Crie Seu Próprio RPG — Guia Prático (com Varsis como estudo de caso)

VARSIS

Autor: Jonathas Costa da Silva

Sumário

1. Introdução rápida

Este é um manual direto e prático para quem quer criar um RPG de mesa original, simples e divertido — que fuja do D&D tradicional sem perder a essência dos jogos de interpretação. Varsis (mundo pirata) é o exemplo prático, mas todas as regras e estruturas aqui são *genéricas* e adaptáveis para fantasia medieval, sci-fi, cyberpunk, pulp, horror e mais.

Leia capítulo por capítulo e use os apêndices como recursos prontos para imprimir e distribuir à mesa.

---

2. O objetivo deste manual

Ensinar, de forma prática e sem rodeios, como:

- projetar mecânicas claras e fáceis de aplicar;
- gerar progressão significativa (níveis 1→500) sem tornar o jogo chato;
- equilibrar raridade e impacto (poderes que quebram o mundo são raros e custosos);
- adaptar tudo para qualquer tema mantendo consistência.

---

3. Filosofia de design — princípios simples

1. Regra mínima, narrativa máxima. Regras resolvem conflitos; o resto é cena.
2. Fórmulas simples = rápida compreensão + emergente complexidade.
3. Escala ampla controlada por custo e consequência.
4. Modularidade: troque uma curva, não todo o sistema.

---

4. Como adaptar o sistema a qualquer tema

- **Varsis** serve como exemplo: tem piratas, navios e rituais. Para outro tema, mapeie apenas nomes:
    - barco → nave espacial
    - capitão → comandante
    - mar → vazio espacial
    - rituais → tecnologia ancestral

As mecânicas seguem idênticas; apenas troque termos e ganhe flavor.

---

5. Regras fundamentais

Apresente isso no verso da ficha do jogador e na página do mestre.

5.1 Atributos

Cada personagem tem seis atributos base:

- Força, Velocidade, Defesa, Inteligência, Sabedoria, Carisma

Atributos base são inteiros (ex.: 1–50). Para jogos padrão, ofereça 40 pontos para distribuir no início.

5.2 Modificadores

Use a função abaixo para derivar um modificador a partir do atributo base.

```
import math

def calcular_modificador(atributo):
    if atributo <= 1:
        return -5
    elif atributo <= 3:
        return -4
    elif atributo <= 5:
        return -3
    elif atributo <= 7:
        return -2
    elif atributo <= 9:
        return -1
    elif atributo <= 11:
        return 0
    elif atributo <= 13:
        return 1
    elif atributo <= 15:
        return 2
    elif atributo <= 17:
        return 3
    elif atributo <= 19:
        return 4
    elif atributo <= 21:
        return 5
    elif atributo <= 23:
        return 6
    elif atributo <= 25:
        return 7
    elif atributo <= 27:
        return 8
    elif atributo <= 29:
        return 9
    elif atributo <= 31:
        return 10
    elif atributo <= 33:
        return 11
    else:
        return math.ceil((atributo - 33) / 2) + 11
```

5.3 Estatísticas finais

Use os atributos base e o nível para gerar estatísticas finais numéricas:

```
def calcular_atributos(level, forca_base, velocidade_base,
defesa_base, inteligencia_base, sabedoria_base, carisma_base):
    forca = forca_base * 5 + level * 5 + 50
    velocidade = velocidade_base * 5 + level * 5 + 50
    defesa = defesa_base * 5 + level * 5 + 50
    inteligencia = inteligencia_base * 5 + level * 5 + 50
    sabedoria = sabedoria_base * 5 + level * 5 + 50
    carisma = carisma_base * 5 + level * 5 + 50
    return forca, velocidade, defesa, inteligencia, sabedoria, carisma
```

Esses valores são usados para dano, HP e velocidade de turno.

5.4 Vida e recursos

```
def calcular_hp(defesa_final):
    return defesa_final * 4
```

Defina recursos extras (Pontos de Foco, Pontos de Lenda) conforme desejar. Pontos de Lenda são a moeda de feitos épicos.

