Extended Abstract

Motivation Quarto is a 2 player board game that consists of a 4 by 4 board with 16 distinctive pieces, where the objective of the game is to get 4 pieces of a common attribute in a row, column, or diagonal. On each turn, a player places a piece on the board and gives the opponent their piece to play next. Similar to other board games, game play in Quarto consists of multi-turn trajectories that end in a terminal state of win, loss or draw. This creates a sparse reward structure which existing works on RL for Quarto use. We are interested in introducing intermediate rewards to encourage certain gameplay strategies using heuristics, to create a dense reward environment for training. This project answers the question: *What is the impact of dense reward environments on offline and online RL methods for Quarto?* We observe the resulting impact of training in a dense reward environment on agent performance, behavior, and rate of learning.

Method Our novel approach was to create a custom, dense reward environment by implementing a heuristic-based reward model that incentivizes certain agent behavior. In addition, we implemented Conservative Q Learning as an offline RL method. We compare the performance of an agent trained using CQL, MaskedPPO, and MaskedPPO with self-play in sparse and dense reward environments, to understand the effects of the intermediate rewards.

Implementation To evaluate agent performance in the game of Quarto, we tracked outcome-based metrics: win rate, loss rate, draw rate, and custom behavior-based metrics: the number of threats created, threats blocked, bad pieces given, and game turns. Our primary success metric was win rate, and our other metrics helped reveal behavioral differences across training regimes. For offline learning, we implemented Conservative Q Learning (CQL) from scratch, generating datasets of 30,000 episodes under both sparse and custom dense reward settings, with games between random opponents adn trained MaskedPPO agents. The CQL agent was trained for 1 million steps using an alpha penalty of 1.0. For online learning, we extended the Quinto codebase to train MaskedPPO agents in both sparse and dense reward environments, against both random opponents and in self-play settings for 1 million timesteps.

Results Our experiments reveal with online MaskedPPO, a dense reward environment led to signficant performance improvement compared to sparse rewards. In self-play, MaskedPPO trained with dense rewards achieved a 98.6% win rate, significantly outperforming the sparse counterpart at 27.7%. Even against random opponents, the dense-reward model achieved 78.6% win rate versus 50.3% in the sparse setting. These performance gaps are accompanied by reductions in bad piece selection and average game turns, suggesting better outcomes and more efficient and strategic play. However, we also discover that other heuristics, such as Threats Created and Threats Blocked, did not correlate with optimal performance. In contrast to MaskedPPO, our offline CQL experiments showed a minimal difference between dense and sparse environments.

Discussion Our results demonstrate that dense reward shaping in online MaskedPPO settings can significantly improve online reinforcement learning performance in Quarto. However, our offline learning experiments using Conservative Q Learning showed more insignificant differences, likely due to the limitation of not enough training on enough high-quality data. This contrast points to a broader insight: while reward shaping is powerful, its effectiveness is conditional on the training setting, and offline RL remains highly sensitive to the diversity and quality of collected data.

Conclusion Beyond Quarto, these findings underscore the value of integrating domain knowledge into reward design for environments with sparse feedback. The benefits of intermediate rewards—incentivizing strategic behavior, and accelerated convergence—can generalize to other sequential decision-making problems such as robotic planning or multi-step control tasks. Future directions could include leveraging learned reward functions via inverse reinforcement learning to better align training signals with expert-level strategy, or scaling to larger and more complex games. Ultimately, our work contributes to a broader understanding of how structure and feedback interact in reinforcement learning, and we hope it informs both future research in game-playing agents and real-world applications with sparse or delayed rewards.

RL on Quarto: Experimenting with Dense Reward Environments

Ishvi Mathai
Department of Computer Science
Stanford University
ishvim@stanford.edu

Humishka Zope
Department of Computer Science
Stanford University
zope@stanford.edu

Aditri Patil
Department of Electrical Engineering
Stanford University
apatil26@stanford.edu

