Minegicka — Résumé IA (sept. 2025)

TL;DR (pour une IA qui reprend)

Objectif : porter **Minegicka3 (MC 1.7.10)** vers **Forge 1.20.1 (Java 17)** avec un workflow reproductible (OpenRewrite, Error Prone/Refaster, Semgrep, DataGen). On a un squelette moderne (registries, capability mana, réseau minimal, HUD), un lifecycle de sorts en place, et un pipeline d'assets fonctionnel. Priorités immédiates : 1) **DataGen/Assets baseline** : générer les JSON pour tout le contenu enregistré, éliminer les warnings au lancement. 2) **Capability partout** : remplacer 100 % des usages legacy (PlayersData & co) par **PlayerManaCapability**. 3) **Réseau** : migrer tous les packets **SimpleNetworkWrapper → SimpleChannel** avec handlers explicites + handshake versionné. 4) **Vertical slice gameplay** : 1 sort jouable (ex. *beam*) + 1 entité + renderer + conso/synchro de mana. 5) **Staffs & recettes** : 2–3 staffs complets (NBT, recettes, modèles, équilibrage basique). 6) **Qualité/outillage** : Spotless/EditorConfig, Semgrep, premières recettes OpenRewrite, tests unitaires. 7) **Étendre** : worldgen, blocks spé/BE, HUD/UX.

Definition of Done (DoD) générique : `compileJava` OK → `runData` OK → `runClient` sans warnings d'assets → (si pertinent) `runGameTestServer` vert → commits atomiques propres.

Contexte & objectifs

Portage du mod **Minegicka3 (1.7.10)** vers **Forge 1.20.1**. Approche : d'abord les migrations mécaniques et l'outillage (OpenRewrite/Semgrep/Error Prone), puis l'import progressif *slice par slice*, avec DataGen pour réduire l'édition manuelle d'assets.

État actuel (20/09/2025)

- **Build & runs** : `compileJava` ✅ `runClient` ✅(quelques warnings d'assets tant que la DataGen n'est pas complète), `runData` OK (providers en place).
- **Registries** : couches `DeferredRegister` pour items/blocks/effects/entity types.
- **Capability joueur** : `PlayerManaCapability` + persistance (SavedData) + hooks login/logout/dimension.
- **Réseau/HUD** : `MinegickaNetwork` (paquets essentiels), overlay mana minimal (`ManaOverlayRenderer`), toasts/updates côté client.
- **Cycle de vie des sorts** : enums `SpellStopReason/SpellForm`, contrats `SpellExecutor/SpellContext/SpellInstance`, gestion centralisée via `ServerSpellManager` (start/stop, caches, raisons d'arrêt, ticks).
- **Exécuteurs & consommation** : `SpellExecutors` (soutien/timers), utilitaires `SpellAttributes/SpellContext`, drain de mana via capability, effets consolidés.
- **Assets & DataGen** : Providers (BlockState/ItemModel/LootTable/Language), **pipeline textures** dédié pour les items (≈38 icônes stylisées), script `tools/generate_textures.py` pour régénération.
- **Points ouverts** : warnings Error Prone (ordinals, lookups dépréciés), entités de sorts à porter, sustain à affiner, nettoyage final des maps legacy.

Environnement & conventions

- **OS** : Windows 11. **Java** : 17 pour Forge/port ; JDK 8 pour RFG (audit compilable).
- **Gradle** : utiliser **le wrapper 8.8** dans les projets Forge/RFG. **Ne pas appeler Gradle 9.x** dans ces projets.
- **Chemins de travail** (exemples) :
- Port Forge 1.20.1 : `C:\Users\ulyss\mods\minegicka_port_1_20_1`
- Legacy lecture : `C:\Users\ulyss\dev\Minegicka3`
- Legacy **RFG** : `C:\Users\ulyss\dev\minegicka3-legacy-rfg`
- **Mod ID & package** : `mod_id=minegicka` ; group/package `com.alco.minegickalegacy` ; classe principale `com.alco.minegickalegacy.MinegickaMod` ; garder la cohérence `@Mod("minegicka")/mods.toml/gradle.properties`.
- **Ressources** : `src/main/resources` + `src/generated/resources` (inclus par MDK récent). Préférer `ResourceLocation.tryParse`.
- **Formatage** : `.editorconfig` + (optionnel) Spotless. **Un seul run Gradle à la fois.**

Outils branchés

- **OpenRewrite** : `rewrite.yml` avec recettes 1:1 (packages/types Java/Minecraft), ré-exécutables.
- **Error Prone/Refaster** : activé, prêt pour patchs ciblés (peut nécessiter contournement au bootstrap Forge).
- **Semgrep** : règles pour détecter IEEP/SimpleNetworkWrapper/renders legacy.
- **RFG** (1.7.10) : projet d'audit compilable (plugin GTNH), utile pour cartographier les erreurs concrètes et produire un `rfg-build.log`.

Workflow recommandé (itératif)

1. **Socle Forge** « vierge » + runs (`genIntellijRuns`, `runClient`).
2. **Brancher outillage** : OpenRewrite/Error Prone/Semgrep → `rewriteRun`.
3. **Audit legacy** : greps + (option) build RFG pour sortir une **carte des migrations**.
4. **Import canari** : petite poche de code → `rewriteRun` → `compileJava`.
5. **Garde-fous** : alignements MODID/group/package/mods.toml, `tryParse`, `.editorconfig`.
6. **Premier vertical slice** : 1 item/bloc avec DataGen → `runClient`.
7. **Automatiser** : recettes OpenRewrite/Refaster pour rejouer à grande échelle.
8. **Réseau & Capabilities** : adaptateurs minces → nettoyage legacy.
9. **Rendu/UI & logique serveur** : tester via `runClient` / `runServer` (un seul run).
10. **Itérer slice par slice** jusqu'au cœur du gameplay (worldgen, entités, GUIs, etc.).

