Scotland Yard

Multi-Agent Mechanism Design

Collective Intelligence – Spring 2026

Teams of 2-3 students | 6-8 weeks

Overview

This assignment extends the Scotland Yard mechanism design project with **three independent research directions**. The codebase provides a working multi-agent RL framework with GNN/MAPPO agents, graph environments, and training infrastructure. Your team will choose **one task** to investigate deeply, implementing new algorithms/architectures/mechanisms and producing rigorous experimental results.

What is already provided

- **Environment**: Scotland Yard on procedural graphs (PettingZoo + TorchRL)
- **Agents**: GNN and MAPPO implementations with action masking
- **Training**: Self-play loops, curriculum learning, reward shaping
- **Infrastructure**: Hydra configs, Docker, visualization, unit tests
- **Evaluation**: Metrics system, exploitability tests, OOD evaluation

Teams & Timeline

- **3 teams** of 2-3 students each
- **Deadline**: 6-8 weeks from start
- **Each team chooses ONE task** from the three options below
- **Tasks are independent** – no inter-team dependencies

Task Options

Task 1: Population-Based Self-Play & Robustness

Research Question: *Are mechanisms trained against diverse opponent populations more robust and less exploitable than single-opponent self-play?*

Hypothesis

Maintaining a population of diverse agents (varying strategies, skill levels) during training produces mechanisms that generalize better to unseen opponents and are harder to exploit.

Implementation Requirements

1. **Population Manager** (`src/training/population_trainer.py`):
   - Maintain pool of 5-10 checkpoint policies for both MrX and Police
   - Implement matchmaking: round-robin, skill-based pairing, or Elo-based
   - Periodic checkpoint saving with diversity metrics
2. **Exploitability Testing** (`src/eval/exploitability.py` extension):
   - Best-response training: freeze target mechanism, optimize attacker
   - Measure win-rate shift under exploitation attempts
   - Compare single-agent vs population-trained exploitability
3. **Diversity Metrics**:
   - Behavioral diversity: action entropy, trajectory clustering
   - Strategic diversity: response to different opponent styles
   - Performance spread: Elo ratings, win-rate distributions

Technical Guidance

- Use `PopulationTrainer` class inheriting from `BaseTrainer`
- Store population in `src/artifacts/populations/`
- Config: `src/configs/training/population.yaml` with pool sizes, matchmaking strategy
- Plot: exploitability curves, diversity metrics over training, population skill distribution

Expected Deliverables

- Working population trainer with 3+ matchmaking strategies
- Exploitability comparison: single-agent SP vs population SP (5+ seeds)
- 4-6 plots: exploitability over training, diversity metrics, Elo distributions, behavioral clustering
- 2-3 GIFs: population agents playing, exploitation attempts
- Analysis: when does population training help? Failure cases?

Task 2: Attention Mechanisms & Architecture Ablations

Research Question: *Do graph attention networks learn better strategic coordination than standard message-passing GNNs?*

Hypothesis

Attention mechanisms allow agents to selectively focus on strategically relevant teammates/opponents, improving sample efficiency and final performance compared to uniform message aggregation.

Implementation Requirements

1. **New Agent Architectures**:
   - `src/agent/gat_agent.py`: Graph Attention Networks (PyTorch Geometric)
   - `src/agent/transformer_agent.py`: Transformer with positional encoding on graphs
   - Follow same interface as `GNNAgent` (inherit from `BaseAgent`)
2. **Ablation Study** (`src/eval/architecture_ablations.py`):
   - Compare: GCN (baseline) vs GAT vs Transformer
   - Vary: number of layers (2-5), hidden dims (64-256), attention heads (1-8)
   - Fixed environment/training config for fair comparison
3. **Attention Visualization**:
   - Extract attention weights at inference time
   - Visualize which agents/nodes receive high attention
   - Correlate attention patterns with strategic events (captures, escapes)

Technical Guidance

- Use `torch_geometric.nn.GATConv` and `TransformerConv`
- Configs: `src/configs/agent/gat.yaml`, `transformer.yaml`
- Attention extraction: hook into `.forward()` or use `return_attention_weights=True`
- Compare on same compute budget (match parameter count or training time)

Expected Deliverables

- 2 new agent implementations (GAT + Transformer) with unit tests
- Ablation results: 3+ architectures × 3+ hyperparameter settings (5 seeds each)
- 4-6 plots: learning curves, sample efficiency, performance vs parameters, attention heatmaps
- 2-3 GIFs: attention-weighted graphs during gameplay
- Analysis: when does attention help? Diminishing returns? Interpretability insights?

