Co-Learning Empirical Games and World Models

Max Olan Smith
University of Michigan
mxsmith@umich.edu

Michael P. Wellman
University of Michigan
wellman@umich.edu

Abstract

Game-based decision-making involves reasoning over both world dynamics and strategic interactions among the agents. Typically, empirical models capturing these respective aspects are learned and used separately. We investigate the potential gain from co-learning these elements: a world model for dynamics and an empirical game for strategic interactions. Empirical games drive world models toward a broader consideration of possible game dynamics induced by a diversity of strategy profiles. Conversely, world models guide empirical games to efficiently discover new strategies through planning. We demonstrate these benefits first independently, then in combination as realized by a new algorithm, Dyna-PSRO, that co-learns an empirical game and a world model. When compared to PSRO—a baseline empirical-game building algorithm, Dyna-PSRO is found to compute lower regret solutions on partially observable general-sum games. In our experiments, Dyna-PSRO also requires substantially fewer experiences than PSRO, a key algorithmic advantage for settings where collecting player-game interaction data is a cost-limiting factor.

1 Introduction

Even seemingly simple games can actually embody a level of complexity rendering them intractable to direct reasoning. This complexity stems from the interplay of two sources: dynamics of the game environment, and strategic interactions among the game's players. As an alternative to direct reasoning, models have been developed to facilitate reasoning over these distinct aspects of the game. ***Empirical games*** capture strategic interactions in the form of payoff estimates for joint policies [82]. ***World models*** represent a game's transition dynamics and reward signal directly [71, 20]. Whereas each of these forms of model have been found useful for game reasoning, typical use in prior work has focused on one or the other, learned and employed in isolation from its natural counterpart.

Co-learning both models presents an opportunity to leverage their complementary strengths as a means to improve each other. World models predict successor states and rewards given a game's current state and action(s). However, their performance depends on coverage of their training data, which is limited by the range of strategies considered during learning. Empirical games can inform training of world models by suggesting a diverse set of salient strategies, based on game-theoretic reasoning [82]. These strategies can expose the world model to a broader range of relevant dynamics. Moreover, as empirical games are estimated through simulation of strategy profiles, this same simulation data can be reused as training data for the world model.

Strategic diversity through empirical games, however, comes at a cost. In the popular framework of Policy-Space Response Oracles (PSRO) [40], empirical normal-form game models are built iteratively, at each step expanding a restricted strategy set by computing best-response policies to the current game's solution. As computing an exact best-response is generally intractable, PSRO uses Deep Reinforcement Learning (DRL) to compute approximate response policies. However, each application of DRL can be considerably resource-intensive, necessitating the generation of a vast amount of gameplays for learning. Whether gameplays, or experiences, are generated via

Preprint. Under review.

simulation [50] or from real-world interactions [25], their collection poses a major limiting factor in DRL and by extension PSRO. World models present one avenue to reduce this cost by transferring previously learned game dynamics across response computations.

We investigate the mutual benefits of co-learning a world model and an empirical game by first verifying the potential contributions of each component independently. We then show how to realize the combined effects in a new algorithm, *Dyna-PSRO*, that co-learns a world model and an empirical game (illustrated in Figure 1). Dyna-PSRO extends PSRO to learn a world model concurrently with empirical game expansion, and applies this world model to reduce the computational cost of computing new policies. This is implemented by a Dyna-based reinforcement learner [69, 70] that integrates planning, acting, and learning in parallel. Dyna-PSRO is evaluated against PSRO on a collection of partially observable general-sum games. In our experiments, Dyna-PSRO found lower-regret solutions while requiring substantially fewer cumulative experiences.

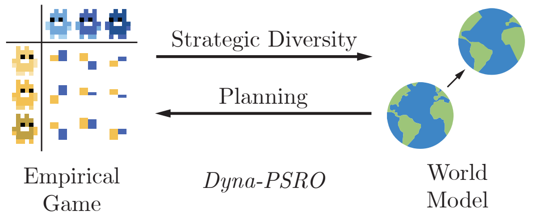

Figure 1: Dyna-PSRO co-learns a world model and empirical game. Empirical games offer world models strategically diverse game dynamics. World models offer empirical games more efficient strategy discovery through planning.

The main points of novelty of this paper are as follows: (1) empirically demonstrate that world models benefit from the strategic diversity induced by an empirical game; (2) empirically demonstrate that a world model can be effectively transferred and used in planning with new other-players. The major contribution of this work is a new algorithm, Dyna-PSRO, that co-learns an empirical game and world model finding a stronger solution at less cost than the baseline, PSRO.

2 Related Work

Empirical Game Theoretic Analysis (EGTA). The core idea of EGTA [82] is to reason over approximate game models (*empirical games*) estimated by simulation over a restricted strategy set. This basic approach was first demonstrated by Walsh et al. [79], in a study of pricing and bidding games. Phelps et al. [53] introduced the idea of extending a strategy set automatically through optimization, employing genetic search over a policy space. Schvartzman & Wellman [60] proposed using RL to derive new strategies that are approximate best responses (BRs) to the current empirical game's Nash equilibrium. The general question of which strategies to add to an empirical game has been termed the *strategy exploration problem* [33]. PSRO [40] generalized the target for BR beyond NE, and introduced DRL for BR computation in empirical games. Many further variants and extensions of EGTA have been proposed, for example those using structured game representations such as extensive-form [45, 36]. Some prior work has considered transfer learning across BR computations in EGTA, specifically by reusing elements of policies and value functions [66, 67].

Model-Based Reinforcement Learning (MBRL). *Model-Based* RL algorithms construct or use a model of the environment (henceforth, *world model*) in the process of learning a policy or value function [71]. World models may either predict successor observations directly (e.g., at pixel level [78, 81]), or in a learned latent space [19, 18]. The world models can be either used for *background planning* by rolling out model-predicted trajectories to train a policy, or by *decision-time planning* where the world model is used to evaluate the current state by planning into the future. Talvitie [73] demonstrated that even in small Markov decision processes (MDP) [54], model-prediction errors tend to compound—rendering long-term planning at the abstraction of observations ineffective. A follow-up study demonstrated that for imperfect models, short-term planning was no better than repeatedly training on previously collected real experiences; however, medium-term planning offered advantages even with an imperfect model [29]. Parallel studies hypothesized that these errors are a result of insufficient data for that transition to be learned [38, 8]. To remedy the data insufficiency, ensembles of world models were proposed to account for world model uncertainty [8, 38, 86], and another line of inquiry used world model uncertainty to guide exploration in state-action space [3, 61]. This study extends this problem into the multiagent setting, where now other-agents may preclude transitions from occurring. The proposed remedy is to leverage the strategy exploration process of building an empirical game to guide data generation.

Multiagent Reinforcement Learning (MARL). Previous research intersecting MARL and MBRL has primarily focused on modeling the opponent, particularly in scenarios where the opponent is fixed and well-defined. Within specific game sub-classes, like cooperative games and two-player zero-sum games, it has been theoretically shown that opponent modeling reduces the sample complexity of RL [75, 87]. Opponent models can either explicitly [48, 16] or implicitly [4, 31] model the behavior of the opponent. Additionally, these models can either construct a single model of opponent behavior, or learn a set of models [13, 22]. While opponent modeling details are beyond the scope of this study, readers can refer to Albrecht & Stone's survey [1] for a comprehensive review on this subject. Instead, we consider the case where the learner has explicit access to the opponent's policy during training, as is the case in empirical-game building. A natural example is that of Self-Play, where all agents play the same policy; therefore, a world model can be learned used to evaluate the quality of actions with Monte-Carlo Tree Search [62, 64, 74, 58]. Li et al. [43] expands on this by building a population of candidate opponent policies through PSRO to augment the search procedure. Krupnik et al. [37] demonstrated that a generative world model could be useful in multi-step opponent-action prediction. Sun et al. [68] examined modeling stateful game dynamics from observations when the agents' policies are stationary. Chockalingam et al. [12] explored learning world models for homogeneous agents with a centralized controller in a cooperative game. World models may also be shared by independent reinforcement learners in cooperative games [83, 88].

3 Co-Learning Benefits

We begin by specifying exactly what we mean by world model and empirical game. This requires defining some primitive elements. Let $t \in \mathcal{T}$ denote time in the real game, with $s^t \in \mathcal{S}$ the ***information state*** and $h^t \in \mathcal{H}$ the ***game state*** at time t. The information state $s^t \equiv (m^{\pi,t}, o^t)$ is composed of the ***agent's memory*** $m^\pi \in \mathcal{M}^\pi$, or recurrent state, and the current ***observation*** $o \in \mathcal{O}$. Subscripts denote a player-specific component s_i, negative subscripts denote all but the player s_{-i}, and boldface denote the joint of all players $\boldsymbol{s}$. The ***transition dynamics*** $p : \mathcal{H} \times \boldsymbol{\mathcal{A}} \rightarrow \Delta(\mathcal{H}) \times \Delta(\boldsymbol{\mathcal{R}})$ define the game state update and reward signal. The agent experiences ***transitions***, or ***experiences***, $(s^t,\ a^t, r^{t+1}, s^{t+1})$ of the game; where, sequences of transitions are called ***trajectories*** τ and trajectories ending in a terminal game state are ***episodes***.

At the start of an episode, all players sample their current ***policy*** π from their ***strategy*** $\sigma : \Pi \rightarrow [0, 1]$, where Π is the ***policy space*** and Σ is the corresponding ***strategy space***. A ***utility function*** $U : \boldsymbol{\Pi} \rightarrow \mathbb{R}^n$ defines the payoffs/returns (i.e., cumulative reward) for each of n players. The tuple $\Gamma \equiv (\boldsymbol{\Pi}, U, n)$ defines a ***normal-form game*** (NFG) based on these elements. We represent empirical games in normal form. An ***empirical normal-form game*** (ENFG) $\hat{\Gamma} \equiv (\hat{\boldsymbol{\Pi}}, \hat{U}, n)$ models a game with a ***restricted strategy set*** $\hat{\boldsymbol{\Pi}}$ and an estimated payoff function $\hat{U}$. An empirical game is typically built by alternating between game reasoning and strategy exploration. During the game reasoning phase, the empirical game is solved based on a solution concept predefined by the modeler. The strategy exploration step uses this solution to generate new policies to add to the empirical game. One common heuristic is to generate new policies that best-respond to the current solution [47, 59]. As exact best-responses typically cannot be computed, RL or DRL are employed to derive approximate best-responses [40].

An ***agent world model*** w represents dynamics in terms of information available to the agent. Specifically, w maps information states and actions to observations and rewards, $w : \boldsymbol{\mathcal{O}} \times \boldsymbol{\mathcal{A}} \times \mathcal{M}^w \rightarrow \boldsymbol{\mathcal{O}} \times \boldsymbol{\mathcal{R}}$, where $m^w \in \mathcal{M}^w$ is the ***world model's memory***, or recurrent state. For simplicity, in this work, we assume the agent learns and uses a deterministic world model, irrespective of stochasticity that may be present in the true game. Specific implementation details for this work are provided in Appendix C.2.

Until now, we have implicitly assumed the need for distinct models. However, if a single model could serve both functions, co-learning two separate models would not be needed. Empirical games, in general, cannot replace a world model as they entirely abstract away any concept of game dynamics. Conversely, world models have the potential to substitute for the payoff estimations in empirical games by estimating payoffs as rollouts with the world model. We explore this possibility in an auxiliary experiment included in Appendix E.4, but our findings indicate that this substitution is impractical. Due to compounding of model-prediction errors, the payoff estimates and entailed game solutions were quite inaccurate.