Abstract

Quarto is a two-player board game with a sparse reward structure, making it a challenging environment for reinforcement learning (RL). This project investigates how introducing dense, heuristic-based intermediate rewards impacts learning performance in both online and offline RL settings. We implemented a custom reward model to incentivize strategic gameplay behaviors and evaluated agent performance using outcome metrics (e.g., win rate) and behavior metrics (e.g., threats created, bad pieces given). For online learning, we trained agents using MaskedPPO with and without self-play in both sparse and dense reward environments. For offline learning, we implemented Conservative Q Learning (CQL). Results show that dense rewards significantly improve online performance: in self-play, dense-trained agents reached a 98.6% win rate compared to 27.7% in the sparse setting. Offline learning with CQL, however, showed minimal difference between reward conditions, likely due to limited high-quality data. These findings suggest that while dense reward shaping can accelerate learning and encourage efficient strategy in online RL, its benefits are conditional in offline settings. More broadly, this work highlights the importance of integrating domain knowledge into reward design, especially for sequential decision-making problems with sparse or delayed feedback.

1 Introduction

In this project, we explore the impact of dense reward environments on both offline and online RL methods on Quarto. Quarto is a 2-player board game where the object is to get 4 pieces, of 16 unique pieces that are characterized by 4 attributes, that share an attribute in a row, column, or diagonal. On each turn, a player places a piece on the board and gives the opponent their piece to play next. A unique aspect of Quarto is that players choose the piece that their opponent places next. Thus, the goal becomes "trapping" the next player into giving you a winning move, making Quarto a great game for strategic and anticipatory thinking and adversarial planning.

However, similar to many other board game structures, Quarto's game ends after multi-turn trajectories in a terminal state of win, draw, or loss. This structures the problem as a sparse reward environment, which can lead to more challenges when training agents to play the game. We found that existing implementations only used sparse reward environments (with rewards given at the terminal states) to train RL methods on Quarto. Furthermore, existing methods have only implemented online Rl

algorithms (such as PPO) largely due to the limitations of a sparse reward environment, and these methods took a relatively large time to train (anywhere from 6 hours to 1 week). Thus, we wondered about the impact of dense reward environments, where we reward not only the final state (for a win, loss, draw) but also use specific heuristics for attributes of game-play that occur during the game to assign intermediate rewards in trajectories. Our specific research question is: *What is impact of a dense reward environment on agent performance, behavior, and rate of learning in both online methods (MaskedPPO) and offline methods (Conservative Q Learning)?*

Our experiments reveal the impact of specific heuristics that incentive better performance (higher win-rate) and improvements in agent strategy. Furthermore, using our dense reward environment, we further explored the performance of offline methods by implementing Conservative Q-Learning. In addition, we explored the performance improvement of dense rewards with MaskedPPO when playing against a randomly trained opponent versus self-play.

2 Related Work

While existing work on RL methods for Quarto is limited, prior research has explored approaches have explored using both traditional and reinforcement learning methods. Pedersen (2019) examined Quarto as a reinforcement learning (RL) problem, implementing various RL techniques trained via self-play. His findings highlighted that while the choice of learning algorithm had limited effect on overall performance, differences in hyperparameter configurations significantly influenced agent behavior and learning dynamics over time. This work underscores the sensitivity of Quarto-playing agents to training regimes and provides foundational insight into neural value function approximation in combinatorial game settings.

Another approach to Quarto AI has relied on classical search methods. Mohrmann et al. (2013) has Java-based implementation used depth-first search with alpha-beta pruning, transposition tables, and constraint solvers to effectively narrow the search space and enable real-time gameplay against human opponents. While not learning-based, this method demonstrated the effectiveness of domain-specific heuristics and optimization in game tree search.

More recently, the Capuano and Matteotti (2023) Quinto project introduced a modern deep reinforcement learning framework for Quarto. This work focuses on an on-policy learning setup enhanced with innovative components such as action masking and incremental self-play. The use of action masking—filtering out illegal or non-optimal actions during learning—aligns particularly well with the complex action space in Quarto and represents a notable step toward scalable learning in constrained environments.

A limitation of these prior works is that they all use sparse reward environments. Our project introduces the novel approach of experimenting with dense reward environments. We build upon Quinto, which provides a setup for MaskedPPO in a sparse environment, and additionally implement an offline technique of Conservative Q Learning.

