NOVUS ORDO

THE ARCHITECT'S ASCENT

Manual de Atualização e Manutenção

Versão 1.0.0 · Novembro de 2026
Documento oficial do projeto

Sumário

1. Política de versões
2. Fluxo de uma atualização
3. Onde alterar cada coisa
4. Criar ou ajustar uma câmara
5. Testes e controle de qualidade
6. Atualizar os manuais e o site
7. Publicar, verificar e reverter no Cloudflare Pages
8. Versionamento no GitHub
9. Compatibilidade de salvamentos
10. Atualizar dependências (Three.js)
11. Checklist de release

1. Política de versões

O projeto segue **Versionamento Semântico** (`MAIOR.MENOR.CORREÇÃO`), registrado no campo `version` do `package.json`.

Tipo	Quando usar	Exemplo
CORREÇÃO (1.0.**1**)	Correção de bug, ajuste de equilíbrio, texto	Alargar um pilar injusto, corrigir colisão
MENOR (1.**1**.0)	Conteúdo ou recurso novo compatível com saves existentes	Nova mecânica, controles por toque
MAIOR (**2**.0.0)	Mudança que quebra saves ou altera a estrutura do jogo	Novas alas, nova estrutura de missões

> Regra de ouro: **uma câmara nunca deve mudar de layout entre versões CORREÇÃO**, pois os jogadores memorizam caminhos. A geração usa sementes fixas por câmara: alterar apenas tamanhos preserva o layout; alterar a ordem das chamadas ao gerador aleatório muda a câmara inteira.

2. Fluxo de uma atualização

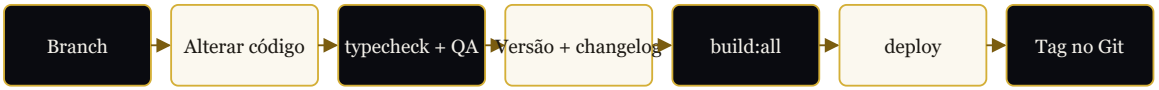


Figura 1 — Do código à produção

```
git checkout -b fix/pilares-largos     # 1. ramo para a mudança
# 2. editar o código em src/
npm run typecheck                      # 3. checagem de tipos
npm run dev                            #    testar em http://localhost:5173/play/
# 4. atualizar version no package.json e o changelog (seção 6)
npm run build:all                      # 5. PDFs + jogo + site em dist/
npm run deploy                         # 6. publicar no Cloudflare Pages
git commit -am "v1.0.1: pilares mais largos" && git tag v1.0.1 && git push --tags   # 7
```

3. Onde alterar cada coisa

Quero mudar...	Arquivo
Nome, descrição, cores, céu, música, mecânicas ou dificuldade de uma câmara	`src/world/levels.ts`
Fragmentos necessários para abrir cada portal	`REQUIRED` em `src/world/levels.ts`
Geometria dos segmentos (vãos, tamanhos, armadilhas)	`src/world/levelBuilder.ts`
Monumento central e decoração de cada tema	`src/world/props.ts`
Grande Átrio (Hub) e portais	`src/world/hub.ts`
Física do personagem (pulos, velocidades, gravidade)	`src/player/player.ts`
Relíquias e quando despertam	`src/player/powers.ts` e `POWER_THRESHOLDS` em `src/core/save.ts`
Diálogos dos NPCs	`attachDialogs` e `hubDialog` em `src/game.ts`
Textos dos pergaminhos	`lore` em `levels.ts` e `GRAND_MASTER_NOTES`
Música e efeitos sonoros	`src/core/audio.ts`
Texturas e modelos	`src/assets/textures.ts` e `src/assets/meshes.ts`
Shaders (água, portal, Visão Astral)	`src/world/shaders.ts`
HUD, menus e estilos do jogo	`src/ui/`
Site oficial	`website/index.html`, `styles.css`, `app.js`
Conteúdo dos manuais em PDF	`scripts/generate-pdfs.js`

4. Criar ou ajustar uma câmara

4.1 Ajustar dificuldade

Em `levels.ts`, o campo `difficulty` (1–10) controla largura das plataformas, vãos, desníveis, velocidade de pêndulos e plataformas e o número de segmentos. O campo `mechanics` define o repertório sorteado; repetir uma mecânica na lista aumenta sua frequência.

4.2 Mecânicas disponíveis

Chave	Efeito	Chave	Efeito
static	Plataforma simples	pendulum	Lâminas pendulares
stairs	Degrau intermediário	sweeper	Laser ou pistão varredor
moving	Plataforma de vaivém	wallkick	Poço entre paredes
vertical	Elevador	falling	Pisos que caem
orbit	Plataformas orbitais	phase	Pontes de luz alternadas
pillars	Pilares	beam	Feixe ascendente
sinking	Chumbo que afunda	wind	Rajadas laterais
enemy	Sentinela patrulheira		

4.3 Adicionar uma câmara nova

1. Acrescente um objeto ao array `RAW` em `levels.ts` e um valor correspondente em `REQUIRED`.
2. Se usar um monumento novo, adicione o tipo em `Centerpiece` e um `case` em `buildCenterpiece` e `scatterProps` (`props.ts`).
3. Posicione o portal no Hub (`hub.ts` , lista `wings`) e atualize os totais de fragmentos (6 por câmara) no HUD e nos menus.
4. Rode o analisador de alcance (seção 5) antes de publicar.

