Causal Expectations for Decentralised Coordination in Multi-Agent Systems

Anonymous Author(s)

Affiliation withheld for review

Abstract. Coordination in multi-agent systems (MAS) is commonly seen as a problem of distributed decision-making under shared constraints. However, coordination within decentralised MAS environments also requires agents to anticipate the causal consequences of their own actions and those of others. This paper explores whether decentralised coordination can emerge through the creation and reuse of explicit causal expectations without communication, centralised control, or hardcoded coordination rules. We implement Popperian Expectations in autonomous agents operating in an intersection environment without coordination mechanisms. We represent causal expectations using the Expectation Event Calculus (EEC), enabling agents to form them internally and revise them when they are falsified. Over 2000 crossing attempts, agents achieved a 98.6% task completion rate with 733,116 total simulations, compared to the baseline model which achieved 99.9% task completion but with a total of 6,530,568 total simulations. The results show that agents converge to a stable set of causal expectation rules while significantly reducing the number of simulations needed, with minimal reduction in safety. The results suggest that explicit causal expectations can provide a sufficient mechanism for decentralised coordination in MAS.

Keywords: Expectation · Coordination · Contention · Beliefs · Popperian Expectations · Expectation Event Calculus · Multi-agent Systems · Causal Expectations · Causal Learning

1 Introduction

Coordination in multi-agent systems (MAS) is commonly framed as a problem of distributed decision-making, in which agents must align their actions to satisfy objectives or constraints using game-theoretic, algorithmic, or logical mechanisms [1]. Traditional approaches to coordination in multi-agent systems primarily focus on selecting mutually compatible strategies under predefined constraints or payoff structures. However, we argue that coordination also requires causal anticipation, in which agents can represent and reason about which actions will cause rather than merely identify compatible strategies.

Some traditional approaches to coordination typically rely on centralised control, explicit hardcoded protocols and rules, reward-based reinforcement learning, or reactive policies. These approaches can be effective, but they each share

certain limitations. Centralised control can reduce the scalability of large systems. Hard-coded rules can restrict the adaptability of agents and impair generalisation and learning new contexts. Reactive agents lack anticipatory reasoning to reason about future states or events. Reinforcement learning approaches often require extended training, which could put agents at risk or fail when key variables change.

Coordination is a key problem in decentralised multi-agent systems. When multiple autonomous agents share an environment, resources, or goals, their actions can become interdependent, so that individual agents' goals cannot be achieved without coordination. Without effective coordination, agents could interfere with each other, create deadlocks, use incompatible strategies, or produce unsafe outcomes for themselves and other agents. These issues often arise when agents face uncertainty. Agents must choose individual actions while anticipating the behaviour of other agents whose intentions and possible actions may be unobservable or unclear. This paper addresses this question: can agents learn to coordinate reliably in shared environments by forming and refining falsifiable causal expectations about their own and other agents' actions without communication, coordination rules, or shared knowledge? Previous work, such as the Popperian Expectations Framework [2] [3], describes the formal mechanisms for extracting, storing, using, and revising explicit causal expectations learnt from internal models.

To accomplish this, we propose implementing Popperian Expectations in decentralised agents to allow agents to create and use causal rules for decision-making. We showcase this in an autonomous intersection navigation tasks where four different agents all approach a shared intersection from different directions. The environment has no traffic signals, means of communication, or specified right-of-way rules. In the environment, agents must learn when it is safe to proceed and when it must stop purely from learnt causal expectations. The coordination between agents forms if they independently learn the same causal structure of the intersection without the use of hardcoded or established rules. Rather than having to rely on explicitly hard-coded rules for coordination, agents can form and use causal expectations about the likely consequences of actions based on certain conditions. These expectations are structured representations of causal anticipation formed through internal simulation. The causal expectations are stored and then reused when an applicable scenario arises. If an expectation is violated, agents can revise their expectations. Through repeated interactions with these same agents that allow for causal rules that are timely and adaptable, expectations can constrain action selection in a way that promotes coordinated behaviour by allowing agents to reason causally about other agents and their intentions. However, relying on simulation alone can be insufficient for long-term coordination. Without a mechanism for memory, agents must repeatedly re-simulate similar scenarios, whereas structured expectations can allow agents to store and reuse causal knowledge for application in current and future decision-making. Popperian Expectations are structured condition-effect rules represented in the Expectation Event Calculus (EEC) [4]. When confidence in a