Having defined the models and established the need for their separate instantiations, we can proceed to evaluate the claims of beneficial co-learning. Our first experiment shows that the strategic diversity embodied in an empirical game yields diverse game dynamics, resulting in the training of a more performant world model. The second set of experiments demonstrates that a world model can help reduce the computational cost of policy construction in an empirical game.

3.1 Strategic Diversity

A world model is trained to predict successor observations and rewards, from the current observations and actions, using a supervised learning signal. Ideally, the training data would cover all possible transitions. This is not feasible, so instead draws are conventionally taken from a dataset generated from play of a ***behavioral strategy***. Performance of the world model is then measured against a ***target strategy***. Differences between the behavioral and target strategies present challenges in learning an effective world model.

We call the probability of drawing a state-action pair under some strategy its ***reach probability***. From this, we define a strategy's ***strategic diversity*** as the distribution induced from reach probabilities. across the full state-action space. These terms allow us to observe two challenges for learning world models. First, the diversity of the behavioral strategy ought to *cover* the target strategy's diversity. Otherwise, transitions will be absent from the training data. It is possible to supplement coverage of the absent transitions if they can be generalized from covered data; however, this cannot be generally guaranteed. Second, the *closer* the diversities are, the more accurate the learning objective will be. An extended formal argument of these challenges is provided in Appendix C.3.

If the target strategy were known, we could readily construct the ideal training data for the world model. However the target is generally not known at the outset; indeed determining this target is the ultimate purpose of empirical game reasoning. The evolving empirical game essentially reflects a search for the target. Serendipitously, construction of this empirical game entails generation of data that captures elements of likely targets. This data can be reused for world model training without incurring any additional data collection cost.

Game. We evaluate the claims of independent co-learning benefits within the context of a *commons game* called "Harvest". In Harvest, players move around an orchard picking apples. The challenging commons element is that apple regrowth rate is proportional to nearby apples, so that socially optimum behavior would entail managed harvesting. Self-interested agents capture only part of the benefit of optimal growth, thus non-cooperative equilibria tend to exhibit collective over-harvesting. The game has established roots in human-behavioral studies [32] and in agent-based modeling of emergent behavior [55, 42, 41]. For our initial experiments, we use a symmetric two-player version of the game, where in-game entities are represented categorically [30]. Each player has a 10×10 viewbox within their field of vision. The possible actions include moving in the four cardinal directions, rotating either way, tagging, or remaining idle. A successful tag temporarily removes the other player from the game, but can only be done to other nearby players. Players receive a reward of 1 for each apple picked. More detailed information and visualizations are available in Appendix D.1.

Experiment. To test the effects of strategic diversity, we train a suite of world models that differ in the diversity of their training data. The datasets are constructed from the play of three policies: a random baseline policy, and two PSRO-generated policies. The PSRO policies were arbitrarily sampled from an approximate solution produced by a run of PSRO. We sampled an additional policy from PSRO for evaluating the generalization capacity of the world models. These policies are then subsampled and used to train seven world models. The world models are referred to by icons ▦ that depict the symmetric strategy profiles used to train them in the normal-form. Strategy profiles included in the training data of the world models are shaded black. For instance, the first (random) policy ▦, or the first and third policies ▦. Each world model's dataset contains 1 million total transitions, collected uniformly from each distinct strategy profile (symmetric profiles are not re-sampled). The world models are then evaluated on accuracy and recall for their predictions of both observation and reward for both players. The world models are optimized with a weighted-average cross-entropy objective. Additional details are in Appendix C.2.

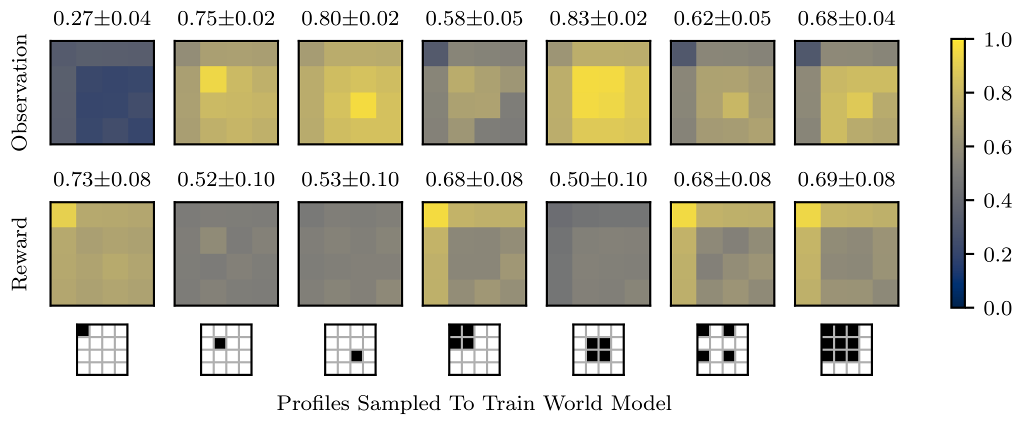

Figure 2: World model accuracy across strategy profiles. Each heatmap portrays a world model's accuracy over 16 strategy profiles. The meta x-axis corresponds to the profiles used to train the world model (as black cells). Above each heatmap is the model's average accuracy.

Results. Figure 2 presents each world model's per-profile accuracy, as well as its average over all profiles. Inclusion of the random policy corresponds to decreases in observation prediction accuracy: ▦ $0.75 \pm 0.02 \rightarrow$ ▦ 0.58 ± 0.05, ▦ $0.80 \pm 0.02 \rightarrow$ ▦ 0.62 ± 0.05, and ▦ $0.83 \pm 0.02 \rightarrow$ ▦ 0.68 ± 0.04. Figure 14 (Appendix E.1) contains the world model's per-profile recall. Inclusion of the random policy corresponds to increases in reward 1 recall: ▦ $0.25 \pm 0.07 \rightarrow$ ▦ 0.37 ± 0.11, ▦ $0.25 \pm 0.07 \rightarrow$ ▦ 0.36 ± 0.11, and ▦ $0.26 \pm 0.07 \rightarrow$ ▦ 0.37 ± 0.11.

Discussion. The PSRO policies offer the most strategically salient view of the game's dynamics. Consequently, the world model ▦ trained with these policies yields the highest observation accuracy. However, this world model performs poorly on reward accuracy, scoring only 0.50 ± 0.10. In comparison, the model trained on the random policy ▦ scores 0.73 ± 0.08. This seemingly counterintuitive result can be attributed to a significant class imbalance in rewards. ▦ predicts only the most common class, no reward, which gives the illusion of higher performance. In contrast, the remaining world models attempt to predict rewarding states, which reduces their overall accuracy. Therefore, we should compare the world models based on their ability to recall rewards. When we examine ▦ again, we find that it also struggles to recall rewards, scoring only 0.26 ± 0.07. However, when the random policy is included in the training data (▦), the recall improves to 0.37 ± 0.11. This improvement is also due to the same class imbalance. The PSRO policies are highly competitive, tending to over-harvest. This limits the proportion of rewarding experiences. Including the random policy enhances the diversity of rewards in this instance, as its coplayer can demonstrate successful harvesting. Given the importance of accurately predicting both observations and rewards for effective planning, ▦ appears to be the most promising option. However, the strong performance of ▦ suggests future work on algorithms that can benefit solely from observation predictions. Overall, these results support the claim that strategic diversity enhances the training of world models.

3.2 Response Calculations

Empirical games are built by iteratively calculating and incorporating responses to the current solution. However, direct computation of these responses is often infeasible, so RL or DRL is used to approximate the response. This process of approximating a single response policy using RL is computationally intensive, posing a significant constraint in empirical game modeling when executed repeatedly. World models present an opportunity to address this issue. A world model can serve as a medium for transferring previously learned knowledge about the game's dynamics. Therefore, the dynamics need not be relearned, reducing the computational cost associated with response calculation.

Exercising a world model for transfer is achieved through a process called ***planning***. Planning is any procedure that takes a world model and produces or improves a policy. In the context of games, planning can optionally take into account the existence of coplayers. This consideration can reduce experiential variance caused by unobserved confounders (i.e., the coplayers). However, coplayer modeling errors may introduce further errors in the planning procedure [22].

Planning alongside empirical-game construction allows us to side-step this issue as we have direct access to the policies of all players during training. This allows us to circumvent the challenge of building accurate agent models. Instead, the policies of coplayers can be directly queried and used alongside a world model, leading to more accurate planning. In this section, we empirically demonstrate the effectiveness of two methods that decrease the cost of response calculation by integrating planning with a world model and other agent policies.

3.2.1 Background Planning

The first type of planning that is investigated is ***background planning***, popularized by the Dyna architecture [69]. In background planning, agents interact with the world model to produce ***planned experiences***[1]. The planned experiences are then used by a model-free reinforcement learning algorithm as if they were ***real experiences*** (experiences generated from the real game). Background planning enables learners to generate experiences of states they are not currently in.

Experiment. To assess whether planned experiences are effective for training a policy in the actual game, we compute two response policies. The first response policy, serving as our baseline, learns exclusively from real experiences. The second response policy, referred to as the planner, is trained using a two-step procedure. Initially, the planner is exclusively trained on planned experiences. After 10 000 updates, it then transitions to learning solely from real experiences. Policies are trained using IMPALA [15], with further details available in Appendix C.1. The planner employs the ▛ world model from Section 3.1, and the opponent plays the previously held-out policy. In this and subsequent experiments, the cost of methods is measured by the number of experiences they require with the actual game. This is because, experience collection is often the bottleneck when applying RL-based methods [50, 25]. Throughout the remainder of this work, each experience represents a trajectory of 20 transitions, facilitating the training of recurrent policies.

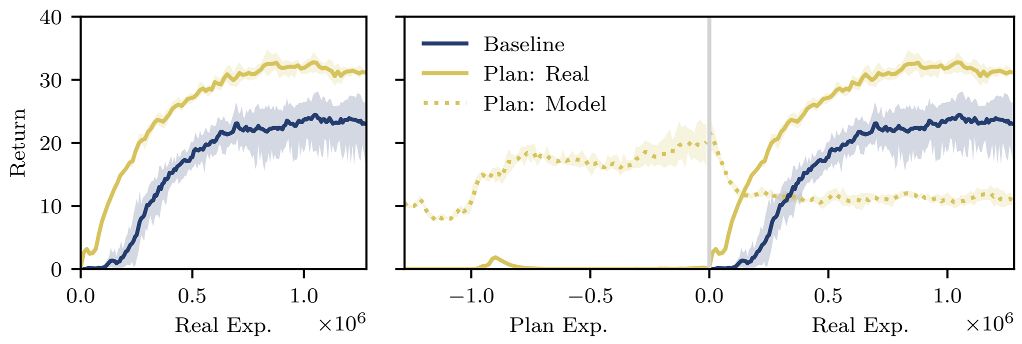

Figure 3: Effects of background planning on response learning. Left: Return curves measured by the number of real experiences used. Right: Return curves measured by usage of both real and planned experiences. The planner's return is measured against the real game and the world model. (5 seeds, with 95 % bootstrapped CI).

Results. Figure 3 presents the results of the background planning experiment. The methods are compared based on their final return, utilizing an equivalent amount of real experiences. The baseline yields a return of 23.00 ± 4.01, whereas the planner yields a return of 31.17 ± 0.25.