3 Method

To understand the impact of dense reward environments, we evaluated performance using Online methods (MaskedPPO) and Offline methods (Conservative Q-Learning). To do this, we created a custom environment with dense rewards and behavior tracking, set up agent training, and implemented Conservative Q Learning. In addition, we evaluated performance in training with both a random opponent and self-play (playing against ourselves) with sparse and dense reward environments. We contrast our agents' behavior and performance with our baselines, MaskedPPO with sparse rewards and CQL with sparse rewards.

Our rationale with implementing both an offline method and an online method was that we wanted to understand whether including intermediate rewards would make offline learning more feasible, as offline learning is generally challenging in sparse reward environments. Furthermore, we decided to experiment with self-play to evaluate whether training against a progressively stronger agent leads to more robust policies, especially in dense reward settings. Self-play has shown success in other strategic games by allowing agents to learn increasingly sophisticated behaviors through iterative competition. By comparing self-play with training against a random opponent, we aimed to isolate

how much of the agent's improvement stems from the reward shaping itself versus the quality of the opponent. This also allowed us to assess whether dense rewards accelerate learning even in more adversarial, non-stationary training regimes.

A full list of experiments we ran:

- Conservative Q-Learning in Dense Reward Environment
- Conservative Q-Learning in Sparse Reward Environment (Baseline)
- Conservative Q-Learning with Self-Play in Dense Reward Environment
- Conservative Q-Learning with Self-Play in Sparse Reward Environment (Baseline)
- MaskedPPO with Random Opponent in Dense Reward Environment
- MaskedPPO with Random Opponent in Sparse Reward Environment (Baseline)
- MaskedPPO with Self-Play in Dense Reward Environment
- MaskedPPO with Self-Play in Sparse Reward Environment (Baseline)

3.1 Dense Reward Environment

We developed four different heuristics to model optimal gameplay in Quarto and designed a custom environment that rewards or penalizes the agent based on the following criteria:

- **Threats Created:** The agent is awarded $+2$ (previously $+6$ in Reward Model #1) every time it places a piece that creates a row, column, or diagonal of pieces with shared attributes.
- **Threats Blocked:** The agent is awarded $+2$ (previously $+7$ in Reward Model #1) every time it places a piece that blocks the opponent from completing a row, column, or diagonal of pieces with a common attribute.
- **Bad Pieces:** The agent is penalized -10 (equivalent to a loss) if it hands the opponent a piece that allows them to win on their next move.
- **Game Turns:** To encourage strategic and efficient gameplay, the agent is penalized $-0.5 \times (\text{number of turns played})$. This discourages the agent from accumulating intermediate rewards unnecessarily and instead focuses on quick victories.

We also define terminal rewards based on the game outcome:

- **Win:** $+10$
- **Lose:** -10
- **Draw:** 0 (previously 0.2 in Reward Model #1)

To determine the appropriate reward values for each heuristic and game state, we ran experiments across a variety of hyperparameter settings. The final reward model was selected based on configurations that maximized the agent's win rate.

3.2 Sparse Reward Environment (Baseline)

Our baseline was running experiments in a sparse reward environment where a reward was only given to the terminal state:

- **Win:** $+1$
- **Lose:** -1
- **Draw:** 0

3.3 Assumptions

We make the following assumptions:

- We assume full observability of the board state and the opponent's given piece.

- The environment is deterministic; randomness arises only from the opponent's policy or initial conditions.
- Agents receive information about the game state as a structured observation, which includes the current board layout and the given piece.
- Opponents are either fully random or use the same policy network (self-play).

3.4 Math/Algorithm Details Behind Conservative Q-Learning

Conservative Q-Learning (CQL) is an offline reinforcement learning algorithm designed to learn policies from fixed datasets without interacting with the environment during training. CQL adds a penalty to the Q-function objective that explicitly reduces overestimation of unseen actions, improving stability and generalization.

From Finn (2025), CQL optimizes the following objective:

$$\mathcal{L}_{\text{CQL}} = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q(s,a) - r - \gamma \max_{a'} Q(s',a')\right)^2\right] + \alpha \cdot \mathbb{E}_{s\sim\mathcal{D}}\left[\log \sum_{a} \exp(Q(s,a)) - \mathbb{E}_{a\sim\pi_\beta} Q(s,a)\right]$$

The first term is the standard Bellman error, and the second is the conservative penalty, where α is a hyperparameter controlling conservatism, and π_β is the behavior policy from the dataset.

