BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

EMERGENT COALITIONS IN MULTI-AGENT REINFORCEMENT LEARNING

EMERGENTNÍ KOALICE V MULTI-AGENTNÍM POSILOVANÉM UČENÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MARTIN ŠEVČÍK**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2026

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

169975

Ústav: Ústav inteligentních systémů (UITS)
Student: **Ševčík Martin**
Program: Informační technologie
Název: **Emergentní koalice v multi-agentním posilovaném učení**
Kategorie: Umělá inteligence
Akademický rok: 2025/26

Zadání:

1. Prostudujte problematiku multiagentního posilovaného učení a seznamte se s aktuálními přístupy k implementaci a s praktickými aplikacemi, zahrnujícími kooperaci a konkurenci.
2. Navrhněte multiagentní úlohu v diskrétním prostředí, vhodnou pro experimenty zahrnující kooperaci i konkurenci mezi agenty.
3. Implementujte prostředí pro experimenty. Vyzkoušejte různé strategie učení a nastavení odměn s cílem pozorovat a vyhodnotit projevy koaličního chování.
4. Navrhněte způsob hodnocení experimentů, včetně metrik a vizualizací, umožňujících analýzu chování agentů a průběhu učení.
5. Proveďte experimentální zkoumání vlivu různých parametrů a strategií učení na chování agentů a možné formování koalic, s ohledem na stabilitu, průběh a variabilitu výsledků učení. Výsledky experimentů vyhodnoťte.

Literatura:

- Changxi Zhu, Mehdi Dastani, and Shihan Wang. 2024. A survey of multi-agent deep reinforcement learning with communication. Autonomous Agents and Multi-Agent Systems 38, 1 (Jun 2024).

Při obhajobě semestrální části projektu je požadováno:
První 4 body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Janoušek Vladimír, doc. Ing., Ph.D.**
Vedoucí ústavu: Kočí Radek, Ing., Ph.D.
Datum zadání: 1.11.2025
Termín pro odevzdání: 13.5.2026
Datum schválení: 3.11.2025

Abstract

Multi-agent reinforcement learning asks whether agents can learn to coordinate, divide roles, and form coalitions purely from experience – with no hand-coded behaviors and no explicit communication. This thesis investigates these questions in Knights, Archers, and Zombies (KAZ), a cooperative game extended with boss enemies, wave-based difficulty scaling, an experience-point (XP) leveling system, and shielded zombies that only knights can break, giving the task genuine role asymmetry. The environment is implemented using JAX, a numerical computing library that compiles Python code for GPU execution, enabling efficient parallel training. We train MAPPO across seven reward functions and develop a metrics framework covering spatial structure, attack coordination, behavioral differentiation, and fairness. Distinct and reproducible coalition styles emerge rather than converging to one solution. Across all policies, knight survival and restraint predict episode length more reliably than attack frequency. Reward shaping steers the form coalitions take, but does not create cooperation from nothing – even the unmodified baseline develops coordination through task structure alone.

Abstrakt

Multi-agentní posilované učení (MARL) zkoumá, zda jsou agenti schopni naučit se koordinovat, rozdělit si role a vytvářet koalice čistě na základě zkušeností, a to bez pevně zakódovaných pravidel a bez explicitní komunikace. Tato práce tyto otázky zkoumá v prostředí Knights, Archers, and Zombies (KAZ), kooperativní hře rozšířené o bossové nepřátele, obtížnost škálovanou vlnami, systém zkušenostních bodů (XP) s úrovňovým postupem a oštítované zombie, které mohou zranit pouze rytíři. Prostředí je implementováno pomocí knihovny JAX, která překládá numerický kód pro běh na GPU, což umožňuje efektivní paralelní trénování. Trénujeme politiku MAPPO se sdílením parametrů pro sedm různých funkcí odměny a vyvíjíme sadu metrik zachycujících prostorovou strukturu, koordinaci útoků, behaviorální diferenciaci a spravedlnost. Analyzované politiky produkují odlišné a reprodukovatelné styly koalic. Míra přežití a nečinnosti rytířů předpovídá délku epizody lépe než frekvence útoků. Tvarování odměny určuje formu koalic, nikoliv to, zda vůbec vznikají – i nezmodifikovaná základní politika rozvíjí koordinaci díky samotné struktuře prostředí.

Keywords

multi-agent reinforcement learning, emergent coordination, coalition formation, reward shaping, MAPPO, parameter sharing, Knights Archers Zombies, behavioral differentiation, JAX, proximal policy optimization

Klíčová slova

víceagentní zpětnovazební učení, emergentní koordinace, tvorba koalic, tvarování odměny, MAPPO, sdílení parametrů, Knights Archers Zombies, behaviorální diferenciace, JAX, proximální optimalizace politiky

Reference

ŠEVČÍK, Martin. *Emergent Coalitions in multi-agent reinforcement learning.* Brno, 2026. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Vladimír Janoušek, Ph.D.

Emergent Coalitions in multi-agent reinforcement learning

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of doc. Ing. Vladimír Janoušek Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis. I have used ChatGPT and Grammarly to correct spelling and other language mistakes. I used GitHub Copilot when working on the software.

.....
Martin Ševčík
April 22, 2026

Acknowledgements

I would like to thank my supervisor, doc. Ing. Vladimír Janoušek, Ph.D., for his kind attitude and help throughout the work on this thesis.

Contents

List of Figures

Chapter 1

Introduction

Reinforcement learning is about agents learning to act in an environment purely from experience. *Multi-agent reinforcement learning* (MARL) extends this to settings where multiple agents share the same environment, which makes everything harder – the joint action space grows exponentially, credit assignment across agents becomes difficult, and the environment turns non-stationary as everyone learns at once [30, 2].

But these difficulties are also what make MARL interesting for studying coordination. In a small heterogeneous team, one of the questions is whether agents settle into stable coordination regimes: who protects whom, how risk gets distributed, and what role spatial structure plays in all of it. Think of a football match, where human players rely on communication and predefined roles, but what happens when RL agents have nothing more than a reward signal? What kinds of structures can emerge, and how do you measure them without over-interpreting reward?

This thesis investigates these questions in Knights, Archers, and Zombies (KAZ), a cooperative environment where two knights and two archers must coordinate to survive increasingly difficult waves of enemies. We review reinforcement learning from tabular methods to multi-agent algorithms, then build a custom KAZ training pipeline using MAPPO, implemented in JAX (a numerical computing library that compiles Python code for GPU execution), with multiple parameter-sharing paradigms. Seven reward schemes are compared through a focused metric suite that measures how agents coordinate, how roles depend on each other, and whether grouping behavior actually emerges. The resulting coordination regimes are then tested under changes in observation range, architecture, and environment mechanics.

The main finding is that reward design does not just shift performance up or down. Different reward functions produce genuinely different coordination regimes – a dense mixed-role cluster, a separated frontline-backline formation, an aggressive hybrid – each with its own spatial signature and failure modes. Intuitive coordination proxies like focus fire, behavioral diversity, and spatial cohesion turn out to describe *which* regime a policy occupies rather than how well it actually performs. Across all regimes, the strongest predictor of team durability is role-protective behavior, specifically knight survival, though this dependence weakens once archers can also break shields. An environment modification and a leave-one-out ablation support this more strongly than correlation alone.

With only four agents of two types, the team is too small for nontrivial coalition partitioning in the game-theoretic sense. Throughout this thesis, "coalition" refers to a group of agents that coordinates in space, timing, or contribution toward a shared sub-goal, consistent with the cooperative MARL literature.

The remainder of this thesis is organized as follows. Chapter 2 introduces single-agent reinforcement learning. Chapter 3 extends these ideas to deep RL with function approximation. Chapter 4 covers multi-agent reinforcement learning, MAPPO, and emergent coordination. Chapter 5 describes the environment. Chapter 6 defines the metrics used to measure coordination regimes. Chapters 7 and 8 present the training experiments and their analysis. Chapter 9 concludes with limitations and directions for future work.

Chapter 2

Reinforcement Learning

Reinforcement learning is learning what to do in order to maximize a numerical reward signal. RL starts with a complete, interactive, goal-seeking agent, rather than focusing on isolated subproblems. It considers the problem of an agent interacting with an uncertain environment [26].

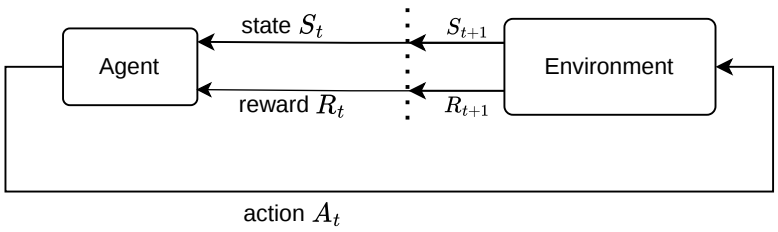

Figure 2.1: The agent-environment interaction in reinforcement learning.

Distinguishing Features in Reinforcement Learning
What makes reinforcement learning so interesting is that, unlike supervised learning, the learner is *not told which actions to take* beforehand, but instead must *discover* which actions yield the most reward by trying them [26]. A cat image classifier must train on thousands of images labeled as 'cat' or 'not cat', whereas if we want to teach AI to be good at chess, instead of labeling every possible combination of moves (which is practically impossible), an RL agent simply plays games and learns from wins and losses. RL agents are not provided a clear "correct answer", only environmental feedback indicating how good/bad an action was [26]. This parallels how humans learn from experience. A child learning to ride a bike receives immediate feedback through balance rather than explicit instructions for every muscle movement. In multi-agent settings, we cannot rely on supervised approaches, because optimal joint behaviors aren't known in advance [30], so to perform optimally, agents must adapt to the environment and discover all sorts of tactical behavior just through experience.

Key Challenges in RL
Consider an agent in a maze. It tries out sequences of actions and eventually finds the exit. But should it keep exploiting that known path, or explore further for a better one?

This is the *exploration-exploitation dilemma* [26]. Each action also changes the state of the environment, affecting what future actions become available. Rewards are often sparse, meaning the agent may go many steps with no signal at all – this is the *temporal credit assignment problem*, where the agent might take a detrimental action at step $t = 0$ but only receive a reward at $t = 100$ [26]. In a cooperative game, for instance, good formation and positioning early on could yield more reward in the long run.

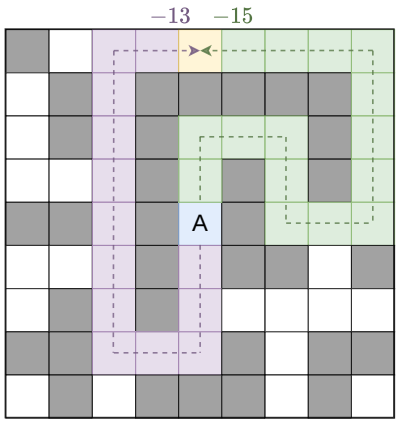

Figure 2.2: Simple example of exploration-exploitation dilemma in a maze game, where for every step agent gets rewarded -1 reward. Agent could take the known green path and get decent reward (exploit), or try a different path (explore), which ultimately leads to a better reward.

2.1 Markov Decision Process

To define a framework in which agents operate, we formalize this with Markov Decision Processes (MDPs). An *MDP* is a tuple $\langle S, A, P, R, \gamma \rangle$ where S is the state space, A is the action space, $P(s'|s, a)$ is the transition probability, $R(s, a)$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor [26]. At each step, the agent receives a representation of the environment's state $S_t \in S$, selects an action $A_t \in A$, and receives a reward R_t and a new state S_{t+1} as consequence. The agent's sole objective is to learn a policy $\pi(a|s)$ that maximizes expected cumulative discounted reward [26]:

$$\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \right]$$

The key assumption here is that all future states depend only on current S_t and A_t, not on the history of past states; this is known as the *Markov Property* [26]. Everything outside the agent's direct control, including other agents' behaviors, is considered part of the environment. However, in multi-agent settings with centralized training (CTDE discussed in Section 4.3.1), critics might have access to privileged joint state information during learning.

2.2 Value Functions and Policies

An RL agent needs a way to measure how good a situation is, and a rule for what to do next. Therefore, this section defines the foundations of every algorithm in reinforcement learning.

Returns and Discounting

Return G_t is the sum of future rewards an agent expects to receive from time step t onward [1]. In episodic tasks, this can be the finite-horizon undiscounted return $G_t = \sum_{k=0}^{T-t-1} R_{t+k}$, where T is the terminal time step [1]. The discount factor (γ) is fundamental, because it has a strong influence on agent's behavior in multi-agent settings, discussed later in Section 4.5.2.

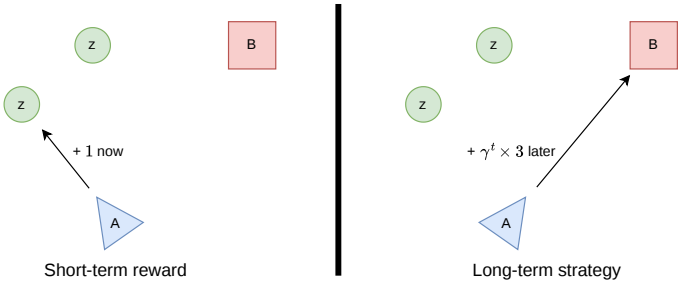

Figure 2.3: Effect of discounting on agent behaviour in KAZ. Left: the agent targets a nearby zombie for an immediate reward of $+1$. Right: the agent instead commits to pursuing the boss, earning $+\gamma^t \times 3$ at a future time step t. A low γ makes the agent prefer the immediate zombie kill; a high γ makes the harder, delayed boss kill worthwhile.

Value Functions

Value functions estimate expected returns from states or state-action pairs under a policy [1]. The *state-value function* $V^\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$ gives the expected return from state s following policy π [1]. The *action-value function* $Q^\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$ gives the expected return from taking action a in state s, then following π [1]. These satisfy the *Bellman equations*, expressing that a state's value equals immediate reward plus discounted next-state value [26, 1]. In MARL, individual value functions require credit assignment to decompose global rewards [30].

Since V^π and Q^π depend on the policy, different policies produce different value functions. The *optimal value function* $V^*(s) = \max_\pi V^\pi(s)$ gives the highest achievable expected return from each state, taken over all possible policies [26]. Any policy that achieves V^* everywhere is an *optimal policy* π^*. Finding π^* (or a good approximation of it) is the central objective of reinforcement learning [26].

Policy Representation

A policy π maps states to actions [1]. Policies can be deterministic ($\mu : \mathcal{S} \to \mathcal{A}$) or stochastic ($\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$) [1]. Discrete action spaces use categorical policies with softmax outputs, while continuous spaces use diagonal Gaussian policies [1]. MARL agents may share parameters or maintain separate policies [30, 12].

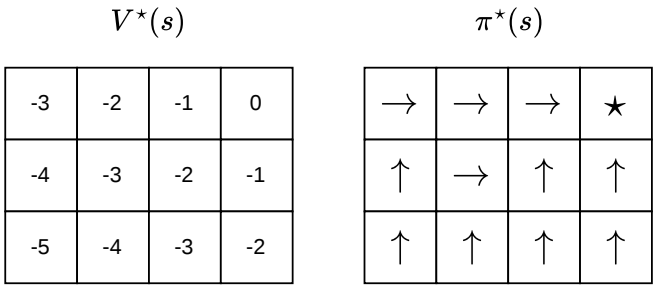

Figure 2.4: Optimal value function and policy for simple grid game.

2.3 Tabular Solution Methods

In an ideal world, where we know exactly every transition probability $P(s'|s,a)$ and reward function $R(s,a)$, we could compute optimal policies using Dynamic Programming. However, DP requires complete knowledge of the environment and scales poorly as state spaces grow [26]. For practical RL, we must instead rely on *model-free methods* that learn from trial-and-error interaction.

Monte Carlo Methods
When the environment's model is unknown, the agents have to estimate values by sampling their trajectories [26]. This leads us to Monte Carlo methods that solve this by letting the agent play out the entire episodes from start to finish and using the actual empirical return G_t to update the values of the states visited [26]. The update rule simply shifts the current value estimate towards the observed return:

$$V(S_t) \leftarrow V(S_t) + \alpha\big[G_t - V(S_t)\big],$$

where α is a constant step-size parameter or learning rate [26]. Because MC methods do not *bootstrap* – meaning they do not rely on other learned value estimates to update their current one – their value estimates are entirely unbiased [26]. However, because G_t tracks every single random choice over a long period, MC updates suffer from severely high variance [26]. In addition, the agent has to wait until the end of the episode before it can make any learning updates [26]. That is where TD-learning is introduced.

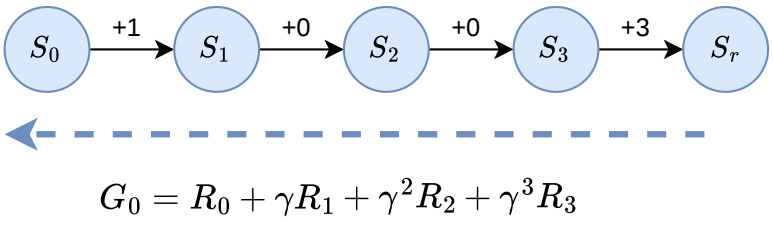

Figure 2.5: Abstract visualization of how the return G_t is computed.

