RL Assignment 2 Super Tic-Tac-Toe

WENG Yanbing 21091234 ywengae@connect.ust.hk

WONG Chongqi

1. Problem Definition

Super Tic-Tac-Toe is an advanced variant of the traditional Tic-Tac-Toe game, characterized by a more complex game board and enhanced rules. Unlike standard Tic-Tac-Toe, this game is played on a cross-shaped board consisting of five interconnected 4×4 squares as the Figure 1 shown. Players alternate turns to place noughts and crosses, attempting to win by aligning 4 markers consecutively in a row or column, or 5 markers diagonally.

The key distinctions of this variant include:

- **Board Configuration:** A uniquely shaped board resembling a cross
- **Probabilistic Move Acceptance:** Each move chosen by the player has only a 50% chance of being placed exactly as intended. If the intended square is not successfully claimed, the piece is instead randomly placed on one of eight adjacent squares (probability of 1/16 each). Moves that land off the board or onto an occupied cell are forfeited entirely.
- **Winning Conditions:** Achieving 4 aligned pieces horizontally or vertically, or 5 diagonally.

Therefore, the primary challenge of this assignment is not merely understanding and implementing game mechanics, but also leveraging reinforcement learning techniques—particularly using TensorFlow's TF-Agents framework—to develop a robust agent capable of effectively playing under these stochastic conditions.

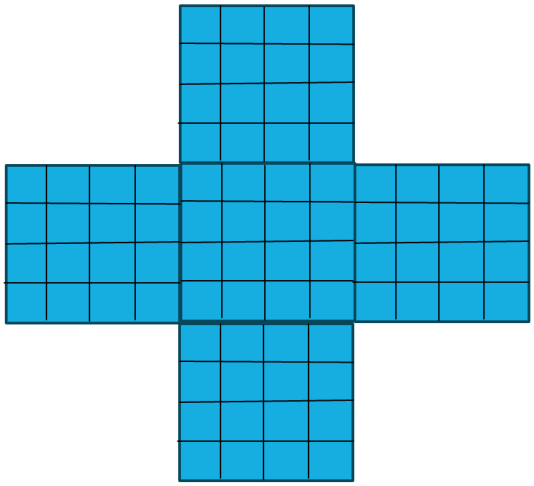


Figure 1: Super Tic-Tac-Toe Board Diagram

2. Model Solution

2.1 Challenges

This Super Tic-Tac-Toe variant presents distinct challenges that require a targeted reinforcement learning approach:

- **Complex State and Action Space:**

  The cross-shaped board significantly increases the complexity of state representation and valid actions compared to traditional Tic-Tac-Toe, necessitating careful state-space design.

- **Stochastic Environment Dynamics:**

  The probabilistic acceptance of moves introduces uncertainty, demanding an RL algorithm robust enough to handle stochastic transitions effectively.

- **Sparse and Shaped Rewards:**

  Winning conditions are sparse events. Therefore, reward shaping must guide intermediate learning effectively without inadvertently creating suboptimal agent behaviors.

- **Illegal Moves and Action Masking:**

  Handling invalid actions is critical, as performing illegal moves wastes opportunities and potentially costs the game.

2.2 Choice of PPO Algorithm

We adopt Proximal Policy Optimization (PPO) as the reinforcement learning algorithm due to its excellent performance in complex stochastic environments.

PPO is a policy-based actor-critic method that iteratively samples trajectories from environment interactions and optimizes a clipped surrogate objective function using stochastic gradient ascent. The actor network updates the policy by maximizing the expected advantage, while the critic estimates the value function to provide the learning signal. The introduction of the clipped objective function is a key innovation as it constrains the policy update to a bounded range, thereby improving stability and preventing overfitting to high-variance gradients. This design is particularly beneficial in environments with stochastic transitions, such as Super Tic-Tac-Toe, where abrupt policy changes may lead to undesirable behavior. PPO also supports flexible neural network architectures that are suitable for the large and complex action space and state representation of Super Tic-Tac-Toe. In addition, the entropy regularization embedded in PPO facilitates efficient exploration, striking a balance between exploiting known policies and discovering new ones.

In this project, I primarily optimized PPO performance by tuning the following hyperparameters:

- **Clip ratio**：constrains the update magnitude between the new and old policies to prevent abrupt changes;
- **GAE lambda**： the smoothing parameter for Generalized Advantage Estimation, balancing bias and variance;
- **Discount factor**： determines the importance of future rewards;
- **Entropy coefficient**：controls the weight of the entropy regularization term to encourage exploration.