We use a dataset of trajectories generated from a expert opponent in the Quarto environment, and train the Q-function using a replay buffer of offline interactions. The policy is implicitly derived from the learned Q-values via softmax.

3.5 Math/Algorithm details behind MaskedPPO

MaskedPPO is a variant of Proximal Policy Optimization (PPO) tailored for environments like Quarto where not all actions are valid at each timestep. At each decision point (e.g., selecting a square or choosing a piece), the agent receives an action mask indicating which actions are valid.

The PPO objective is:

$$\mathcal{L}_{\text{PPO}} = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the ratio of new to old policy probabilities, and $\hat{A}_t$ is the advantage estimate at time t.

To account for invalid actions, we apply a masking operation during both sampling and policy evaluation, ensuring that probability mass is only assigned to legal actions.

MaskedPPO agents are trained using environments where the observation includes both the board state and the given piece, with dense or sparse rewards depending on the experiment. Self-play training alternates between the agent playing as the piece-placer and board-player, mirroring turn-based play.

4 Experimental Setup

4.1 Metrics

We used these metrics:

1. Win-rate: percent of game trajectories agent won
2. Loss-rate: percent of game trajectories agent lost
3. Draw-rate: percent of game trajectories agent drew

To track agent behavior, we also track the average of Threats Created, Threats Blocked, Bad pieces, and Game Turns. The primary metric used to evaluate overall performance was win rate.

We used tensorboard and wandb to create visualizations of the game trajectories, and log metrics and loss curves.

4.2 Implementation Details for Conservative Q Learning

To run our experiments for CQL, we implemented CQL from scratch. This means first generating the dataset with 30,000 episodes. We chose this size after experimenting as it provided diverse data and manageability. We generated rollouts in our custom dense rewards environment vs the prior sparse environment. We also compare rollouts containing games of 2 random opponents versus self-play of 2 MaskedPPO v3 opponents. Each transition contains the info dictionary with fields like "threat_blocked" and "bad_piece". Then we load this data and train a CQL agent with an alpha penalty weight of 1.0. We train for 1M steps and after each step the Bellman error and CQL penalty are logged to the tensor board. We choose to evaluate against a random opponent to keep it consistent across the parameters.

4.3 Implementation Details for MaskedPPO Learning

To run our experiments using the MaskedPPO algorithm, we utilized the existing Quinto codebase . We first created a custom environment with a different reward function to capture all the intermediate rewards in our dense environment. We then trained an agent using a random opponent, and then the agent itself for self-play. We also enhanced the Quinto base environments to keep track of our intermediate metrics: "threats created", "threats blocked", "game turns", and "bad pieces given" to track the effects of the dense environment as compared to the baseline sparse environment performance. We also used wandb to track sample trajectories to qualitatively analyze the training and performance of the agent in the dense reward environment. We trained for 1M timesteps for each run during which we have already found comparable and superior results to Quinto's performance.

5 Results

5.1 Quantitative Evaluation

Metric	Offline CQL (Dense)	Offline CQL (Sparse)
Avg. Win Rate	33.9%	32.2%
Avg. Loss Rate	64.2%	65.7%
Avg. Draw Rate	1.9%	2.1%
Threats Created	1.83	1.85
Threats Blocked	0.57	0.64
Bad Pieces	2.34	2.21
Game Turns	6.62	6.69

Table 1: Comparison of Offline CQL performance under dense and sparse reward settings, trained with offline trajectories of random opponents and evaluated against a random opponent.

Metric	Offline CQL Self-Play (Dense)	Offline CQL Self-Play (Sparse)
Avg. Win Rate	32.15%	30.42%
Avg. Loss Rate	67.85%	69.58%
Avg. Draw Rate	0%	0%
Threats Created	1.84	1.74
Threats Blocked	0.48	0.71
Bad Pieces	2.30	3.41
Game Turns	6.62	6.60

Table 2: Comparison of Offline CQL performance under dense and sparse reward settings with offline dataset generated with self-play, evaluated against a random opponent.