TD-Learning

Temporal-Difference (TD) learning resolves the problem of waiting until the episode terminates by bootstrapping [26]. What bootstrapping does is that it substitutes the true return by updating value estimates *online after every single step* [26]. It uses the immediate reward and the current value estimate of the next state to form a proxy for the actual return [26]. The simplest variant, known as TD(0), updates the state-value function as follows:

$$V(S_t) \leftarrow V(S_t) + \alpha \underbrace{\Big[\underbrace{R_t + \gamma\, V(S_{t+1})}_{\text{TD target}} - V(S_t)\Big]}_{\delta_t \text{ (TD error)}}$$

where α is the learning rate. The TD error δ_t is the continuous correction signal, measuring the difference between what the agent predicted and what it actually observed one step later [26]. When $\delta_t > 0$, the outcome was better than predicted, so the value estimate is nudged upward, and when $\delta_t < 0$, the reverse applies [26].

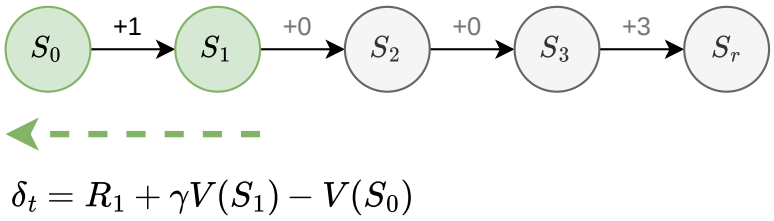

Figure 2.6: TD error δ_t computed from just two consecutive states (green). Unlike Monte Carlo, the rest of the episode (grey) is not needed.

Q-Learning

Now Q-Learning extends TD-learning where instead of simply evaluating policies, it *directly finds optimal actions* by learning the action-value function $Q(s, a)$ [26]. Q-learning applies the following update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_t + \gamma \underbrace{\max_{a'} Q(S_{t+1}, a')}_{\text{greedy next value}} - Q(S_t, A_t)\Big].$$

The max operator evaluates the best possible action, assuming the agent will act optimally from the next state onward [26].

This brings up a key distinction in reinforcement learning: *on-policy* versus *off-policy* methods. On-policy methods (like SARSA or later, PPO) learn the value of the exact policy the agent is currently executing. Off-policy methods, however, separate the behavior policy used to explore the environment from the target policy being learned [26]. Q-learning is strictly off-policy because the max operator always evaluates the greedy (optimal) target policy, regardless of which behavior policy the agent actually follows for exploration. This separation allows the agent to explore freely while still converging to the optimal action-value function Q^* [26].

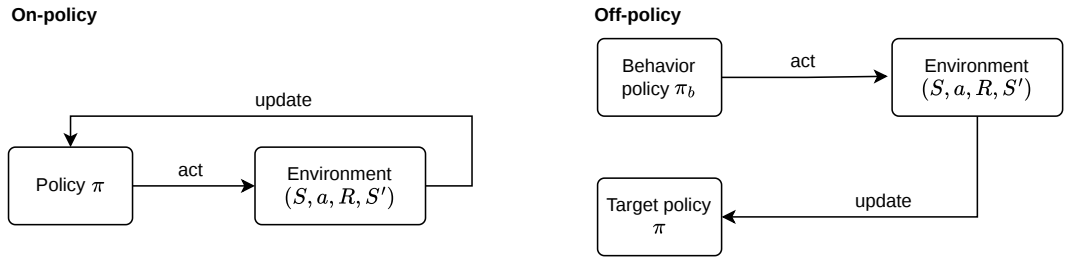

Figure 2.7: Diagram showing distinction between on-policy and off-policy methods.

The TD error and Q-learning update rule form the foundation used in virtually every modern algorithm discussed in this thesis. However, these tabular methods cannot scale to environments with large or continuous state spaces, which motivates the introduction of function approximation in the following chapter.

Chapter 3

Deep Reinforcement Learning

Classical reinforcement learning often represents value functions in tabular form, but as state and action spaces grow from discrete settings to high-dimensional continuous domains, storing all value estimates in lookup tables becomes computationally impossible [26]. The solution is to approximate value functions using parametric models, and neural networks have become the standard *function approximators* for these larger problems [1]. Formally, instead of storing $v_\pi(s)$ for every state s, we introduce an approximate value function $\hat{v}(s, w)$ with a weight vector $w \in \mathbb{R}^d$ so that $\hat{v}(s, w) \approx v_\pi(s)$ [26]. Since the number of parameters d is typically much lower than the number of possible states, updating $\hat{v}$ for one state also indirectly changes the value estimates for many similar states; this is a form of *generalization* [26].

3.1 Neural Networks as Function Approximators

A deep neural network can approximate value functions or policies across vast state spaces without explicitly storing each state-action pair [19]. Apart from scaling to very high-dimensional inputs, neural networks in particular are great at representing highly non-linear relationships between states and values or policies [1]. They also naturally support feature learning, where networks learn intermediate representations that are useful for predicting returns or action advantages [19]. In multi-agent environments with continuous and combinatorial observations, deep neural networks are necessary because the joint state of all agents cannot feasibly be represented in a tabular value function.

Deep Q-Network (DQN)
A big breakthrough in the 2010s was the example of Deep Q-Network (DQN), which showed that a convolutional neural network can approximate the action-value function $Q(s, a)$ directly from raw Atari game pixels and achieve human-level performance across many games [19]. The network takes a stack of 4 recent grayscale frames as input and outputs estimated Q-value for each possible action, while trying to minimize the temporal difference target [19]. DQN stabilizes learning with two techniques: *experience replay*, which randomly samples past transitions to break correlation in the observation sequence, and a separate *target network*, which periodically freezes Q-value estimates to provide a stable learning target [19]. This is the core update rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left(R_t + \gamma \max_{a'} \hat{Q}(S_{t+1}, a'; \mathbf{w}^-) - \hat{Q}(S_t, A_t; \mathbf{w}_t) \right) \nabla_{\mathbf{w}} \hat{Q}(S_t, A_t; \mathbf{w}_t)$$

where α is the learning rate and the network is updated using stochastic gradient descent [19]. DQN demonstrated the viability of combining deep learning with RL and inspired many subsequent advances in both value-based and policy-based methods.

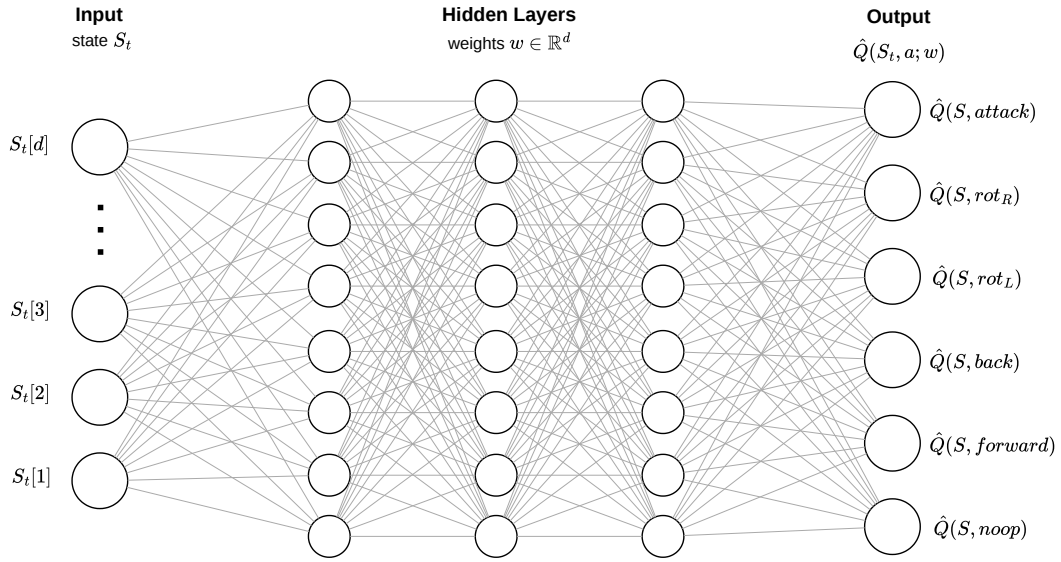

Figure 3.1: Simple abstraction of neural network used for estimating $\hat{Q}(S_t, a)$.

3.2 Policy Gradient Methods

There is also another set of methods that are essential to understand; particularly methods that instead of learning an action-value function $Q_\theta(s, a)$ like DQN, these methods learn the *policy directly* as $\pi_\theta(a \mid s)$ and optimize expected return $J(\theta)$ by gradient ascent on the policy parameters [26].

Because it is not possible to directly backpropagate through the environment's state transitions due to unknown dynamics, the gradient of the performance must be reformulated [1]. By applying the Policy Gradient Theorem, the gradient can be computed purely from sampled trajectories without requiring the derivative of the environment's transition probabilities [26]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \underbrace{\nabla_\theta \log \pi_\theta(A_t \mid S_t)}_{\text{push direction}} \underbrace{Q^{\pi_\theta}(S_t, A_t)}_{\text{scale by return}}\right].$$

Actions with high returns are reinforced; those with low returns are suppressed [1].

REINFORCE and Advantage

The foundational policy gradient algorithm is REINFORCE, which uses the actual observed episodic returns (Monte Carlo style) to estimate the true action-value function [26]. The agent plays out a trajectory and then increases the log-probability of actions proportional to the total return earned [26]. While conceptually simple and unbiased, relying on full episodic returns causes severe variance, leading to slow and unstable learning [26].

To stabilize training, a standard improvement is to subtract a baseline value function $V^{\pi_\theta}(S_t)$ from the return, arriving at the *advantage function* [26]:

$$A^{\pi_\theta}(S_t, A_t) = \underbrace{Q^{\pi_\theta}(S_t, A_t)}_{\text{this action}} - \underbrace{V^{\pi_\theta}(S_t)}_{\text{state average}},$$

This makes gradient updates focus on relative improvements and is the standard for data efficiency and stability in modern RL algorithms [26, 1].

However, a key practical question remains: how do we obtain $V^{\pi_\theta}(S_t)$ to compute the advantage, and must we still wait for full episode rollouts like in REINFORCE [26]? To address both questions simultaneously, let's introduce actor-critic methods.

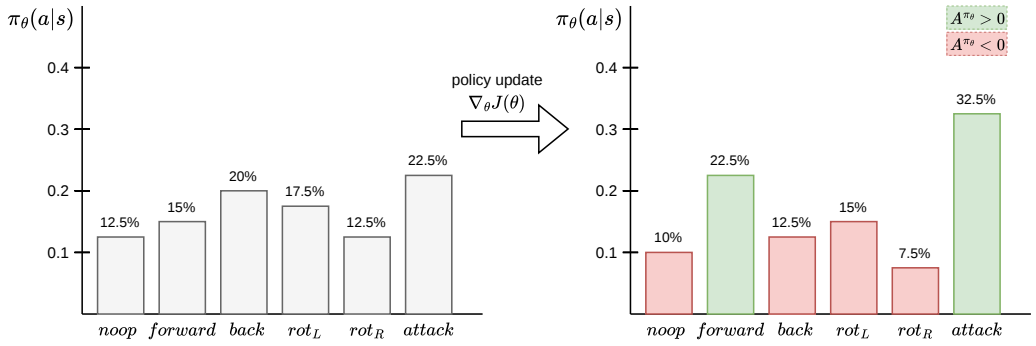

Figure 3.2: Policy gradient update over the six KAZ actions. Actions with positive advantage $A^{\pi_\theta} > 0$ (green) gain probability mass; those with negative advantage (red) lose it.

3.3 Actor-Critic Methods

Actor-critic methods sit between pure value-based methods and pure policy-gradient methods by maintaining two separate function approximators: a *policy* (actor) and a *value function* (critic) [26].

Architecture and Roles

The actor is a parametrized policy $\pi_\theta(a \mid s)$ that maps states to a distribution over actions and is the component responsible for making decisions [26]. The critic is a parametrized value function, usually a state-value $V_w(s)$ or an action-value $Q_w(s, a)$, that predicts the expected return when following the current policy [26, 30]. Basically, the actor *proposes* behavior and the critic *judges* it by summarizing the performance of the policy into a scalar value [30]. This architecture differs from pure value-based methods, where the policy is implicitly 'greedy' with respect to a learned Q-function [26]. In actor-critic frameworks, the policy and value function are conceptually independent of each other and are represented by two distinct neural networks. However, in deep RL practice, they generally share a common feature extractor with separate output heads for the policy and value [1].

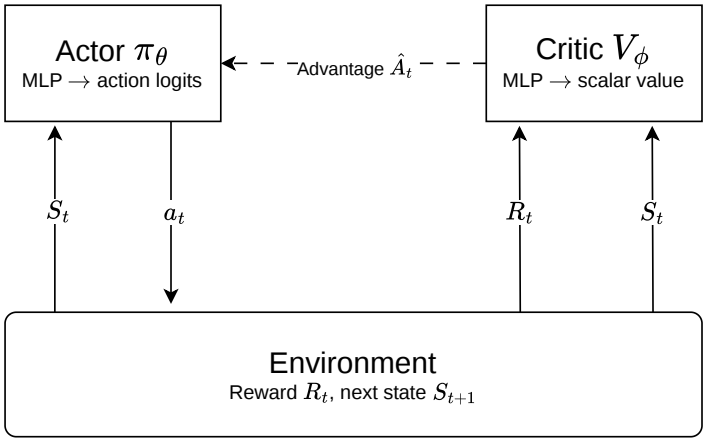

Figure 3.3: Actor-critic architecture. The actor π_θ selects actions from state S_t; the critic V_ϕ evaluates them by computing the advantage $\hat{A}_t$ from the environment reward, which is fed back to guide the actor's updates.

Learning Signal
The actor uses the TD error δ_t directly as a learning signal: if $\delta_t > 0$, the taken action was better than expected, so the actor increases the probability of similar actions; if $\delta_t < 0$, it decreases it [26]. This way, the critic effectively provides an online advantage estimate without waiting for full episode returns, which leads to faster and more stable learning overall. However, this could introduce bias if the critic's signal is imperfect [13, 1]. Most modern deep RL algorithms – such as A2C/A3C, DDPG, TD3, TRPO, and PPO – all share this actor-critic backbone, although with different architectural additions and design choices [1].

3.4 Proximal Policy Optimization

The algorithm chosen for this environment and thesis scenario is MAPPO, which is a multi-agent extension of Proximal Policy Optimization (PPO) [29]. Before adding ideas and extensions from the multi-agent layer, it is helpful to first thoroughly explain its fundamental algorithm. PPO builds on the actor-critic framework [13] but with an innovation, which is preventing policy from changing *too drastically* in a single update, leading to significantly more stable training [24].

3.4.1 The Policy Update Problem

Imagine an agent that has been exploring a maze and has finally learned a decent path to the exit. Now the standard policy gradient update comes, and it happens to be too large, and completely diverges the agent's policy, making it run into walls again. This is a problem because our collected data were generated under the old policy, so large policy changes make the updates unreliable and can cause sudden training collapse. PPO solves this with a simple clipping mechanism [24].

The Probability Ratio and Clipping

To take clipping into effect, we first determine *how much* the policy has changed. PPO does this through the *probability ratio*, which compares the probability of taking the same action under the new policy versus the old one:

$$r_t(\theta) = \frac{\pi_\theta(A_t \mid S_t)}{\pi_{\theta_{\text{old}}}(A_t \mid S_t)}.$$

When $r_t(\theta) = 1$, the policies are identical for action A_t; values above or below 1 indicate the new policy makes A_t more or less likely. Then, PPO takes this ratio and clips it to stay within $[1-\epsilon, 1+\epsilon]$ (typically $\epsilon \approx 0.2$), so once the policy has changed noticeably, further updates in that direction do not improve the objective. The clipped surrogate is

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\,\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t\Big)\Big],$$

where $\hat{A}_t$ is the advantage estimate and the *min* ensures conservative updates [24]. Because clipping constrains how much the policy can change, the same rollout can be safely reused for multiple epochs of minibatch gradient descent without the updates becoming too stale or destructive, making PPO significantly more sample-efficient than classic policy gradient methods [1].

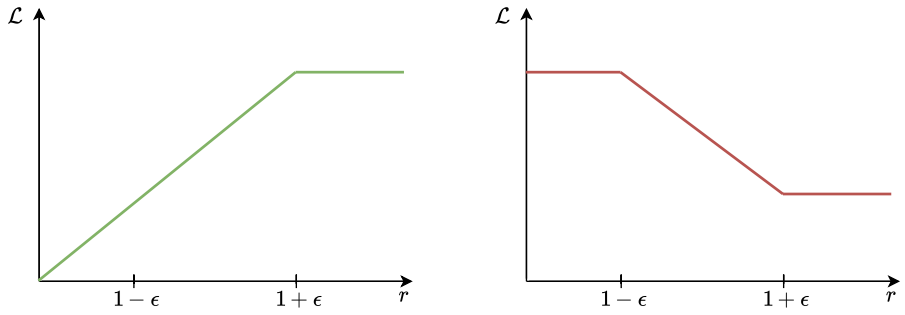

Figure 3.4: PPO clipping for positive advantage $\hat{A} > 0$ (left, green) and negative advantage $\hat{A} < 0$ (right, red). Once the probability ratio r leaves $[1-\epsilon,\ 1+\epsilon]$, the objective flattens, blocking further updates in that direction.

3.4.2 Generalized Advantage Estimation

To compute the advantage $\hat{A}_t$, PPO uses Generalized Advantage Estimation (GAE), that finds a middle ground by blending TD errors across multiple time steps with an exponential decay [23]:

$$\hat{A}_t = \sum_{l=0}^{T-1-t} (\gamma\lambda)^l\, \delta_{t+l},$$

where $\delta_t = R_t + \gamma V_\phi(S_{t+1}) - V_\phi(S_t)$ is the TD error, $\lambda \in [0, 1]$ balances bias and variance; when $\lambda = 1$ it becomes the full Monte Carlo return (in practice, $\lambda \approx 0.95$ is used) [23]. The critic is then trained to minimize the squared error between predictions $V_\phi(S_t)$ and the target returns $\hat{R}_t = \hat{A}_t + V_\phi(S_t)$ [24]. MAPPO directly builds on this foundation by extending it to the multi-agent setting.