causal expectation exceeds a threshold, the agent can reuse the expectations to guide their action selection, reducing redundant simulations and prevents risky behaviours. When expectations are confirmed, their confidence increases; when expectations are violated, their confidence weakens, or they revise the expectation. This mechanism for confirmation-violation of causal expectations enables adaptive reasoning even when states and environments change without the need for specific norm encoding. In this paper, we will show that decentralised coordination can be achieved through the use of explicit causal expectations where agents use confidence-gated causal expectations to bypass simulations, falling back onto simulations only when no causal expectation exists for that scenario. No coordination protocols, priority rules, or normative constraints are hard-coded or defined in the programme prior to execution. Coordination can arise only from locally formed expectations within the individual agents. In an experimental extension of this concept, agents achieved a 98.63% task completion rate without collisions or persistent deadlocks in a decentralised contention environment over 2000 total attempts. These results suggest that implementing Popperian Expectations can support a sufficient and stable level of coordination in decentralised environments. This paper seeks to make these contributions:

1. Implementation and use of explicit causal expectations in MAS using Popperian Expectations.
2. A proof of concept demonstration that causal expectations can support sufficient decentralised coordination.
3. Provide an outline for how causal expectations can be implemented in MAS.

2 Related Work

2.1 Dennett's Tower of Generate and Test

Dennett's Tower of Generate and Test [6] [5] represents a hierarchy of cognitive thinking in agents. It includes four distinct levels, each of which contains a creature that displays a level of cognitive ability; Darwinian creatures (survival of the fittest), Skinnerian creatures (reinforcement learning), Popperian creatures (internal reasoning), and Gregorian creatures (tool makers and users). The focus of this work will focus on Popperian creatures which have the ability to internalise the trial-and-error process to learn and test potential what-ifs. Popperian creatures have been used as the inspiration behind many of the internal simulation frameworks used in this paper.

2.2 Winfield's Consequence Engine

The architecture first described by Winfield [7] [8] introduced the framework for consequence engines within autonomous agents. The consequence engine framework enabled agents to reason internally about their actions and internally test different hypotheses in their heads before acting upon them in the real world. It gave agents the ability to internally evaluate different causal actions by asking

and evaluating what would happen if a certain action or event occurred. This framework introduced four key components: Object Tracker-Localiser (senses and captures environmental data), Internal Model (simulates potential what-if actions), Consequence Engine (evaluates the effect of potential actions), and the Robot Controller (executes applicable actions). Within this framework agents could reason internally about events within the world. However, their knowledge was transient, so they did not possess the ability to have long-term causal reasoning within their environment.

2.3 Popperian Expectations

The architecture for Popperian Expectations [2] [3] introduced an extension to Winfields consequence engine framework by allowing agents to capture and use causal knowledge using the EEC. The architecture added three key components to the consequence engine framework, EEX Expectation Formation (creates simulation results into causal EEC rules), Expectation Memory (stores causal rules with specific metadata for confidence), and Expectation Update (revises stored expectations based on real-world interactions). This enabled agents to reason about causal expectations within their environment and capture previously transient knowledge as causal rules to be reused and updated instead of relying purely on simulation. This allows agents to apply causal knowledge to real-world decisions and reason about cause-and-effect relationships.

2.4 Expectation Event Calculus

The Expectation Event Calculus (EEC) is a formal language to represent and reason about causal expectations [4]. It was introduced as an extension of the Event Calculus (EC) [9], by incorporating additional semantics for expectations, fulfilment, and violations. The EEC allows for the representation of events and their effects allowing explicit representation of cause-and-effect relationships. $Happens(e, t)$ is used to track what event occurred, storing the event that happened and at what time. $Initiates(e, f, t)$ stores what event caused a state to become true at what time. Allowing the causal relationship between the two to be captured and how e will affect fluent f. $Terminates(e, f, t)$ functions in a similar manner but tracks which fluent became false. Finally, $HoldsAt(f, t)$ tracks which fluent is true at the current time, allowing agents to track constants in the environment.