Discussion. In this experiment, the planner converges to a stronger policy, and makes earlier gains in performance than the baseline. Despite this, there is a significant gap in the planner's learning curves, which are reported with respect to both the world model and real game. This gap arises due to accumulated model-prediction errors, causing the trajectories to deviate from the true state space. Nevertheless, the planner effectively learns to interact with the world model during planning, and

[1]Other names include "imaginary", "simulated", or "hallucinated" experiences.

this behavior shows positive transfer into the real game, as evidenced by the planner's rapid learning. The exact magnitude of benefit will vary across coplayers' policies, games, and world models. In Figure 15 (Appendix E.2), we repeat the same experiment with the poorly performing ▦ world model, and observe a marginal benefit (26.05 ± 1.32). The key take-away is that background planning tends to lead towards learning benefits, and not generally hamper learning.

3.2.2 Decision-Time Planning

The second main way that a world model is used is to inform action selection at ***decision time [planning] (DT)***. In this case, the agent evaluates the quality of actions by comparing the value of the model's predicted successor state for all candidate actions. Action evaluation can also occur recursively, allowing the agent to consider successor states further into the future. Overall, this process should enable the learner to select better actions earlier in training, thereby reducing the amount of experiences needed to compute a response. A potential flaw with decision-time planning is that the agent's learned value function may not be well-defined on model-predicted successor states [73]. To remedy this issue, the value function should also be trained on model-predicted states.

Experiment. To evaluate the impact the decision-time planning, we perform an experiment similar to the background planning experiment (Section 3.2.1). However, in this experiment, we evaluate the quality of four types of decision-time planners that perform one-step three-action search. The planners differ in the their ablations of background planning types: (1) ***warm-start background planning (BG: W)*** learning from planned experiences before any real experiences, and (2) ***concurrent background planning (BG: C)*** where after BG: W, learning proceeds simultaneously on both planned and real experiences. The intuition behind BG: C is that the agent can complement its learning process by incorporating planned experiences that align with its current behavior, offsetting the reliance on costly real experiences. Extended experimental details are provided in Appendix C.

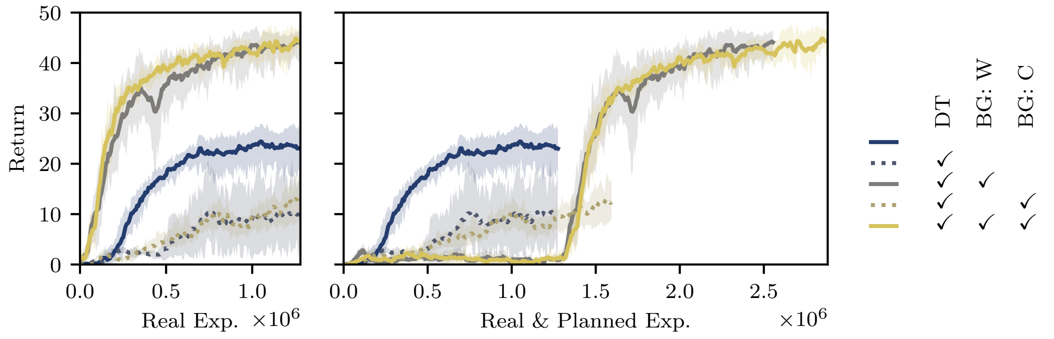

Figure 4: Effects of decision-time planning on response learning. Four planners using decision-time planning (DT) are shown in combinations with warm-start background planning (BG: W) and concurrent background planning (BG: C). (5 seeds, with 95 % bootstrapped CI).

Results. The results for this experiment are shown in Figure 4. The baseline policy receives a final return of 23.00 ± 4.01. The planners that do not include BG: W, perform worse, with final returns of 9.98 ± 7.60 (DT) and 12.42 ± 3.97 (DT & BG: C). The planners that perform BG: W outperform the baseline, with final returns of 44.11 ± 2.81 (DT & BG: W) and 44.31 ± 2.56 (DT, BG: W, & BG: C).

Discussion. Our results suggest that the addition of BG: W provides sizable benefits: 9.98 ± 7.60 (DT) $\to 44.11 \pm 2.81$ (DT & BG:W) and 12.42 ± 3.97 (DT & BG: C) $\to 44.31 \pm 2.56$ (DT, BG: W, & BG: C). We postulate that this is because it informs the policy's value function on model-predictive states early into training. This allows that the learner is able to more effectively search earlier into training. BG: C appears to offer minor stability and variance improvements throughout the training procedure; however, it does not have a measurable difference in final performance. This result suggests using planning methods in combination to reap their respective advantages.

However, we caution against focusing on the magnitude of improvement found within this experiment. As the margin of benefit depends on many factors including the world model accuracy, the opponent policy, and the game. To exemplify, similar to the background planning section, we repeat the same experiment with the poorly performing ▦ world model. The results of this ancillary experiment are in Figure 16 (Appendix E.3). The trend of BG: W providing benefits was reinforced: 6.29 ± 5.12 (DT) $\rightarrow 20.98 \pm 9.76$ (DT & BG: W) and 3.64 ± 0.26 (DT & BG: C) $\rightarrow 33.07 \pm 7.67$ (DT, BG: W, & BG: C). However, the addition of BG: C now measurably improved performance 20.98 ± 9.76 (DT & BG: W) $\rightarrow 33.07 \pm 7.67$ (DT, BG: W, & BG: C). The main outcome of these experiments is the observation that multi-faceted planning is unlikely to harm a response calculation, and has a potentially large benefit when applied effectively. These results support the claim that world models offer the potential to improve response calculation through decision-time planning.

4 Dyna-PSRO

In this section we introduce Dyna-PSRO, *Dyna*-Policy-Space Response Oracles, an approximate game-solving algorithm that builds on the PSRO [40] framework. Dyna-PSRO employs co-learning to combine the benefits of world models and empirical games.

Dyna-PSRO is defined by two significant alterations to the original PSRO algorithm. First, it trains a world model in parallel with all the typical PSRO routines (i.e., game reasoning and response calculation). We collect training data for the world model from both the episodes used to estimate the empirical game's payoffs, and the episodes that are generated during response learning and evaluation. This approach ensures that the world model is informed by a diversity of data from a salient set of strategy profiles. By reusing data from empirical game development, training the world model incurs no additional cost for data collection.

The second modification introduced by Dyna-PSRO pertains to the way response policies are learned. Dyna-PSRO adopts a Dyna-based reinforcement learner [69, 70, 72] that integrates simultaneous planning, learning, and acting. Consequently, the learner concurrently processes experiences generated from decision-time planning, background planning, and direct game interaction. These experiences, regardless of their origin, are then learned from using the IMPALA [15] update rule. For all accounts of planning, the learner uses the single world model that is trained within Dyna-PSRO. This allows game knowledge accrued from previous response calculations to be transferred and used to reduce the cost of the current and future response calculations. Pseudocode and additional details for both PSRO and Dyna-PSRO are provided in Appendix C.4.

Games. Dyna-PSRO is evaluated on three games. The first is the harvest commons game used in the experiments described above, denoted "Harvest: Categorical". The other two games come from the MeltingPot [41] evaluation suite and feature rich image-based observations. "Harvest: RGB" is their version of the same commons harvest game (details in Appendix D.2). "Running With Scissors" is a temporally extended version of rock-paper-scissors (details in Appendix D.3). World model training and implementation details for each game are in Appendix C.2, likewise, policies in Appendix C.1.

Experiment. Dyna-PSRO's performance is measured by the quality of the solution it produces when compared against the world-model-free baseline PSRO. The two methods are evaluated on SumRegret (sometimes called *Nash convergence*), which measures the regret across all players $\text{SumRegret}(\boldsymbol{\sigma}, \overline{\boldsymbol{\Pi}}) = \sum_{i \in n} \max_{\pi_i \in \overline{\Pi}_i} \hat{U}_i(\pi_i, \sigma_{-i}) - \hat{U}_i(\sigma_i, \sigma_{-i})$, where $\boldsymbol{\sigma}$ is the method's solution and $\overline{\boldsymbol{\Pi}} \subseteq \boldsymbol{\Pi}$ denotes the deviation set. We define deviation sets based on policies generated across methods (i.e., regret is with respect to the *combined game*): $\overline{\boldsymbol{\Pi}} \equiv \bigcup_{\text{method}} \hat{\boldsymbol{\Pi}}^{\text{method}}$, for all methods for a particular seed (detailed in Appendix C.5) [2]. We measure SumRegret for intermediate solutions, and report it as a function of the cumulative number of real experiences employed in the respective methods.

Results. Figure 5 presents the results for this experiment. For Harvest: Categorical, Dyna-PSRO found a no regret solution within the combined-game in 3.2e6 experiences. Whereas, PSRO achieves a solution of at best 5.45 ± 1.62 within 2e7 experiences. In Harvest: RGB, Dyna-PSRO reaches a solution with 0.89 ± 0.74 regret at 5.12e6 experiences. At the same time, PSRO had found a solution with 6.42 ± 4.73 regret, and at the end of its run had 2.50 ± 2.24 regret. In the final game, RWS, Dyna-PSRO has $2\text{e}{-3} \pm 5\text{e}{-4}$ regret at 1.06e7 experiences, and at a similar point (9.6e6 experiences), PSRO has $6.68\text{e}{-3} \pm 2.51\text{e}{-3}$. At the end of the run, PSRO achieves a regret $3.50\text{e}{-3} \pm 7.36\text{e}{-4}$.

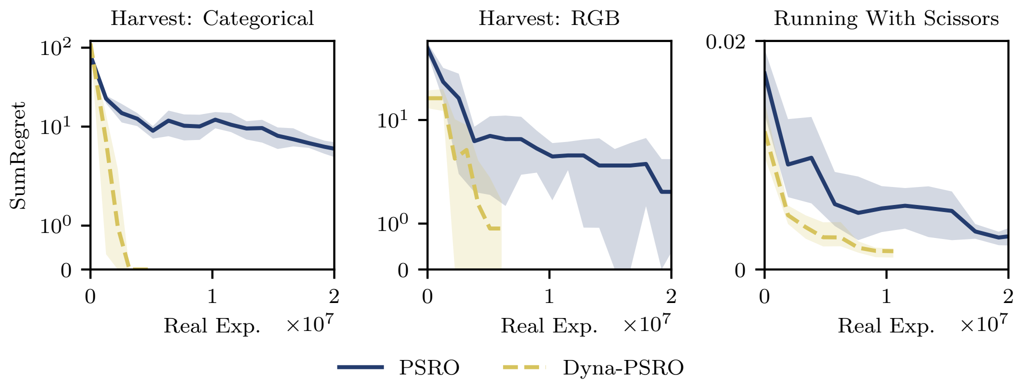

Figure 5: PSRO compared against Dyna-PSRO. (5 seeds, with 95 % bootstrapped CI).

Discussion. The results indicate that across all games, Dyna-PSRO consistently outperforms PSRO by achieving a superior solution. Furthermore, this improved performance is realized while consuming fewer real-game experiences. For instance, in the case of Harvest: Categorical, the application of the world model for decision-time planning enables the computation of an effective policy after only a few iterations. On the other hand, we observe a trend of accruing marginal gains in other games, suggesting that the benefits are likely attributed to the transfer of knowledge about the game dynamics. In Harvest: Categorical and Running With Scissors, Dyna-PSRO also had lower variance than PSRO.

5 Limitations

Although our experiments demonstrate benefits for co-learning world models and empirical games, there are several areas for potential improvement. The world models used in this study necessitated observational data from all players for training, and assumed a simultaneous-action game. Future research could consider relaxing these assumptions to accommodate different interaction protocols, a larger number of players, and incomplete data perspectives. Furthermore, our world models functioned directly on agent observations, which made them computationally costly to query. If the generation of experiences is the major limiting factor, as assumed in this study, this approach is acceptable. Nevertheless, reducing computational demands through methods like latent world models presents a promising avenue for future research. Lastly, the evaluation of solution concepts could also be improved. While combined-game regret employs all available estimates in approximating regret, its inherent inaccuracies may lead to misinterpretations of relative performance.