Chapter 4

Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) extends RL to settings where multiple agents operate in a shared environment. This raises problems that barely exist in standard RL: the environment becomes non-stationary from each agent's perspective as others update their policies concurrently, and credit assignment across agents grows difficult [2, 30]. But MARL also opens up genuinely interesting phenomena – how agents communicate, how *coordinated strategies can emerge* from simple reward signals. In a football game, explicit communication between players is essential for optimal play, but what happens when the players are RL agents with nothing more than a reward function? Before addressing such questions, it is useful to see how multi-agent environments are defined in game-theoretic terms.

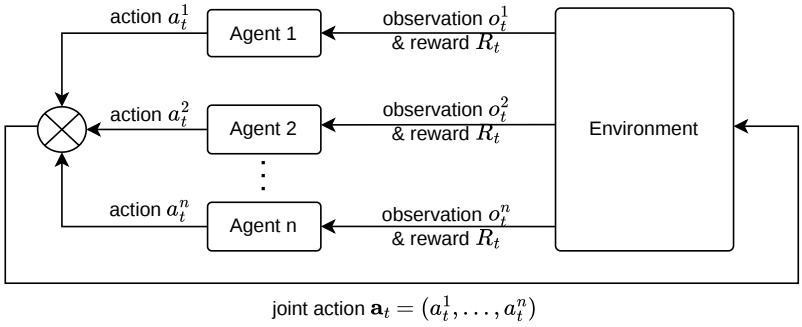

Figure 4.1: Schematic of multi-agent reinforcement learning. Joint actions modify environment state.

4.1 Types of Games

A multi-agent environment has many connections to game theory, especially in how these environments are formally modeled as types of games [2]. At the one-step level, these reduce to the simplest normal-form (matrix) games. For environments that have sequential interaction, these are known as stochastic or Markov games. Markov games are a multi-agent generalization of Markov decision processes where several players choose actions,

receive (possibly different) rewards, and jointly influence the state transition dynamics [16, 2].

An important distinction is between different payoff structures (types of games). Games like chess are competitive *zero-sum games* because from one player's perspective you either get +1 reward for a win or -1 reward for a loss, therefore the sum of possible rewards is 0. In a cooperative environment where all share the same reward, this is known as a *common reward game* [2]. There are also *general-sum games*, where agents have individual reward functions that are neither strictly opposed nor perfectly aligned [2]. But sometimes, the agent cannot see the entire playing field or exactly every player at every step. This is known as *partial observability* and leads to *partially observable stochastic games* (POSGs) and, in fully cooperative common reward games, to *decentralized partially observable MDPs* (Dec-POMDPs), which are the usual formal model for cooperative MARL tasks [20, 2].

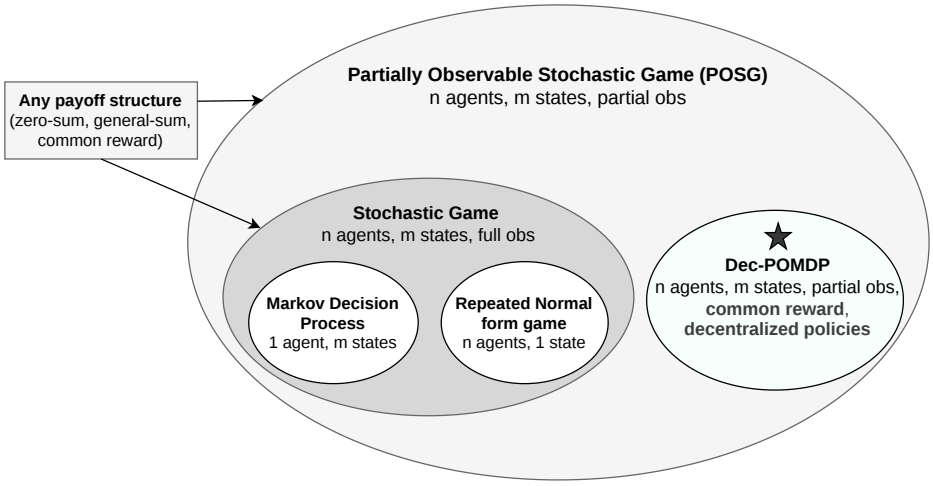

Figure 4.2: Hierarchy of game models. POSGs include stochastic games as a special case, where both can be of any payoff structure. Dec-POMDP is another special case with common reward payoff structure and decentralized observation policies, and is the natural starting formal model for the original KAZ mechanics when all agents share a team reward.

In this thesis, the transition dynamics, observations, and termination conditions stay fixed across experiments, but the reward function is varied. Formally, this means the underlying game mechanics stay the same while the payoff structure changes: some schemes remain strongly cooperative, while others introduce more individualized or mixed-motive incentives. So Dec-POMDP is the right starting model for the base task, but not every reward scheme used later stays common-reward in the strict sense.

4.2 The Multi-Agent Problem

In single-agent RL, the environment is usually modeled as a stationary MDP. In MARL, this assumption breaks down because other learning agents are part of the environment – transition and reward distributions shift as agents update their policies concurrently. This *non-stationarity* makes even simple independent algorithms unstable, especially in general-sum settings where agents' objectives are not aligned [2, 30].

The second issue is scalability. With 2 agents each having 6 actions, the joint action space is $6^2 = 36$; with 4 agents it grows to $6^4 = 1{,}296$. This interlocks with the *multi-agent credit assignment problem*: how to determine which agents and actions were responsible for a change in returns when a single global reward provides sparse and entangled feedback [2].

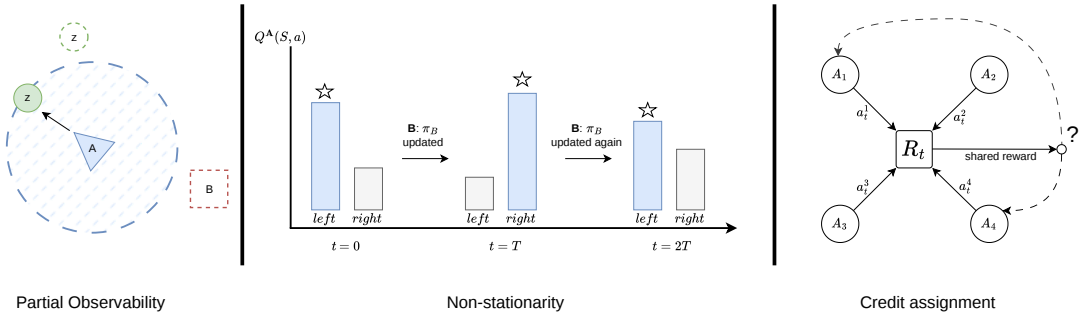

Figure 4.3: Three MARL problems. Left: *partial observability* – agent A targets a zombie in its vision radius while a boss outside escapes undetected. Centre: *non-stationarity* – A's optimal $Q^A(s,a)$ shifts each time B's policy updates. Right: *credit assignment* – four agents produce a shared reward R_t but no signal indicates which action caused it.

Partial observability compounds both problems. When agents can only access local information, finding exact equilibria is often computationally intractable, and agents struggle to distinguish environmental noise from the evolving strategies of teammates [2].

Finally, MARL often involves social dilemma settings where agents face competitive tendencies to maximize individual reward rather than collective welfare, getting trapped in inefficient outcomes even when mutual coordination would pay off. Unlike the classic Prisoner's Dilemma, these are not one-off choices but play out as complex, long-term policies [15].

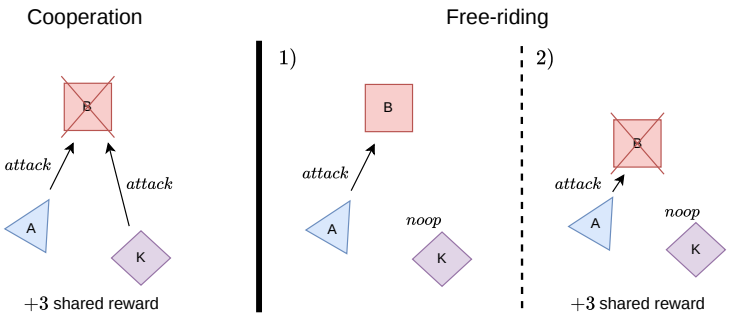

Figure 4.4: Stylized mixed-incentive setting. Coupled rewards can blur credit assignment by letting one agent benefit from an engagement without matching the other agent's contribution exactly, even in an otherwise cooperative task.

Avoiding these problems requires learning mechanisms that go beyond simple, short-term reward maximization [2].

4.3 Solution Concepts and Algorithms

Now considering all these problems we've defined, how should the agents be trained and deployed? Modern cooperative MARL approaches consist of three training schemes: *centralized training and execution* (CTE), *decentralized training and execution* (DTDE), and *centralized training with decentralized execution* (CTDE) [2].

4.3.1 Centralized Training with Decentralized Execution (CTDE)

CTDE has become the dominant paradigm in modern MARL. During training, agents are optimized using centralized or "privileged" information, but at deployment each agent acts using only its local observation. This split addresses non-stationarity and credit assignment while preserving decentralized control at test time [2].

CTDE methods fall into two families. *Value-decomposition* methods such as VDN or QMIX learn per-agent utilities combined by a centralized mixing function to approximate a joint action-value function [25, 22]. *Centralized-critic* methods such as MADDPG, COMA, or MAPPO instead give each decentralized policy a critic with access to global state and joint actions during training [18, 10, 29]. In cooperative tasks with homogeneous agents, these decentralized policies can use parameter sharing – a single shared policy network fed with each agent's local observations, while the critic trains on global information.

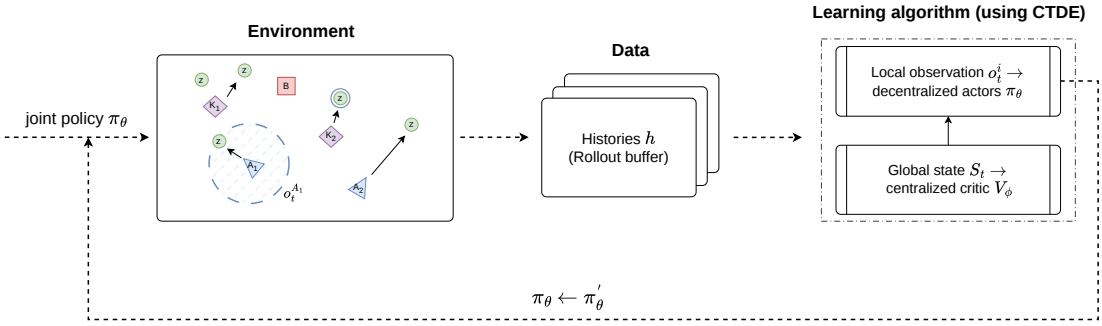

Figure 4.5: Simplified MARL learning loop highlighting CTDE. The joint policy runs in the KAZ environment; trajectories are stored in a rollout buffer. The shared actor is updated on local observations o_t^i (decentralized) and the critic on global state S_t (centralized).

4.4 Multi-Agent Proximal Policy Optimization

Multi-agent proximal policy optimization (MAPPO) is a fairly straightforward multi-agent extension of PPO with a CTDE architecture. Compared to other centralized-critic algorithms, such as COMA [10], MAPPO is architecturally simpler, but still achieves state-of-the-art performance on cooperative benchmarks such as SMAC, Google Research Football, and Hanabi [29], which makes it a clean testing ground for analyzing emergent behavior.

4.4.1 Distinctions from Single-Agent PPO

Structurally, there are only three differences from single-agent PPO. First, instead of collecting experience from one agent, we collect from all n agents simultaneously. At each time step t, every agent i receives its own local observation o_t^i, picks an action A_t^i, and

the environment emits reward signal(s) [20, 2]. In the textbook cooperative case this is a single shared reward R_t; in our experiments the mechanics stay fixed while the reward function is varied, so some schemes also introduce agent-specific components. Second is the CTDE principle; the centralized critic $V_\phi(S_t)$ is given the full state of the environment during training, while each actor (policy) only has access to its own local observation o_t^i [10, 2]. Lastly, the PPO clipped objective and critic loss are computed per agent and then averaged across all agents and time steps [29]. The clipping mechanism itself, GAE, and minibatch reuse are otherwise identical.

Formal Setup

MAPPO maintains a shared decentralized policy $\pi_\theta(A_t^i \mid o_t^i, h_t^i)$ for all agents i, with parameters θ and optional recurrent hidden state h_t^i to handle partial observability [29]. And a centralized value function $V_\phi(S_t)$ (or $V_\phi(\tilde{S}_t)$, where $\tilde{S}_t$ is any global information available during training), with parameters ϕ [29]. In the clean common-reward setting, the goal is to learn a decentralized joint policy $\pi_\theta = (\pi_\theta^1, \ldots, \pi_\theta^n)$ that maximizes expected discounted return

$$J(\theta) = \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t R_t\Big],$$

where, at execution time, each agent submits only its own local information [29]. This is the standard Dec-POMDP/common-reward formulation. In the experimental chapters, the same CTDE setup is retained while the payoff mapping is varied by reward shaping, so some configurations are better viewed as fixed-mechanics POSGs rather than strict common-reward Dec-POMDPs.

4.4.2 MAPPO Objective

MAPPO extends the PPO clipped surrogate objective for multiple agents by aggregating the per-agent PPO losses over all agents and time steps [29, 24]. To track how much the policy has changed for each specific agent i, given a trajectory collected under the old policy $\pi_{\theta_{\text{old}}}$, the per-agent probability ratio is [29, 24]:

$$r_t^i(\theta) = \frac{\pi_\theta(A_t^i \mid o_t^i, h_t^i)}{\pi_{\theta_{\text{old}}}(A_t^i \mid o_t^i, h_t^i)}.$$

Typically in environments with shared reward and a single centralized value function, agents share the same advantage estimate $\hat{A}_t$, computed via GAE from the *centralized* critic exactly as described in Section 3.4 – the only difference being that the critic now operates on the global state S_t instead of a single agent's observation. The clipped policy loss then simply averages the per-agent PPO objectives [29, 24]:

$$L_\pi^{\text{MAPPO}}(\theta) = \mathbb{E}_{t,i}\Big[\min\big(r_t^i(\theta)\,\hat{A}_t,\ \text{clip}(r_t^i(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t\big)\Big],$$

and the critic loss remains the same squared error between predicted and target returns [29]:

$$L_V^{\text{MAPPO}}(\phi) = \mathbb{E}_t\Big[(V_\phi(S_t) - \hat{R}_t)^2\Big].$$

The final combined loss is:

$$\mathcal{L}(\theta, \phi) = \underbrace{-L_\pi^{\text{MAPPO}}(\theta)}_{\text{policy gain}} + \underbrace{c_v L_V^{\text{MAPPO}}(\phi)}_{\text{critic error}} - \underbrace{c_e H[\pi_\theta]}_{\text{entropy bonus}},$$

where c_v weights the critic loss and c_e weights the entropy bonus, which keeps the policy stochastic by stopping it from prematurely converging to a single action [29].

Intuitively, the centralized critic $V_\phi(S_t)$ learns to predict the *joint* return of the whole team, while each decentralized actor update pushes the shared policy π_θ towards increasing team advantage, but only as long as the probability ratio $r_t^i(\theta)$ stays within the PPO clipping range. This preserves the stability benefits of PPO while exploiting extra information during training for faster and more stable learning in multi-agent tasks [29].

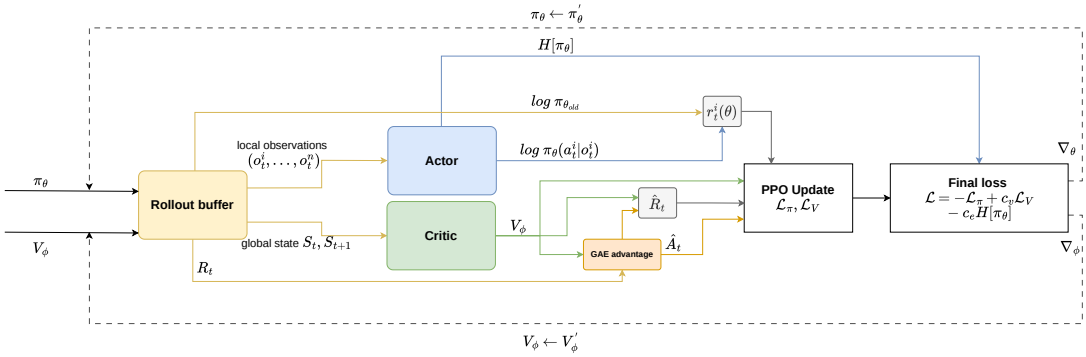

Figure 4.6: MAPPO data flow. The rollout buffer feeds local observations to the actor and global state to the critic. The GAE block combines critic values and rewards into $\hat{A}_t$ and $\hat{R}_t$. The PPO update computes $\mathcal{L}_\pi$ and $\mathcal{L}_V$, producing gradients ∇_θ and ∇_ϕ for the shared networks.

4.4.3 MAPPO Algorithm

Below is the simplified logic of the MAPPO training loop. It follows the standard PPO structure but collects data from all agents simultaneously before updating the shared networks.

Algorithm 1 Multi-Agent Proximal Policy Optimization (MAPPO), adapted from [29, 24, 23].