2.3 Tailored PPO Implementation for Super Tic-Tac-Toe

2.3.1 States Presentation

In traditional Tic-Tac-Toe implementations, the board state is often represented as a 2D array with scalar values indicating empty cells, player X's moves, and player O's moves. While this representation suffices for simple scenarios, it lacks the capacity to convey additional contextual information, such as the current player's turn or potential strategic patterns. To address this limitation, our design employs a three-channel tensor representation of shape (BOARD,

BOARD, 3). The first channel encodes the positions occupied by player X, the second channel represents player O's moves, and the third channel indicates the current player's turn. This enriched representation provides the agent with comprehensive spatial and temporal context, facilitating more informed decision-making and enabling the neural network to learn complex patterns and strategies inherent in the game's dynamics.

2.3.2 Reward Reshaping

In reinforcement learning (RL), sparse reward environments are characterized by infrequent and delayed feedback, where the agent receives significant rewards only when it achieves a specific goal or reaches a terminal state. Super Tic-Tac-Toe is a typical example of such an environment, where the agent usually receives a large reward only after winning the game, with little or no feedback in the intermediate steps. This sparsity poses a significant challenge to policy gradient methods such as proximal policy optimization (PPO), because these methods rely on frequent and informative reward signals to effectively update policies. Sparse rewards make it difficult for the agent to associate specific actions with the final results, making it difficult for the agent to even learn basic rules. In addition, without effective intermediate rewards, the agent has difficulty obtaining sufficient guidance to explore the environment, and is easily trapped in local optimization. It is also easy to cause high variance in the policy update process, which affects the stability of the learning process and hinders convergence. To address these challenges, we design specialized reward and punishment logic and obtain reasonable and effective reward and punishment strengths through experiments.

Our four key components of reward shaping approach are as follows:

1. **Penalties for Invalid Moves and Rewards for Valid Moves**

   Facing more complex invalid areas in the Super Tic-Tac-Toe, we assign positive reward and negative reward to legal and illegal moves respectively to encourage adherence to game rules and promote valid action selection. Illegal moves include placing pieces on occupied grids or outside the board boundaries. It is worth noting that we shouldn’t only give penalties to illegal moves. If only illegal moves are punished without encouraging legal moves, it will be difficult for the agent to obtain rewards with guidance signs in the first few steps.

2. **Rewards for Creating Aligned Sequences**

   To guide the agent toward the ultimate goal of winning the game, we provide incremental rewards for forming partial sequences:

Two-in-a-Row: A moderate reward is given for creating two consecutive aligned pieces, encouraging the agent to build upon existing sequences.

Three-in-a-Row: A higher reward is awarded for forming three aligned pieces, signaling proximity to a winning condition.

Four-in-a-Row (Diagonal Only): A substantial reward is granted upon achieving four aligned pieces on a diagonal, as this configuration constitutes a winning condition in Super Tic-Tac-Toe.

These rewards provide the agent with intermediate objectives, facilitating the learning of strategies that lead to victory.

3. **Penalties for Failing to Block Opponent Threats**

To instill defensive strategies, we penalize the agent for neglecting to counteract the opponent's potential winning moves:

Unblocked Two-in-a-Row by Opponent: A minor penalty is applied if the agent fails to block the opponent's formation of two aligned pieces in horizontal or vertical directions, where both ends are empty and valid, indicating a clear threat of forming a four-in-a-row.

Unblocked Three-in-a-Row by Opponent: A more significant penalty is imposed for not preventing the opponent from achieving three aligned pieces, which poses an immediate threat.

This component encourages the agent to monitor and respond to the opponent's actions, promoting a balanced approach that combines both offensive and defensive tactics.

4. Penalties for Occupying Dead-End Positions

To discourage the agent from placing pieces in positions with limited strategic value, we introduce negative rewards for dead-end moves. When the agent places a piece in a position surrounded by occupied or invalid squares, it receives a small penalty , which leads the agent to consider the broader impact of its moves and encourages it to make moves that are more beneficial to future strategic development.

2.3.3 Network Design

In PPO, the actor network is responsible for learning the policy — it decides which actions the agent should take given the current state. It takes the state as input and outputs a probability distribution over possible actions. The value network, on the other hand, evaluates the goodness

of a given state by predicting the expected future reward. It also takes the state as input, but outputs a single scalar value representing the estimated reward. The actor guides the agent's behavior, while the value network provides feedback to help improve the policy more effectively. Therefore, it is important to choose the right network structure for the problem.