Metric	MaskedPPO (Dense)	MaskedPPO (Sparse)
Avg. Win Rate	78.6%	50.3%
Avg. Loss Rate	19.5%	47.4%
Avg. Draw Rate	1.9%	2.3%
Threats Created	1.67	1.85
Threats Blocked	0.73	0.55
Bad Pieces	1.54	2.33
Game Turns	11.04	12.21

Table 3: Comparison of MaskedPPO against Random opponent under dense and sparse reward settings.

Metric	MaskedPPO Self-Play (Dense)	MaskedPPO Self-Play (Sparse)
Avg. Win Rate	98.6%	27.7%
Avg. Loss Rate	1.4%	70.3%
Avg. Draw Rate	0.0%	2.0%
Threats Created	0.19	1.25
Threats Blocked	0.06	0.30
Bad Pieces	0.03	1.40
Game Turns	5.62	11.13

Table 4: Comparison of MaskedPPO with Self-play performance under dense and sparse reward settings.

5.1.1 CQL Results Analysis

When comparing Offline CQL performance trained on trajectories from random opponents (in Table 1), we observe minimal differences between the dense and sparse reward conditions. The average win rate was slightly higher in the dense setting (33.9%) compared to sparse (32.2%), with a corresponding marginal decrease in loss rate (64.2% vs. 65.7%). However, the difference is very slight, which suggests that introducing intermediate feedback did not significantly improve performance in this setting. Metrics related to gameplay behavior, such as threats created and game turns, were nearly identical across both reward schemes. Overall, these results indicate that when the quality of the training data is low (e.g., random play), dense rewards do not provide a substantial advantage in offline CQL training.

In contrast, our results in Table 2 show that when the offline dataset is generated from self-play, a clearer distinction emerges between the dense and sparse reward settings. The model trained with dense rewards achieved a higher average win rate (32.15%) than the sparse counterpart (30.42%), with a corresponding reduction in loss rate (67.85% vs. 69.58%). Although these gains are modest, they are more consistent than in the random-opponent case. A notable behavioral difference appears in the number of bad pieces given—models trained with sparse rewards gave significantly more bad pieces on average (3.41) than those trained with dense rewards (2.30), suggesting that intermediate feedback may have helped the model avoid tactically harmful moves. However, we also see that the sparse-reward model blocked more threats (0.71 vs. 0.48), implying that 'threats blocked' as a heuristic behavior might not be completely aligned with winning strategies. These findings suggest that dense rewards can improve learning from higher-quality data by reinforcing more beneficial intermediate decisions, however further investigation is required (limitations discussed in Discussion section).

Interestingly, we see that overall, the win rate for CQL trained with random trajectories is slightly higher than our CQL models trained with self-play, regardless of the environment. We hypothesize that this is because using random opponents to generate trajectories leads to a more diverse dataset where more states and actions are explored, compared to expertwhere the agent may tend to overfit to a narrower distribution of game scenarios.

5.1.2 MaskedPPO Results Analysis

When running MaskedPPO against Random opponents (Table 3), the difference between sparse and dense environments becomes more significant. Our model trained with dense rewards achieved

a markedly higher win rate (78.6%) compared to its sparse counterpart (50.3%), along with a significantly lower loss rate (19.5% vs. 47.4%). This suggests that intermediate rewards provided meaningful guidance during training, enabling the policy to converge on more successful strategies. Our results also show the impact of specific heuristics that were optimal for agent strategy. The dense-reward agent gave far fewer bad pieces (1.54 vs. 2.33) and blocked more threats (0.73 vs. 0.55), suggesting that these two heuristics led to better strategy. However, we see that the the sparse-reward agent created slightly more threats (1.85 vs. 1.67), suggesting that rewarding threat creation is not very useful for optimal agent performance. Overall, our results underscore that dense rewards significantly improved game win rate and agent game-play.

When looking at the self-play model performance (Table 4), the difference between dense and spare environments becomes even more significant. We see a more significant difference in win rate between our models, and also see a significant decrease in the average number of game turns in dense versus sparse environments. This suggests that because the agent is learning optimal strategy in the dense environment, it is able to win the game in much fewer moves. Interestingly, self-play leads to worse performance than random-opponent training in the sparse reward setting, likely because the agent overfits to its own suboptimal strategies; however, in the dense reward setting, self-play significantly outperforms random data, suggesting that intermediate feedback enables the agent to effectively bootstrap and refine its own play over time.