Backlog priorisé (prochaines étapes)

- **4.1 DataGen & assets minimum** : modèles JSON items/blocks, lang, loot de base → `runData` sans erreur, `runClient` sans warnings d'assets.
- **4.2 Capability 100 %** : retirer PlayersData/maps globales → la capability couvre tous les appels.
- **4.3 Réseau** : migrer tous les packets en `SimpleChannel`, versioning, ACK/logging, handlers explicites.
- **4.4 Vertical slice gameplay** : porter 1 sort (beam) + 1 entité (`EntityType`, tick, renderer) ; lier à `SpellCasting`/mana/overlay.
- **4.5 Staffs & équilibrage** : sous-classes, stats/NBT, recettes.

- **4.6 DataGen complète** : recipes/tags/loot/lang pour tout le contenu migré ; check CI locale (fail si assets manquants).
- **4.7 HUD & UX** : keybinds (cast/quick-select), HUD mana enrichi, sons/particles, option client.
- **4.8 Qualité & outillage** : Spotless/EditorConfig/Semgrep, recettes OpenRewrite (ex. `GameRegistry.* → DeferredRegister`), tests unitaires (capability, sérialisation, logique mana). Décider du traitement Error Prone au bootstrap.
- **4.9 Worldgen/BE** : BlockEntityRenderers, placed/configured features, biome modifiers.
- **Nouveaux chantiers** : étendre `SpellExecutors` (drain/intervals/spawns d'entités), ajuster sustain, tests d'interruption/répétition, adresser les warnings Error Prone.

Commandes utiles

- **Forge 1.20.1** : `./gradlew genIntellijRuns`, `runClient`, `runServer`, `runData`, `runGameTestServer`, `rewriteRun`, `clean compileJava`.
- **Semgrep** : `semgrep --config .\semgrep-rules\java\port.yml`.
- **RFG** : `./gradlew wrapper`, `tasks`, `build` (génère `rfg-build.log`).

Guardrails & pièges

- Toujours le **wrapper Gradle 8.8** dans Forge/RFG.
- Ne jamais coller le **prompt PowerShell** (évite des erreurs de parsing).
- Fichiers Gradle **sans BOM/ZWSP**.
- Cohérence **MODID/group/package/mods.toml**.
- **ResourceLocation.tryParse** pour éviter warnings.
- **Un seul run** (client/serveur) actif.

« Brief IA » — bloc à copier/coller

- **Contexte** : Migration Minegicka3 (1.7.10) → Forge 1.20.1. Java 17 (port), wrappers Gradle 8.8 (Forge & RFG). RFG dispo pour audit 1.7.10 (JDK 8).
- **Chemins** : `LEGACY=...\Minegicka3`, `RFG=...\minegicka3-legacy-rfg`, `FORGE=...\minegicka_port_1_20_1`.
- **État** : `compileJava` / `runClient` OK, DataGen providers en place, capability mana opérationnelle, réseau minimal, overlay, lifecycle de sorts/execs en place, pipeline textures items (38 icônes) + script de génération.
- **Objectifs immédiats** : 4.1–4.6 (DataGen baseline → capability 100 % → réseau complet → vertical slice → staffs → DataGen complète) + qualité/outillage.
- **Contraintes** : wrapper 8.8 uniquement, pas de Gradle 9.x dans Forge, cohérences MODID/group/package, `.editorconfig`/Spotless, `tryParse`.
- **Livrables** : `runData` propre, `runClient` sans warnings d'assets, packets migrés, 1 sort/entité jouable, 2–3 staffs complets, tests verts.

Tâches que l'IA peut exécuter directement (exemples)

- **Recettes OpenRewrite** : `GameRegistry.* → DeferredRegister`, `cpw.mods.fml → net.minecraftforge.fml`, `ResourceLocation` 1.20.
- **Réseau** : générer/migrer les classes de paquets vers `SimpleChannel` (encode/decode/handler) + handshake versionné, mini-ACK/logging.
- **Capability partout** : proposer des patchs pour remplacer `PlayersData`/maps globales par `PlayerManaCapability` et helpers.

- **DataGen** : compléter `ItemModelProvider/BlockStateProvider/LootTableProvider/LanguageProvider` + exécuter `runData`.
- **Vertical slice** : créer une entité *beam* (type, tick, renderer), relier `SpellCasting` → entité, consommer/synchroniser la mana, valider en jeu.
- **Staffs** : générer 2–3 items (NBT/stats/récupération mana via capability), modèles et recettes basiques.
- **Tests** : GameTests (ex. sérialisation capability), unit tests sur logique mana.
- **HUD/UX** : keybinds (cast/quick-select), options client, messages/lang.
- **CI locale** : tâche qui échoue si assets manquants (vérif de `src/generated/resources`).

Ce document est conçu comme **mémoire opérable** pour copiloter la migration avec une IA : il résume l'état, fixe les conventions, et fournit un plan d'action + DoD pour des itérations *slice par slice*. Mettez-le à jour à chaque jalon (DataGen, réseau, capability, vertical slice, etc.).