6 Conclusion

This study showed the mutual benefit of co-learning a world model and empirical game. First, we demonstrated that empirical games provide strategically diverse training data that could inform a more robust world model. We then showed that world models can reduce the computational cost, measured in experiences, of response calculations through planning. These two benefits were combined and realized in a new algorithm, Dyna-PSRO. In our experiments, Dyna-PSRO computed lower-regret solutions than PSRO on several partially observable general-sum games. Dyna-PSRO also required substantially fewer experiences than PSRO, a key algorithmic advantage for settings where collecting experiences is a cost-limiting factor.

References

[1] Stefano V Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

[2] David Balduzzi, Karl Tuyls, Julien Pérolat, and Thore Graepel. Re-evaluating evaluation. In *32nd Conference on Neural Information Processing Systems*, 2018.

[3] Philip Ball, Jack Parker-Holder, Aldo Pacchiano, Krzysztof Choromanski, and Stephen Roberts. Ready policy one: World building through active learning. In *37th International Conference of Machine Learning*, 2020.

[4] Nolan Bard, Michael Johanson, Neil Burch, and Michael Bowling. Online implicit agent modelling. In *12th International Conference on Autonomous Agents and Multiagent Systems*, 2013.

[5] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *28th Conference on Neural Information Processing Systems*, pages 1171–1179, 2015.

[6] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[7] George W Brown. Iterative solution of games by fictitious play. In *Activity analysis of production and allocation*, volume 13, pages 374–376, 1951.

[8] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *22nd Conference on Neural Information Processing Systems*, 2018.

[9] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019.

[10] Albin Cassirer, Gabriel Barth-Maron, Eugene Brevdo, Sabela Ramos, Toby Boyd, Thibault Sottiaux, and Manuel Kroiss. Reverb: A framework for experience replay, 2021.

[11] Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. In *5th International Conference on Learning Representations*, 2017.

[12] Valliappa Chockingam, Tegg Taekyong Sung, Feryal Behbanai, Rishab Gargeya, Amlesh Sivanantham, and Aleksandra Malysheva. Extending world models for multi-agent reinforcement learning in malmö. In *Joint AIIDE 2018 Workshops co-located with the 14th AAAI conference on artificial intelligence and interactive digital entertainment*, 2018.

[13] Brian Collins. Combining opponent modeling and model-based reinforcement learning in a two-player competitive game. Master's thesis, University of Edinburgh, 2007.

[14] B. Curtis Eaves. The linear complementarity problem. *Management Science*, 17(9):612–634, 1971.

[15] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *35th International Conference on Machine Learning*, 2018.

[16] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018.

[17] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20:121–136, 1975.

[18] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In *36th International Conference on Machine Learning*, volume 97, pages 2170–2179, 2019.

[19] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *31st Conference on Neural Information Processing Systems*, 2018.

[20] David Ha and Jürgen Schmidhuber. World models. In *arXiv preprint arXiv:1803.10122*, 2018.

[21] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *9th International Conference on Learning Representations*, 2021.

[22] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. Opponent modeling in deep reinforcement learning. In *33rd International Conference on Machine Learning*, 2016.

[23] Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020.

[24] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

[25] Todd Hester and Peter Stone. Texplore: Real-time sample-efficient reinforcement learning for robots. In *Machine Learning for Robotics (MLR)*, 2012.

[26] Geoffrey Hinton. Coursera neural networks for machine learning lecture 6. `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, 2018. Accessed: 2023-04-18.

[27] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[28] Matthew W. Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Nikola Momchev, Danila Sinopalnikov, Piotr Stańczyk, Sabela Ramos, Anton Raichuk, Damien Vincent, Léonard Hussenot, Robert Dadashi, Gabriel Dulac-Arnold, Manu Orsini, Alexis Jacq, Johan Ferret, Nino Vieillard, Seyed Kamyar Seyed Ghasemipour, Sertan Girgin, Olivier Pietquin, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Abe Friesen, Ruba Haroun, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

[29] G. Zacharias Holland, Erin Talvitie, and Michael Bowling. The effect of planning shape on dyna-style planning in high-dimensional state spaces. In *FAIM workshop "Prediction and Generative Modeling in Reinforcement Learning"*, 2018.

[30] HumanCompatibleAI. `https://github.com/HumanCompatibleAI/multi-agent`, 2019.

[31] Pararawendy Indarjo. Deep state-space models in multi-agent systems. Master's thesis, Leiden University, 2019.

[32] Marco A. Janssen, Robert Holahan, Allen Lee, and Elinor Ostrom. Lab experiments for the study of social-ecological systems. *Science*, 328(5978):613–617, 2010.

[33] Patrick R. Jordan, L. Julian Schvartzman, and Michael P. Wellman. Strategy exploration in empirical games. In *9th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1131–1138, 2010.

[34] Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In *1st Conference on Robot Learning*, pages 195–206, 2017.

[35] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations*, 2015.

[36] Christine Konicki, Mithun Chakraborty, and Michael P. Wellman. Exploiting extensive-form structure in empirical game-theoretic analysis. In *Web and Internet Economics: 18th International Conference*, 2022.

[37] Orr Krupnik, Igor Mordatch, and Aviv Tamar. Multi-agent reinforcement learning with multi-step generative models. In *4th Conference on Robot Learning*, pages 776–790, 2020.

[38] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *6th International Conference on Learning Representations*, 2018.

[39] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. OpenSpiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019.

[40] Marc Lanctot, Vinicius Zambaldi, Audrūnas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *31st Conference on Neural Information Processing Systems*, page 4193–4206, 2017.

[41] Joel Z. Leibo, Edgar Duéñez-Guzmán, Alexander Sasha Vezhnevets, John P. Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charles Beattie, Igor Mordatch, and Thore Graepel. Scalable evaluation of multi-agent reinforcement learning with melting pot. PMLR, 2021.

[42] Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *16th International Conference on Autonomous Agents and Multiagent Systems*, 2017.

[43] Zun Li, Marc Lanctot, Kevin McKee, Luke Marris, Ian Gemp, Daniel Hennes, Paul Muller, Kate Larson, Yoram Bachrach, and Michael P. Wellman. Search-improved game-theoretic multiagent reinforcement learning in general and negotiation games (extended abstract). In *32nd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS, 2023.

[44] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *11th International Conference on Machine Learning*, pages 157–163, 1994.

[45] Stephen McAleer, John Lanier, Kevin Wang, Pierre Baldi, and Roy Fox. XDO: A double oracle algorithm for extensive-form games. In *35th Conference on Neural Information Processing Systems*, 2021.

[46] Richard D. McKelvey, Andrew M. McLennan, and Theodore L. Turocy. Gambit: Software tools for game theory. `http://www.gambit-project.org/`, 2016.

[47] H. Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *20th International Conference on Machine Learning*, pages 536–543, 2003.

[48] Richard Mealing and Jonathan L Shapiro. Opponent modeling by expectation–maximization and sequence prediction in simplified poker. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 9, pages 11–24, 2015.

[49] Vicent Michalski, Roland Memisevic, and Kishore Konda. Modeling deep temporal dependencies with recurrent grammar cells. In *27th Conference on Neural Information Processing Systems*, 2014.

[50] Johan S. Obando-Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *38th International Conference on Machine Learning*, 2021.

[51] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *28th Conference on Neural Information Processing Systems*, 2015.

[52] Junhyuk Oh, Satinderg Singh, and Honglak Lee. Value prediction network. In *30th Conference on Neural Information Processing Systems*, pages 6118–6128, 2017.

[53] S. Phelps, M. Marcinkiewicz, and S. Parsons. A novel method for automatic strategy acquisition in N-player non-zero-sum games. In *Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, page 705–712, 2006.

[54] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[55] Julien Pérolat, Joel Z. Leibo, Vinicius Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. In *31st Conference on Neural Information Processing Systems*, 2017.

[56] Stéphane Ross and J. Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *CoRR*, abs/1406.5979, 2014.

[57] Stéphane Ross, Goeffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *14th International Conference on Artificial Intelligence and Statistics*, 2011.

[58] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588:604–609, 2020.

[59] L. Julian Schvartzman and Michael P. Wellman. Exploring large strategy spaces in empirical game modeling. In *AAMAS-09 Workshop on Agent-Mediated Electronic Commerce*, 2009.

[60] L. Julian Schvartzman and Michael P. Wellman. Stronger CDA strategies through empirical game-theoretic analysis and reinforcement learning. In *8th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 249–256, 2009.

[61] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *37th International Conference of Machine Learning*, pages 8583–8592, 2020.

[62] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[63] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[64] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[65] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In *34th International Conference on Machine Learning*, volume 70, pages 3191–3199, 2017.

[66] Max Olan Smith, Thomas Anthony, Yongzhao Wang, and Michael P. Wellman. Learning to play against any mixture of opponents. *CoRR*, 2020.

[67] Max Olan Smith, Thomas Anthony, and Michael P. Wellman. Iterative empirical game solving via single policy best response. In *9th International Conference on Learning Representations*, 2021.

[68] Chen Sun, Per Karlsson, Jiajun Wu, Joshua B Tenenbaum, and Kevin Murphy. Stochastic prediction of multi-agent interactions from partial observations. In *7th International Conference on Learning Representations*, 2019.

[69] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *7th International Workshop on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990.

[70] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. In *SIGART Bulletin*, volume 2, pages 160–163. ACM, 1991.

[71] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[72] Richard S Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael P. Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. In *28th Conference on Uncertainty in Artificial Intelligence*, 2012.

[73] Erin Talvitie. Model regularization for stable sample rollouts. In *30th Conference on Uncertainty in Artificial Intelligence*, 2014.

[74] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[75] Zheng Tian, Ying Wen, Zhichen Gong, Faiz Punakkath, Shihao Zou, and Jun Wang. A regularized opponent model with maximum entropy objective. In *International Joint Conference on Artificial Intelligence*, 2019.

[76] Karl Tuyls, Julien Pérolat, Marc Lanctot, Edward Hughes, Richard Everett, Joel Z. Leibo, Csaba Szepesvári, and Thore Graepel. Bounds and dynamics for empirical game theoretic analysis. *Autonomous Agents and Multi-Agent Systems*, 34(7), 2020.

[77] Yevgeniy Vorobeychik. Probabilistic analysis of simulation-based games. *ACM Transactions on Modeling and Computer Simulation*, 20(3), 2010.

[78] Niklas Wahlström, Thomas B. Schön, and Marc Peter Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*, 2015.

[79] William Walsh, Rajarshi Das, Gerald Tesauro, and Jeffrey Kephart. Analyzing complex strategic interactions in multi-agent systems. In *AAAI-02 Workshop on Game Theoretic and Decision Theoretic Agents*, 2002.

[80] Rose E Wang, Chase Kew, Dennis Lee, Edward Lee, Brian Andrew Ichter, Tingnan Zhang, Jie Tan, and Aleksandra Faust. Model-based reinforcement learning for decentralized multiagent rendezvous. In *Conference on Robot Learning*, 2020.

[81] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *28th Conference on Neural Information Processing Systems*, pages 2746–2754, 2015.

[82] Michael P. Wellman. Methods for empirical game-theoretic analysis. In *21st National Conference on Artificial Intelligence*, page 1552–1555, 2006.

