Project Adam — Design Document

Submitted By:

1. Muhammad Saad Khan (457796)
2. Muhammad Ali Imran (455280)

Executive summary

Project Adam is a multi-agent simulation that evolves from a small set of simple agents into a complex society. Agents have internal needs, personality traits, and the ability to discover and craft objects. They learn daily behavior with reinforcement learning (RL), make strategic social decisions based on game-theoretic payoffs, and pass traits across generations via genetic algorithms (GA). The system aims to explore emergent social dynamics: cooperation vs cheating, ideology formation, tool discovery, and collective outcomes such as stability or movement toward a "utopia" metric.

Definition:

Project Adam is a tile-based 2D multi-agent environment where agents learn through experience, interact socially, craft items from raw resources, reproduce, and influence global metrics. No behavior is hardcoded beyond primitive actions and perceptual structure; higher-level strategies emerge from learning, evolution, and interaction.

Goals

- **Short-term:** Build a stable, observable simulation for 5–50 agents demonstrating emergent cooperation, tool discovery, and reputation dynamics.
- **Mid-term:** Scale to 100 agents, add richer ideologies, and stable reproduction/evolution cycles.
- **Long-term:** Use the simulated framework as a testbed for robotics transfer and for studying cheating and deception dynamics.

Key concepts and components

World / Environment

- Grid-based 2D tile world (N x N) with various terrain types (tree, stone, water, food).
- Discrete time ticks or turn-based episodes.
- Items: Objects with property vectors (weight, hardness, function tags, durability) storable in inventory or on tiles.

Agents

- **Internal attributes:** name, sex, base curiosity, honesty bias, extroversion, metabolism, learning rate, ideology vector.
- **Dynamic state:** position, hunger, thirst, energy, mood, inventory, discovered recipes, reputation, current goal.
- **Action set (primitive):** move, gather, pick-up/drop, combine(A,B), use, trade, talk, rest, reproduce, attack/defend.
- **Perception:** local grid view (radius r), adjacent item/agent lists.

Knowledge & Tool-making

- No scripted recipes. Agents discover new items via combine(itemA, itemB).
- Property-based crafting: A rule engine determines viable composites (e.g., stick + stone -> hammer) by matching item properties and tags.
- Curiosity parameter increases exploratory crafting attempts.

Ideology & Social beliefs

- Ideology represented as a low-dimensional vector (altruism vs selfishness, aggression vs pacifism).
- Agents' ideologies shift slightly toward partners after interactions, weighted by trust.
- Group formation occurs as clusters in ideology space.

Popularity / Reputation

- An agent-level scalar updated after social events (gifts, trades, cooperation, betrayal).
- Uses short-term (decaying) and long-term (persistent) reputation scores.
- Influences success of social requests, leadership selection, etc.

Global metric: Utopia score

- A monitored metric composed of: violence rate (inverse), resource inequality (Gini), cooperation rate, and happiness average.
- Observational only; not directly part of the agent reward function.

Algorithms and learning

Reinforcement Learning (RL)

- **Goal:** Learn per-agent policies (observation -> action) to maximize lifetime reward.
- **Observation space:** Local tile features, nearby agent reputation, inventory, internal needs, discovered items, traits.
- **Action space:** Discrete set of primitive actions.
- **Reward design:**
    - **Local:** +food, +resource, +need satisfaction, +craft/trade success.
    - **Social:** +/- popularity based on social actions (cooperation, cheating).

    - **Survival:** +for living longer, +reproducing.
    - **Intrinsic:** +curiosity bonus for novel actions or discoveries.
- **Algorithm choices:** PPO (recommended) or DQN. Start with parameter sharing. Consider centralized training with decentralized execution (CTDE).

Game theory & Social decisions

- Model interactions via dynamic payoff matrices (resource state, popularity consequences).
- Use tit-for-tat or trust tracking for repeated interactions to allow reciprocity.
- Cheating: An action with short-term gain but (probabilistic) long-term reputation loss if discovered.

Genetic Algorithms (GA) for evolution

- Determines agent traits at birth (honesty, curiosity, metabolism, aggression).
- Reproduction triggered by age, resources, or social thresholds.
- Child traits: blend parents via crossover + Gaussian mutation.

Implementation: architecture and tech stack

Suggested stack

- **Simulation core:** Python
- **RL libraries:** Stable Baselines3, RLlib, or PyTorch
- **Visualization/UI:** HTML5 canvas + WebSocket, pygame, or React
- **Data storage:** JSON/CSV logging, optional SQLite.

Module boundaries

- Environment module
- Agent module
- RL module
- GA module
- UI module
- Metrics & evaluation module

Tool-making mechanics (details)

- Items as feature vectors: {mass, stiffness, sharpness, tags[]}
- Craft attempts combine(A,B) succeed if a property rule template is matched.
- Example rule: if (stick.length > min && stone.sturdiness > min) -> produce hammer {tag: "strike"}
- Discovery reward for first agent to produce a new functional tag.
- Limited durability encourages maintenance and replacement.

Visualization and debugging

- 2D bird's-eye grid with agent icons and status bars.
- Click-to-inspect agent details (ideology, recipes, history).
- Live charts: Utopia score, population, ideology clusters, popularity distribution.
- Action traces for debugging.

Evaluation and experiments

- Controlled experiments varying: curiosity, punishment strength, resource abundance.
- Metrics: mean lifetime, resource inequality, cooperation/cheating frequency, leader emergence, tool diversity.
- Ablation studies: remove GA, curiosity, or game-theory payoffs to measure impact.

Risks and mitigation

- **Multi-agent instability:** Mitigate with CTDE, reward clipping, population curriculum.
- **Reward hacking:** Monitor policies and favor sparse rewards + intrinsic curiosity.
- **Compute cost:** Use shorter episodes, smaller grids, and fewer agents initially.
- **Interpretability:** Use visualization and logs to inspect emergent behavior.

Appendices

A. Reward formulas (examples)

- survival_reward = 0.1 per tick alive
- eat_reward = +5 on successful eating
- craft_reward = +2 on successful craft; +10 for novel discovery
- help_reward = +3 if action benefits another agent and increases their resource
- cheat_reward = +8 immediate if successful; discovery_penalty = -20
- curiosity_bonus = k / sqrt(1 + n_visits(state))

B. Data structures (compact)

- **Agent:** {id, traits:{...}, state:{...}, inventory:[itemIDs], policy_state}
- **Item:** {id, properties:{...}, durability}
- **Tile:** {type, itemIDs[], agentIDs[]}

Conclusion

Project Adam is ambitious but feasible if built incrementally. Core strengths are that it combines RL, game theory, and GA to explore social emergence and tool discovery. The main technical challenges are reward design and multi-agent stability. With careful stepwise

development and strong visualization, the project will produce valuable insights and a demonstrable prototype.

Bottom line: Build minimal core, validate emergent behaviors, then add complexity in controlled steps.