Roadmap of New Skills

Phase 1 — RL Foundations (Q-learning, Gymnasium)
Why: Gives you intuition for states/actions/rewards before touching Minecraft.

- Q-learning basics: states, actions, rewards, Q-table updates.
- Implement epsilon-greedy exploration.
- Practice with **Gymnasium** (CartPole, FrozenLake).
- Evaluate policies with metrics (success rate, average steps).

Phase 2 — Contracts & AI Bridge (Schemas + Transport)
Why: Everything else plugs into this.

- **Data formats**: JSON for dev, Protobuf/MsgPack for later performance.
- **Schema design**: versioned Observation, Action, Event, Episode messages.
- **Networking**: WebSockets first; basics of gRPC for future scaling.
- **Concurrency patterns**: async (Python asyncio / Java coroutines).
- Skill exit: Build a ping/pong dummy client ↔ model loop with versioned messages.

Phase 3 — Minecraft Client Control & Vertical Slice (Obs + Act)
Why: Get a loop running with minimal signals + low-level actions.

- **Java modding basics**: Forge/Fabric for in-game control loop, hotkey toggling AI↔human.
- **Game state exposure**: health, hunger, position/orientation, minimal inventory.
- **Low-level action interface**: move, jump, strafe, look deltas.
- **Safe fallback logic**: idle mode when bridge drops.

- Skill exit: AI↔human hotkey works; dummy obs stream + dummy actions at steady cadence.

Phase 4 — Evaluation Harness + Baseline Policy
Why: You need metrics and a reference before doing RL.

- **Minecraft world seeding** and repeatable test environments.
- **Metrics collection**: success %, steps taken, latency.
- **Logging**: structured obs/act/event logs with IDs & timestamps.
- **Baseline scripted policy**: simple heuristic (walk forward if ray clear, else turn).
- Skill exit: Run 10 min with zero stalls; auto-generate eval report.

Phase 5 — Learning & Inference v1 (PyTorch + Mock Env)
Why: Train fast in a mirror environment, then swap in live.

- **PyTorch fundamentals**: tensors, DQN architecture, replay buffers.
- **Config management**: Hydra or YAML configs for reproducibility.
- **Mock environment dev**: simulate obs/act loop using your schemas.
- **Deployment**: connect live Minecraft bridge once mock policy is stable.
- Skill exit: Learned policy outperforms scripted baseline in at least one eval scenario.

Phase 6 — Curriculum + Richer Capabilities
Why: Scale up to survival, combat, crafting with progression.

- **Curriculum design**: navigation → resource collection → survival/combat.
- **Rollout collection**: automate logging and checkpointing.
- **Early stopping rules**: detect when training plateaus.
- **Expand obs & actions**: richer signals (entities, crafting affordances), macro actions (go-to, mine, craft).

- Skill exit: Policy climbs at least one rung of the curriculum with clear metric deltas.

Phase 7 — Tutorial Website (Teaching Track)
Why: Share your work & make it reproducible.

- **Frontend basics**: React or Next.js for structured walkthrough pages.
- **Static site generation**: Markdown docs + starter code downloads.
- **Design skills**: diagrams, gifs/videos, clear explanations.
- **Optional backend**: lightweight API for customization.
- Skill exit: New user can follow guide, install mod, and run the agent.

Phase 8 — Testing & Documentation (Wrap-Up)
Why: Harden everything for deliverables + demos.

- **Unit testing**: Python agent (training loop, action decoder).
- **Integration testing**: Minecraft action responses, schema validation.
- **Load testing**: action bursts at 5–15 Hz without freezing game.
- **Website flow testing**: confirm reproducibility for outsiders.
- Skill exit: Final report + video demo; CI pipeline catches regressions.