[83] Daniël Willemsen, Mario Coppola, and Guido CHE de Croon. MAMBPO: Sample-efficient multi-robot reinforcement learning using learned world models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021.

[84] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2), 1989.

[85] Fan Yang, Gabriel Barth-Maron, Piotr Stańczyk, Matthew Hoffman, Siqi Liu, Manuel Kroiss, Aedan Pope, and Alban Rrustemi. Launchpad: A programming model for distributed machine learning research. *arXiv preprint arXiv:2106.04516*, 2021.

[86] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: Model-based offline policy optimization. In *33rd Conference on Neural Information Processing Systems*, 2020.

[87] Kaiqing Zhang, Sham Kakade, Tamer Basar, and Lin Yang. Model-based multi-agent rl in zero-sum markov games with near-optimal sample complexity. In *33rd Conference on Neural Information Processing Systems*, 2020.

[88] Qizhen Zhang, Chris Lu, Animesh Garg, and Jakob Foerster. Centralized model and exploration policy for multi-agent RL. In *21st International Conference on Autonomous Agents and Multiagent Systems*, 2022.

A Broader Impact

There are no direct broader impacts from this work. However, this work promotes the adoption of both empirical games and world models. Potential negative impacts may arise due to errors introduced when compressing the true game into the confines of *any* model, which could lead to negative consequences. World model errors within Dyna-PSRO are transferred across response calculations potentially reinforcing biases about the world. If these biases are not rectified, they could negatively influence policies learned from these models. The strategic diversity component of this work aims to mitigate these potential biases, though it represents only the initial step in addressing this concern. When considering empirical games, inaccuracies within them can lead to the suggestion of flawed solutions. The adoption of these inaccurate solutions could have negative repercussions for practitioners or other stakeholders involved in the game. Vigilance and thorough evaluation are required to prevent these potential issues.

B Compute

GPUs are used for training world models, and policies within Dyna-PSRO. Two types of GPUs were used throughout this work interchangeably: TITAN X and GTX 1080 Ti. All other computation was completed using CPUs. Each response calculation had additional CPUs corresponding to the number of experience generation arenas described in Appendix C. Experiments were run on internal clusters.

C Methods Details

In this work, the both the policies and world models are implemented in JAX [6] with Haiku [23]. The software is architected using Launchpad [85] with design patterns inspired by ACME [28]. All replay buffers are implemented using Reverb [10]. Gambit [46] is used as a game solver via linear complementarity [14].

C.1 Policy Implementation & Training

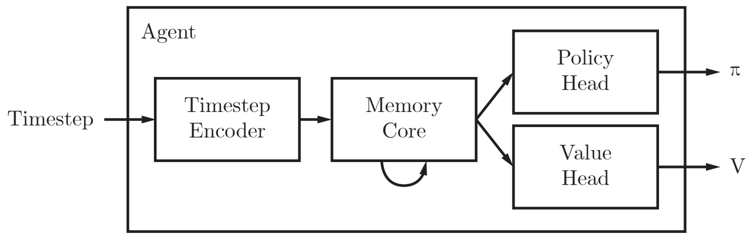

Figure 6: Agent Architecture.

All policies follow the general architecture depicted in Figure 6. This consists of four modules:

- *Timestep Encoder*: Processes all of the current observation's information into a single embedding vector. The timestep includes the new observation and the policy's previous action.
- *Memory Core*: The component of the agent that maintains and update's the agent's memory.
- *Policy Head*: Computes the agent's policy.
- *Value Head*: Computes the agent's state value function.

All of the components are simultaneously trained and their joint parameters are $\theta^{\pi} \in \Theta^{\pi}$. The policies are trained using the IMPALA algorithm [15]. For the IMPALA loss, the coefficients for each component loss are:

$$\mathcal{L}_{\text{IMPALA}} = \lambda_{\pi} \cdot \mathcal{L}_{\pi} + \lambda_{\text{V}} \cdot \mathcal{L}_{\text{V}} + \lambda_{\text{entropy}} \cdot \mathcal{L}_{\text{entropy}},$$

with a discount factor of 0.99. The training details for each specific response calculation are itemized below.

Baseline Parameters The learning rate begins is linearly decayed over $10\,000$ updates. Each update is computed from a mini-batch of 128 examples that are generated from 8 arenas[2] Policy parameters are synchronized at the beginning of each episode. Each example in the mini-batch is a sequence of 20 transitions. Moreover, sequences are stored in a replay buffer with a period of 19, to ensure that the action played at the end of a sequence is trained. Sequences are stored in a replay buffer with a max capacity of $1\,000\,000$, and are evicted once sampled. Additional hyperparameters are specified in Table 1.

Table 1: Baseline policy hyperparameters per game.

Hyperparameter	Harvest: Categorical	Harvest: RGB	Running with Scissors
Optimizer	Adam [35]	RMSProp [26]	RMSProp [26]
λ_π	1.0	1.0	1.0
λ_V	0.2	0.5	0.2
λ_entropy	0.04	0.01	0.003
Learning Rate Start	$6\text{e}{-6}$	$6\text{e}{-4}$	$1\text{e}{-4}$
Learning Rate Stop	$6\text{e}{-9}$	$6\text{e}{-9}$	$1\text{e}{-4}$
Max Grad Norm	10.0	1.0	0.1
Batch Size	128	128	128

Harvest: Categorical module implementations:

- *Timestep Encoder*: The encoder processes two timestep components: the current observation and the previous action the policy took. First the observation is passed through a two-layer fully connected neural network with hidden sizes of $[256, 256]$. The representation of the observation is then concatenated with the previous action (represented as a one-hot vector), and passed together through a second neural network with sizes $[256, 256]$. All of the layers have ReLU [17] activations including the final layers of both networks. The final representation is the output of the timestep encoder.
- *Memory Core*: A single-layer LSTM [27] with $256\,\text{units}$.
- *Policy Head*: A single linear layer of size 8.
- *Value Head*: A single linear layer of size 1.

Harvest: RGB and Running with Scissors module implementations:

- *Timestep Encoder*: The encoder processes two timestep components: the current observation and the previous action the policy took. The observation is first process by a two-layer convolutional neural network with ReLU activations [17]. The first layer has 16 channels, a kernel with shape $[8, 8]$, and a stride of $[8, 8]$. The second layer has 32 channels, a kernel shape of $[4, 4]$, and a stride of $[1, 1]$. The output of this layer is then flattened and concatenated with a one-hot encoding of the policy's previous action. The resulting embedding is then passed through a two-layer fully connected neural network with hidden sizes of $[128, 128]$, and ReLU activations.
- *Memory Core*: A single-layer LSTM [27] with $128\,\text{units}$.
- *Policy Head*: A single linear layer of size 8.
- *Value Head*: A single linear layer of size 1.

Planning Parameters The planners have the same hyperparameters as the baseline method, but with the addition of planning-specific settings. For all planners, an additional 4 arenas are used to generate planned experiences (for background planning). The additional settings for each version of planning are as follows:

- *Warm-Start Background Planning*: An additional $10\,000$ updates are performed on exclusively planned experiences before play in the real game occurs.

[2]The term *arena* is used to refer to an experience generation process. This is more commonly referred to as an "actor"; however, this terminology may be confounding with language in RL, Dyna, or multiagent learning.

- *Concurrent Background Planning*: Each mini-batch sampled after warm-starting contains $25\,\%$ planned experiences, and $75\,\%$ real experiences.
- *Decision-Time Planning*: In the training arenas (those that have the real game, and are not used for evaluation), the agent selects actions with a beam-search of width 3 and depth 1.

Background planning also requires defining a *search control* procedure [69, 70, 71]. Search control defines how the agent prioritizes selecting starting states and actions for background planning. This work considers the simplest search-control method: maintain a buffer of the initial states and uniformly sample.

C.2 World Model Implementation & Training

C.2.1 Action-Conditioned Scheduled Sampling

As noted by Talvitie [73], rolling out trajectories with an imperfect model tends to result in compounding errors in prediction. Their work suggests training a Markovian world model with previous predictions (referred to as "hallucinated replay"), to train the model to correct errors. For stateful world models, as studied in this work, it has been demonstrated that curricula of n-step future predictions can train a fruitful world model [49, 51, 11]. All of the preceding work was studying single-agent systems; therefore, they could assume a much more stable data distribution for training. As a result, these fixed curricula style approaches may prove fatal as the data distribution may change dramatically throughout training based on the coplayers' strategies.

Algorithm 1: Action-Conditioned Scheduled Sampling

```
$m \leftarrow$ Initial recurrent state
**for** $t \in T$ **do**
    $\boldsymbol{o} \leftarrow \boldsymbol{o}^t$ if Unif$[0,1] < \epsilon(t)$ else $\hat{\boldsymbol{o}}^t$
    $\hat{\boldsymbol{o}}^{t+1}, \hat{\boldsymbol{r}}^{t+1}, m \leftarrow w(\boldsymbol{o}, \boldsymbol{a}^t, m)$
**Output:** Predicted trajectory $(\hat{\boldsymbol{o}}^{0:T}, \hat{\boldsymbol{r}}^{0:T})$
```

Instead, this work adapts the scheduled sampling [5] algorithm as a stochastic curricula, which will allow both short- and long-term predictions throughout the course of training. Scheduled sampling is an algorithm for training auto-regressive sequence prediction models where at each predictive step during training the model input is sampled from either the previous prediction or the ground truth. Adapting this algorithm for world model rollouts requires biasing each predictive step with the true actions while sampling between the predicted successor observation and the true successor observation. Therefore, the predictions will always be biased on true actions, but must learn to handle model-predicted observation. The sampling follows a schedule $\epsilon : \mathbb{Z} \to [0,1]$ that determines the probability of sampling the true observation over the previous prediction. When ϵ is 1.0, the algorithm behaves akin to teacher forcing [84] (with the same action-conditional modification); whereas, as it approaches 0.0 it becomes fully auto-regressive.

C.2.2 Implementation

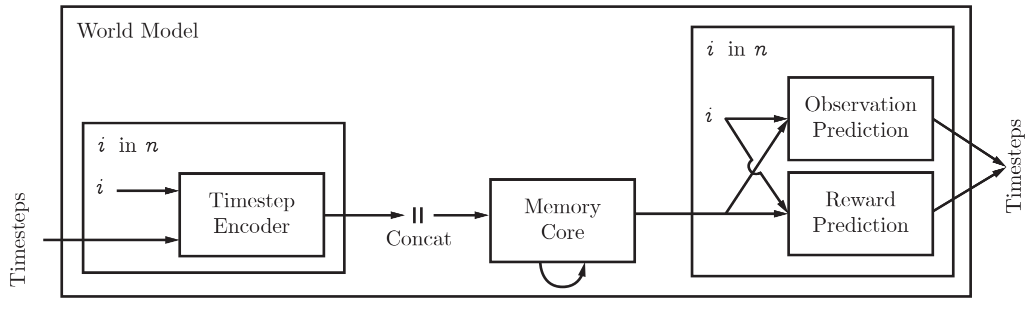

Figure 7: World Model Architecture.

The high-level architecture of the world model is illustrated in Figure 7. The world model is composed of several modules that are quite similar to the policy:

- *Timestep Encoder*: Processes all of the current observation's information into a single embedding vector. The timestep includes all new observational data that the agent gains at the current point in time. Different from the agent's timestep encoder, this encoder also receives the ID that corresponds with the timestep.
- *Memory Core*: The component of the agent that maintains and update's the agent's memory. Different from the agent's timestep encoder, this memory core receives the representation of each player's timestep concatenated.
- *Observation Prediction (Head)*: Predicts the successor observation for each player. As all games considered in this work are gridworld games, the predicted observation is a classification task for each future grid cell (that are within the respective player's observation window).
- *Reward Prediction (Head)*: Predicts the reward received for each player. Rewards are treated as categorical values.