2.5 Monte Carlo

Monte Carlo methods estimate action values by repeatedly randomly sampling rather than doing exhaustive searches [10]. Monte Carlo approaches sample actions from a given state and select the actions that produce the best average outcome across rollouts. Flat Monte Carlo is used for its generality as it requires no domain-specific heuristics, only a forward model of the environment [10]. For

each action, a bounded forward rollout is run, and the action with the greatest success is selected. Flat Monte Carlo does not build trees, store any context, or reuse knowledge from previous decision-making, meaning an agent using flat Monte Carlo must repeatedly simulate every possible time step.

3 Decentralised Coordination Setting

3.1 Environmental Model

We consider a decentralised multi-agent environment in which multiple agents operate within a shared environment. Agents pursue individual goals that require them to travel through shared spaces where they can potentially collide with other agents. The environment evolves in discrete timesteps and coordination in the environment must arise without centralised control or predefined rules.

Let $A = \{a_1, a_2, \ldots, a_n\}$ represent a finite set of agents. There are four types of identical, distinguished only by their colour: green, red, blue, and yellow. Each agent functions identically, maintains its own local state and expectation memory, and selects actions independently at each timestep. There is no global controller, shared policy, or communication mechanism to promote coordination.

Each agent occupies a discrete position in the environment and follows a fixed trajectory along a road towards a designated goal. The environment contains four roads entering from the north, south, east, and west, converging at a central intersection. Each road has two lanes corresponding to opposite directions of travel. Each agent starts from a designated road segment and must successfully traverse the intersection to reach the opposite end.

At every timestep t, an agent selects one action from a finite action set

$$U = \{\text{SPEED}, \text{GO}, \text{SLOW}, \text{STOP}\}.$$

Each action corresponds to a different movement speed along the agent's trajectory and determines the agent's position at timestep $t + 1$. Action selection is synchronous across all agents.

Each of the agents perceives their surroundings through a limited sensor radius. Observations consist of the relative positions of nearby agents. Agents cannot access other agents internal states, expectation memories, or chosen actions prior to execution, and do not possess knowledge of future states beyond what is currently observable and what can be inferred about potential future states from current observations. The environment contains spatial regions that multiple agents may try to occupy at the location within overlapping timesteps. A collision will occur when two or more agents occupy the same spatial cell simultaneously. Collisions are treated as safety violations and in that particular agent failing the task attempt. Within this environment, each agents goal is to reach the designated goal region while avoiding collisions and deadlocks. An agent is considered successful if they can reach their target region without collisions or running out of fuel. The environment has no hardcoded right-of-way

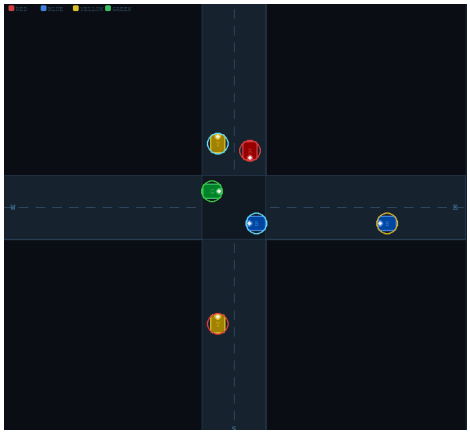

Fig. 1. Model of the MAS environment used within the experiment.

rules, priority hierarchies, normative constraints, or arbitration mechanisms. Coordination must emerge from local decision-making. That is, agents must reason about the potential consequences of both their actions and the actions of other agents in order to select behaviour that is compatible with their goals and the expected behaviour of other agents.