Super Tic-Tac-Toe is a board game with structured space, and recognizing patterns such as configuration and arrangement of pieces is crucial to gameplay. Convolutional Neural Networks (CNNs) are particularly well suited for tasks involving spatial hierarchies and local dependencies because they can effectively capture and process spatial patterns in data, such as by using the same weights (filters) in different regions of the input, CNNs can detect the same features in different locations. Our Super Tic-Tac-Toe is a 12x12 grid, and then the player only needs to achieve 4 horizontal and vertical connections or 5 diagonal connections to win the game, which is mainly local recognition, and CNN is very suitable for this scenario. In board games, CNNs have been successfully applied to games such as Go and Four in a Row.

Our Actor network comprises three convolutional layers followed by a flattening layer that connects to the subsequent fully connected layer. A masking layer is applied before the final output to filter invalid actions. The Actor and Value networks share the convolutional base to encourage efficient feature extraction and parameter sharing.

In addition to the invalid action penalty mentioned above, we also added a mask mechanism to the network structure. For invalid actions, we reduce their output logits to an extremely low level. Under the categorical sampling mechanism, the probability of such invalid actions being sampled is greatly reduced. If the mask mechanism is not used, despite the penalty signal, the quality of the early experiences in the training will be greatly reduced due to too many invalid actions, making it difficult to learn truly effective game strategies. Therefore, the mask mechanism is necessary.

2.3.4 Training Strategy

In the initial phase of training, the agent competes against an opponent employing a **random policy**, wherein moves are selected uniformly at random from the set of valid actions. This approach facilitates the agent's understanding of fundamental game mechanics, such as legal move selection and basic pattern recognition, without the complexity introduced by strategic adversaries.

To monitor the agent's progress, we track its win rate against the random policy opponent. If the agent's win rate remains below 90% over an extended period, indicating a plateau in learning, we transition to a more challenging **mixed policy** opponent.

The mixed policy opponent comprises a combination of strategies designed to introduce varying levels of complexity:

- **20% Simple Rule-Based Strategy**: This component follows predefined heuristics, such as prioritizing moves adjacent to the agent's existing pieces, occupying central positions, and blocking the opponent's potential winning sequences.
- **80% Random Strategy**: Maintains an element of unpredictability by selecting moves randomly, ensuring the agent encounters a diverse range of scenarios.

This progression from random to mixed policy opponents allows the agent to incrementally adapt to more sophisticated adversaries, enhancing its strategic capabilities and robustness.

3. Implementation

The agent training pipeline is implemented using the TF-Agents library, which provides modular and extensible components for reinforcement learning in TensorFlow. TF-Agents allows for clear separation of environment logic, policy definition, training loops, and evaluation workflows.

3.1 Environment Integration

We defined a custom environment class `SuperTicTacToe` by subclassing `py_environment.PyEnvironment`, which conforms to TF-Agents' required API:

`observation_spec()`, `action_spec()` define the structure of inputs and actions.

`_step()` and `_reset()` control the transition logic and reward return.

The environment is then wrapped using `TFPyEnvironment`, allowing it to be compatible with TensorFlow graphs and batched execution.

3.2 Policy and Network Wrapping

The actor and value networks are implemented as TensorFlow `tf.keras.Model` subclasses and passed to TF-Agents via the standard interfaces. These models are used by the `PPOAgent`, which wraps policy construction, loss computation, and training logic internally.

The actor policy returns a `Categorical` distribution, with a custom masking operation applied to ensure that only legal actions are sampled. The value network estimates the state value for the critic.

3.3 Experience Collection and Replay

Experience is collected using `TF-Agents.TFUniformReplayBuffer` with rollouts from 8 parallel environments. In each step, the agent acts first, followed by the opponent. However, the environment only returns the reward corresponding to the agent's action, ensuring that all learning signals are directly tied to the agent's decisions.

Transitions are stored using `from_transition(...)`, and PPO updates are performed after gathering a full batch of experiences. This setup maintains consistency with on-policy learning and allows for efficient batch training across parallel rollouts.

3.4 Training Loop and Scheduling

The entire training process is implemented using a simple Python loop with `tqdm` progress bars for visualization. Within each epoch, the agent collects rollouts, performs PPO updates, and undergoes evaluation against a fixed opponent.