Input: shared policy θ_0, value function ϕ_0, n agents
Hyperparameters: clip ϵ, discount γ, GAE λ, rollout N, epochs K, minibatch M
for iteration $k = 0, 1, 2, \ldots$ **do**
  $\theta_{\text{old}} \leftarrow \theta_k$
  // Collect rollout from all agents
  for $t = 0$ to $N - 1$ **do**
    for each agent $i = 1, \ldots, n$ **do**
      Sample $A_t^i \sim \pi_{\theta_{\text{old}}}(\cdot \mid o_t^i)$ *(decentralized: local obs only)*
    end for
    Execute joint action $\mathbf{A}_t$, observe team reward R_t, global state S_{t+1}, local obs $\{o_{t+1}^i\}$
  end for
  // Compute GAE advantages using centralized critic
  Compute $\delta_t = R_t + \gamma V_\phi(S_{t+1}) - V_\phi(S_t)$ for all t *(centralized: global state)*
  Compute $\hat{A}_t = \sum_{l=0}^{T-1-t} (\gamma\lambda)^l \delta_{t+l}$ for all t
  Set target returns $\hat{R}_t \leftarrow \hat{A}_t + V_\phi(S_t)$ and normalize advantages
  // Update shared networks
  for epoch $e = 1$ to K **do**
    for each minibatch $\mathcal{B}$ of size M **do**
      Compute per-agent ratio $r_t^i(\theta) = \pi_\theta(A_t^i \mid o_t^i) / \pi_{\theta_{\text{old}}}(A_t^i \mid o_t^i)$
      $L_\pi = \mathbb{E}_{(t,i)\in\mathcal{B}}[\min(r_t^i \hat{A}_t, \text{clip}(r_t^i, 1-\epsilon, 1+\epsilon)\hat{A}_t)]$
      $L_V = \mathbb{E}_{t\in\mathcal{B}}[(V_\phi(S_t) - \hat{R}_t)^2]$
      Update (θ, ϕ) via gradient descent on $-L_\pi + c_v L_V - c_e H[\pi_\theta]$
    end for
  end for
end for

4.4.4 Parameter Sharing

With n agents we need n *separate* policy networks, which would give each agent full independence, but would be expensive to train. To address the scalability problem, it is often better to employ *parameter sharing* in the actor structure, especially if agents share the same action space and play the same role [27, 2]. All agents are controlled by a single shared policy network, but each agent still develops diverse behavior because they receive different observations, ending up in different situations during play. The number of trainable parameters therefore stays constant regardless of team size, and every agent's experience contributes to training the same network [29]. This works well for homogeneous teams, but in heterogeneous settings, where different sets of agents have different jobs, full sharing may suppress this role-specific specialization. *Type-based sharing* addresses this by maintaining one network per agent type. This thesis experiments with all three variants to analyze how this choice influences emergent coordination regimes and role structure [2].

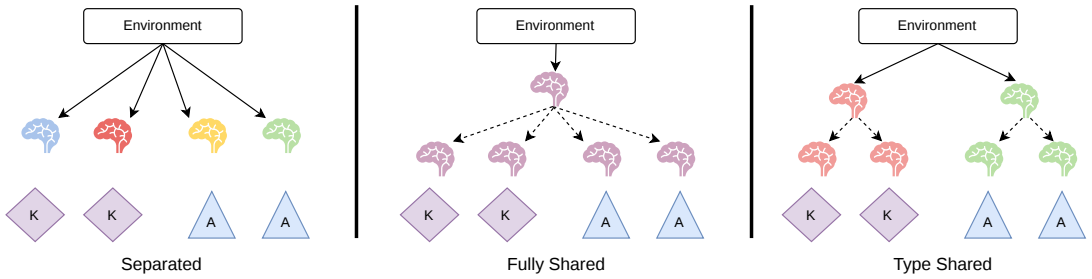

Figure 4.7: Parameter sharing variants explored in this thesis. The same brain color indicates the same policy function part. Four agents are separated into two heterogeneous roles.

4.5 Emergent Coordination, Roles, and Communication

After establishing the fundamental concepts of MARL, it is time to focus on what makes multi-agent RL truly interesting: the situations where agents start to coordinate, divide roles, or develop stable conventions without these patterns being scripted in advance. In small heterogeneous teams, emergence often appears less like randomly forming subgroups and more like developing stable roles where agents actively rely on each other. These phenomena are *emergent* in the sense that they arise from the interaction of learning dynamics, reward functions, and environment mechanics rather than from hard-coded tactical rules.

4.5.1 Emergent Coordination

Emergent coordination means that when agents learn simultaneously, their joint behavior starts to reveal a certain global structure that cannot be understood by just looking at any single isolated agent. For example, when agents learn to cover different parts of the map, or to take complementary roles in an attack-defense pattern, or to time their actions so that they help each other instead of getting in each other's way. In cooperative MARL benchmarks like SMAC, strong policies almost always display this behavior with respect to scalar reward signals [29, 22]. In this thesis, we design situations where mutual support and shared risk could benefit the team, and then empirically test whether agents actually learn to exploit these opportunities and whether doing so turns out to be optimal.

4.5.2 Reward Design for Coordination Regimes

Emergent behavior is heavily shaped by reward function design. Leibo et al. demonstrated this in simple gridworld harvesting games, where a small change to the reward structure was enough to flip the system from peaceful cooperation to aggressive overexploitation [15]. Counter-intuitively, in resource-scarce environments, agents who care more about the future are actually more likely to learn aggressive policies because they recognize the long-term payoff of eliminating competition [15]. Network capacity also matters: complex behaviors only emerge if the agent has sufficient representational capacity to execute them [15]. In mixed-motive settings, short-term selfish behavior such as free riding can be tempting even when long-term group welfare would be higher under cooperation, so naive reward design alone is rarely sufficient [15, 2]. In this thesis, reward engineering is not treated as a source of sociality by itself. KAZ already contains hard interdependence through asymmetric roles

and shared failure conditions. Reward design is instead used as an experimental lever for probing which coordination regime becomes stable under the same base task.

4.5.3 Emergent Communication

Another interesting emergent behavior is how agents communicate with each other and what drives them to do so. In many MARL instances, agents are given *differentiable communication channels*, where agents send discrete or continuous messages, which are added to their observation or policy networks, and eventually learn an artificial "language" specialized for the task [14]. Though, measuring whether such communication is truly useful is tricky. Lowe et al. showed that naive metrics like mutual information between messages and actions can be misleading, since agents might send messages that correlate with their behavior but have no real effect on teammates [17]. The recommended solution here is to compare tasks with and without channels or to scramble the channel and verify if performance differs. This thesis doesn't explore explicit channels; instead, it looks for communication-like coordination through behavioral metrics.

4.5.4 Coalitions, Roles, and Social Structure

Multi-agent systems can develop complex social structures beyond simple collective cooperation. Rachum et al. showed that agents in small environments can spontaneously form *dominance hierarchies* as a tool for conflict resolution [21]. Such structural differences can be quantified by frameworks like Role Diversity, which measure differences between actions, trajectories, and value contributions [12]. Importantly, these differences interact with architectural choices: environments with high action-based diversity can degrade under full parameter sharing, while settings with high trajectory divergence may benefit less from explicit communication [12].

Chapter 5

Chosen Environment

Picking the right environment matters. Since the primary goal is studying reward-sensitive coordination in a heterogeneous team, the chosen environment needs asymmetric roles, potential for partial credit assignment, and room for spatial coordination. The initial plan was to build a custom environment from scratch: a 10×10 grid with resources, enemies, and a boss. In practice, agents failed to learn even basic tasks, so coordination analysis on top of that was pointless. This led to KAZ.

5.1 Knights Archers Zombies

Knights Archers Zombies (KAZ) is a cooperative grid-world-style game where knights and archers defend the bottom of the screen from zombies descending from the top. It is part of the PettingZoo "butterfly" environments and is commonly used as a benchmark in MARL libraries.

Figure 5.1: Knights Archers Zombies entities visualized. The sprites originate from the PettingZoo environment [9].

Original Knights Archers Zombies
Agents are separated into two roles – knights and archers – and share a set of six discrete actions: NOOP, FORWARD, BACKWARD, ROTATE_LEFT, ROTATE_RIGHT, and ATTACK. The attack action is distinct for each role: knights swing a mace and archers shoot long-range arrows. The goal is to survive as long as possible by killing zombies that spawn at fixed intervals at a random horizontal position along the top of the screen and descend with a zig-zag movement (they are not aggressive towards agents). The episode ends when all agents die or a zombie reaches the bottom of the screen. The two distinct roles make the agents heterogeneous. Because zombie spawn positions and lateral movement vary with the random seed, dealing with them requires cooperation. The screen is large enough for agents to move freely, which makes spatial behavior easy to observe. This raises several

questions worth investigating: what coordination regimes emerge, how much does reward design actually matter, and which environment mechanics dominate the resulting team structure?

5.1.1 Custom KAZ Implementation

While the environment is promising, the first training runs exposed two problems. First, archers can spam arrows from a distance while knights struggle to get close without dying. This made archers overpowered – they ended up stealing most kills, leaving the environment too imbalanced for meaningful cooperation analysis. Second, there is no rising difficulty, so agents could settle on a local optimum and survive indefinitely until the step limit. To get a fair but not stale baseline for cooperation analysis, we customized the environment with the following additions.

Boss enemy and wave system
Bosses add a second enemy type with different dynamics. Like regular zombies, they spawn at fixed intervals (every 200 steps) at a random horizontal position. They have more HP, slightly different movement, and their attack instantly kills any agent close by. Because bosses take considerably more effort to bring down, agents need to commit attention to them. Killing a boss advances the wave counter, which increases both zombie and boss HP. Waves act as a difficulty scaler that prevents agents from settling into a static strategy, and the wave count itself serves as a rough measure of team progress.

Leveling / XP system
During testing, the wave system overwhelmed agents too quickly because their damage output stayed flat while enemy HP grew. To address this, a leveling system was added: agents receive XP for each kill (amounts differ by enemy type), and after accumulating enough XP they level up, gaining more base HP and ATK. Beyond offsetting the rising difficulty, leveling creates asymmetric agent states within an episode, which makes kill distribution matter more – an agent that falls behind in XP becomes weaker relative to the wave, so level balance across the team becomes a real concern and a target for reward shaping.

Shielded Zombies
Because archers still dominated the kill count, the last addition was shielded zombies. These spawn less frequently and carry a shield that only a knight can break, so archers cannot damage them until the shield is down. This forces a dependency on knights: if all knights die, shielded zombies become unstoppable and can walk through to the bottom. It also gives archers a reason to protect their knights rather than ignoring them, encouraging mutual protection. In later chapters, we isolate the effect of this mechanic by testing a variant in which archers can also break shields and receive the same shield-break reward.

Figure 5.2: Knights Archers Zombies with new custom additions.

Statistics
Episodes last at most 900 steps on a 1280×720 px screen. Enemies spawn at $y = 5$ with $x \sim \mathcal{U}[150, 1130]$ and descend with zig-zag lateral movement.

Entity	HP(w)	Speed	Score → XP	Spawn
Archer	3	25 px/step†	—	fixed
Knight	5	35 px/step†	—	fixed
Zombie	$2 + 2(w-1)$	5 / 10 px/step	$+1.0 \to 1$	20 steps
Shielded	$2 + 2(w-1)$	5 / 10 px/step	$+1.5$ kill, $+0.75$ break → 2/1	80 steps
Boss	$5 + 3(w-1)$	4 / 8 px/step	$+3.0 \to 3$	200 steps

† Agent speed is omnidirectional; enemy speed listed as down / lateral.

Table 5.1: Compact environment statistics. Wave index w starts at 1 and increments when any boss dies. Baseline XP is round(Δscore). For level $L \to L+1$, required XP is $5+5(L-1)$; each level gives +1 max HP, +0.5 ATK, full heal, and XP carry-over. Contact damage: zombies/shielded = 1, boss = 999.

5.2 JAX Implementation

The original KAZ environment is built on PyGame, which works well for playing games but is not suitable for large-scale MARL training, where each step goes through the Python interpreter sequentially, and GPU utilization is poor. The solution is to rebuild the environment entirely in JAX, which compiles and executes array operations directly on GPU hardware [7].

Functional Design and Immutable State
JAX requires *pure* functions with no side effects or mutable state [7]. The custom KAZ environment is built around two pure functions: `reset(rng, config)` and `step(state, actions, config)`, neither of which modifies anything in place. The entire game state is packed into `KAZState`, a `NamedTuple` of JAX arrays that forms a valid *pytree* for the compiler to traverse and transform [7]. Because JAX requires fixed array shapes at compile time, the state always allocates slots for all possible entities, and the unused ones are simply masked out with a boolean flag.

JIT Compilation and Parallel Environments
Expressing the environment as pure functions over pytree state unlocks two JAX primitives [7]. `jax.jit` compiles a function on its first call via XLA and runs all subsequent calls as

a native GPU kernel. `jax.vmap` vectorizes over a batch dimension, lifting `step` to operate on 256 independent environment states in a single GPU call.

Scan-Based Rollout

Python `for` loops inside JIT-compiled code get statically unrolled into the computation graph. `jax.lax.scan` avoids this by expressing the rollout as a single compiled loop: it applies the environment step and action selection for T steps, accumulating transitions into one stacked array. Combined with `vmap`, the full collection phase of each MAPPO update is one fused GPU operation [7].

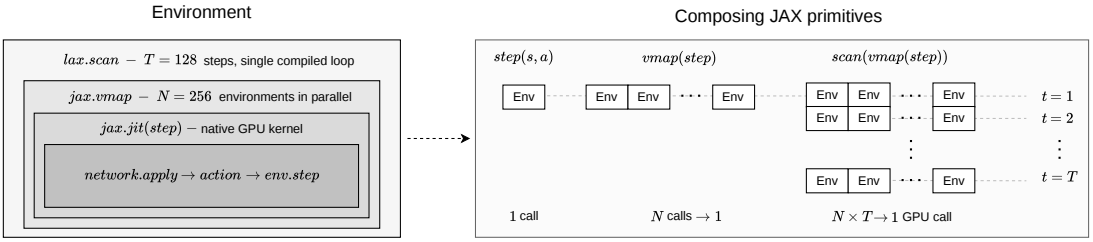

Figure 5.3: Each JAX layer adds a transformation – `jax.jit` compiles the step into a native GPU kernel, `jax.vmap` lifts it over $N = 256$ parallel environments, and `lax.scan` unrolls $T = 128$ timesteps – leading to the full rollout of $N \times T$ transitions in a single GPU call.

Observation Space

From a single agent's perspective, the environment is not encoded as a grid or an image. Instead, we use a *relative, entity-centric* representation where all positions are given as coordinates relative to the observing agent, normalized by screen dimensions, with distances normalized by the screen diagonal. Each agent receives a 2D observation matrix shaped (num_rows, 8), where every row represents a different entity in the environment. All entity types share the same schema:

$$\Big[\ \underbrace{d_{\text{norm}}}_{\text{distance}},\ \underbrace{\Delta x,\ \Delta y}_{\text{relative pos.}},\ \underbrace{\text{dir}_x,\ \text{dir}_y}_{\text{facing}},\ \underbrace{\text{hp\_ratio}}_{\text{health}},\ \underbrace{\text{extra}_1,\ \text{extra}_2}_{\text{type-specific}}\ \Big],$$

where the two extra columns carry type-specific information such as level, shield, status, or type marker that distinguishes bosses from regular zombies. Partial observability is controlled by a configurable *vision radius* that zeroes out entities beyond this distance.

Consistent with the CTDE paradigm, the centralized critic's global world state is formed by concatenating the full observations of all agents into a single flat vector while each actor still operates only on its own local observation.

Neural Network Stack

Depending on the parameter-sharing setting, we use a shared/separate actor network that maps local observations to action logits, and a centralized critic that maps the global world state to a scalar value. Both are three-layer MLPs with 256 hidden units and tanh activations. The implementation uses Flax Linen [11] for the network definitions, Distrax [3] for sampling from the categorical action distribution, and Optax [3] for Adam with gradient clipping. All parameters, optimizer state, and random keys are bundled into a single `TrainState` pytree, which is the only object carried forward across update iterations.

5.3 Reward Design

As discussed in Section 4.5.2, reward shaping is the main experimental lever for encouraging or suppressing specific coordination patterns. To study its effect, seven distinct reward schemes were designed. The original KAZ environment used a simple +1.0 reward per zombie kill, which is a solid baseline for extensions. With the environment customization, the base rewards were extended to +1.5 per shielded zombie, +0.75 per shield break, and +3.0 per boss. All seven policies share the same transition dynamics, observation model, and episode rules; what changes is only the reward function. Formally, the mechanics stay fixed while the payoff structure is varied. Importantly, the environment is already cooperation-biased through shield breaking and shared failure conditions, so these rewards should be read as steering the *style* of coordination more than creating interdependence from nothing.

Baseline
Only base kill rewards, no modifications. Each agent is rewarded solely for its own kills. Any coordination that shows up under this policy must come from the environment structure rather than reward engineering.

Shared Reward
Pools 50% of all kill rewards into a team pot each step, split equally among agents who took the attack action. The remaining 50% stays with the killer. So if an archer lands a kill, the knight swinging nearby also gets a cut.

Egalitarian
Penalizes XP inequality. *Gini penalty*: $-0.015 \times (G - 0.3)$ applied to all agents when the Gini coefficient exceeds 0.3. *Catch-up bonus*: +1.5 when the lowest-XP agent lands a kill.

Survival
Alive reward: $+0.025 \times$ (alive count/total agents) per step per alive agent. *Death penalty*: -2.0 to all surviving agents when a teammate dies.