3.2 Coordination Criteria

To determine whether successful decentralised coordination has been achieved, three criteria are defined: safety, stability, and task completion. This criterion specifies the conditions for what we will consider as coordinated behaviour within this environment. The safety criterion requires the absence of collisions or fuel exhaustion. A collision will occur in the environment when two or more agents occupy the same area at the same timestep. Fuel exhaustion occurs when an agent depletes its fuel supply before crossing the intersection. An agent is considered safe if no collisions or fuel exhaustion occur during its movement towards its end goal. When an agent does collide with another agent both of those agents attempts are deemed fails and are recorded as crashes. When an agent runs out of fuel, the agent car is destroyed. Stability requires that agents avoid deadlock. Deadlock arises when two or more agents block each others progress preventing one another from advancing towards their goal even without the risk of a collision. A deadlock is characterised as a finite time interval in which multiple agents remain stationary in a mutually obstructive configuration. A deadlock is not considered present if all agents are making progress toward their goals over time. Task completion requires that agents reach their dedicated goal regions. Each agent starts from outside the contention area and has to traverse it to reach its goal. If the agent successfully reaches its goal region, then the task is considered successful. Using all three criteria, we consider the agents to be

showing coordination if all three can be reasonably satisfied, meaning the behaviour must be safe, stable, and successful in completing its tasks. For overall purposes, we will specifically measure the number of successful crosses compared to the number of failed attempts. This can show a measure of coordination in a decentralised environment by using Popperian Expectations.

4 Expectation-Driven Coordination

4.1 Expectation Definition

An expectation is a structured causal representation capturing the predicted relationship between conditions and their anticipated effects. Within the Popperian Expectations framework, expectations are formed through internal simulation or observation and stored for reuse in future decision-making. They function as reusable causal commitments that reduce the need for repeated simulation when similar scenarios are encountered. In this paper, expectations are represented as bounded executable EEC-style rules of the form `exp _rule (Cond, Exp)`. `Cond` is a set of predicates that describe the conditions under which the rule applies. `Exp` specifies the causal constraint expected to hold (or not hold) after the action is completed.

Condition (Cond) The condition defines when an expectation rule is applicable. Conditions are represented as sets of EEC-style predicates that include both events and fluents. Events capture the occurrence of an action at the current timestep using `Happens(action, t)`, while fluents capture contextual features using `HoldsAt(fluent, t)`. An `exp_rule` is triggered when its stored condition is satisfied by the current state and the selected action matches the specified event.

Expectation (Exp) The expected consequence is represented using Event Calculus predicates such as `Initiates(action, fluent, t)` and `Terminates(action, fluent, t)`. These encode whether an action is expected to cause a fluent to become true or false at the next timestep. Within this system, causal expectations are interpreted according to Event Calculus principles to derive predicted `HoldsAt` conclusions at time $t+1$. This allows an `exp_rule` to be used to reason about future fluents rather than serving as a purely descriptive record.

Confidence Each expectation contains a confidence value that represents how strongly an agent supports the rule based on previous confirmations or violations of the rule. Expectations that are confirmed are reinforced, whereas expectations that are violated are weakened and could be revised if their confidence is low enough. Confidence is used operationally as a gating mechanism where only expectations above a specified confidence threshold are permitted to skip simulations and be reused, in our programme, confidence had to be at least 68%. This helps ensure that poorly supported or incorrect expectations will not be used in decisions.

4.2 Expectation Lifecycle

Within the Popperian Expectations architecture, expectations evolve through four key stages: formation, storage, reuse, and revision. These stages allow agents to create hypothetical consequences using internal simulations, encode them as causal rules, reuse them when they are confident in them, and update them based on observed outcomes.

Formation Expectation formation occurs when an agent performs an internal simulation to evaluate candidate actions under current conditions. If there is no sufficiently confident expectation for a given context, the agent performs a forward simulation of that action. The simulated outcome predicts changes in safety or other progress relevant fluents. From the simulation, the agent can construct a structured causal rule linking the simulated condition to its resulting effect. The condition captures contextual predicates that are present along with the selected action. The effect captures the fluent state predicted by the simulation. Together, this represents a newly formed expectation.

Storage Expectations are encoded as structured EEC rules in the form of

```
exp_rule(Cond, Exp)
```

where cond represents the triggering predicate set and exp represents the expected causal effect. The rule is saved in the agents expectation memory along with its associated confidence value. Storage preserves causal knowledge across timesteps and interactions. Unlike purely reactive or purely-simulation agents, the agent can avoid creating identical predictions when confident expectations are available. This allows expectations to function as reusable causal abstractions that can persist over time in an agents memory.