We also implement detailed logging to monitor key training metrics and ensure training stability. During each epoch, the following metrics are recorded and printed:

- **Policy Loss**: Indicates how much the policy network is adjusting. This helps evaluate the direction and magnitude of policy updates.
- **Value Loss**: Measures how accurately the critic estimates returns. A decreasing value loss typically signals improved value function approximation.
- **Entropy**: Represents the uncertainty in the agent's action selection. High entropy promotes exploration; low entropy suggests the policy is converging toward exploitation.
- **Win Rate**: Computed over 10 evaluation episodes using the current policy against a rule-based opponent, this provides a high-level indicator of the agent's strategic competence.

These metrics are printed to the console and can optionally be recorded for later analysis. Combined with periodic checkpointing, this allows for both qualitative and quantitative tracking of training progress.

4. Results and Analysis

To explore a tractable solution space and accelerate policy convergence, we initially conducted training in a simplified version of the full environment. While the official project specification introduces two challenging mechanics.

- victory is achieved by forming four aligned pieces horizontally or vertically, but five diagonally
- there is a 50% chance that a move fails to land on the intended cell, and if the randomly selected fallback cell is invalid, the agent forfeits its turn—we relaxed both constraints to facilitate early experimentation and model prototyping.

Specifically, our modified environment differs in two key ways:

- Diagonal Victory Condition: A win is declared if four aligned pieces are formed in any direction—including diagonals—rather than requiring five in the diagonal direction.
- Fallback Placement Mechanism: When a move fails to land on the selected cell (with 50% probability), it is redirected to a randomly chosen valid adjacent cell only. If no valid adjacent cell exists, the move is simply skipped in the project setting, but in our simplified environment, we ensure that the fallback always occurs on a valid cell, thus forfeits never occur.

These simplifications reduce the stochasticity and sparsity of the learning signals in early training, allowing us to better isolate the effects of architectural and reward design choices. All experimental results reported in the following sections are based on this simplified environment.

4.1 Parameter Settings

To ensure efficient training in a stochastic and partially sparse-reward environment, we adopted a carefully tuned set of hyperparameters (see Figure 5). The training process spanned **1,000 epochs**, using **8 parallel environments**, each collecting **64 steps per epoch**, resulting in a batch size of 512 for policy updates. The agent was evaluated over **80 episodes per epoch**, with model checkpoints saved every 200 epochs.

Our final hyper-parameter are as follows：

Parameter	Value	Description
`EPOCHS`	1000	Total training epochs
`PRETRAIN_EPOCHS`	20	Epochs using only random opponents before switching to mixed

Parameter	Value	Description
NUM_ENVS	8	Number of parallel environments
COLLECT_PER_ENV	64	Number of steps collected per environment per epoch
EVAL_EPIS	80	Evaluation episodes per epoch
SAVE_EVERY	200	Checkpoint saving interval (epochs)
TARGET_WIN	0.90	Target win rate threshold for strategy switch
Optimization Parameters	Value	Description
LR_BASE	1e-4	Initial learning rate (cosine decay)
ENTROPY_INIT	0.2	Initial entropy coefficient for exploration
MIN_ENT_COEF	0.05	Minimum entropy coefficient after decay
CLIP_RATIO	0.30	PPO clip ratio to limit policy update size
NUM_PPO_EPOCH	5	PPO update iterations per epoch
VALUE_COEF	1.5	Weight for value function loss in total loss
GAE_LAMBDA	0.80	GAE lambda for advantage estimation smoothing