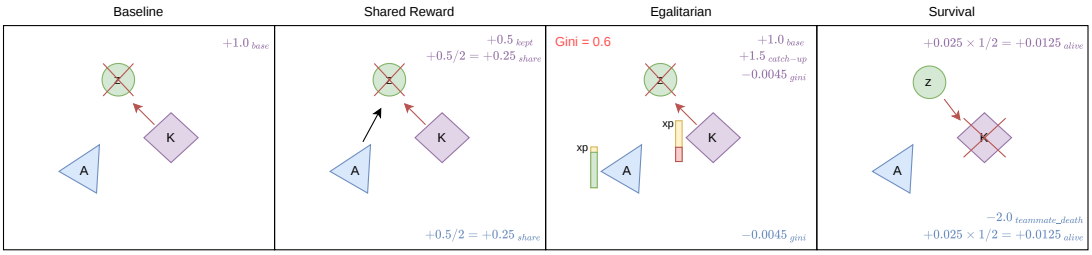

Figure 5.4: Visualization of one step of Baseline, Shared Reward, Egalitarian, and Survival policies. Arrows signify attacking, red arrow means the entity gave the killing blow. Different color rewards in the corners distinguish which agent receives them.

Coalition
When an agent kills an enemy, 70% of the kill reward is shared equally among alive allies

within 200 px. The killer keeps full base reward. The 200 px radius matches the clustering coefficient threshold used in Chapter 6.

Zero-Sum
The killer receives $+0.2\times$ kill value as bonus while all other alive agents split a penalty of equal magnitude. The signal is individually competitive, where helping a teammate actively hurts you.

Territorial
Screen split at $x = \text{width}/2$. Archers own the left half, knights the right. Kill bonus of $+0.5\times$ kill value in own zone; -0.005 per-step penalty outside. The idea is to see whether explicit spatial ownership pushes agents to divide the map rather than cluster together.

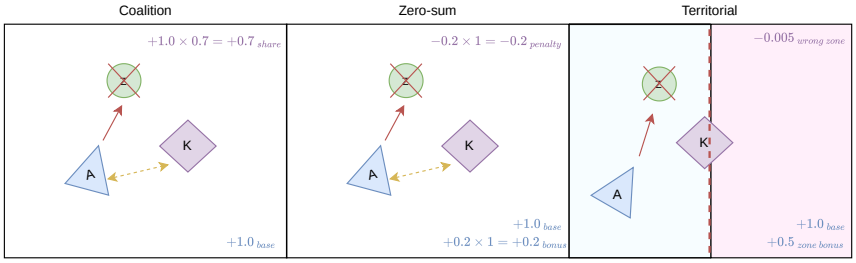

Figure 5.5: Visualization of one step of Coalition, Zero-Sum, and Territorial policies. Arrows signify attacking, red arrow means the entity gave the killing blow. Different color rewards in the corners distinguish which agent receives them.

Chapter 6

Measuring Emergent Coalitions

How do we determine whether two policies survive for the same reason? It is easy to eyeball patterns, which are not really there, and reward alone cannot answer this. A team of agents could achieve the same total return through completely different social structures. To distinguish between these outcomes, this chapter introduces a set of post-hoc metrics computed over evaluation trajectories after training, consistent with prior work on emergent coordination, role diversity, and behavioral heterogeneity in cooperative MARL [2, 12, 5].

Twelve metrics form the core of the analysis, organized into multiple sections. No single metric proves coalition formation on its own – a team could be clustered tightly yet never coordinate attacks, or survive well while depending entirely on one role, which is why the metrics are most informative when read *together*. Appendix A.2 collects simpler supplementary diagnostics such as archer attack frequency and idle ratio, which are occasionally useful for replay interpretation but are not part of the primary comparison.

Replay Browser

In addition to quantitative metrics, we built a web-based replay browser (implemented in Svelte/SvelteKit) for visual inspection of agent behavior. Users export episodes from selected checkpoints into slim replay files and load them into interactive playback with frame-by-frame navigation, per-step analytics, and toggleable metric overlays. This was especially useful for debugging gameplay and checking whether custom metrics matched observed behavior. Appendix A.1 briefly documents the tool and gives the hosted deployment link.

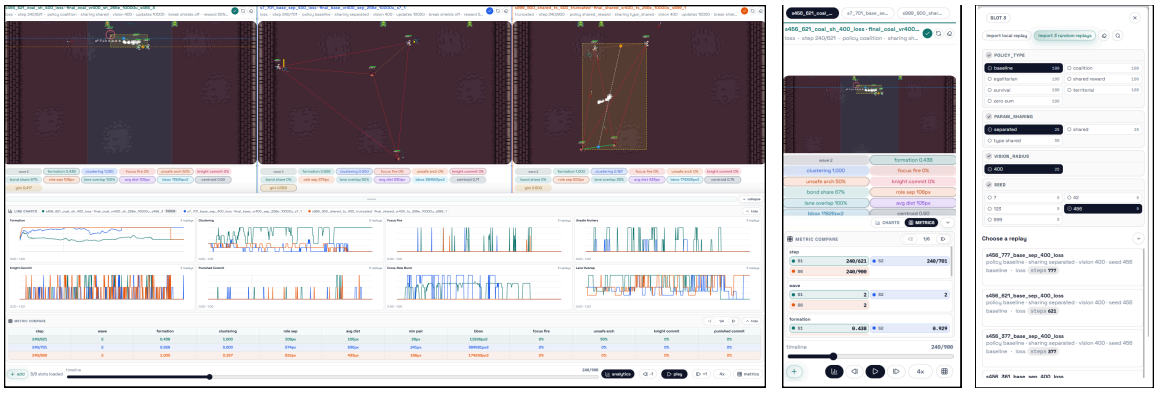

Figure 6.1: Replay browser with three exported episodes. Step-wise metrics below gameplay can be toggled as in-scene overlays. The analytics panel provides trend plots and a cross-episode comparison table; images on the right show mobile layouts.

6.1 Spatial Structure and Cohesion

The most immediate thing about a team is how its members are positioned. Knights need to be close because of their melee attacks; archers can stay back because they shoot from range. If agents learn role-appropriate positioning, their spatial arrangement may be a useful indicator of effective coordination. Prior work on multi-robot formation control [4] has analyzed team behavior through clustering and relative positioning, and it was one of the inspirations for the following metrics.

6.1.1 Clustering Coefficient

The starting point is mean pairwise inter-agent distance, but that number is sensitive to outliers – a single agent far from the group inflates the average even when the rest form a tight cluster, so a threshold-based metric handles this better. The clustering coefficient is the fraction of alive pairs within a fixed radius r_c:

$$C_t = \frac{1}{|P_t|} \sum_{(i,j)\in P_t} \mathbf{1}\left\{\|\mathbf{p}_i(t) - \mathbf{p}_j(t)\|_2 \le r_c\right\}, \qquad \bar{C} = \frac{1}{T}\sum_{t=1}^{T} C_t.$$

The threshold $r_c = 200$ pixels matches the coalition reward sharing radius, allowing direct comparison of spatial compactness with incentive range. $C_t \approx 1$ means nearly every agent pair is within 200 pixels; $C_t \approx 0$ means otherwise.

6.1.2 Formation Score

Proximity alone is not enough; an effective KAZ team should also show role-aware positioning. Intuitively, knights should be in front to absorb damage and break shields, and archers behind to deal sustained ranged damage. To measure this, let

$$\bar{y}_A(t) = \frac{1}{|A_t|}\sum_{i\in A_t} y_i(t), \qquad \bar{y}_K(t) = \frac{1}{|K_t|}\sum_{j\in K_t} y_j(t),$$

where A_t and K_t are the sets of archers and knights alive at the time step t. Since enemies descend from the top of the screen, lower y-values correspond to positions closer to the

front line. The formation score is then

$$F_t = \text{clip}\left(0.5 + \frac{\bar{y}_A(t) - \bar{y}_K(t)}{H}, 0, 1\right), \qquad \bar{F} = \frac{1}{T}\sum_{t=1}^{T} F_t.$$

where H is the screen height in pixels ($H = 720$) normalized to $[0, 1]$. A value above 0.5 means knights are, on average, positioned in front of archers.

6.1.3 Role Separation

Role separation measures how far apart two role groups are overall by measuring Euclidean distance between the centroids of alive archers and alive knights:

$$R_t = \|\mathbf{c}_A(t) - \mathbf{c}_K(t)\|_2, \qquad \bar{R} = \frac{1}{T}\sum_{t=1}^{T} R_t,$$

where $\mathbf{c}_A(t)$ and $\mathbf{c}_K(t)$ are the centroids of alive archers and knights, respectively. High role separation with high formation score indicates spatially distinct sub-teams with a clear front-back structure. Low role separation with high clustering suggests the team moves as a single mixed group.

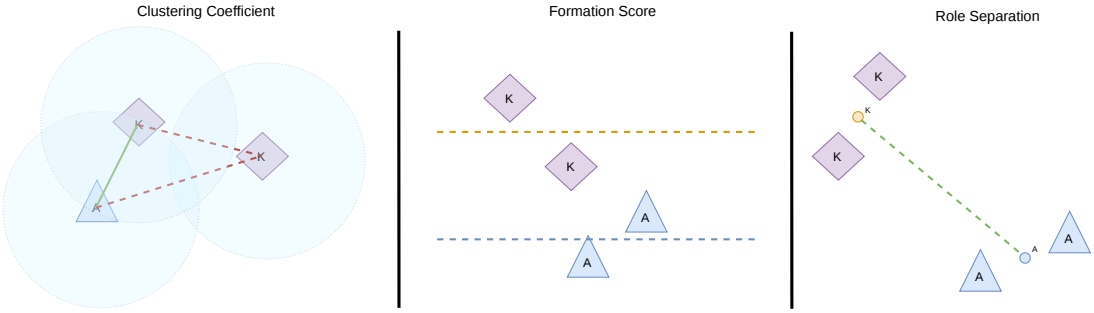

Figure 6.2: Spatial structure metrics. Left: *clustering coefficient* – agent pairs within $r_c = 200$ px are connected by green lines; pairs beyond it by red. Here two agents cluster together and one falls outside. Middle: *formation score* – the orange dashed line is the mean y-position of alive knights; the blue marks that of alive archers. When both knights are above the blue line, the front-back formation holds. Right: *role separation* – the green dashed line connects the archer centroid (**A**) to the knight centroid (**K**).

6.1.4 Coalition Lifetime Median

Since spatial snapshots only tell us who is near whom at a given moment, it's useful to track how long pairs of agents stay close together over time. Two agents are considered close at timestep t if both are alive and within radius $r = 350$px. For each pair, we record each continuous time interval during which they remain close. If R_r is the set of all such intervals and ℓ_k is the length of interval k, then

$$\tilde{\ell}_{350} = \text{median}\{\ell_k\}_{k \in R_{350}}.$$

High values mean agents tend to stay close for longer. Low values mean they only come close briefly. We use $r = 350$ px instead of 200 px to make the metric more relaxed and not limited to only very close pairs.

6.2 Coordination, Differentiation, and Engagement

Spatial proximity is a prerequisite for coordination, but agents can be clustered in the same area while each attacks a different enemy. Measuring coordinated target selection is useful as a descriptive indicator, although higher target concentration does not necessarily imply better teamwork, as later analysis shows.

6.2.1 Focus Fire Score

The focus fire score measures whether multiple agents attack the same enemy at the same timestep. We geometrically resolve each agent's attack: an archer is treated as targeting an enemy if the enemy is within 400 pixels and falls inside a roughly 60-degree forward cone, and a knight targets an enemy if it is within its 70-pixel sword arc radius.

Given the set of attacking agents T_t at timestep t, we define $\text{coord}_i(t) = 1$ if agent i attacks an enemy that is simultaneously targeted by at least one teammate, and 0 otherwise. The focus fire score is then

$$FF_t = \frac{1}{|T_t|} \sum_{i \in T_t} \text{coord}_i(t), \qquad \overline{FF} = \frac{1}{|\{t : |T_t| > 0\}|} \sum_{t:|T_t|>0} FF_t.$$

High focus fire means agents often attack the same enemy. However, this should be read as a descriptive signal rather than direct evidence of coordinated decision-making.

To complement this action-level metric, we also measure policy-level diversity with System Neural Diversity [5].

6.2.2 System Neural Diversity

System Neural Diversity (SND), introduced by Bettini et al. [5], measures how different agents' action distributions are. We compute it in discrete KAZ action space using Jensen–Shannon (JS) divergence, which is symmetric, bounded in $[0, \log 2]$, and well-defined for categorical distributions.

For each pair of alive agents (i, j), the per-timestep JS divergence is averaged over the episode, then averaged over all pairs:

$$D_{JS}^{ij} = \frac{1}{T} \sum_{t=1}^{T} D_{JS}(\pi_i(\cdot \mid o_i(t)) \,\|\, \pi_j(\cdot \mid o_j(t))),$$

$$SND = \frac{2}{n(n-1)} \sum_{i<j} D_{JS}^{ij}.$$

High SND signals systematically different action preferences across agents. Low SND means agents behave similarly, either because the policy has not differentiated or because roles genuinely require similar actions.

6.2.3 Knight Close-Commit Rate

This metric measures how much knights actively press near engagements by focusing on meaningful knight commits rather than raw action counts. For each alive knight i, let $d_i^\star(t)$ denote distance to the nearest active enemy. A *close commit* is recorded when the knight is already near contact range (105px) and chooses to move forward or attack:

$$\text{commit}_i(t) = \mathbf{1}\{\text{alive}_i(t) \wedge d_i^\star(t) \le 105 \wedge a_i(t) \in \{1, 5\}\}.$$

The knight close-commit rate is then

$$\mathrm{KCCR} = \frac{\sum_t \sum_{i \in K} \mathrm{commit}_i(t)}{\sum_t \sum_{i \in K} \mathbf{1}\{\mathrm{alive}_i(t)\}}.$$

In simple terms, it measures how often knights go for a nearby enemy when they have the chance.

6.3 Protection and Role Balance

A team can survive and coordinate while still exposing one role to much more risk than the other. This section therefore focuses on direct protection and outcome balance.

6.3.1 Unsupported Archer Exposure

One of the supposed optimal playstyles for archers is to attack from a safer distance, catching stray enemies, rather than being forced into near-contact range without knight cover. For each alive archer i at timestep t, let $e_i^\star(t)$ be the nearest active enemy and let

$$d_i^\star(t) = \|\mathbf{p}_i(t) - \mathbf{e}_i^\star(t)\|_2.$$

The archer is marked as *unsupported and exposed* if $d_i^\star(t) < 150\,\mathrm{px}$ and no alive knight is closer to that same enemy by a small support margin $m = 5\,\mathrm{px}$:

$$\neg \exists\, k \in K_t : \|\mathbf{p}_k(t) - \mathbf{e}_i^\star(t)\|_2 + m < d_i^\star(t).$$

The episode-level rate is

$$\mathrm{UAER} = \frac{\sum_t \sum_{i \in A} \mathbf{1}\{\mathrm{alive}_i(t) \wedge d_i^\star(t) < 150 \wedge \neg \mathrm{supported}_i(t)\}}{\sum_t \sum_{i \in A} \mathbf{1}\{\mathrm{alive}_i(t)\}}.$$

Lower values mean archers spend less of their alive time in dangerous unsupported states. This metric is more mechanistic than a simple action-frequency summary because it measures positional vulnerability directly.

6.3.2 Survival Ratio

The simplest outcome measure is how long each role survives. For an agent class $\mathcal{C}$ (archers or knights), the survival ratio is

$$\mathrm{SR}_{\mathcal{C}} = \frac{1}{T|\mathcal{C}|} \sum_{t=1}^{T} \sum_{i \in \mathcal{C}} \mathbf{1}\{\mathrm{alive}_i(t)\}.$$

This is reported separately for archers and knights. A large gap reveals asymmetric survival: if knights die early, archers face shielded zombies they cannot damage, leading to rapid team collapse.

6.3.3 Knight HP Bleed Rate

Survival alone does not show how much damage agents take while still alive. To measure this, we track how quickly knights lose normalized HP over time. For knight i, let

$$h_i(t) = \frac{\mathrm{HP}_i(t)}{\mathrm{HP}_i^{\max}(t)}, \qquad \Delta^- h_i(t) = \max(h_i(t) - h_i(t+1),\, 0)\,.$$

The knight HP bleed rate is then

$$\mathrm{BR}_K = \frac{\sum_{t=1}^{T-1} \sum_{i \in K} \mathbf{1}\{\mathrm{alive}_i(t)\}\, \Delta^- h_i(t)}{\sum_{t=1}^{T-1} \sum_{i \in K} \mathbf{1}\{\mathrm{alive}_i(t)\}}.$$

This metric measures how quickly knights lose health, on average. Low values mean they take damage more gradually, while high values mean they are being worn down more quickly. This helps determine if knights ram into enemies, or if their attacks are more reserved.

6.3.4 Gini Coefficient

To measure inequality in end-of-episode experience points, we compute the Gini coefficient over the final XP values of the agents that are still alive at episode end:

$$G = \frac{\sum_{i=1}^{n} (2i - n - 1) x_i}{n \sum_{j=1}^{n} x_j},$$

where $x_1 \le x_2 \le \cdots \le x_n$ are the sorted XP values [8].

The Gini coefficient ranges from 0 (perfect equality) to 1 (all XP concentrated in one agent). This is a deliberate late-episode metric: it measures how unevenly progression is distributed among the agents that survive to the end, rather than assigning zero XP to dead agents.

6.4 Interpreting Metrics Together

No single metric tells the full story. High clustering with low focus fire means agents stay together without fighting together. High focus fire with high role separation could mean coordinated attacks from distinct positions, or everyone reacting to the same crisis. Low unsupported archer exposure only looks good when accompanied by strong knight survival, not by archers refusing to engage. The analysis in Chapter 8 reads metrics together rather than trusting any single scalar alone.