Reuse During decision-making, the agent evaluates whether it contains stored expectations that are applicable in the current context. An expectation is eligible to be reused when its condition matches the current context and its confidence exceeds a predefined inference threshold. When an expectation exists, the stored `exp_rule` is triggered and the associated Initiates or Terminates predicates are used to predict fluent states at the next timestep. If a confident expectation is available, the agent may bypass or shorten simulations by reusing the saved expectation rule from the previous time it saw the scenario.

Revision After the execution of an action, the agent will observe the resulting state transition. The observed result is compared with the effect that was predicted by the causal expectations used. If the predicted fluent holds as anticipated, then the expectation is confirmed, and confidence in the expectation is increased. If the predicted effect fails, then the expectation is weakened or

possibly revised. This mechanism for confirmation and violation of stored expectations ensures that expectations remain empirically grounded, where rules that repeatedly fail will lose influence in decision-making, causing the agent to resume simulating for that context. This enables agents to also adapt overtime since no rules are ever hardcoded within the agent itself.

5 Agent Architecture

5.1 Agent Control Loop

At each timestep every agent executes a fixed control loop that consists of perception, expectation evaluation, action selection, execution, and revision. The agent first constructs a representation of the current scene from local perceptual inputs. This includes spatial relationship with the nearby agents along with fluents related to safety and blocking. These predicates form the basis for matching the expectations stored. For each admissible action in the action set U = Speed,GO,SLOW,STOP the agent performs a forward simulation with a fixed horizon that limits how far the simulator can look ahead when evaluating actions, and will evaluate whether it contains a sufficiently confident expectation. If a relevant high-confidence expectation exists that predicts the consequence of executing that action under the current conditions, then the agent may use the expectation. In the agent implementation, this reuse is governed by a short-circuit confidence threshold, where only expectations that exceed the required confidence bounds can bypass simulation If no confident expectations exist, then the agent will perform a bounded forward simulation. The simulation predicts relevant safety and progress outcomes and can generate new expectations based on observed transitions. This ensures that the internal simulation of an agent will only be used when its memory is not sufficient to make a decision. The agent will select the action with the highest utility evaluated, incorporating collision and blocking risk, and forward progress. After execution of the action, the agent will observe the resulting state transition and perform confirmation and violation checks against any expectations that were triggered during decision-making and update their confidence values accordingly.

5.2 Expectation Memory

Expectation memory is implemented as a structured set of ExpRule(cond, exp) objects. Each rule corresponds to an EEC style predicate `exp_rule(Cond, Exp)`. Cond represents a set of predicates that includes action predicates of the form Happens(action, t) and contextual fluents, and Exp represents causal constraints such as Initiates(action, fluent) or Terminates(action, fluent). Each stored expectation rule also maintains associated metadata, which includes:

- Confirmation count.
- Violation Count.
- Accumulated confirmation weight.

– Confidence score.

Confidence is computed as a bounded scalar and change based on changes in value due to confirmations or violations of the expectation. Expectations that are below the threshold for use are retained, but do not influence deduction until they are higher confidence.

5.3 Expectation Reuse

Matching expectations occurs by evaluating the stored `exp_rule` predicates. An expectation rule can trigger when its condition is satisfied by an agent's current predicates. Triggered expectation rules are passed to the ECReasoner which reasons about predicted future fluents. Particularly, Initiates and Terminates predicates are interpreted to generate predicted HoldsAt conclusions in the next timestep. Rules that have a confidence value exceeding the inference threshold can participate in deduction, but higher confidence values are required to directly bypass simulation.

5.4 Confidence Updating

After executing an action, the agent compares the observed state with the predicted effects of the triggered expectations. If the expectation holds, then the confidence in that expectation increases. However, if the expectation is proven wrong, then the expectation is violated, and confidence in the expectation decreases. When confidence in an expectation falls below the confidence threshold, the expectation is pruned, thus removing the corresponding rule and preventing further reuse. The confidence update ensures that agents retain only valuable and useful expectations.