4.2 Results

4.2.1 Training Results and Analysis

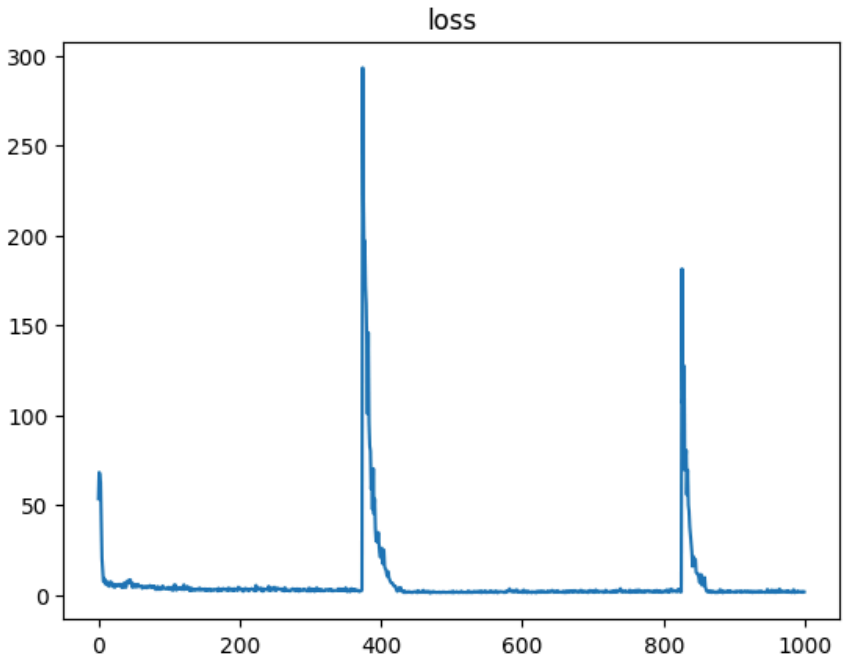


Figure 2: Training loss

The training loss curve in Figure 2 illustrates stable convergence behavior of the PPO agent across 1,000 epochs. The loss rapidly declines within the first 20–30 epochs, suggesting the agent quickly learns basic rules and achieves early success—prompting an early switch to the mixed-strategy opponent.

Two sharp spikes in loss near epochs 380 and 800 are not due to opponent changes but likely stem from rare learning instabilities, such as encountering atypical states or entropy schedule shifts. In both cases, the loss quickly recovers within 20–30 epochs, demonstrating the robustness of PPO’s clipped objective and learning rate schedule in stabilizing training under stochastic dynamics.

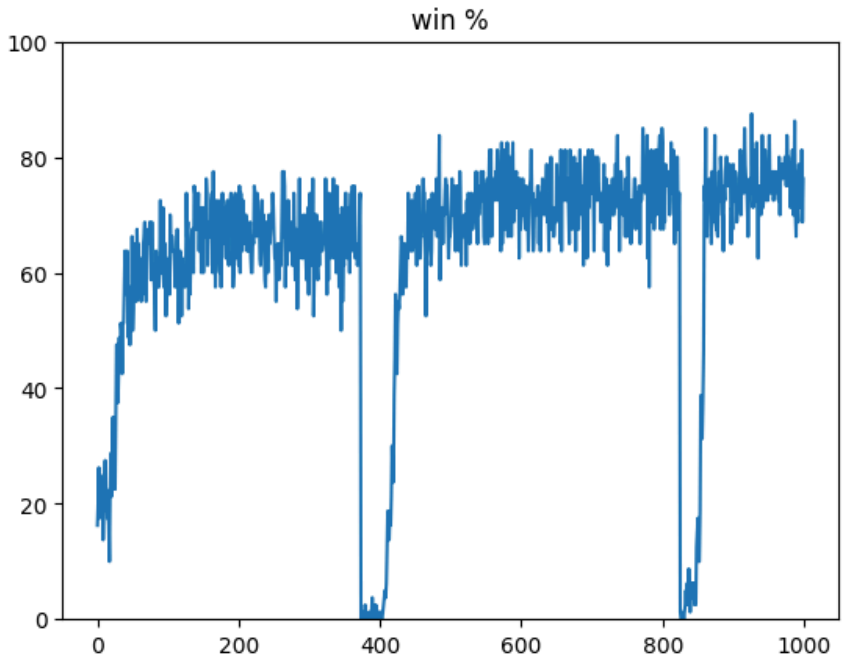
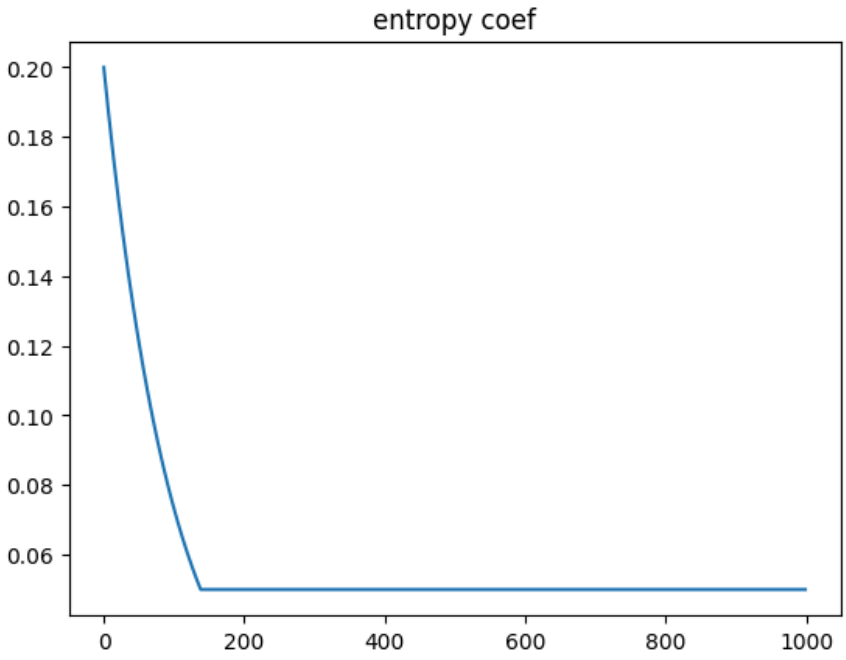


