e arrays para armazenar e manipular dados** no jogo.
6. **Consolidação do Aprendizado 3:**
    - Pratique manipulando strings e arrays para **gerenciar inventário, recursos e atributos do personagem**.
7. **Consolidação do Aprendizado 4:**
    - Aplique conceitos de strings e arrays em exercícios adicionais.
8. **Assista à Aula 04 - Objetos, Classes e Métodos:**
    - Escreva sobre a **definição de classes, criação de objetos e métodos em Java**.
9. **Projeto: Diagrama de Classes:**
    - Crie um **diagrama de classes** que inclua os principais elementos do jogo: **Personagem, Ambiente, Evento, Item, Criatura**.
10. **Projeto: Identificação de Classes e Atributos:**
    - Liste as classes necessárias para o jogo e identifique seus **atributos e métodos principais**.
11. **Projeto: Métodos de Acesso:**
    - Defina os métodos de acesso para os atributos usando **encapsulamento adequado**.
12. **Projeto: Implementação Inicial:**
    - Implemente a **estrutura básica do jogo**, permitindo a movimentação do personagem entre ambientes.
13. **Consolidação do Aprendizado 5:**
    - Revise o progresso do projeto e integre feedback para melhorias.
14. **CHECKPOINT 1:**
    - Faça o upload do projeto para **revisão e feedback**.

**Módulo 3: Herança e Polimorfismo**

15. **Assista à Aula 05 - Herança:**
    - Escreva sobre a **herança e sua aplicação no jogo**, como subclasses para diferentes tipos de ambientes e eventos.
16. **Projeto: Aplicação de Herança:**
    - Utilize herança para criar subclasses de **Ambiente, Evento e Item**.
17. **Projeto: Polimorfismo em Eventos e Itens:**
    - Implemente **polimorfismo** para definir diferentes comportamentos de eventos e interação com itens.
18. **Projeto: Sistema de Inventário:**
    - Desenvolva um sistema de **inventário para armazenar e manipular itens** no jogo.
19. **Consolidação do Aprendizado 6:**
    - Teste e refine a aplicação de **herança e polimorfismo**.
20. **CHECKPOINT 2:**
    - Faça o upload do projeto atualizado para **revisão e feedback**.

**Módulo 4: Encapsulamento e Estruturação**

21. **Assista à Aula 06 - Encapsulamento, Pacotes e Modificadores:**
    - Escreva sobre o que aprendeu sobre **organização de código em pacotes e uso de modificadores de acesso**.
22. **Projeto: Organização em Pacotes:**
    - Estruture o código separando as classes em pacotes lógicos como `personagens`, `ambientes`, `itens` e `eventos`.
23. **Projeto: Revisão de Código e Boas Práticas:**
    - Revise o código garantindo que **todos os atributos sensíveis estejam encapsulados**.
24. **Consolidação do Aprendizado 7:**
    - Aplique melhorias no projeto com base nos conceitos de **encapsulamento e estruturação**.

**Módulo 5: Sistema de Turnos e Eventos Dinâmicos**

25. **Assista à Aula 07 - Implementação de Turnos:**
    - Escreva sobre **como implementar um sistema de turnos e tomada de decisão** no jogo.
26. **Projeto: Implementação do Sistema de Turnos:**
    - Implemente a estrutura de turnos com **ações do jogador, atualização do ambiente e eventos aleatórios**.
27. **Projeto: Eventos Aleatórios:**
    - Crie eventos aleatórios que **modifiquem o jogo de forma dinâmica**.
28. **Projeto: Mecânica de Sobrevivência:**
    - Adicione a **geração e consumo de recursos essenciais** (comida, água, energia).
29. **Consolidação do Aprendizado 8:**
    - Teste e refine o sistema de **turnos e eventos dinâmicos**.

**Módulo 6: Classes Abstratas, Interfaces e Exceções**

30. **Assista à Aula 08 - Classes Abstratas e Interfaces:**
    - Escreva sobre o que aprendeu sobre **uso de classes abstratas e interfaces** no projeto.
31. **Projeto: Aplicação de Classes Abstratas e Interfaces:**
    - Transforme `Evento` e `Ambiente` em classes abstratas e implemente **interfaces para ações comuns**.
32. **Assista à Aula 09 - Exceções e Tratamento de Erros:**
    - Escreva sobre o **tratamento de exceções** para lidar com erros no jogo.
33. **Projeto: Implementação de Exceções:**
    - Adicione tratamento de erros para evitar problemas como **inventário cheio, ambiente inacessível, morte por fome ou sede**.
34. **CHECKPOINT 3:**
    - Faça o upload do projeto para revisão, incluindo melhorias e ajustes feitos.