5.5 Decision Hierarchy

At each timestep, the agent evaluates potential actions through a four-layered decision hierarchy. Each layer represents a distinct area of intellectual trust. A layer is only consulted when higher-priority layers lack sufficient confidence or knowledge to commit to an action. This helps ensure that simulation is used only when the agent lacks causal knowledge, reducing redundant simulations while still preserving safety. The hierarchy consists of four layers, KB Deduction, Short Circuit, EC Reasoning, and Monte Carlo. As agents accumulate experience and knowledge, the two top layers will take on much of the decision-making.

KB Deduction Layer The agent first consults the available causal knowledge it contains. If a rule of the form Initiates(action, fluent, context) or Terminates(action, fluent, context) exists where the context matches the current scene and the confidence exceeds the threshold, then the agent will reason the predicted outcome using Event Calculus forward chaining. This layer contains the

highest-confidence rules, which have been observed multiple times and have survived real-world falsification. If the layer produces high-confidence predictions, then the simulation is completely bypassed.

Short Circuit Layer If the KB layer does not contain sufficiently confident rules and is not used the agent will then directly consult its expectation memory. If a stored expectation rule exists whose condition matches the current context and has the required confidence level, the agent will reuse that expectation without simulation. This layer differs from the KB layer because it operates on individual expectations rather than generalised causal laws; it reasons about pattern recognition rather than abstract causal knowledge.

EC Reasoning Layer If no layer up to this point can commit to an action, the agents use the Event Calculus to reason over a short forward horizon using whatever expectation rules they currently possess, even if they are of low confidence. This layer looks to create predictions in contexts where KB rules have incomplete coverage. It focuses on reasoning about potential future fluents rather than simulating.

Monte Carlo Layer If all three layers fail to produce an action, the agent will fall back to a bounded forward simulation using Monte Carlo. The trials are simulated in the current context, sample possible outcomes, and estimate the collision probability for each potential action. The agent with the lowest collision risk and the highest utility is then selected. This layer represents a purely empirical base that uses no causal knowledge; its results will both select an action and create new expectations for future use.

6 Methodology

This experiment is intended as a feasibility demonstration of the use of expectations to enable coordination in decentralised environments. The main objective is to determine whether decentralised coordination can emerge when agents rely solely on internally formed causal expectations, without centralised control or hard-coded rules. The simulation was run for a total of 2000 crossing attempts, with each successful cross counted as an attempt, and each collision counted as two attempts, since two cars are involved. In total, there were four agents, each distinguished by a different colour: red, blue, yellow, and green. All blue cars, for example, were operated by a single agent who could control multiple of their own cars; however, these cars always stayed in their corresponding lanes. Each agent would randomly spawn cars, with at most one car per agent present in the environment at any given time. All four agents had the same capacity to create and use causal expectations and were internally identical. When the programme started, each agent had no knowledge of the capabilities of other agents and knew only what it itself was capable of; it could learn only through observation

and internal simulations to create causal rules. The agents have no ability to communicate or coordinate with one another; the only mechanism they use to coordinate across the intersection is the creation and use of causal rules. The agent could create as many causal rules as it found applicable, and, if a causal rule was falsified in the simulation, the agent could update that rule.

Monte Carlo Baseline Comparison To evaluate the contribution and usefulness of using causal expectation learning, we compare our results with a Monte Carlo baseline agent. This agent was implemented using an identical simulation environment and action set. The baseline agents uses the same bounded forward simulation present within the Monte Carlo layer of the EEC agent. The baseline does not create, store, or reuse any causal knowledge it has learnt from its simulations which requires it to resimulate every potential action regardless of how often it has seen that scenario. The agent selects actions purely by sampling rollout trials and choosing the action with the lowest collision risk. All other parameters in the agent are identical to the EEC agent.

7 Results

Over 2000 total attempts for all of the agents combined, there were a total of 1946 successful crossings, and a total of 27 crashes. Overall, this yielded a success rate of 98.63% across the simulation experiment. Throughout the total simulation duration, the agents learnt a combined number of 40,544 causal expectation rules with a total of 856 causal rules being pruned for low confidence. Over the total run the agents used causal expectations a total of 245,286 times and resorted to simulation results 733,116 times.