Compared to Capuano and Matteotti (2023), our model achieves 98% win rate, comparable to the best model in the Quinto project. However, we see that our model required 1M initial training steps + 200k self-play whereas their model used 120M training steps.

In the figure below, we visualize the training of a MaskedPPO model in the dense reward environment for 1M training steps (grey) alongside self-play training using that model for 1M steps (pink). We see that the better model trained using self-play learns to give less bad pieces, which is strongly correlated with its win rate. However, the plots for "threats created", "threats blocked", "game turns" and "reward" seem to show that our reward model was not entirely optimal and that more unnecessary threats are being created and blocked without adversely affecting win rate to increase reward.

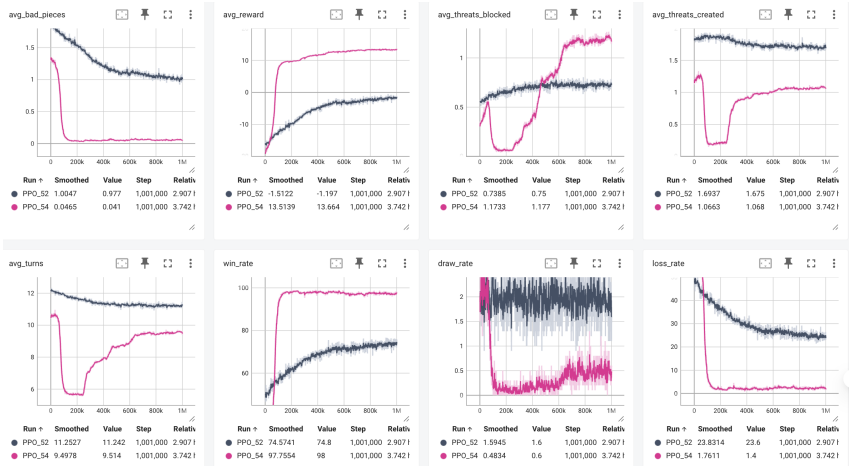

Figure 1: Visualization of a random opponent (grey) vs. self play (pink) training in dense reward environment.

5.1.3 Analysis of optimal game strategy

Based on our results and by contrasting the behavior of our models, we see that the most significant difference in agent behavior comes from the "Bad Pieces" heuristic. This penalizes the agent for giving the opponent a piece which then can be placed on the board for the opponent to win. Our results show better performing agents signficantly reduce the number of bad pieces given.

Our results also show how our reward model was not fully optimal, as metrics like threats created or blocked do not correlate with performance as strongly as the bad piece metric. In fact, some agents

that created more threats performed worse overall. Overall, our analysis highlights that optimal strategy in Quarto may depend more critically on foresight and safe piece-giving than on aggressive play or threat generation alone.

5.2 Qualitative Analysis

To perform a qualitative analysis of our best model that was trained using the MaskedPPO algorithm and self-play, we analyze a sample gameplay trajectory after convergence (Figure 1).

In the first play, the agent chooses piece 11 for the opponent, and the opponent places it at (2,0).

In the second play, the agent places piece 5 that was chosen by the opponent at (1,2) and chooses 13 for the opponent which places the piece at (2,3).

In the next play, the agent places piece 3 at (3,2) and gives the opponent piece 9 which it places at (2,1).

The following play is very interesting because the agent uses the chosen piece 0 and places it at (2,2) which blocks the opponent's threat of shared first attribute along the 3rd row, but also creates a threat of 3 pieces sharing their first attribute along the 3rd column. Here we see the effects of the threat blocked and threat created rewards. The agent also chooses not to give the opponent a piece with first attribute False that would allow the opponent to win the game. Here we see the effect of a penalty for bad piece given. Instead the agent gives the opponent piece 10 which it places at (1,0).

On the next turn, agent gets piece 1 which has the first attribute False and places it at (0,2) to use the threat created to win!

The agent choosing to place this piece to win shows a culmination of effects of choosing a reward for winning far greater than any intermediate rewards in the environment, and also penalizing number of game turns to encourage the agent to win the game strategically and quickly.