Note, that the timestep encoder, observation prediction head, and reward prediction head each use the same parameters across each player. Similar to the agent, all components are simultaneously trained and their joint parameters are referred to as $\theta^w \in \Theta^w$. Both observation and reward losses are optimized with a cross entropy objective, and averaged across players. The total world model loss is as follows:

$$\mathcal{L}_w = \lambda_{\text{observation}} \cdot \mathcal{L}_{\text{observation}} + \lambda_{\text{reward}} \cdot \mathcal{L}_{\text{reward}}.$$

The implementation of each component is as follows:

- *Timestep Encoder*: The same as the agent's timestep encoder, but the player's ID is also provided alongside the action into the second neural network.
- *Memory Core*: Identical to the agent.
- *Observation Prediction (Head)*: The observation prediction is based on the memory core's output and a one-hot ID of the predicted player's ID. These inputs are concatenated and fed into an transposed version of the timestep encoder.
- *Reward Prediction (Head)*: A linear layer of size one. For Harvest: Categorical this output is handled as a discrete prediction; whereas, it is continuous for the other games.

A world model is trained for $1\,250\,000$ updates. Each example in the mini-batch is a sequence of 20 transitions, where the first 5 timesteps are used to burn-in the memory. Burn-in does not occur for examples where the first 5 transitions are at the beginning of the episode. Moreover, sequences are added into the replay buffer at a period of 14 so that all timesteps show up as prediction targets.

The world model is trained using action-conditioned scheduled sampling (Appendix 1, Algorithm 1). The schedule ϵ follows the following schedule:

$$\epsilon(t) = \begin{cases} 1.0 & t < 250000 \\ \frac{4}{3} - \frac{t}{750000} & 250000 \leq t \leq 1000000 \\ 0.0 & t > 1000000. \end{cases}$$

This schedule starts out training as a variation of teacher forcing [84], and slowly transitions to fully auto-regressive. Additional hyperparameters are specified in Table 2.

Table 2: World model hyperparameters per game.

Hyperparameter	Harvest: Categorical	Harvest: RGB	Running with Scissors
$\lambda_{\text{observation}}$	1.0	1.0	1.0
λ_{reward}	10.0	0.01	0.01
Optimizer	Adam [35]	Adam [35]	Adam [35]
Learning Rate	3e−4	3e−4	3e−4
Max Grad Norm	10.0	10.0	10.0
Batch Size	32	24	24

C.3 Strategic Diversity

Learning a general world model assumes that the transitions are drawn from the space of all possible transitions. This is typically not tractable, but instead draws are taken from a dataset generated from play of a *behavioral strategy* $\boldsymbol{\sigma}$. And the performance of the world model is measured under a *target strategy* $\boldsymbol{\sigma}^*$, instead of all possible strategies $\boldsymbol{\Sigma}$. Differences between $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}^*$ present challenges in learning an effective world model.

We call the probability of drawing a state-action pair $\boldsymbol{s},\ \boldsymbol{a}$ under some strategy its *reach probability* $\eta^{\hat{\boldsymbol{\sigma}}}$ under joint strategy $\hat{\boldsymbol{\sigma}}$. From this, we define *strategic diversity* as the distribution induced from reach probabilities. These terms allow us to observe two challenges for learning world models.

First, the diversity of the behavioral strategy *cover* the target strategy's diversity:

$$\eta^{\boldsymbol{\sigma}^*}(\boldsymbol{s}, \boldsymbol{a}) \rightarrow \eta^{\boldsymbol{\sigma}}(\boldsymbol{s}, \boldsymbol{a}). \tag{1}$$

Otherwise, transitions will be absent from the training data. As an aside, it is possible to construct a weaker claim for coverage. This is done through making additional assumptions about the generalization capacity of a world model across transitions. For example, if transitions are drawn from two discrete latent variables, unseen combinations of these variables may be generalized if the individual values are known. However, generalization cannot be generally guaranteed, so we consider coverage.

The second challenge is that the *closer* the diversities are, the more accurate the learning objective will be. In other words, we want

$$\eta^{\boldsymbol{\sigma}^*}(\boldsymbol{s}, \boldsymbol{a}) \approx \eta^{\boldsymbol{\sigma}}(\boldsymbol{s}, \boldsymbol{a}). \tag{2}$$

If closeness is not ensured, crucial dynamics knowledge may not be learned as the learning signal is dominated from unimportant transitions. An example of the issue of closeness can be seen in the "noisy TV problem," [9]. This exploration problem poses that novelty-seeking agents may be stuck forever watching the ever new TV static, and not experiencing practical novelty. In the same vein, if a world model is trained almost entirely on "noisy TV"-like experiences, and as a rarely on the few salient experiences, it may never learn. Therefore, we should strive to correct the distribution of experiences to be informed by a target strategy.

By design, empirical-game building algorithms offer a means to construct the target world model objective. These algorithms require the specification of a solution concept that serves the dual roll as the target strategy for a world model. Then through an iterative process, the empirical-game constructs strategies that progressively approach the target. In turn, generating transitions that match the target world model objective.

Claim 1. *Dyna-PSRO produces a correct world-model objective $\eta^{\boldsymbol{\sigma}^*}$ with a best-response oracle and a correct empirical game for a game with a unique Nash Equilibrium $\boldsymbol{\sigma}^*$.*

Proof. Following McMahan et al. [47], the Double Oracle algorithm will converge to a NE in the limit of enumerating the full strategy space. Let $\boldsymbol{\sigma}^0, \boldsymbol{\sigma}^1, \ldots, \boldsymbol{\sigma}^e$ be the solutions discovered for each epoch, ending at epoch e. Then a dataset composed of experiences generated by the current empirical game solution evolves as follows:

$$\eta^{\boldsymbol{\sigma}^0} \rightarrow \eta^{\boldsymbol{\sigma}^1} \rightarrow \ldots \rightarrow \eta^{\boldsymbol{\sigma}^e} = \eta^{\boldsymbol{\sigma}^*}. \tag{3}$$

$\square$

The previous claim contains two strong assumptions: an exact best-response oracle and error-less empirical game. These assumptions must be made, because PSRO is parameterized by its choice of response oracle and empirical game model; therefore, PSRO's convergence must be proven for each choice. Theoretically PSRO has been shown to converge to an ϵ-NE, where ϵ depends on the empirical game's modelling error, to a corresponding NE in the true game [76, 77]. Therefore, in practice Dyna-PSRO produces $\eta^{\boldsymbol{\sigma}^e} \approx \eta^{\boldsymbol{\sigma}^*}$, which supports the weaker claim that Dyna-PSRO generally improves the quality of a world model.

It is also worth noting the connections between this analysis and MARL regimes that seek to find any solution the game. In these regimes, the priority is finding *any* performant strategy. This matches the approach taken by the majority of studies in MARL falling under paradigms such as Independent RL or Self-Play. Therefore, their target distribution is the best-response to the previous strategy $\eta^{\text{BR}(\boldsymbol{\sigma}^{i-1})}$ and changes in tandem with the strategies. When no best-response can be found, then the current strategy matches the solution and the dataset correspondingly reflects this.

C.4 Dyna-PSRO

The Dyna-PSRO builds upon PSRO (Algorithm 3) by including the co-learning of a world model. The high-level pseudocode of Dyna-PSRO is provided in Algorithm 5 an a high-level application architecture diagram is depicted in Figure 8. There are three main co-routines of Dyna-PSRO: response computation, world-model learning, and empirical-game simulation. The details of each routine are first provided; then, how the routines interact with each other is explained.

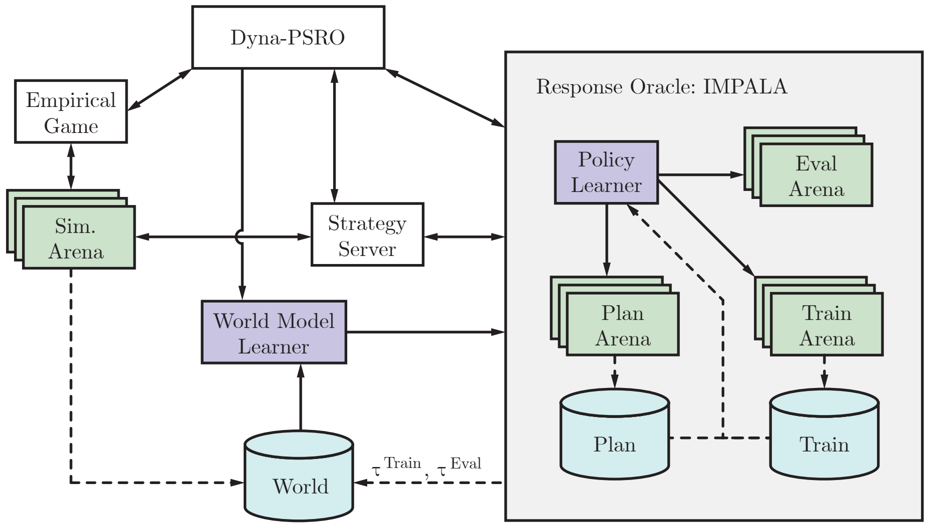

Figure 8: Overview of the major Dyna-PSRO processes.

C.4.1 Empirical Game

The empirical game routine is responsible for maintaining the empirical game, including simulating new payoffs and game reasoning. New profiles are sent to *simulation (sim.) arenas* for payoff estimation in parallel. Once all profiles are estimated, the game is solved, and the solution is based to the main Dyna-PSRO process. In the experiments in this work, the chosen solution is Nash Equilibrium, and it is solved through the linear complementarity [14] algorithm that is implemented by Gambit [46].

C.4.2 World Model

The world model routine is responsible for training the world model and serving its parameters. This routine's pseudocode is provided in Algorithm 2, and follows mostly the same method details as the strategic diversity experiment. The difference is that instead of there being a precomputed fixed dataset, the world model is now trained over a dynamic dataset. The dataset is represented by a replay buffer that is populated from: (1) trajectories from the simulation arena used for expanding the empirical game, and (2) trajectories from the training and evaluation arenas from the response calculation. Notably, all of this data must be generated in the standard PSRO procedure, so it collected with no additional cost. The world-model learner samples and evicts data randomly from this buffer.

Algorithm 2: World Model Learner

```
Input: World model $w$ and data buffer $\mathcal{B}^w$
Input: $n$ no. of updates (default: $\infty$).
for $i \in [[n]]$ do
    Train $w$ over $\tau \sim \mathcal{B}^w$
Output: $w$
```

C.4.3 Response Oracle

The response oracle uses the IMPALA [15] algorithm to compute an approximate best-response to the opponent's strategy according the the current empirical game. IMPALA uses several processes that generate experiences for the agent to train on. These process are referred to in this work as arenas. The *train arenas* generate real experiences, and the *plan arenas* generate planned experiences. If the learner is using decision-time planning they will only use it in the train arenas. A third set of arenas called *eval arenas* periodically evaluate the performance of the greedy policy and record additional metrics. The arenas attempt to synchronize all parameters at the start of each episode.

The policy learner runs for a fixed number of updates, querying the datastores for experiences to learn from. The specifics of how each policy learns is described in Appendix C.1.

C.4.4 Runtime Procedure

A sketch of the respective processes runtime is shown in Figure 9 As in PSRO, the main empirical-game building loop iterates between response computation and empirical-game simulation.