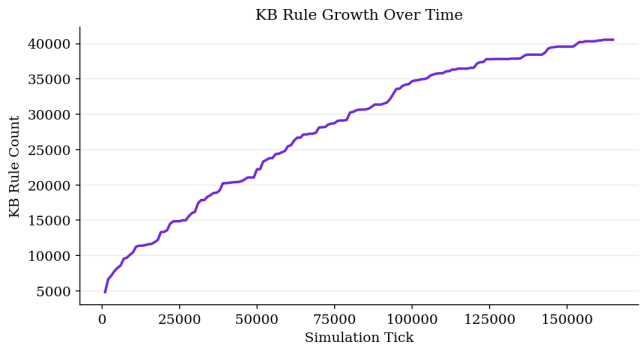

Fig. 2. The growth of KB rules over time

As seen in 2, agents rapidly were able to learn expectations about the environment with a plateau in learnt expectations beginning to happen around

tick 12k. This means that, roughly over halfway through the simulation, the agents were able to identify enough rules to accurately model and anticipate the environment. This suggests that agents' causal knowledge can converge over time rather than grow indefinitely. This means that agents are not merely creating rules at random, but seem to converge on a stable representation of their environment.

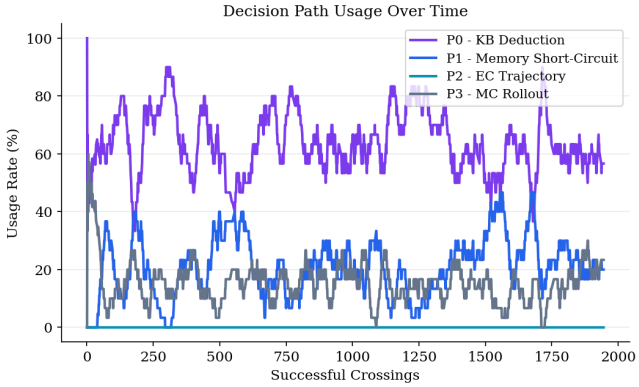

Fig. 3. Shows how frequently each layer is firing based on continued success

3 shows the rate at which each of the four layers is being utilised over the course of the agent's success. As 1 shows, once the agent begins achieving successful crossing of the levels, each of which the four layers are utilised, the level remains fairly stable over the entire course of the experiment, with the deductive layer being used a majority of the time in the experiment. This suggests that agents are quickly generalising behaviours and that using those generalised behaviours leads to continued successful crossings. To further analyse this, however, we have to look at how coordination within agents also evolved.

This result in 4 suggests that the use of explicit causal rules within a decentralised MAS environment does enable a stable level of coordination and that this area of causal reasoning in MAS provides a promising starting point for the use of causal expectations to enable coordination in decentralised tasks. Coordination within the simulation did indeed arise without the use or implementation of hardcoded priors or explicit frameworks for coordination but rather through the application of what-if hypotheses internally tested by agents and stored and used as causal rules using the EEC.

7.1 Baseline comparison

The baseline was able to achieve a marginally higher success rate, which can be expected, an agent that is able to resimulate every decision without being constrained can retest every decision it makes. However, the EEC agent

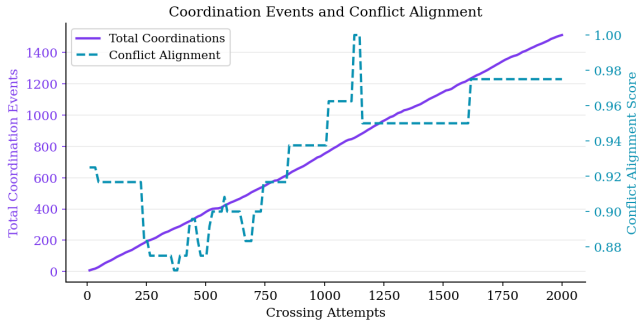

Fig. 4. The coordination events and conflict alignment of agents over time

Metric	EEC	Baseline
Success rate	98.63%	99.95%
Crashes	27	1
Total simulations	733,116	6,530,568
Simulations per attempt	366.6	3,265.3
KB coverage	67.6%	–
Conflict alignment	0.975	0.917
Rules learned	40,544	–
Rules pruned	856	–

Table 1. Performance comparison between EEC and the baseline simulation.

achieved comparable performance with 8.9 times fewer total simulations. This efficiency gain arises from the use of the decision hierarchy. In 85% of crossings, the agent chose a successful action using KB deduction or memory reuse without simulation, and only fell back to simulation in 15% of cases. The EEC agent also achieved a higher conflict alignment score than the baseline, indicating that causal expectation learning produced a more consistent role differentiation. When an agent learned a causal rule about when to yield, it consistently applied it in subsequent encounters, whereas a purely simulation-based agent could yield different outcomes each time, which does not lead to coordination.