5.5 Ordem de combate (turnos contínuos)

Sistema simples e fluido:

```
def calcular_turnos(personagens, num_turnos=100):
    for p in personagens:
        p['intervalo'] = 1000 / p['velocidade']
        p['proximo_turno'] = p['intervalo']

    turnos = []
    for _ in range(num_turnos):
        ativo = min(personagens, key=lambda x: x['proximo_turno'])
        tempo = ativo['proximo_turno']
        turnos.append({'personagem': ativo['nome'], 'tempo': tempo})
        ativo['proximo_turno'] += ativo['intervalo']
    return turnos
```

5.6 Dano e resolução

Assuma d20 para checks. Fórmula direta para dano efetivo:

```
def calcular_dano(dado_atq, dado_def, dano, defesa, crit=False):
    if crit:
        return (dado_atq / 20) * dano * 2 - (dado_def / 20) * defesa
    else:
        return (dado_atq / 20) * dano - (dado_def / 20) * defesa
```

Aplique `max(0, dano_efetivo)` e subtraia do HP.

5.7 Testes e perícias

- Use `d20 + modificador` vs DC (difculdade) ou vs `d20 + modificador` do oponente.
- Proficiências: aplique bônus fixo (ex.: +2) ou multiplicador simples.

---

6. Progressão de níveis e tiers

6.1 Tiers

- Iniciado: 1–20
- Veterano: 21–80
- Herói: 81–200
- Lendário: 201–350
- Ascendido/Deificado: 351–500+

6.2 Curva de XP (sugerida)

Para gerar XP_next para nível `n` use fórmulas por tier:

```
def xp_next(n):
    if n <= 20:
        return int(500 * (n ** 1.5))
    elif n <= 80:
        return int(2000 * (n ** 1.6))
    elif n <= 200:
        return int(8000 * (n ** 1.7))
    elif n <= 350:
        return int(30000 * (n ** 1.75))
    else:
        return int(100000 * (n ** 1.8))
```

Gere tabela completa automaticamente e disponibilize ao grupo.

---

7. Recompensas, notoriedade e bounties

7.1 Faixas de recompensa (R$)

- 0 – 100.000: Pirata sem reconhecimento
- 100.000 – 1.000.000: Pirata pouco reconhecido
- 1.000.000 – 30.000.000: Pirata reconhecido em algumas cidades
- 30.000.000 – 100.000.000: Pirata reconhecido em várias cidades
- 100.000.000 – 500.000.000: Pirata reconhecido a nível continental
- 500.000.000 – 1.000.000.000: Pirata reconhecido mundialmente
- 1.000.000.000 – 3.000.000.000: Perigo mundial
- 3.000.000.000 – ∞: Ameaça crítica

7.2 Como bounties afetam o jogo

- Portos podem negar serviços para bounties altos
- Facções montam caçadas com recompensas
- Autoridades aplicam restrições e sanções
- Jogadores podem reduzir bounties com missões políticas/corruptas

---

8. Poderes épicos: ritual, custo e consequência

Sistema pronto: Ritual de Cataclisma.

```
ritual = {
    'artefatos_requeridos': 3,
    'pontos_lenda': 10,
    'nivel_minimo_executor': 450,
    'fases': 3,
}

def executar_ritual(checks):
    sucesso = sum(checks)
    if sucesso == 3:
        return 'sucesso_total'
    elif sucesso >= 1:
        return 'sucesso_parcial'
    else:
        return 'falha_critica'
```

Consequências práticas:

- Sucesso total: efeito catastrófico controlado
- Sucesso parcial: efeito reduzido + fendas
- Falha crítica: repercussão negativa, corrupção ou reversão

Cada execução aumenta o bounty do executor para a faixa "Perigo mundial" ou "Ameaça crítica".

---

9. Criando aventuras que funcionam em qualquer escala

1. Decida o **escopo** (local, regional, continental, planar).
2. Defina 3 objetivos por sessão: derrotar inimigo, ganhar recurso, tomar decisão moral.
3. Para tiers altos, troque "matar inimigo" por "resolver consequência política/ritual".
4. Sempre mostre repercussões: o mundo reage e as decisões têm custos.