Chapter 7

Training and Experiments

With the environment, reward schemes, and metrics all in place, it is time to start training. This chapter covers the training setup, experimental protocol, and the hypotheses that guide the analysis.

7.1 Training Configuration

All agents are trained with MAPPO in the JAX-based environment. The primary setup uses *type-shared* parameter sharing, where all agents of the same role share a single network. Two additional parameter-sharing variants are also trained: *non-shared* (independent networks per agent) and *fully shared* (one network for all agents regardless of role). The second variable is the agent's *vision radius*: the default limits each agent's observation to entities within 400 px, while a comparison condition removes this limit and gives agents full-map visibility.

The actor and critic are both three-layer MLPs with 256 hidden units and tanh activations. The actor outputs 6 action logits, the critic takes the concatenated global state and outputs a scalar value estimate.

Hyperparameters are fixed across all reward schemes (Table 7.1). The goal is to isolate the effect of reward design, not to tune each policy individually. Each of the seven policies from Section 5.3 is trained with *five predetermined seeds* (42, 7, 999, 123, 456) to control for training variance.

Each seed trains on 256 parallel environments with 128 steps per rollout and 10 000 update iterations, producing roughly 328 million environment frames. On a single NVIDIA GPU, the JAX implementation sustains close to 130 000 frames per second, so a full run finishes in under four hours. All runs are tracked with Weights & Biases [6].

In total, the primary evaluation covers 35 type-shared runs with limited vision (7 policies $\times$ 5 seeds), supplemented by 35 runs with unlimited vision, 70 runs across the two alternative parameter-sharing paradigms, and 35 additional runs for the archers-break-shields environment modification (7 policies $\times$ 5 seeds, with matched controls from the primary setup).

7.2 Experimental Protocol

Every 25 update iterations, 200 deterministic episodes are run (argmax actions, no exploration). This produces snapshots of episode reward and all analysis metrics from Chapter 6

Hyperparameter	Value
Learning rate	3×10^{-4}
Discount factor γ	0.99
GAE λ	0.95
PPO clip ϵ	0.2
Entropy coefficient c_e	0.01
Value function coef. c_v	0.5
Gradient clipping norm	1.0
Parallel environments	256
Steps per rollout	128
PPO epochs per update	10
Minibatches per epoch	4
Hidden layer size	256
Hidden layers	3
Activation	tanh
Total update iterations	10000

Table 7.1: MAPPO training hyperparameters, shared across all reward policies.

at regular intervals throughout training. The evaluation uses a separate random key chain to avoid leaking entropy into the training loop.

Final evaluation
After the 10 000th update, 500 deterministic episodes are run and all metrics are computed, producing mean and standard deviation estimates that form the basis of Chapter 8. Across the main analysis axes, this yields 35 evaluated configuration groups and 175 selected run artifacts used for plotting and replay inspection.

7.3 Training figures

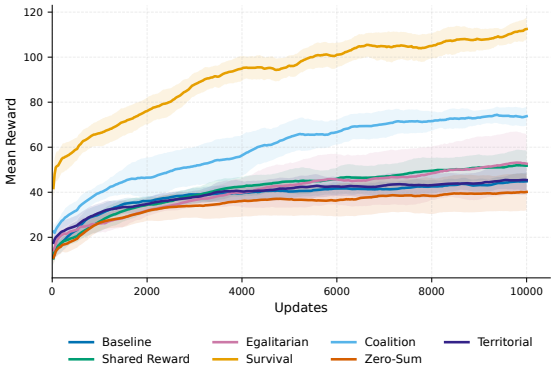

Figure 7.1: Mean reward

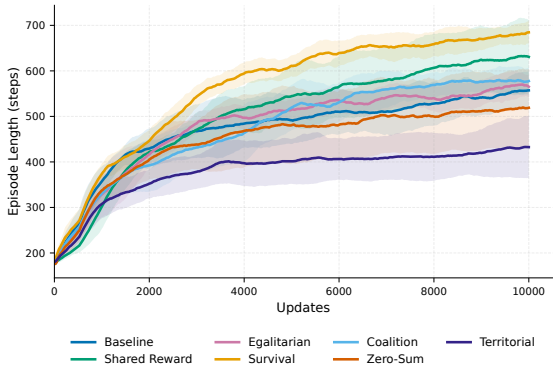

Figure 7.2: Episode length

Since every policy defines a different reward function, the mean reward curves cannot be compared across policies – though they still confirm that training converges. Episode length

is the cleaner signal, directly telling us how long the team survives. Survival and Shared Reward end up highest; Territorial trails. The early read is that policies rewarding team preservation outperform policies that push for aggressive engagement or territorial control.

7.4 Hypotheses

These hypotheses follow from the reward mechanisms in Section 5.3 and from prior work on reward shaping in multi-agent systems [15, 28]. The first five address reward design and environment mechanics; the last two address the architectural comparison axes.

1. **Reward design selects coordination style, not just performance.** Different incentive structures – proximity bonuses, reward sharing, territorial ownership – should produce measurably different spatial and behavioral regimes, even when final episode lengths are comparable. *Coalition* should produce a compact local regime, *Territorial* should push agents toward divided map usage, and *Shared Reward* should differ from the *Baseline* in spacing and role burden.

2. **Equality incentives reduce XP inequality but may hurt durability.** The *Egalitarian* policy should produce the lowest Gini coefficient, but forcing redistribution might reduce total team output and weaken the protective role structure needed for survival.

3. **Survival pressure produces the strongest role-protective structure.** The *Survival* policy should develop the clearest frontline-backline separation with high survival ratios, possibly at the cost of a slower, more selective attack tempo.

4. **Surface coordination proxies do not predict robustness.** The strongest policies should not simply maximize attack frequency, focus fire, or other immediately intuitive coordination indicators. Instead, durable regimes should combine sensible pacing with strong knight-side protection.

5. **Coordination structure depends on environment mechanics, not just reward.** Changing the role-dependency rules – such as allowing archers to break shields – should shift which coordination regime performs best, even with identical reward functions.

6. **Limited vision promotes structured local cooperation.** Agents with restricted vision (400 px radius) should develop stronger local or formation-based behavior than agents with full-map visibility.

7. **Type-shared parameters enable role specialization.** Type-shared networks should produce stronger role differentiation and better performance than fully shared networks, because the architecture allows different roles to develop different policies while still benefiting from within-role generalization.

Chapter 8

Analysis and outcomes

The chapter has five parts: the primary comparison of seven reward schemes under type-shared parameters with limited vision (Section 8.1), vision settings (Section 8.4), parameter-sharing paradigms (Section 8.5), an environment modification where archers can break shields (Section 8.6), and cross-play plus leave-one-out robustness analysis (Section 8.7).

All evaluation results in this chapter use the final evaluation: 10 000-update runs, five seeds per configuration, with deterministic evaluation episodes.

	Primary	**Unlimited**	**Param. sharing**		**ABS**
Policy	**(TS, VR 400)**	**vision**	**SEP**	**SH**	
Survival	701.1 ± 6.0	583.9 ± 32.9	509.8 ± 44.6	504.3 ± 41.4	572.9 ± 57.3
Shared Reward	608.8 ± 63.6	666.4 ± 30.7	508.2 ± 95.1	585.5 ± 48.2	668.3 ± 55.8
Coalition	583.3 ± 22.7	539.3 ± 20.4	538.1 ± 38.3	479.7 ± 47.7	633.8 ± 26.6
Baseline	556.3 ± 28.2	501.6 ± 52.6	510.0 ± 65.8	379.0 ± 63.8	571.6 ± 29.8
Egalitarian	552.8 ± 31.9	477.5 ± 53.8	456.5 ± 66.5	439.0 ± 73.1	551.8 ± 36.9
Zero-Sum	534.4 ± 81.3	431.5 ± 66.4	428.4 ± 53.6	402.3 ± 20.3	539.7 ± 63.8
Territorial	431.7 ± 62.7	474.3 ± 84.0	431.2 ± 48.2	361.1 ± 23.6	525.2 ± 72.7

Table 8.1: Mean episode length (steps) ± standard deviation across five seeds. Primary: type-shared parameters with 400 px vision. Unlimited vision: full-map visibility, type-shared. SEP/SH: non-shared and fully shared parameters with 400 px vision. ABS: archers-break-shields environment modification (all seven policies).

8.1 Primary evaluation: type-shared, limited vision

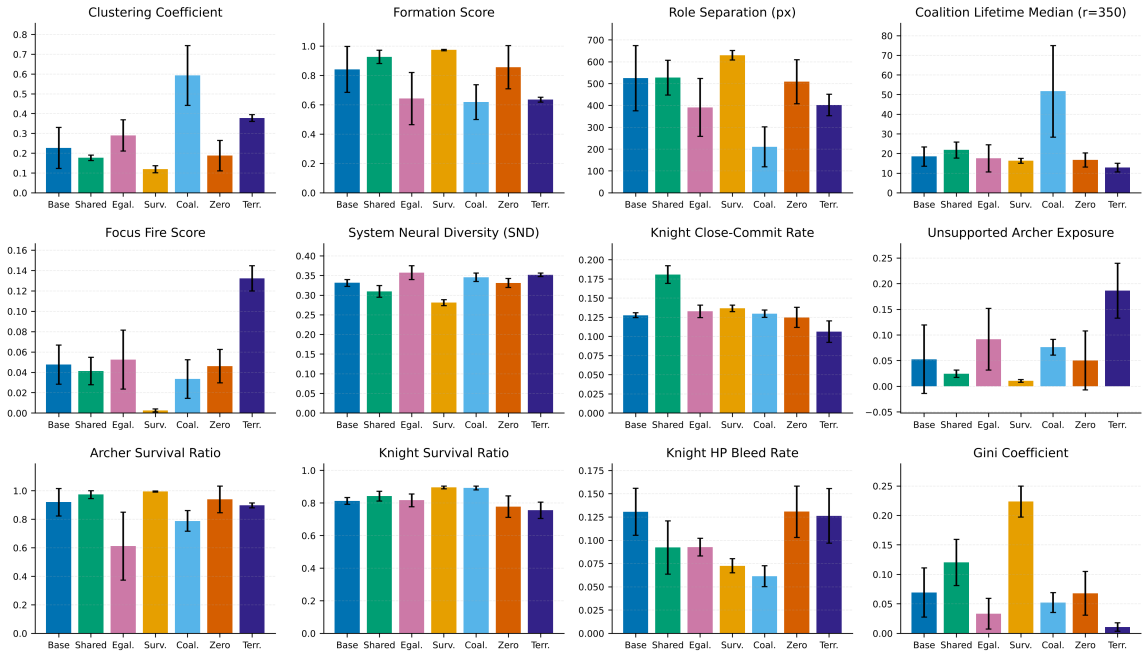

Figure 8.1: Comparison of seven reward policies on the main coordination metrics. Bars show final means with standard deviation over five seeds.

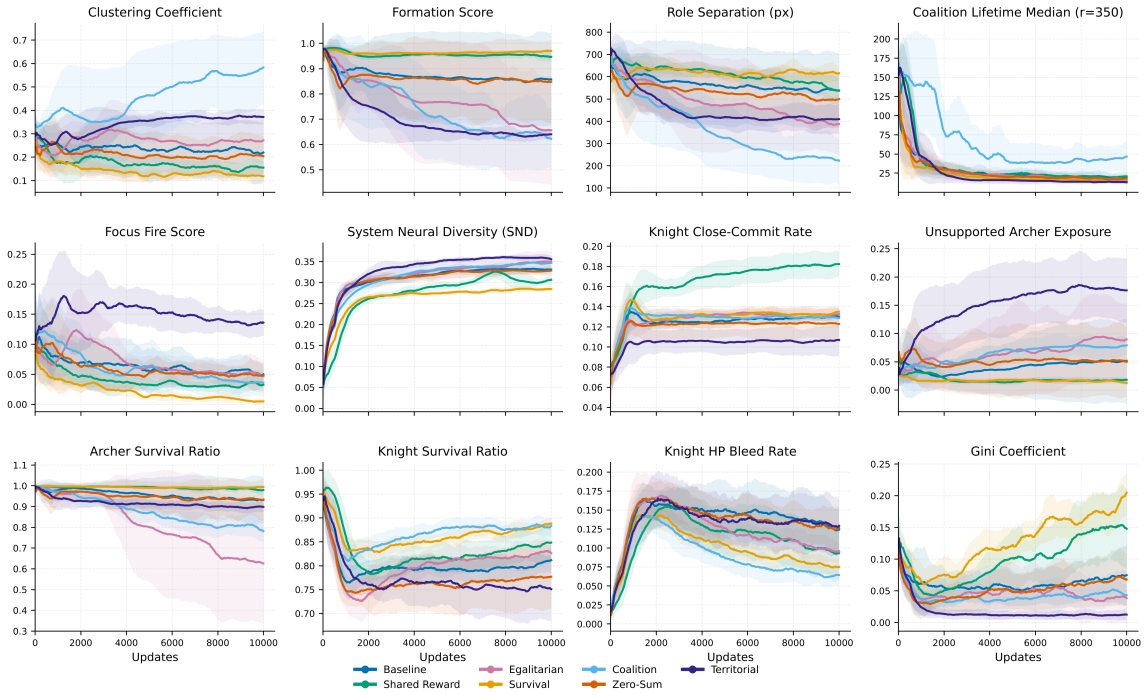

Figure 8.2: Coordination metrics during training. Points show means over 200 episodes from checkpoints every 25 update iterations.

Figures 8.1 and 8.2 summarize the final deterministic evaluations. The question now is what coordination regime each reward design produced.

In performance terms, Survival leads at 701.1 steps with the lowest seed variance in the set (±6.0). Territorial trails at 431.7. The full ranking is in Table 8.1. More interesting than the ranking itself is that these policies do not converge to a single solution. Different reward functions produce genuinely different coordination styles, and several of them work.

Survival

Survival is the most structured policy. It has the highest formation score and role separation in the set, with both survival ratios near their maximum and the lowest unsupported archer exposure. Yet it does not maximize pressure – archer attack frequency is only 0.272 and knight HP bleed stays at 0.073. Its Gini coefficient is the highest (0.223), so late-episode XP concentrates among a subset of the surviving agents. In practice, knights absorb pressure up front while archers fire from behind, and the team picks fights rather than maximizing attack volume.

Coalition and Shared Reward

Coalition and Shared Reward contrast sharply. Coalition is the most compact policy with the highest clustering, lowest role separation, and by far the longest coalition lifetime median (51.7 steps), so the groupings are persistent. It has the lowest knight HP bleed and the formation score is only 0.619, so archers are commonly in front. One caveat is that the Coalition bonus uses the same 200 px distance as the clustering metric, so this particular metric is intentionally aligned with the incentive rather than fully independent of it.

Shared Reward is the opposite: low clustering, high formation, and the highest knight close-commit rate. The archers are often separated shooting from the bottom, while knights are more aggressive near the top. These are two different styles of play but with similar performance – Coalition relies on density and temporal cohesion, while Shared Reward is more spatially distinct and aggressive.

Baseline and Zero-Sum

Baseline and Zero-Sum are controls and produce very similar results across the metrics. The competitive reward transfer in Zero-Sum neither helps nor hurts coordination, possibly because the cooperative pressure of the environment neutralizes the negative component.

Egalitarian and Territorial

Egalitarian and Territorial serve as counterexamples, where policies do achieve the targeted design goal but with weaker performance. Egalitarian has the lowest late-episode Gini coefficient by a wide margin, but also the worst archer survival ratio. Forced equalization leaves ranged agents idle rather than firing from protected positions, breaking the durable role structure that the stronger policies rely on. Territorial tries to split the arena into two role-exclusive zones, which are initially followed, but this incentive conflicts with shielded zombies spawning across the full arena, so knights are forced to act on the other side, breaking the formation and ultimately spiraling towards confusion. This can also be seen in the policy's highest focus fire, yet the shortest episodes, highest unsupported archer exposure, and elevated knight HP bleed.

8.2 Metrics correlation

Metric	Overall ρ	Within-policy ρ
Knight survival ratio	+0.827	+0.895
Focus fire score	−0.795	−0.117
Gini coefficient	+0.747	+0.364
Knight HP bleed rate	−0.691	−0.712
Knight close-commit rate	+0.628	+0.763
Unsupported archer exposure	−0.602	−0.354
SND	−0.550	−0.003
Clustering coefficient	−0.445	−0.011
Formation score	+0.429	+0.017
Archer survival ratio	+0.306	+0.047
Role separation	+0.288	−0.212
Coalition lifetime median $r = 350$	+0.116	−0.402

Table 8.2: Spearman correlation of the twelve thesis metrics with episode length, shown both across all seed-level evaluations and after demeaning within policy families. Rows are sorted by absolute overall correlation.

Cross-policy and within-policy correlates
Several metrics strongly correlate with episode length. The knight survival ratio and the knight close-commit rate correlate strongly, while the knight HP bleed gives a large negative signal. Unsupported archer exposure and archer survival ratio still have strong signals, but clearly the environment is more knight-dependent. The Gini coefficient is also strongly positive, indicating that durable policies tend to preserve a more uneven late-episode progression profile among the agents that survive, rather than forcing equality. Focus fire and SND have the strongest *negative* association; this could mean that optimal play consists of each agent attacking its own enemy while still behaving similarly. High formation and role separation also incentivize greater episode length, as discussed in the previous section.

The picture sharpens further, when we remove the variance between policies. Knight metrics stay dominant, but spatial metrics (formation, clustering, role separation) lose all significance. Spatial structure differs reliably *between* policies but does not predict which seeds within a policy do best. What predicts within-policy success is knight-side burden management.