An area in which the EEC agent showed dramatic improvements compared to the baseline agent was in terms of simulation overhead; the EEC agent was able to achieve 98.6% success using only 733,116 simulations, while the baseline agent did achieve 99.9% success but had to use 6,530,568 simulations to accomplish this. As show in 5 the number of simulations used by the baseline drastically exceeds the number used by the EEC program. In the 27 crashes experienced by the EEC agent most were concentrated in contexts that fell through to simulation use and consisted of scenes that were not yet fully covered by KB rules. This does represent one of the tradeoffs for using causal expectations and knowledge over simulations, which is that an agent will always pay some inherent safety or efficiency cost while creating and learning rules.

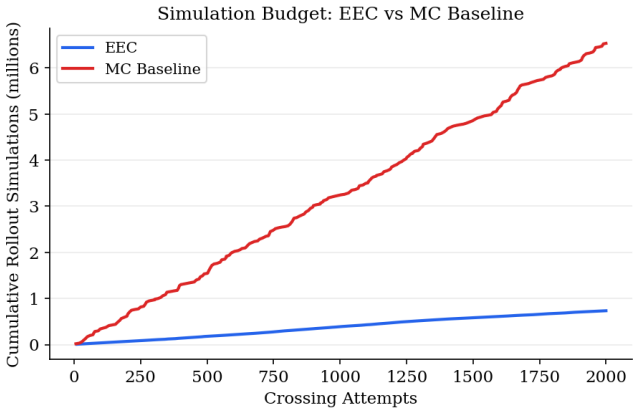

Fig. 5. The number of simulations used by each agent program over time

8 Conclusion

The demonstration of explicit causal rules for coordination in MAS yields encouraging results for future work in causal learning for MAS. It shows how causal learning can enable a sufficient level of reasoning and coordination within MAS and the importance of agents that can causally reason when in these environments. This is not without potential limitations though as the baseline did achieve a success rate 1.4% higher than the EEC agents, however most of the crashes with the EEC came from simulations. This could potentially be due to the agent not properly simulating the potential actions of an agent who already possesses its own expectation and therefore may act differently. Potential future work in this area could examine how causal learning and causal expectations can be combined with more specific coordination frameworks for MAS and implemented to improve upon existing causal MAS learning frameworks.

References

1. Shoham, Y., Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press, Cambridge (2008)
2. Anonymous Authors: Title withheld for review, ACSOS-C 2025
3. Anonymous Authors: Title withheld for review, ACSOS-C 2025
4. Cranefield, S.: Agents and Expectations. In: Balke, T., Dignum, F., van Riemsdijk, M.B., Chopra, A.K. (eds.) Coordination, Organizations, Institutions, and Norms in Agent Systems IX. Springer International Publishing, pp. 234–255 (2014)
5. Dennett, D.C.: Kinds of Minds: Toward an Understanding of Consciousness. Basic Books, New York (1996)
6. Dennett, D.C.: Darwin's Dangerous Idea: Evolution and the Meanings of Life. Simon & Schuster, New York (1995)
7. Winfield, A.F.: Robots with Internal Models: A Route to Self-Aware and Hence Safer Robots. In: The Computer After Me, pp. 237–252 (2014).

8. Winfield, A.F.T.: How Intelligent is your Intelligent Robot? arXiv preprint arXiv:1712.08878 (2017). DOI: 10.48550/arXiv.1712.08878
9. Shanahan, M.: The Event Calculus Explained. In: Wooldridge, M., Veloso, M. (eds.) Artificial Intelligence Today: Recent Trends and Developments. Springer, Berlin, Heidelberg, pp. 409–430 (2001)
10. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games 4(1), 1–43 (2012)