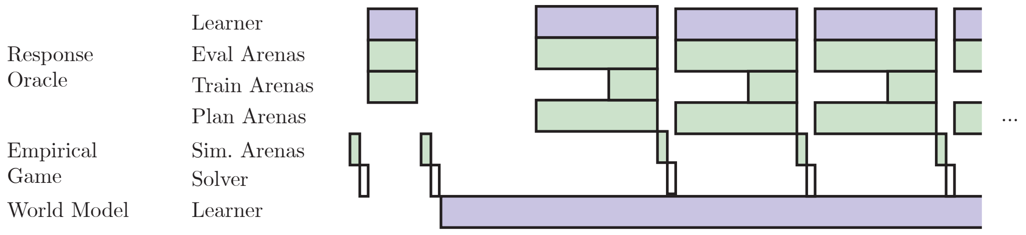

Figure 9: Example Dyna-PSRO runtime. Planning is set to occur after the first epoch. Each players' response oracle runs in parallel.

The runtime is defined by a parameter specifying on which PSRO epoch to begin planning. Before that epoch, the response oracles do not use planning at all, because the world models are untrained. All of these policies therefore are trained exclusively on real experiences just like standard PSRO. However, these experiences are also being used to populate the world model's replay buffer. Once the first planning epoch has arrived, computing responses is temporarily paused. The world model is then given a set number of updates to warm-start its parameters, before being used in response calculation. Once the world model's warm-start phase is over, all process proceed concurrently.

Throughout this work planning begins on the second epoch. The world models is given 1 million updates of warm starting.

Algorithm 3: Policy-Space Response Oracles [40]

```
Input: Initial strategy sets for all players $\mathbf{\Pi}^0$
Simulate utilities $\hat{U}^{\mathbf{\Pi}^0}$ for each joint $\boldsymbol{\pi}^0 \in \mathbf{\Pi}^0$
Initialize solution $\sigma_i^{*,0} = \text{Uniform}(\Pi_i^0)$
while epoch e in $\{1, 2, \ldots\}$ do
    for player $i \in [[n]]$ do
        // Algorithm 4.
        $\pi_i^e, \_ = \text{response\_oracle}(\sigma_{-i}^{*,e-1})$
        $\Pi_i^e = \Pi_i^{e-1} \cup \{\pi_i^e\}$
    Simulate missing entries in $\hat{U}^{\Pi^e}$ from $\mathbf{\Pi}^e$
    Compute a solution $\sigma^{*,e}$ from $\hat{\Gamma}^e$
Output: Current solution $\sigma_i^{*,e}$ for player $i$
```

Algorithm 4: Response Oracle

```
Input: Coplayer strategy profile $\sigma_{-i}$
Input: Num updates $k$
$\pi_i \leftarrow \theta^\pi$
$\mathcal{B} \leftarrow \{\}$            // Replay Buffer.
for many async episodes do
    $\pi_{-i} \sim \sigma_{-i}$
    $\mathcal{B} = \mathcal{B} \cup \{\tau \sim (\pi_i, \pi_{-i})\}$
for $i \in [[k]]$ do
    Train $\pi_i$ over $\tau \sim \mathcal{B}$
Output: $\pi_i, \mathcal{B}$
```

Algorithm 5: Dyna-PSRO

```
Input: Initial strategy sets for all players $\mathbf{\Pi}^0$
Input: No. of world model head-start updates $n_w$
Input: Epoch to begin planning $e^{\text{plan}}$
Simulate utilities $\hat{U}^{\mathbf{\Pi}^0}$ for each joint $\boldsymbol{\pi}^0 \in \mathbf{\Pi}^0$
Initialize solution $\sigma_i^{*,0} = \text{Uniform}(\Pi_i^0)$
$w \leftarrow \theta^w$
$\mathcal{B}^w \leftarrow \{\}$                                   // World Model's Replay Buffer.

while epoch e in $\{1, 2, \dots\}$ do
    for player $i \in [[n]]$ do
        if $e > e^{\text{plan}}$ then
            $\pi_i^e, \tau = \text{async}(\text{planner\_oracle}(\sigma_{-i}^{*,e-1}, w))$          // Algorithm 6.
        else
            $\pi_i^e, \tau = \text{async}(\text{response\_oracle}(\sigma_{-i}^{*,e-1}))$            // Algorithm 4.
        $\mathcal{B}^w = \mathcal{B}^w \cup \{\tau\}$
        $\Pi_i^e = \Pi_i^{e-1} \cup \{\pi_i^e\}$
    Wait on all futures $\boldsymbol{\pi}^e, \tau$

    Simulate missing entries in $\hat{U}^{\mathbf{\Pi}^e}$ from $\mathbf{\Pi}^e$
    Add $\tau$ from simulation to $\mathcal{B}^w$
    Compute a solution $\boldsymbol{\sigma}^{*,e}$ from $\hat{\Gamma}^e$

    if e == 1 then
        $w = \text{world\_model\_learner}(w, n_w)$                       // Algorithm 2.
        $w = \text{async}(\text{world\_model\_learner}(w))$        // Parameters periodically sync.
Output: Current solution $\sigma_i^{*,e}$ for player $i$
```

Algorithm 6: Planner Oracle

```
Input: Coplayer strategy profile $\sigma_{-i}$
Input: World model $w$, real game dynamics $p$
Input: Warm-start background planning updates $n^{\text{BG:WS}}$
Input: Training updates $n$
Input: Concurrent background planning fraction $f^{\text{BG:C}}$
$\pi_i \leftarrow \theta^\pi$
$\mathcal{B}^{\text{plan}} \leftarrow \{\}$                    // Replay Buffer with planned experience.
$\mathcal{B}^{\text{train}} \leftarrow \{\}$                      // Replay Buffer with real experience.

// Asynchronously generate data on arenas.
for many async episodes do
    $\pi_{-i} \sim \sigma_{-i}$
    $\mathcal{B}^{\text{plan}} = \mathcal{B}^{\text{plan}} \cup \{\tau \sim (\pi_i, \pi_{-i}, w)\}$
for many async episodes do
    $\pi_{-i} \sim \sigma_{-i}$
    $\mathcal{B}^{\text{train}} = \mathcal{B}^{\text{train}} \cup \{\tau \sim (\pi_i, \pi_{-i}, p)\}$

// Train the response policy.
for $i \in [[n^{\text{BG:WS}}]]$ do
    Train $\pi_i$ over $\tau \sim \mathcal{B}^{\text{plan}}$
for $i \in [[n]]$ do
    Train $\pi_i$ over $\tau \sim \left\{(1.0 - f^{\text{BG:C}}) \cdot \mathcal{B}^{\text{train}}\right\} \cup \left\{f^{\text{BG:C}} \cdot \mathcal{B}^{\text{plan}}\right\}$
Output: $\pi_i, \mathcal{B}$
```

C.5 Combined-Game Regret

Combined-game regret is an approximate measure of regret that all available estimates to approximate the regret within the true game. Intuitively, combined-game regret is the regret of a strategy with respect to all discovered policies. When comparing empirical-game building algorithms this is formalized as follows:

$$\text{SumRegret}(\boldsymbol{\sigma}, \overline{\boldsymbol{\Pi}}) = \sum_{i \in n} \max_{\pi_i \in \overline{\Pi}_i} \hat{U}_i(\pi_i, \sigma_{-i}) - \hat{U}_i(\sigma_i, \sigma_{-i}), \qquad \overline{\boldsymbol{\Pi}}_i \equiv \bigcup_{\text{method}} \hat{\boldsymbol{\Pi}}_i^{\text{method}}, \tag{4}$$

where $\hat{\Pi}$ is the restricted strategy set from one of the algorithms.

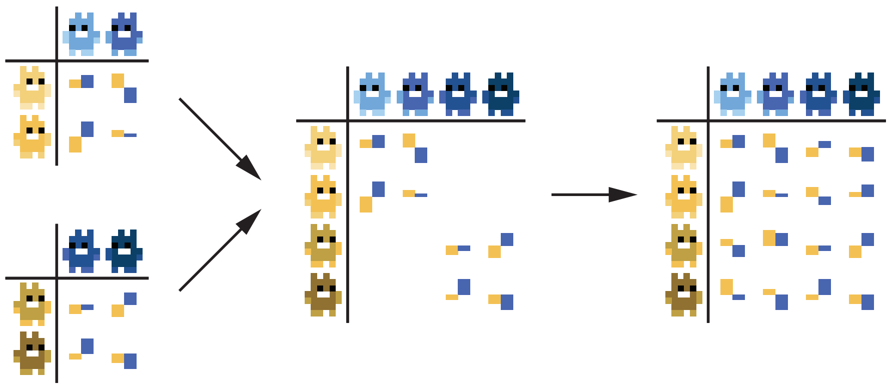

Figure 10: Combined-game construction. Left: Constituent empirical games. Middle: Combination of the strategy sets and payoff functions. Right: Completion of the empirical game by estimating new strategy profile payoffs.

The process of constructing a combined-game is illustrated in Figure 10. Where, the strategy sets (depicted by the toons) across methods are first combined. The new *combined game* that results from this can be initialized with the payoff estimates from the constituent empirical games. Unestimated payoffs must then be simulated for the new strategy profiles. Then the complete combined game can be used to compute the combined-game regret from the solutions computed throughout the empirical-game building algorithms.

D Games

D.1 Harvest: Categorical

In Harvest, players move around an orchard picking apples. The challenging commons element is that apple regrowth rate is proportional to nearby apples, so that socially optimum behavior would entail managed harvesting. Self-interested agents capture only part of the benefit of optimal growth, thus non-cooperative equilibria tend to exhibit collective over-harvesting. The game has established roots in human-behavioral studies [32] and in agent-based modeling of emergent behavior [55, 42, 41].

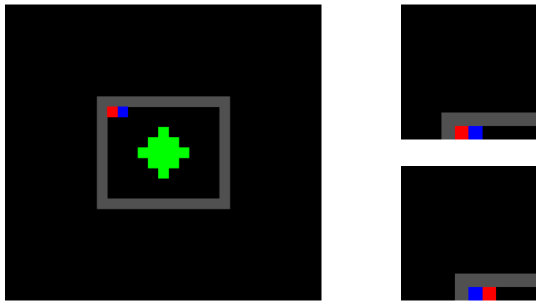
Figure 11: Harvest: Categorical. Left: game state. Right: player observations.

For our initial experiments, we use a symmetric two-player version of the game, where in-game entities are represented categorically [30]. This categorical representation facilitates faster experimentation and simplifies the interpretation of results. Figure 11 depicts the game state and player observations. Each player has a 10×10 viewbox within their field of vision. The cells of the grid world can be occupied by either agent shown in red and blue, the apples shown in green, or a wall in gray. The possible actions include moving in the four cardinal directions, rotating either way, tagging, or remaining idle. A successful tag temporarily removes the other player from the game, but can only be done to other nearby players. Players receive a reward of 1 for each apple picked. Episodes are limited to 100 timesteps.

D.2 Harvest: RGB

Harvest: RGB is a different implementation of the Harvest game introduced by Harvest: Categorical (Appendix D.1. Harvest: RGB is exactly the harvest implementation from MeltingPot [41] with the same orchard map. A rendering of the game state and observations is shown in Figure 12. The main difference between the Harvest versions is that the observations are $88 \times 88 \times 3$ images of the 11×11 viewbox in front of them. There are also minor differences in the implementation of tagging and apple respawn mechanism. Episodes play for 1000 timesteps.