Correlation Matrix
The metric matrix shows that several metrics mostly describe the same coordination regime rather than independent effects. Formation and role separation move together (0.91), while both oppose clustering $(-0.84, -0.81)$. This structure is closely related to archer protection, where formation correlates strongly with archer survival (0.95) and negatively with archer exposure (-0.91), confirming that frontline-backline play protects archers largely by construction. For the Gini coefficient, we can actually see that here it behaves less like a fairness score and more like a summary of how progression concentrates within the separated regime.

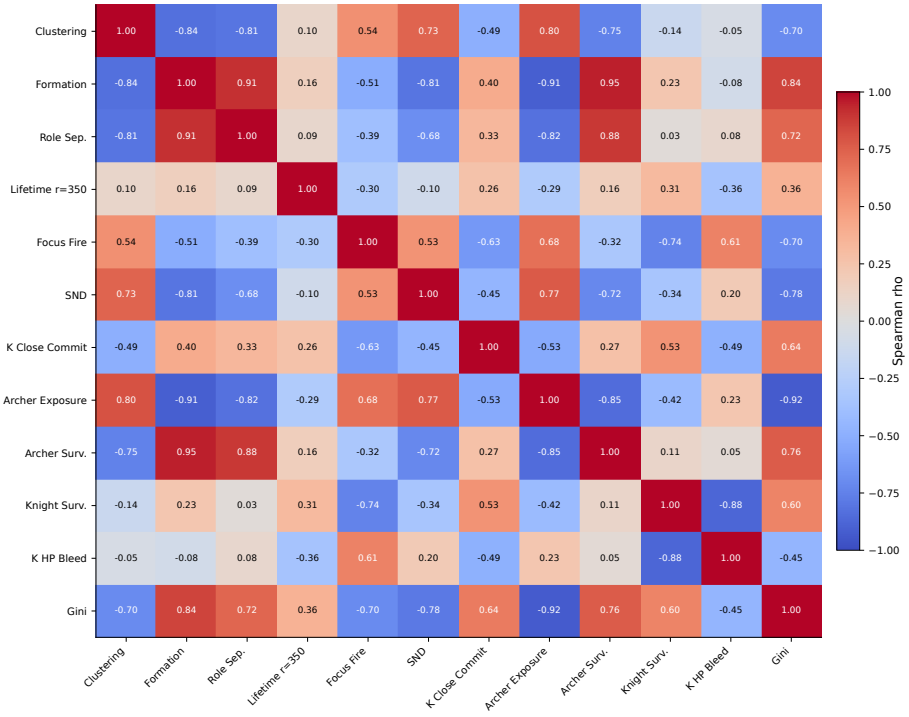

Figure 8.3: Spearman rank correlation matrix across the final seed-level metric summaries for the seven reward policies. Red indicates positive association, blue negative association.

8.3 Spatial heatmaps and qualitative interpretation

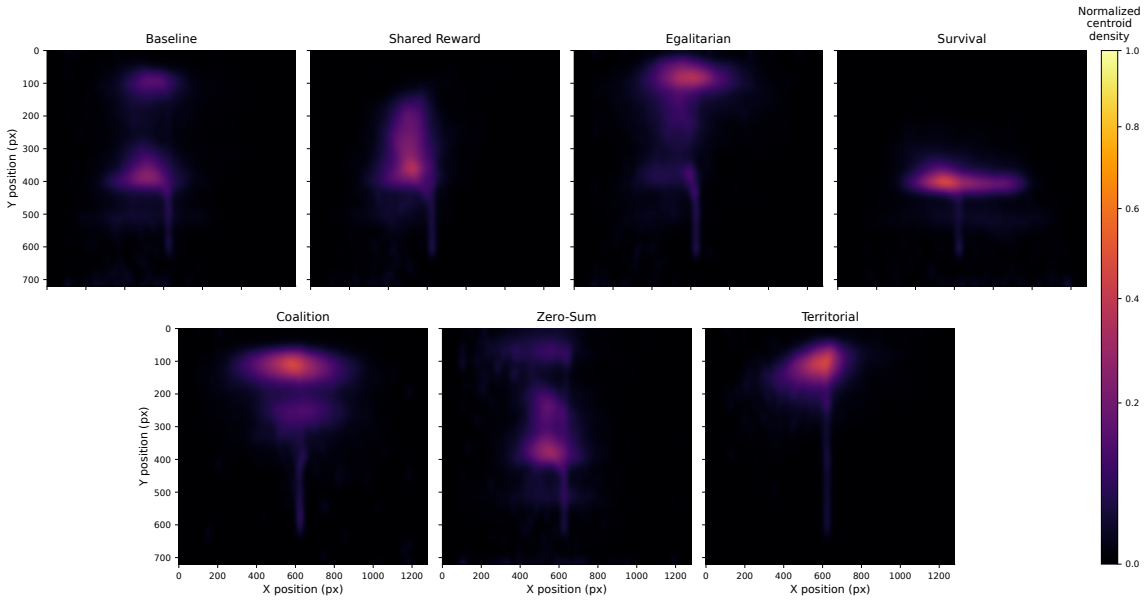

Figure 8.4: Average team-centroid occupancy heatmaps for the seven policies. Brighter regions indicate locations where the team centroid appears more often during deterministic evaluation episodes.

The centroid heatmaps support the earlier interpretation. Survival occupies a deeper mid-map space in a wide horizontal band, accomplishing a near-perfect front-back structure. Coalition remains compact in a narrow upper part, Baseline and Zero-Sum behave almost identically, and Territorial still fails to show a clean lasting split of the arena. Shared Reward sits between these extremes, keeping a more vertical lane structure rather than a compact cluster.

8.4 Vision radius comparison

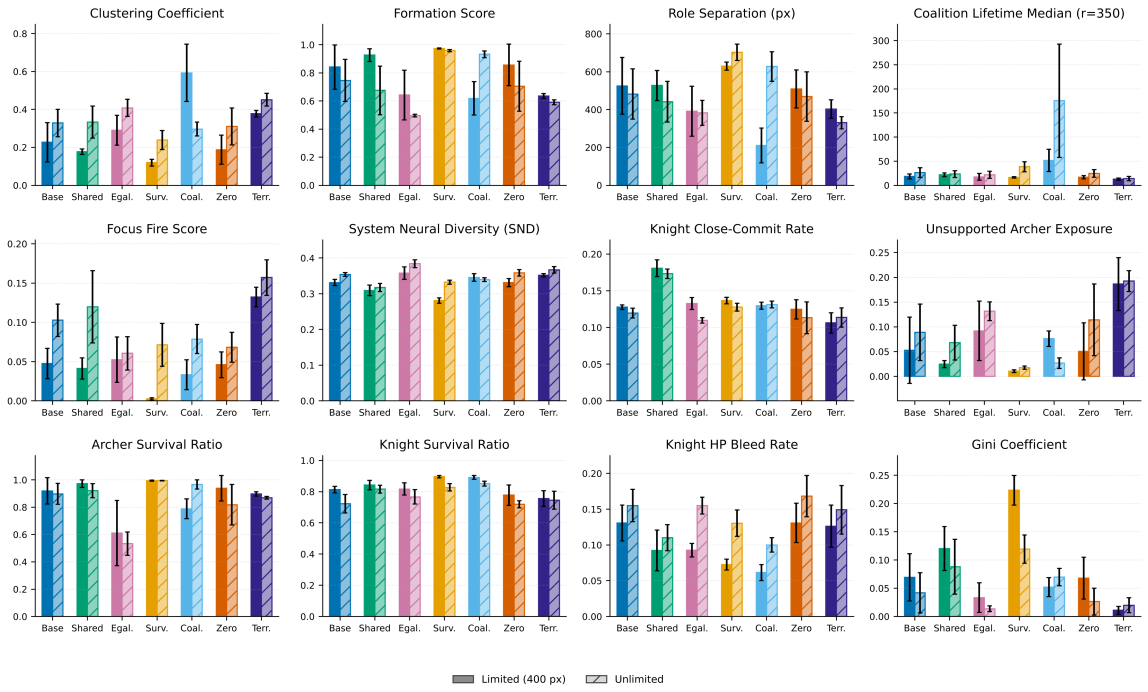

Figure 8.5: Bar comparison of limited (400 px) and unlimited vision across the seven policies.

Each of the seven policies was also trained with unlimited vision (full-map visibility) under the same type-shared setup, to see whether restricting the observation window actually changes the learned coalition style.

For five of seven policies, limited vision produces equal or better performance; only Shared Reward (608.8 → 666.4) and Territorial (431.7 → 474.3) benefit from full-map visibility. Among the five that decline, Survival shows the largest drop (−117 steps), while the Baseline, Zero-Sum and Egalitarian decline moderately. Why does more information about the environment reduce performance in many cases? It suggests that in KAZ, full-map visibility might simply add noise that distracts policies from handling the most important local decisions.

Behavior differences

Coalition is the most dramatic case. Under unlimited vision, it *reverses* its defining compact style: formation score jumps from 0.619 to 0.932, role separation rises from 210 to 627 px, and clustering drops from 0.593 to 0.296. When Coalition agents can see the entire

map, they stop clustering and spread into a formation-based structure resembling Survival. But this more "organized" Coalition actually performs worse (539.3 vs 583.3 steps). The compact style was therefore not a failure to learn formation, but an effective local strategy that depends on limited vision.

Survival retains high formation under both conditions (0.974 vs 0.958), but the 117-step drop with unlimited vision suggests the extra information is just more noise. The policy already has everything it needs within 400 px.

But Shared Reward goes the other way is the clearest case where unlimited vision is more effective. Formation drops (0.927 to 0.676) and clustering rises (0.177 to 0.333), yet performance goes up. This makes sense for a more aggressive policy, where seeing more of the map possibly helps agents react and target enemies earlier.

8.5 Parameter sharing comparison

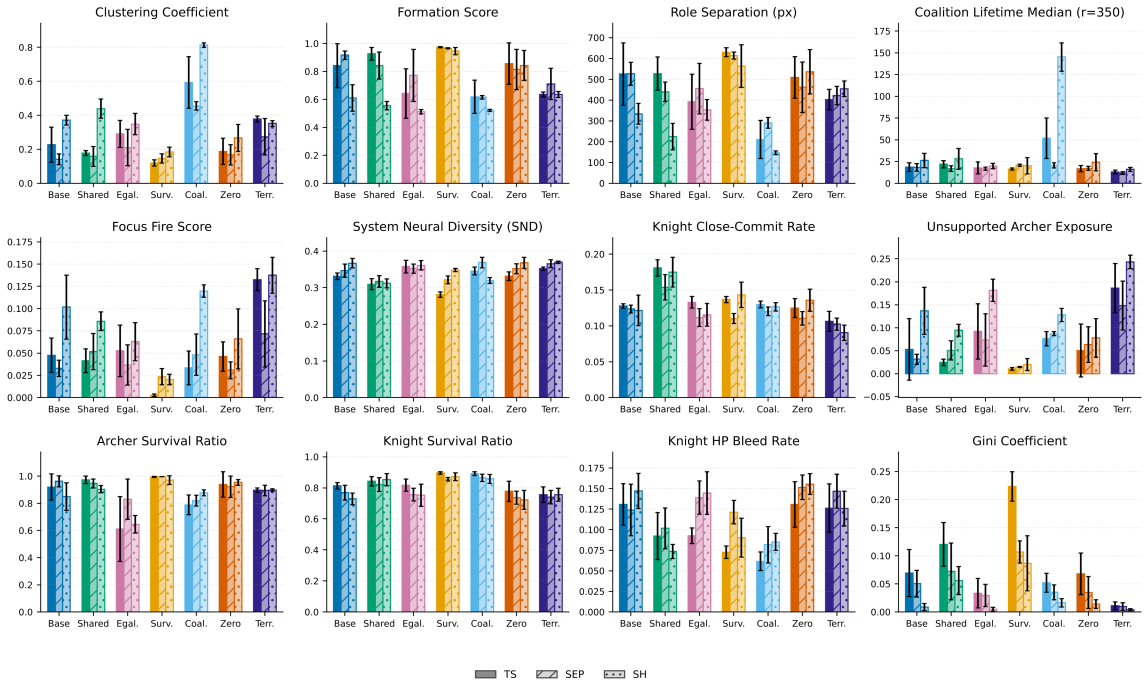

Figure 8.6: Episode length comparison across type-shared (TS), non-shared (SEP), and fully shared (SH) paradigms.

Each policy was also trained under three parameter-sharing settings: type-shared (TS, the primary setup), non-shared (SEP, independent networks per agent), and fully shared (SH, one network for all agents). Type-shared produces the longest episodes for all seven policies. The gap is largest for Survival (TS 701.1 vs SEP 509.8 vs SH 504.3), with Coalition and Baseline showing the same pattern. The most architecture-robust policy is Shared Reward, where its fully shared variant comes close to type-shared (SH 585.5 vs TS 608.8).

Failure modes of the alternatives

Non-shared networks sometimes retain or even increase visible structure, for example in

formation score (Baseline SEP 0.916 vs TS 0.842), but it does not translate into longer survival. Interestingly enough, fully shared networks achieve almost no XP inequality across agents, with Gini coefficients near zero across most policies, though they perform worst overall. This suggests that one shared network suppresses role specialization. Survival is the clearest example of this effect. Under type-shared parameters it reaches near-perfect formation (0.974) and the longest episodes in the study, but under both alternatives it drops back to the middle of the pack.

8.6 Environment modification: archers break shields

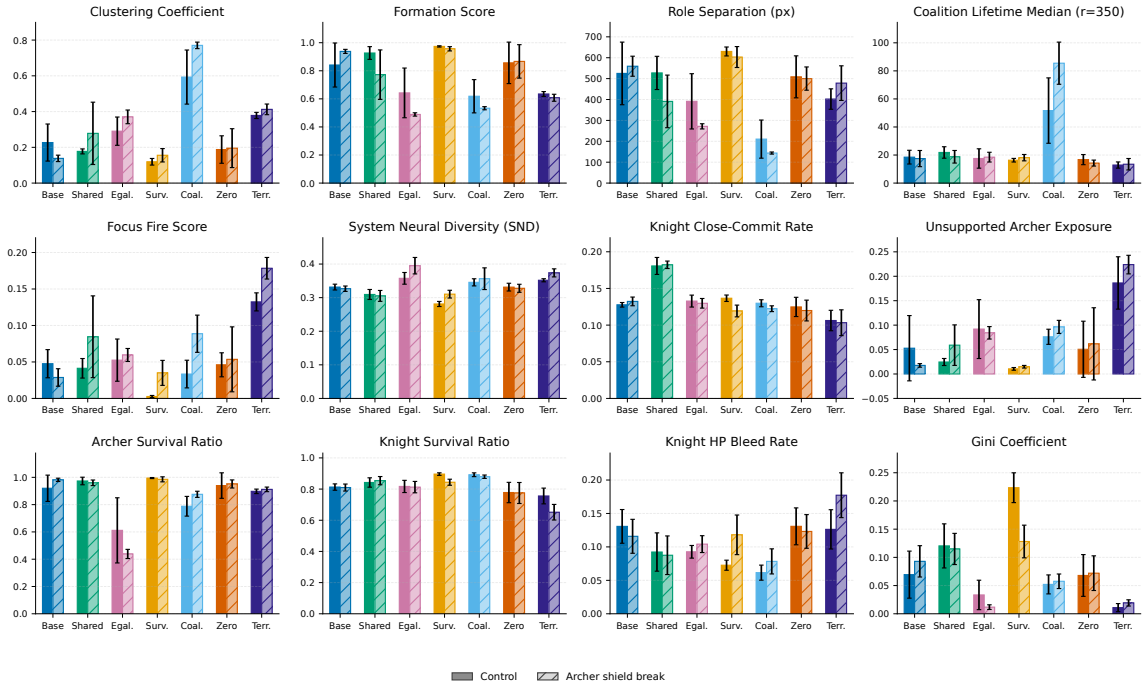

Figure 8.7: Bar comparison of the archers-break-shields environment modification across all seven evaluated policies.

So far, knight survival has been the strongest predictor of episode length across the primary evaluation. But that finding depends on an environment rule: only knights can destroy zombie shields. To test how much of the coordination structure actually rests on this asymmetry, we trained all seven policies in a modified environment where archer projectiles can also break shields and receive the standard shield-break reward. Everything else stays identical to the primary setup.

Performance shifts

Shared Reward now leads at 668.3 steps, Coalition rises to 633.8, and Survival falls sharply from 701.1 to 572.9, nearly tying Baseline at 571.6. The most surprising change comes from Territorial: it remains the weakest ABS policy, but gains 93.5 steps, the largest improvement in the set. Egalitarian and Zero-Sum barely move (552.8 $\rightarrow$ 551.8 and 534.4 $\rightarrow$ 539.7), so relaxing the mechanic does not help every reward equally.

Policy-specific shifts

Shared Reward and Coalition both move even more toward a more attack-oriented play style. Coalition's clustering rises from 0.593 to 0.770, role separation falls from 210 to 144 px, focus fire nearly triples (0.033 → 0.089), and archer attack frequency rises from 0.213 to 0.281. Shared Reward shows the same: role separation drops from 527 to 391 px, clustering rises from 0.177 to 0.278, focus fire doubles (0.041 → 0.084), and it becomes the best-performing policy in the modified environment even though archer attack frequency falls (0.666 → 0.547). Supposedly, once archers can help clear shields, compact pressure becomes directly useful for these regimes.