---

10. Ferramentas práticas: fichas e templates

10.1 Ficha de jogador (pronta)

```
Nome:
Raça:
Level:
Força(base):
Velocidade(base):
Defesa(base):
Inteligência(base):
Sabedoria(base):
Carisma(base):

Força(final):
Velocidade(final):
Defesa(final):
Inteligência(final):
Sabedoria(final):
Carisma(final):

HP:
Pontos de Foco:
Pontos de Lenda:
Bounty:
Reputação:

Habilidades:

Equipamento:
```

10.2 Ficha NPC (rápida)

```
Nome:
Tipo:
Level:
Atributos(base):
Força(final):
Velocidade(final):
Defesa(final):
HP:
Papel:
Recompensa sugerida:
Notas:
```

---

11. Exemplo de campanha: plano macro 1→500

- Nível 1–10: estabelecer vínculo, contrabando e pequenos conflitos
- Nível 11–50: infiltração, artefatos regionais
- Nível 51–100: confrontos por recursos estratégicos
- Nível 101–250: guerras entre facções e ritos antigos
- Nível 251–400: eventos que mudam geografia e estrutura política
- Nível 401–500+: confrontos com entidades e decisões que moldam o mundo

Inclua ganchos políticos e morais em cada transição de tier.

---

12. Produção do eBook: layout e checklist rápido

- Escreva no formato Markdown mestre
- Gere PDF (A5) e EPUB
- Inclua sumário navegável e links internos
- Geração de capa: título, subtítulo, imagem de impacto
- Validar ortografia e padronização de termos

---

13. Apêndices

13.1 Ranks de Missões (níveis)

- F (Pedra): 0–5
- E (Carvão): 5–15
- D (Bronze): 15–30
- C (Prata): 30–60
- B (Ouro): 60–100
- A (Platina): 100–250
- S (Diamante): 250–500
- H (Herói): 500+

13.2 Scripts prontos (sem comentários)

```
import math

def calcular_modificador(atributo):
    if atributo <= 1:
        return -5
    elif atributo <= 3:
        return -4
    elif atributo <= 5:
        return -3
    elif atributo <= 7:
        return -2
    elif atributo <= 9:
        return -1
    elif atributo <= 11:
        return 0
    elif atributo <= 13:
        return 1
    elif atributo <= 15:
        return 2
    elif atributo <= 17:
        return 3
    elif atributo <= 19:
        return 4
    elif atributo <= 21:
        return 5
    elif atributo <= 23:
        return 6
    elif atributo <= 25:
        return 7
```

```
    elif atributo <= 27:
        return 8
    elif atributo <= 29:
        return 9
    elif atributo <= 31:
        return 10
    elif atributo <= 33:
        return 11
    else:
        return math.ceil((atributo - 33) / 2) + 11


def calcular_atributos(level, forca_base, velocidade_base,
defesa_base, inteligencia_base, sabedoria_base, carisma_base):
    forca = forca_base * 5 + level * 5 + 50
    velocidade = velocidade_base * 5 + level * 5 + 50
    defesa = defesa_base * 5 + level * 5 + 50
    inteligencia = inteligencia_base * 5 + level * 5 + 50
    sabedoria = sabedoria_base * 5 + level * 5 + 50
    carisma = carisma_base * 5 + level * 5 + 50
    return forca, velocidade, defesa, inteligencia, sabedoria, carisma


def calcular_hp(defesa_final):
    return defesa_final * 4


def calcular_dano(dado_atq, dado_def, dano, defesa, crit=False):
    if crit:
        return (dado_atq / 20) * dano * 2 - (dado_def / 20) * defesa
    else:
        return (dado_atq / 20) * dano - (dado_def / 20) * defesa


def calcular_turnos(personagens, num_turnos=100):
    for p in personagens:
        p['intervalo'] = 1000 / p['velocidade']
        p['proximo_turno'] = p['intervalo']
    turnos = []
    for _ in range(num_turnos):
        ativo = min(personagens, key=lambda x: x['proximo_turno'])
        tempo = ativo['proximo_turno']
        turnos.append({'personagem': ativo['nome'], 'tempo': tempo})
        ativo['proximo_turno'] += ativo['intervalo']
    return turnos
```