5. Testes e controle de qualidade

5.1 Checagem de tipos

```
npm run typecheck
```

5.2 Analisador de alcance (obrigatório antes de cada release)

O jogo inclui uma ferramenta de QA que reconstrói cada câmara, amostra as plataformas móveis ao longo do tempo e verifica, com a física real do personagem, se o cume, o Caminho Oculto, o selo de pedra, os 8 pergaminhos e os anéis da prova são alcançáveis. Abra `/play/`, pressione F12 e execute no console:

```
await novusOrdoQA({ noFloor: true })   // rota pela espiral, sem atalhos pelo chão
await novusOrdoQA()                    // incluindo o piso sólido
```

Cada item do resultado traz `ok`, a lista `missing` (objetivos inalcançáveis, com coordenadas) e `needTriple` (objetivos que exigem pulo triplo). Para confirmar que o analisador detecta falhas, use os controles negativos `{ noBeams: true }` e `{ shrink: 0.4 }`: eles devem acusar falhas.

5.3 Simulação quadro a quadro

`novusOrdo.frame(1/60)` avança um quadro de jogo; `novusOrdo.renderEnabled = false` desliga a renderização para simular rapidamente. Isso permite roteiros automatizados com teclas simuladas.

5.4 Teste manual mínimo

- Hub: andar até um portal aberto e um fechado.
- Uma câmara de cada grau: pular, pulo triplo, salto em distância, wall kick, Ground Pound no selo, nadar (Atlântida).
- Coletar um fragmento e recarregar a página: o progresso deve persistir.
- Sanctum: os três selos derrotam o Guardião do Véu.

6. Atualizar os manuais e o site

Os PDFs são gerados a partir de `scripts/generate-pdfs.js`. Para registrar uma versão nova:

1. No Manual I, acrescente uma seção ao changelog e ajuste o roadmap.
2. Atualize a versão na capa (função `cover`) e no rodapé do site (`website/index.html`).
3. Rode `npm run build:manuals`: os PDFs são gravados em `public/manuals/` (publicados no site) e em `docs/manuals/` (versionados no Git).

O gerador usa o Edge ou o Chrome instalado. Se nenhum for encontrado, defina `CHROME_PATH` com o caminho do executável.

7. Publicar, verificar e reverter no Cloudflare Pages

```
npx wrangler login     # uma vez por máquina
npm run deploy         # build + upload para o projeto "novus-ordo"
```

Cada publicação gera uma URL própria (`https://<hash>.novus-ordo.pages.dev`) e atualiza a produção em `https://novus-ordo.pages.dev`.

Verificação pós-publicação

- Abrir o site, `/play/` e um PDF em `/manuals/`.
- No console de `/play/`, rodar `await novusOrdoQA({noFloor:true})` e confirmar 18/18.
- Conferir que não há erros no console.

Reverter uma versão com problema

No painel da Cloudflare: **Workers & Pages → novus-ordo → Deployments**, escolha a publicação anterior e use **Rollback to this deployment**. A reversão é imediata e não exige novo build.

8. Versionamento no GitHub

```
git init && git add . && git commit -m "Novus Ordo v1.0.0"
git branch -M main
git remote add origin https://github.com/<usuario>/novus-ordo-game.git
git push -u origin main
```

Item	Convenção
Ramos	`feat/…` recurso, `fix/…` correção, `docs/…` documentação
Commits	Mensagem curta no imperativo, citando a versão quando for release
Tags	`v1.0.1`, `v1.1.0` … uma por publicação
Ignorados	`node_modules/`, `dist/`, `.wrangler/` (recriados por `npm install` e `npm run build`)

Publicação automática (opcional): no painel da Cloudflare, conecte o projeto Pages ao repositório do GitHub com comando de build `npm run build` e diretório de saída `dist`. Cada *push* na `main` passa a publicar sozinho.

9. Compatibilidade de salvamentos

O progresso fica no `localStorage` do navegador, na chave `novus-ordo-save-v1`, com a estrutura: fragmentos por câmara (6 booleanos), pergaminhos por câmara (8 booleanos), melhor contagem de esquadros, flag de final e configurações.

- **Compatível:** acrescentar campos novos (eles recebem valores padrão ao carregar) ou acrescentar câmaras ao fim da lista.
- **Incompatível:** reordenar câmaras, mudar a ordem das missões ou o número de pergaminhos. Nesse caso, crie a chave `novus-ordo-save-v2` e escreva uma migração em `SaveSystem` que leia a v1 e converta os dados.

10. Atualizar dependências (Three.js)

A versão do Three.js está fixada em `package.json` (`"three": "0.170.0"` e `"@types/three"` igual). Para atualizar:

1. Altere as duas versões e rode `npm install --ignore-scripts`.
2. Rode `npm run typecheck` e corrija as mudanças de API indicadas.
3. Confira em `scripts/build-game.js` se as pastas de `examples/jsm` copiadas para `dist/play/vendor` continuam cobrindo os módulos usados (a partir do r171 também existe `three.core.js`, que o script já copia).
4. Teste o jogo (seção 5) antes de publicar.

> **Windows com Controle de Aplicativo:** o build não depende de binários nativos (usa somente `tsc` e Node). Instale sempre com `--ignore-scripts`; ferramentas como Vite, esbuild e rollup nativo podem ser bloqueadas pela política.

11. Checklist de release

✓	Etapa
☐	Mudanças feitas em um ramo próprio
☐	`npm run typecheck` sem erros
☐	Analisador de alcance: 18/18 com `{noFloor:true}`
☐	Teste manual mínimo (seção 5.4)
☐	Versão atualizada em `package.json`, capa dos manuais e rodapé do site
☐	Changelog e roadmap atualizados no Manual I
☐	`npm run build:all` concluído
☐	`npm run deploy` e verificação pós-publicação
☐	Commit, tag `vX.Y.Z` e `git push --tags`