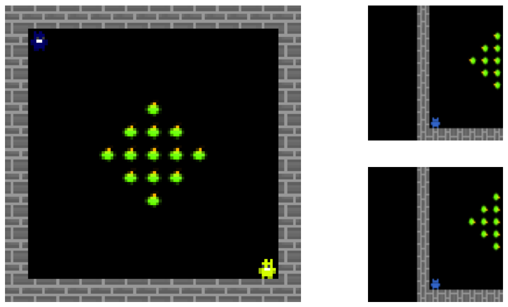
Figure 12: Harvest: RGB. Left: game state. Right: player observations.

D.3 Running With Scissors

Running With Scissors (RWS) is a temporally extended version of rock-paper-scissors (RPS). In it, players collect rock, paper, and scissor items into their inventory. At any point the player has the option to tag their opponent if they're nearby. Then they play RPS corresponding to the distribution of items in their inventories. The agents have the same action space as in the previous games. The observation space is $40 \times 40 \times 3$ image-based viewbox in fromt of them corresponding to a 6×6 grid around them. A portion of items are placed within the game deterministically, the rest are randomly sampled before play. If neither player tags each other before 1000 timesteps, the players are forced into playing RPS.

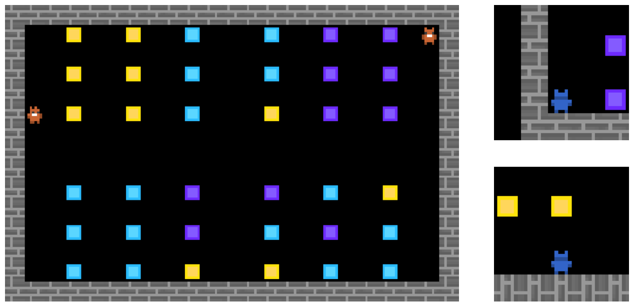
Figure 13: Running With Scissors. Left: game state. Right: player observations.

E Additional Results

E.1 Strategic Diversity

Figure 14 displays the recall results that correspond to the accuracies portrayed in Figure 2. See Section 3.1 for a discussion of these results.

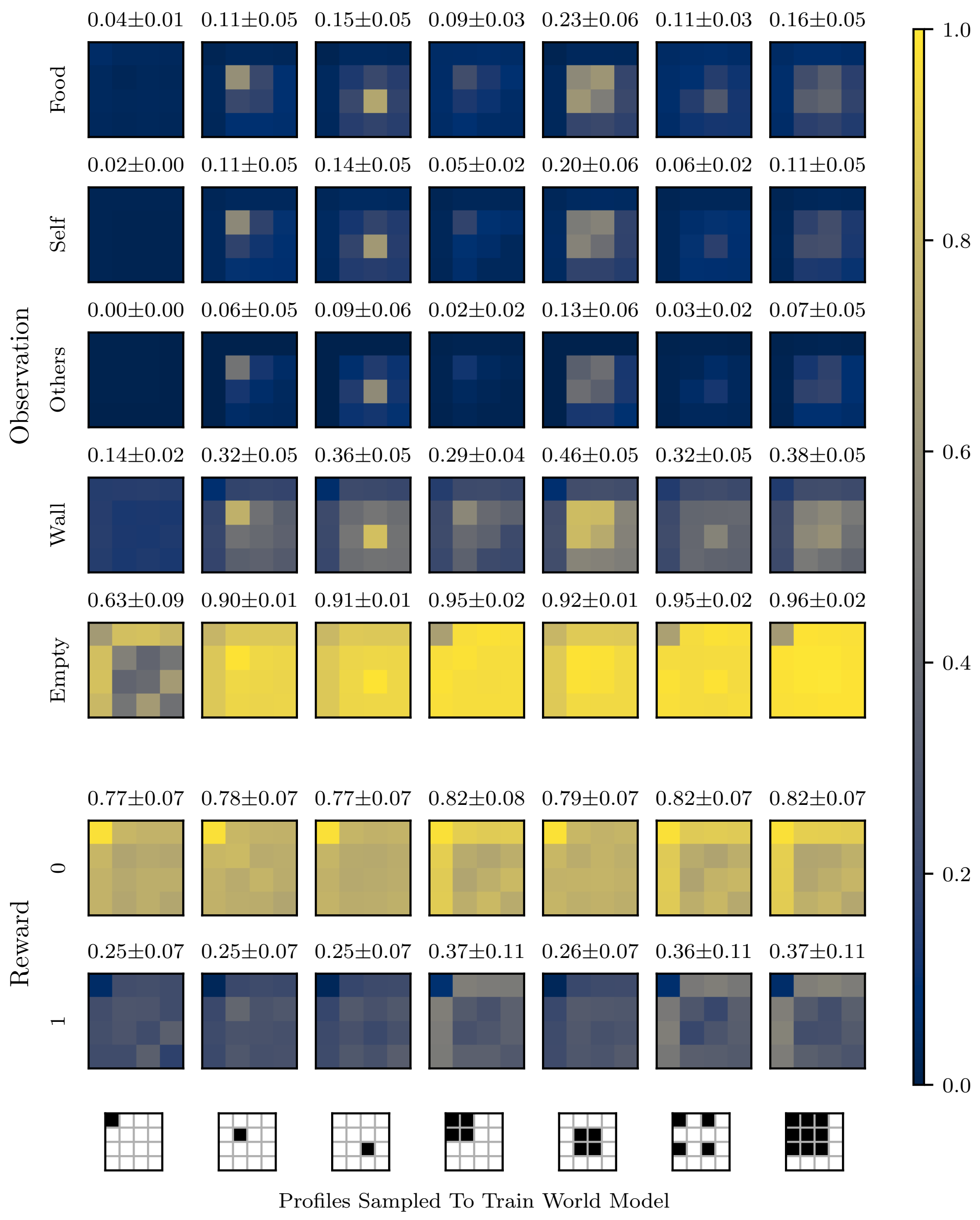

Figure 14: World model recall on Harvest: Categorical.

E.2 Background Planning

Figure 15 shows the results of repeating the background planning experiment with world model ▦. Besides changing the world model, the methodology is consistent with that described in Section 3.2.1. This result shows the planner achieving results comparable to the baseline method. Supporting the adoption of planning, as it tends to not negatively impact the learning process.

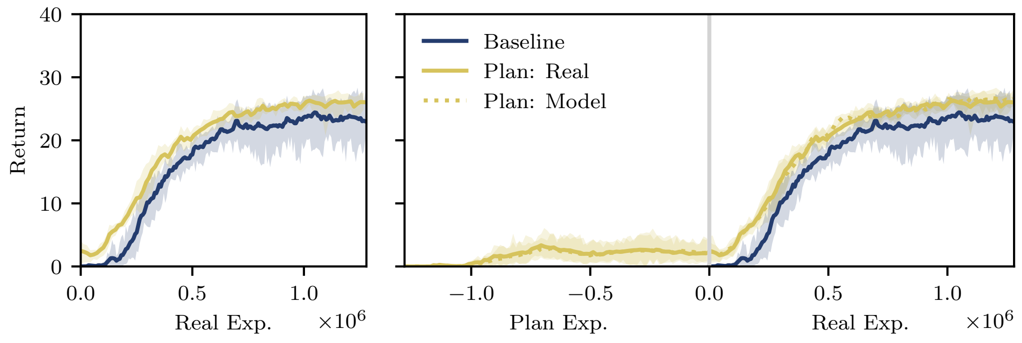

Figure 15: Effects of background planning on response computation using world model ▦. (5 seeds, with 95 % bootstrapped CI).

E.3 Decision-Time Planning

Figure 16 shows the results of repeating the decision-time planning experiment with world model ▦. Besides changing the world model, the methodology is consistent with that described in Section 3.2.2. This result further exemplifies the trend shown in Figure 4, where the planners that did not use BG: W failed to learn an effective policy. The planner that used BG: W achieved performance comparable to the baseline. Finally, the planner that used both BG: W and BG: C achieves the strongest performance at 33.07 ± 6.76. These results support the benefit of BG: W when using DT, and that effective planning performs as least as good as the baseline.

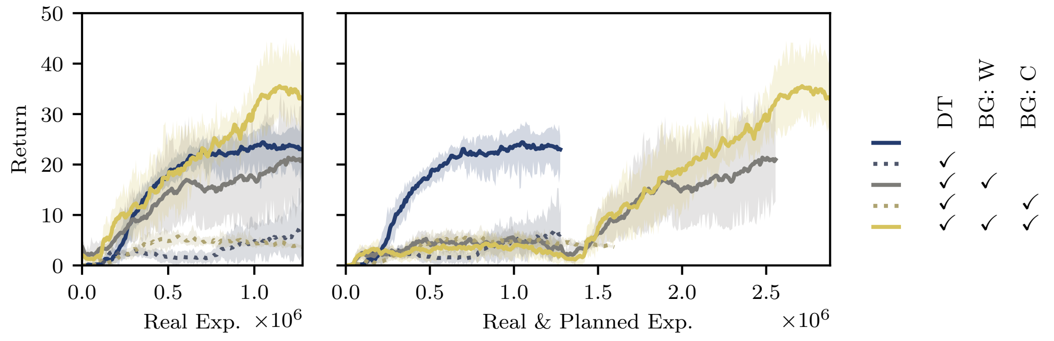

Figure 16: Effects of decision-time planning on response computation using world model ▦. (5 seeds, with 95 % bootstrapped CI).

E.4 World Models as Empirical Games

In this section, we verify the need for separate models.

First, consider if an empirical game can substitute for a world model. The majority of previous work on empirical games represents the model in the normal form. This representation abstracts away any notion of dynamics within an episode into a choice in policy and the resulting payoff. Since empirical games currently lack dynamics information completely, this supports the choice of separate models.

This is not without any exceptions. If the original game is one-shot and stateless (i.e., an episode is played through a single action), then a normal-form empirical game is exactly a world model.

Now, consider if a world model can substitute for an empirical game. World models predict successor states and rewards; and thus, can rollout planned trajectories to estimate payoffs. Note, that rolling out a trajectory a trajectory with a world model is an auto-regressive prediction that tends to result in compounding errors [73, 29]. Despite this, it is plausible that a world model can substitute as a high-fidelity empirical game.

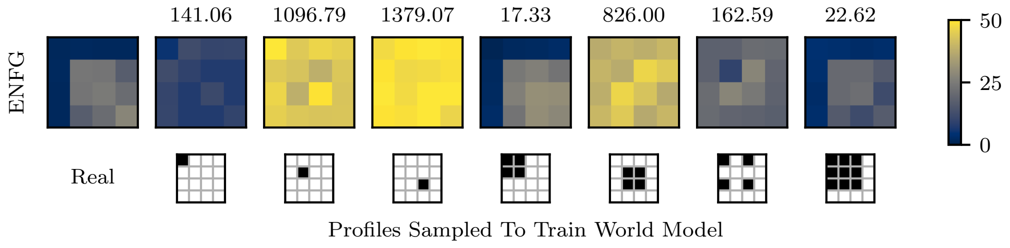

Figure 17: Empirical normal-form games (ENFG) estimated by world model rollouts. The title of each plot is its L2 distance with the real ENFG.

Figure 17 compares an empirical game estimated from real game payouts empirical games estimated with payouts predicted by a world model. In this experiment, the world models are the same that were used in Section 3.1. In general, the empirical games estimated by world models have large errors (L2 >100), with several having exceptionally large errors (L2 >1000). These result suggest that this direction may be possible with future algorithmic improvements; however, currently, the prediction errors are too large to substitute empirical games with world models. Especially in games with long time horizons.