Baseline moves in the opposite direction: clustering drops from 0.227 to 0.138 while formation rises from 0.841 to 0.938 and unsupported archer exposure falls from 0.053 to 0.018. And Survival keeps its separated frontline-backline signature (formation 0.974 → 0.957, role separation 630 → 603 px), but loses its advantage because the old knight bottleneck is gone, so the policy keeps doing its specialized knight front-line, but now with less payoff.

Territorial benefits the most in raw episode length. Once archers can break shields in their own lane, the intended split becomes more viable: role separation increases from 402 to 478 px, focus fire rises from 0.132 to 0.178, and archer attack frequency from 0.322 to 0.409. Knight survival actually worsens (0.756 → 0.652) and it still remains last, but it does overall suggest that knight-breakable shields had been one of the main reasons its left-right ownership scheme failed.

Egalitarian drives its Gini coefficient even lower (0.033 → 0.012) and pushes archers to participate more aggressively, but archer survival falls sharply (0.611 → 0.438) and no durable protective structure emerges. Zero-Sum shows no meaningful metric changes.

Correlation structure

We also reran correlation analysis and across all 70 runs, knight survival remains the strongest positive correlate of episode length ($\rho = 0.772$), but weaker than in the primary evaluation ($\rho = 0.827$), while knight HP bleed remains strongly negative ($\rho = -0.720$).

Focus fire no longer acts as a uniformly harmful signal. Its overall correlation is still negative ($\rho = -0.398$), but within more tightly packed Coalition and Shared Reward it becomes positive ($\rho = 0.661$) and 0.697, whereas within more spaced out policies like Survival and Egalitarian it remains strongly negative ($\rho = -0.842$) and (-0.818). Which makes sense, because all are relatively high-performing policies, so agents in Coalition policy, moving very close together would have naturally better focus fire, whereas Survival, which thrives in separation, naturally has a lower one.

The overall takeaway is that the same policy can produce different spatial organizations under different environment rules. What counts as the "best" coalition style depends on what the environment actually requires, so reward design alone does not determine it.

8.7 Cross-play and leave-one-out analysis

The final experiments test how tightly these coordination styles depend on the specific teammates seen during training. Cross-play tests whether role-conditioned coordination survives mixing, while leave-one-out asks which roles are load-bearing. Both experiments cover all seven reward policies under the standard type-shared primary setup.

8.7.1 Cross-play

For each seed, two archers from one policy were paired with two knights from another, and the resulting mixed team was evaluated. Table 8.3 reports normalized compatibility: each off-diagonal entry is the mixed-team episode length divided by the average of the two source policies' self-play episode lengths.

Archers ↓ / Knights →	Base	Coal	Egal	Shared	Surv	Terr	Zero
Baseline	1.000	0.755	0.786	0.988	0.835	0.573	0.885
Coalition	0.760	1.000	0.776	0.748	0.717	0.578	0.784
Egalitarian	0.708	0.796	1.000	0.708	0.694	0.592	0.673
Shared Reward	0.795	0.541	0.621	1.000	0.638	0.486	0.814
Survival	0.693	0.645	0.550	0.743	1.000	0.464	0.722
Territorial	0.875	1.019	0.931	0.955	0.892	1.000	0.940
Zero-Sum	0.883	0.759	0.730	0.933	0.867	0.623	1.000

Table 8.3: Mean normalized compatibility across five seeds. Values express cross-play episode length relative to the arithmetic mean of both source policies' self-play episode lengths. Diagonal is self-play (1.0).

The matrix confirms that mixing policies usually reduces performance: 41 of 42 off-diagonal entries are below 1.0. The only positive transfer is Territorial archers with Coalition knights (1.019), while Baseline archers with Shared Reward knights remain near parity (0.988). The weakest pairing is Survival archers with Territorial knights (0.464), indicating a sharp mismatch between a separated front-back style and strict zone ownership. Looking at source portability, Territorial archers are the most transferable (off-diagonal row mean 0.935), Shared Reward knights are the most transferable knight source (off-diagonal column mean 0.846). Territorial knights are the least transferable (0.553), implying that Territorial's strict one-side-only policy for knights is a real burden for all other policy archers.

8.7.2 Leave-one-out

Lastly, we directly test which removals the learned coordination regimes can survive. Across all seven policies, removing a knight causes larger performance drops than removing an archer:

Policy	**Archer Δ ep**	**Knight Δ ep**
Baseline	−35.5	−299.5
Coalition	−79.5	−320.8
Egalitarian	−74.2	−293.1
Shared Reward	−2.0	−360.3
Survival	−37.1	−451.0
Territorial	−5.6	−121.6
Zero-Sum	−3.8	−274.1

Table 8.4: Mean episode length deltas when removing an archer versus a knight. Values averaged across both agents of each role and across five seeds.

Survival remains the most knight-dependent policy, losing 451.0 steps when one knight is removed. Coalition and Shared Reward are also heavily knight-dependent (-320.8 and -360.3), while Territorial is the least knight-dependent at -121.6.

Archer removal is much more policy-dependent. Coalition and Egalitarian drop the most (-79.5 and -74.2), while Shared Reward, Territorial, and Zero-Sum are nearly archer-redundant under this ablation (-2.0, -5.6, and -3.8).

Taken together, the leave-one-out results identify knights as the key part across all seven policies, while archer importance depends strongly on the specific coordination regime.

8.8 Hypothesis evaluation

1. **Reward design selects coordination style, not just performance.** The results support this hypothesis. Coalition has the highest clustering (0.593), Survival the clearest formation (0.974), and Shared Reward sits between them with an aggressive spatially-spread style. All three finish in the top four, but through genuinely different regimes. Territorial does push agents toward divided map usage as predicted, though the regime collapses under shielded zombie pressure.

2. **Equality incentives reduce inequality but may hurt durability.** Yes, Egalitarian achieves the lowest late-episode Gini coefficient (0.033) but the worst archer survival (0.611). Forced equalization leaves archers idle rather than firing from protected positions, breaking the role structure that other policies rely on.

3. **Survival pressure produces the strongest role-protective structure.** Correct. Survival leads at 701.1 steps through selective engagement. It has the highest formation score, highest role separation, and lowest unsupported archer exposure, while knight HP bleed stays at 0.073 and archer attack frequency at only 0.272. The leave-one-out ablation strengthens this: Survival loses 451 steps when a single knight is removed, the largest drop of any policy, confirming that the frontline-backline structure really is load-bearing rather than incidental.

4. **Surface coordination proxies do not predict robustness.** Yes. Focus fire correlates negatively with episode length (-0.759). Territorial has the highest focus fire and the shortest episodes. SND is also negatively associated. After removing between-policy variance, spatial metrics lose all significance. Cross-play added more evidence, where almost all mixed-team pairings performed worse than self-play, showing that these regimes are not just interchangeable approximations of one solution. Leave-one-out also confirms severe knight-side burden, meaning one the biggest predictors for policy success are knights.

5. **Coordination structure depends on environment mechanics, not just reward.** Yes. When archers gain shield-breaking ability and the corresponding reward, Shared Reward rises to 668.3, Coalition to 633.8 steps, Territorial gains 93.5 steps, while Survival falls to 572.9 and loses its lead. The same reward functions produce different spatial organizations under different environment rules. Knight survival remains the strongest correlate but weaker (0.772 vs. 0.827), and focus fire becomes regime-dependent.

6. **Limited vision promotes structured local cooperation.** Mostly. Five of seven policies perform better with limited vision. Coalition being the clearest: with un-

limited vision it reverses its compact style entirely, spreading into a formation-based structure that actually performs worse. Shared Reward is the exception, gaining a real boost from full-map visibility.

7. **Type-shared parameters enable role specialization.** Yes. Type-shared outperforms both alternatives for all seven policies. The gap is largest for Survival (701.1 vs. 509.8 vs. 504.3) and Coalition. Non-shared keeps some visible structure, but underperforms. Fully shared networks suppress role specialization, achieving near-zero Gini coefficients across most policies while performing worst overall.

Chapter 9

Conclusion

This thesis asked whether different reward functions produce different coordination structures in a small heterogeneous MARL team, and whether those structures can be meaningfully distinguished beyond reward curves. The answer to both is yes, but with caveats worth talking about.

Seven reward schemes were trained in a custom JAX-based KAZ environment using MAPPO primarily with type-shared parameters. The resulting analysis found distinct coordination archetypes, such as Survival's separated frontline-backline formation, Coalition's dense mixed-role cluster, and Shared Reward's aggressive spatially-spread style. Each achieves competitive performance through a different mechanism. Several of the intuitive coordination, spatial cohesion, and behavioral metrics turn out to describe which regime a policy occupies rather than how durable it is. The strongest predictor across all regimes is knight survival, and more broadly, how well the team manages frontline burden.

The archers-break-shields modification and leave-one-out ablation support this interpretation more strongly than correlation alone. When archers were allowed to break shields, Shared Reward became the best policy, Coalition also improved, Territorial improved sharply, and Survival lost its lead. Knight survival still remained the strongest correlate, but the advantage of high knight-centered separation weakened. Removing a knight caused far larger performance drops than removing an archer across all seven leave-one-out policies. These results show that the coordination structure is shaped jointly by reward design and environment mechanics, and changing either one can shift which regime works best.

On the architectural side, type-shared parameters outperformed both alternatives for all seven policies, and limited vision (400 px) helped five out of seven. More information does not always help; in most cases it appears to add noise that distracts from the local decisions that actually matter.

9.1 Limitations

The study has several limitations that should be kept in mind when interpreting the results.

The team consists of four agents of two types. This is enough to observe coordination regimes but too small for meaningful coalition partitioning in the game-theoretic sense. So whether the same reward-regime relationships hold at larger scales is an open question.

All findings come from a single KAZ environment, which has specific properties, such as asymmetric roles, descending enemies, and shield mechanics that clearly shape what

emerges. Different environments with different interdependence structures could produce entirely different relationships between reward design and coordination.

We deliberately used fixed hyperparameters across all reward schemes to isolate the effect of reward design, but this could mean that some policies may be undertrained relative to what they could achieve with per-policy tuning. Also, we used only one algorithm; different algorithms could push policies to learn different coordination structures under the same rewards.

The metrics are post-hoc and descriptive. They characterize what happened in evaluation trajectories, not *why* agents coordinate the way they do. The replay browser helped bridge this gap by providing frame-by-frame gameplay context that grounded the metric interpretations in the analysis, but it remains a qualitative tool. The ablation experiments go further than metrics alone, but they still cannot fully explain the learning dynamics.

9.2 Future Work

Scaling the team to more agents and introducing additional roles would test whether the coordination archetypes observed here generalize or whether new ones appear. Larger teams would also make genuine coalition partitioning possible.

Testing the same reward schemes in a different cooperative environment would help separate which findings are KAZ-specific and which reflect more general properties of reward-conditioned coordination.

Adding explicit communication channels [14] and comparing them to the implicit coordination found here would help clarify whether the observed structure is driven more by behavioral conventions or by learned communication.

Bibliography

[1] Achiam, J. *Spinning Up in Deep Reinforcement Learning* https://spinningup.openai.com/. 2018. OpenAI educational resource.

[2] Albrecht, S. V.; Christianos, F. and Schäfer, L. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024. Available at: https://www.marl-book.com.

[3] Babuschkin, I.; Baumli, K.; Bell, A.; Bhupatiraju, S. et al. *The DeepMind JAX Ecosystem* http://github.com/google-deepmind. 2020.

[4] Balch, T. and Arkin, R. C. Behavior-Based Formation Control for Multirobot Teams. *IEEE Transactions on Robotics and Automation*, 1998, vol. 14, no. 6, p. 926–939.

[5] Bettini, M.; Shankar, A. and Prorok, A. System Neural Diversity: Measuring Behavioral Heterogeneity in Multi-Agent Learning. *Journal of Machine Learning Research*, 2025, vol. 26, no. 163, p. 1–27. Available at: https://www.jmlr.org/papers/v26/24-1477.html.

[6] Biewald, L. *Experiment Tracking with Weights and Biases* https://www.wandb.com/. 2020. Software available from wandb.com.

[7] Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C. et al. *JAX: composable transformations of Python+NumPy programs* http://github.com/google/jax. 2018. Version 0.3.13.

[8] Ceriani, L. and Verme, P. The origins of the Gini index: extracts from *Variabilità e Mutabilità* (1912) by Corrado Gini. *The Journal of Economic Inequality*. Springer, 2012, vol. 10, no. 3, p. 421–443.

[9] Farama Foundation. *Knights Archers Zombies (KAZ) — PettingZoo Documentation* https://pettingzoo.farama.org/environments/butterfly/knights_archers_zombies/. 2024. Accessed: 2026-04-01.

[10] Foerster, J. N.; Farquhar, G.; Afouras, T.; Nardelli, N. and Whiteson, S. Counterfactual Multi-Agent Policy Gradients. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2018, vol. 32, p. 2974–2982.

[11] Heek, J.; Levskaya, A.; Oliver, A.; Ritter, M.; Rondepierre, B. et al. *Flax: A neural network library and ecosystem for JAX* http://github.com/google/flax. 2020. Version 0.12.0.

[12] Hu, S.; Xie, C.; Liang, X. and Chang, X. Policy Diagnosis via Measuring Role Diversity in Cooperative Multi-Agent RL. In: *Proceedings of the 39th International Conference on Machine Learning (ICML)*. 2022, p. 9041–9071. PMLR volume 162.

[13] Konda, V. R. and Tsitsiklis, J. N. Actor-Critic Algorithms. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2000, vol. 12, p. 1008–1014.

[14] Lazaridou, A. and Baroni, M. Emergent Multi-Agent Communication in the Deep Learning Era. *ArXiv preprint*, 2020, arXiv:2006.02419.

[15] Leibo, J. Z.; Zambaldi, V.; Lanctot, M.; Marecki, J. and Graepel, T. Multi-Agent Reinforcement Learning in Sequential Social Dilemmas. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2017, p. 464–473.

[16] Littman, M. L. Markov Games as a Framework for Multi-Agent Reinforcement Learning. In: *Proceedings of the 11th International Conference on Machine Learning (ICML)*. 1994, p. 157–163.

[17] Lowe, R.; Foerster, J. N.; Boureau, Y.-L.; Pineau, J. and Dauphin, Y. On the Pitfalls of Measuring Emergent Communication. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2019, p. 693–701.

[18] Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, P. et al. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, vol. 30.

[19] Mnih, V.; Kavukcuoglu, K.; Silver, D. et al. Human-Level Control Through Deep Reinforcement Learning. *Nature*, 2015, vol. 518, no. 7540, p. 529–533.

[20] Oliehoek, F. A. and Amato, C. *A Concise Introduction to Decentralized POMDPs*. Springer, 2016.

[21] Rachum, R.; Nakar, Y.; Tomlinson, B.; Alon, N. and Mirsky, R. Emergent Dominance Hierarchies in Reinforcement Learning Agents. *ArXiv preprint*, 2024, arXiv:2401.12258.

[22] Rashid, T.; Samvelyan, M.; Witt, C. S. de; Farquhar, G.; Foerster, J. et al. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018.

[23] Schulman, J.; Moritz, P.; Levine, S.; Jordan, M. I. and Abbeel, P. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In: *Proceedings of the 4th International Conference on Learning Representations (ICLR)*. 2016.

[24] Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A. and Klimov, O. Proximal Policy Optimization Algorithms. *ArXiv preprint*, 2017, arXiv:1707.06347.

[25] Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W. M. et al. Value-Decomposition Networks for Cooperative Multi-Agent Learning Based on Team Reward. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2018.

[26] Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press, 2018.

[27] Tan, M. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In: *Proceedings of the 10th International Conference on Machine Learning (ICML)*. 1993, p. 330–337.

[28] Wolpert, D. H. and Tumer, K. *An Introduction to Collective Intelligence*. NASA-ARC-IC-99-63. NASA Ames Research Center, 1999.

[29] Yu, C.; Velu, A.; Vinitsky, E.; Wang, Y.; Bayen, A. M. et al. The Surprising Effectiveness of MAPPO in Cooperative Multi-Agent Reinforcement Learning. *ArXiv preprint*, 2021, arXiv:2103.01955.

[30] Zhang, K.; Yang, Z. and Başar, T. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. In: *Handbook of Reinforcement Learning and Control*. Springer, 2021, p. 321–384.

Appendix A

Additional Material

A.1 Replay Browser

The replay browser is a small web-based tool implemented in Svelte/SvelteKit and used for qualitative inspection of exported episodes.

Hosted deployment: https://eightl.github.io/replay-browser/.

A.2 Supplementary Metrics

Chapter 6 defines the twelve core metrics. This appendix collects simple supplementary diagnostics that were useful for replay interpretation and debugging but are not part of the primary comparison.

A.2.1 Archer Attack Frequency

Archer attack frequency measures how often alive archers choose the attack action:

$$\mathrm{AAF} = \frac{\sum_t \sum_{i \in A} \mathbf{1}\{\mathrm{alive}_i(t) \wedge a_i(t) = \textsc{attack}\}}{\sum_t \sum_{i \in A} \mathbf{1}\{\mathrm{alive}_i(t)\}}.$$

High values mean archers spend a larger fraction of their alive timesteps firing. Low values indicate more movement, repositioning, or idling.

A.2.2 Idle Ratio

Idle ratio measures how often alive agents choose NOOP. For any class $\mathcal{C}$:

$$\mathrm{IR}_{\mathcal{C}} = \frac{\sum_t \sum_{i \in \mathcal{C}} \mathbf{1}\{\mathrm{alive}_i(t) \wedge a_i(t) = \textsc{noop}\}}{\sum_t \sum_{i \in \mathcal{C}} \mathbf{1}\{\mathrm{alive}_i(t)\}}.$$

This is mainly a debugging diagnostic. High values can mean passivity, but in some situations they also reflect deliberate pacing or waiting for range.