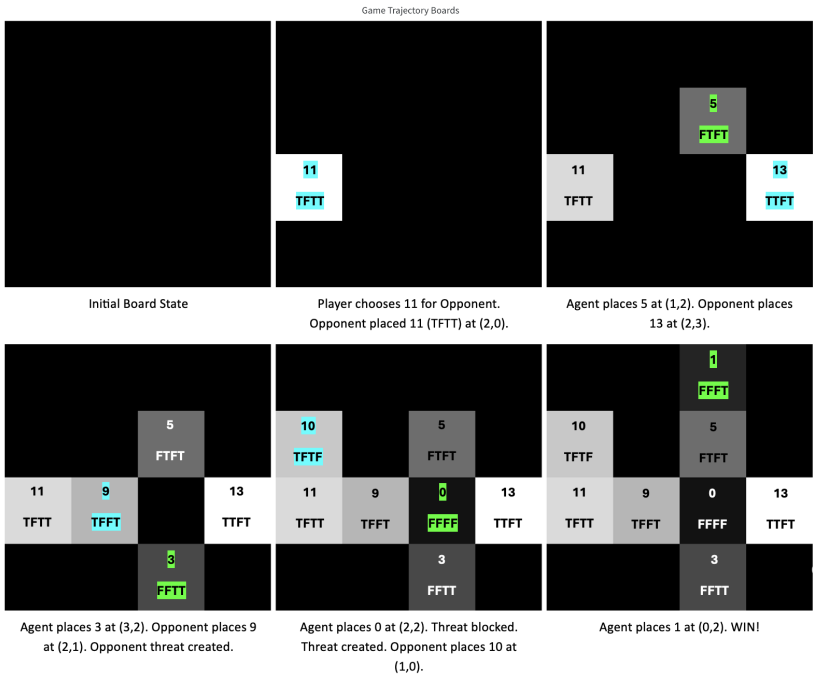

Figure 2: Visualization of a gameplay trajectory.

6 Discussion

6.1 Limitations and challenges

We faced a challenge when generating the offline dataset for Conservative Q learning. Specifically, we faced memory issues with storing large datasets. As a result, the largest dataset size we were able to generate for offline learning was 30,000 trajectories, as our AWS instance would crash with larger dataset sizes. Moreover, the process would get killed both on the local machine and on AWS. However, as Quarto is a complex game with 16 board positions and 16 distinct pieces, this lead to a large amount of possible board configurations, all of which could not have been covered by our offline dataset. And thus, it was difficult to achieve high diversity and quality of the dataset, not allowing the model to generalize and exploit novel strategies. This is a limitation of our Conservative Q Learning results, making it more difficult to view the difference in results between our experiments with CQL.

Another challenge we faced was shaping an effective reward function. While we observed that "bad pieces given" was a good heuristic for strategic gameplay, when the values for "threats created" and "threats blocked" were too high, this caused the model to prefer prioritizing them over winning the game by unnecessarily increasing trajectory lengths to boost reward. Moreover, the values used for penalizing every turn and the win reward and loss penalty were tuned to ensure that the model learned that winning was the priority.

6.2 Broader Impact

Overall, our experiments show that dense rewards do have a significant impact on improving online RL for Quarto, and this difference is further exaggerated in a self-play environment. More generally, our project demonstrates that adding intermediate rewards has potential to improve agent strategy in board games. Since board gamesoften serve as standard benchmarks for evaluating planning and decision-making in RL, our findings go beyond just Quarto. They show that careful reward-shaping and well-chosen heuristics for intermediate feedback in sparse environments can improve sample efficiency, increase win rates and accelerate convergence. These benefits could apply to other domains such as robotic manipulation and autonomous decision-making tasks. By integrating domain knowledge with RL, we show that agent behavior can be improved in challenging environments with delayed and sparse rewards.

7 Conclusion

The ability to play and win board games has often been related to signs of reasoning, anticipatory and strategic thinking. Board games, such as Chess and Go, are considered some of the hardest tasks for computers to beat humans at, and are often used as a testbed to measure AI agents. Through our project, we demonstrate the potential of adding intermediate awards to Quarto by showing how it improves game win rate, agent strategy, and optimizes training time with a variety of methods (PPO, Conservative Q Learning, and self-play). Future work in this direction could include exploring other heuristics that could lead to optimal strategy, as well as explored learned reward models using techniques such as inverse reinforcement learning to better align reward shaping with human strategy. Through our project, we hope to inspire future work on improving agents performance on other board games with similar methods.