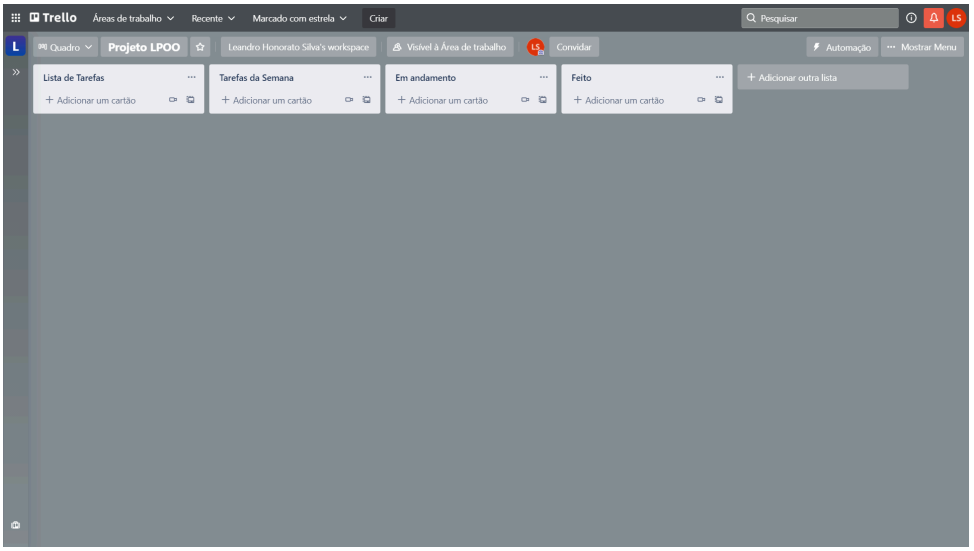
**Módulo 7: Estruturas de Dados Avançadas e Interface Gráfica**

35. **Assista à Aula 10 - Generics e Estruturas de Dados:**
    - Escreva sobre **o uso de listas, filas e coleções para otimizar o gerenciamento de recursos no jogo**.
36. **Projeto: Melhorias no Sistema de Inventário:**
    - Utilize **estruturas de dados avançadas**, como listas ordenadas para organizar os itens.
37. **Projeto: Interface Gráfica (Opcional):**
    - Implemente uma **interface gráfica básica para exibir status do personagem e opções de ações**.
38. **CHECKPOINT FINAL:**
    - Faça o upload final do projeto completo para **avaliação e feedback final**.

# Anexo I

## Trello

O Trello será o nosso ambiente colaborativo para o gerenciamento do projeto. Para isso, cada membro da equipe deve criar uma conta no Trello e criar um Quadro. O nome do Quadro deve ser “LPOO 2024.2 <nome equipe>”. O quadro deve possuir quatro listas, conforme ilustrado abaixo: Lista de tarefas, Tarefas da Semana, Em andamento e Feito.




A Lista de Tarefas deve conter todas as tarefas previstas ao longo do semestre (podem ser adicionadas mais tarefas ao longo da execução do projeto). **Semanalmente, haverá uma reunião de acompanhamento do projeto.** Nesta reunião, o grupo deve apresentar as atividades que foram realizadas na semana que passou (estão na lista Feito) e quais atividades serão realizadas na semana seguinte (Tarefas da Semana).

# Anexo II

## Github

O GitHub é o local onde os códigos desenvolvidos devem ser armazenados. As etapas para criação de um repositório no GitHub são encontradas aqui: Introdução ao GitHub - GitHub Docs.

De início, você não precisa se preocupar com funcionalidades mais avançadas como *branch* (bifurcações). É possível utilizar exclusivamente o *browser* para interagir com o repositório, embora seja mais fácil utilizar um cliente Git (conforme descrito no tutorial).

Para fins de avaliação, será avaliado o registro de *commits* no projeto.

O projeto deve conter um arquivo README, descrevendo as eventuais particularidades do projeto, instruções e demais informações relevantes.