Task 3: Multi-Objective Mechanisms (Fairness vs Efficiency)

Research Question: *Can priority edges (metro/ship) enable Pareto-optimal trade-offs between catch efficiency and detective workload fairness?*

Hypothesis

Adding high-speed "metro" edges that MrX can use creates interesting mechanism design trade-offs: faster games but potential unfairness in which detectives get catches. Optimizing both objectives reveals a Pareto frontier.

Implementation Requirements

1. **Priority Edge System** (`src/environment/graph_generator.py` extension):
   - Generate graphs with 10-20% "metro" edges (weight 0.5-1.0 of normal)
   - MrX can use all edges; Police restricted to normal edges (configurable)
   - Visualize priority edges distinctly (different colors/line styles)
2. **Multi-Objective Rewards** (`src/environment/multi_objective_reward.py`):
   - Objective 1 (Efficiency): time-to-catch, total travel distance
   - Objective 2 (Fairness): Gini coefficient of catches per detective, workload balance
   - Scalarization: weighted sum $\alpha \cdot \text{eff} + (1 - \alpha) \cdot \text{fair}$
3. **Pareto Frontier Exploration**:
   - Train mechanisms with $\alpha \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$
   - Compare standard graphs vs metro-enhanced graphs
   - Identify Pareto-optimal configurations

Technical Guidance

- Extend `GraphGenerator.generate()` with `priority_edge_ratio` parameter
- Config: `src/configs/environment/priority_edges.yaml`
- Fairness metrics: Gini coefficient $G = \frac{\sum_{i,j} |c_i - c_j|}{2n \sum_i c_i}$ where c_i = catches by detective i
- Use existing `RewardCalculator` as base, add multi-objective wrapper

Expected Deliverables

- Priority edge generation + visualization (clearly distinguishable in GIFs)
- Multi-objective reward system with 2+ metrics per objective
- Pareto frontier: 5+ α values $\times$ 2 graph types (5 seeds each)
- 4-6 plots: Pareto curves, efficiency-fairness scatter, Gini distributions, catch-per-detective histograms
- 2-3 GIFs: standard vs metro graphs in action
- Analysis: optimal trade-offs? When do priority edges help/hurt? Policy differences?

Shared Requirements (All Tasks)

All teams must follow these common guidelines to ensure quality and reproducibility.

Code Quality

- Follow existing code structure (inherit from base classes)
- Update relevant folder READMEs documenting new modules
- Add 2-3 unit tests for core functionality
- Hydra configs for all experiments

Experiments

- Run on both **small** (5-7 agents, 20-30 nodes) and **large** (15-20 agents, 50-80 nodes) graphs

- Use fixed seeds for reproducibility (report all seeds in README)
- Minimum 5 seeds per experimental condition

Documentation

- Update root `README.md` with new "Semester Contribution" section:
  - Research question & hypothesis
  - Implementation summary (what was added/modified)
  - Key results (embed 2-3 key plots/GIFs)
  - Conclusions & limitations
  - Future work
- Keep it concise (500-800 words max)

Grading Rubric

Total: 100 points

Each task has the same grading structure. All requirements must use Hydra configs and be reproducible.

Implementation (40 pts)

- Core functionality working (new modules/classes follow codebase conventions) **20 pts**
- Integration with existing system (configs, training loops, evaluation) .... **10 pts**
- Code quality (documentation, tests, readable) .... **10 pts**

Experiments (30 pts)

- Rigorous experimental design (controls, baselines, multiple seeds) .... **10 pts**
- Sufficient scale (small + large graphs, 5+ conditions) .... **10 pts**
- Reproducibility (fixed seeds, configs, clear instructions) .... **10 pts**

Results & Analysis (20 pts)

- Clear visualizations (4-6 plots + 2-3 GIFs with captions) .... **10 pts**
- Insightful analysis (hypothesis testing, failure cases, limitations) .... **10 pts**

Documentation (10 pts)

- Semester Contribution in root README (clear, well-structured) .... **6 pts**
- Updated folder READMEs for modified modules .... **4 pts**

Bonus (up to +10 pts)

- Novel extension beyond task requirements (e.g., additional architectures, metrics, theoretical analysis) .... **+5 pts**
- Exceptional results (paper-quality plots, surprising insights, strong baselines) **+5 pts**

Presentation

Brief 10-minute presentation with slides covering: (1) Research question, (2) Implementation highlights, (3) Key results (3-4 plots/GIFs), (4) Failure analysis, (5) Future work. Live demo or pre-recorded GIF required.

Submission

- **GitHub**: Push all code, configs, and updated READMEs to repository
- **Canvas**: Submit PDF slides and link to GitHub branch/PR
- **Artifacts**: Include plots/GIFs in `src/artifacts/semester_contribution/`
- **Team**: Document individual contributions in README

Prepared by: Tamás Takács
Semester: 2025/26/2