8 Team Contributions

- **Group Member 1: Ishvi Mathai** Defined the dense reward environment and implemented a custom dense environment. Ran experiments with random opponent and self-play using MaskedPPO in sparse and dense reward environments. Analyzed results to create reward model #2 for better dense environment. Analyzed trajectories for qualitative analysis.
- **Group Member 2: Humishka Zope** Defined the dense reward environment. Ran experiments with MaskedPPO with random opponent. Analyzed results to create reward model #2 for better dense environment for training. Analyzed results for quantitative analysis.

- **Group Member 3: Aditri Patil** Defined the dense reward environment. Analyzed results to create reward model #2 for better dense environment for training. Implemented and ran experiments for CQL training with random opponent and self-play in sparse and dense environments.

References

Francesco Capuano and Matteo Matteotti. 2023. Quinto Report. https://github.com/fracapuano/Quinto/blob/main/Quinto_Report.pdf. https://github.com/fracapuano/Quinto/blob/main/Quinto_Report.pdf Final project report for the "Computational Intelligence" course @ PoliTo, 2022/2023.

Chelsea Finn. 2025. CS224R Spring 2025 Homework 3: Offline Reinforcement Learning. https://cs224r.stanford.edu/material/CS224r_Homework3_updated.pdf. Accessed: 2025-06-09.

Jochen Mohrmann, Michael Neumann, and David Suendermann. 2013. An artificial intelligence for the board game 'Quarto!' in Java. 141–146. https://doi.org/10.1145/2500828.2500842

Frode Stig Nørgaard Pedersen. 2019. Quarto as a Reinforcement Learning problem. https://api.semanticscholar.org/CorpusID:207902811

A Additional Experiments

One influential experiment that pointed us towards Reward Model #2 was training with Reward Model #1 and noticing the very high rewards. We observed that despite the fall of the win rate after peaking around 75%, the reward continued to increase, as well as the threats created and blocked. Through qualitative analysis of the trajectories we noticed longer games where threats were being created and blocked unnecessarily when more direct wins were possible. SO we adjusted the rewards in Reward Model #2 to reduce the weight given to threats created and blocked, penalized number of turns, so as to prioritize winning over accumulating intermediate rewards.

We also ran an experiment using Quinto's best fine-tuned model (120M training steps) as an opponent. This gave us nearly a 100% win rate and showed us that the agent can quickly learn from the opponent in the dense reward environment. This led us to experimenting with better opponents than the random opponent, i.e., self-play as a promising next step to better results.

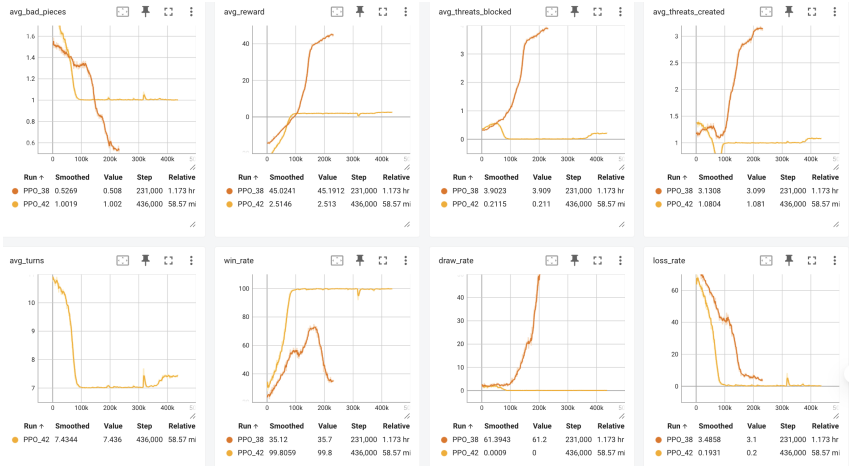

Figure 3: Visualization of training metrics with Reward Model #1 (orange) and Quinto opponent (yellow).