Figure 3: Win rate over 10 evaluations

The win rate curves in Figure 3 reflect the evolution of the agent's strategy strength during training. In the initial stage (epochs 0-20), the win rate quickly rises from below 30% to above 60%, indicating that the agent is able to quickly adapt to the basic game mechanics against random opponents. This performance improvement quickly reaches the preset switching threshold, leading to an early transition to a more challenging mixed-strategy opponent.

After the opponent changes (around the 20th-25th epoch), the win rate drops slightly as the agent begins to face complex random and rule-based behaviors. However, the agent shows strong adaptability: the win rate steadily recovers and reaches more than 65% in the 200th epoch. When the loss changes suddenly, the win rate also changes suddenly, but recovers quickly. Overall, the agent's performance has remained at a high level, and the final win rate stabilizes at around 75% to 80%.

This progress shows that evolutionary opponent design, coupled with intensive reward shaping, can effectively promote rapid skill acquisition and continuous improvement in complex strategic scenarios.



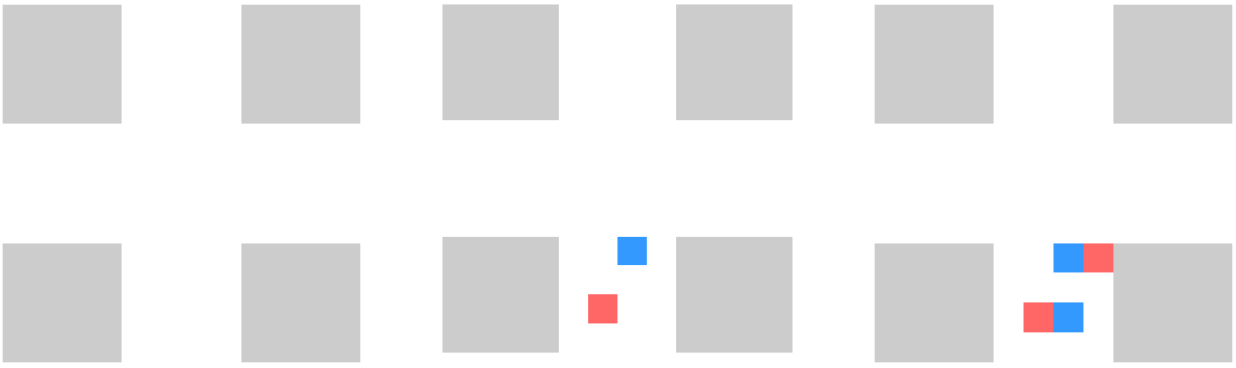
The entropy coefficient table (Figure 3) gradually decays from 0.2 to 0.05, which is consistent with our program settings. In the first 100-150 epochs, the entropy value remains relatively high, which coincides with the stage of a sharp increase in the win rate. As training progresses, the entropy gradually decays, and the win rate reaches about 60%, and the agent transitions to an exploitation-dominated strategy mode, focusing on improving and stabilizing previously learned strategies. Importantly, there is no sharp drop in the win rate during this transition, which indicates that the agent effectively uses the exploration phase to build a robust and transferable policy foundation.

Overall, the entropy regularization strategy proves to be effective: it promotes early policy discovery and late policy consolidation, thereby helping to achieve a smooth and consistent performance trajectory.

4.2.2 Simulation Results and Analysis

The simulation below illustrates a match between the PPO-trained agent (blue) and a rule-based opponent (red) after 1000 training epochs. The agent demonstrates clear strategic behavior by gradually building aligned formations from the early steps, prioritizing central and connected positions. As the game progresses, it consistently places pieces around existing ones to form potential win paths, rather than scattering moves randomly. The agent avoids isolated or dead-end placements and maintains offensive momentum, even when facing defensive responses from the opponent. By Step 8, it is close to completing a four-in-a-row, indicating effective long-term

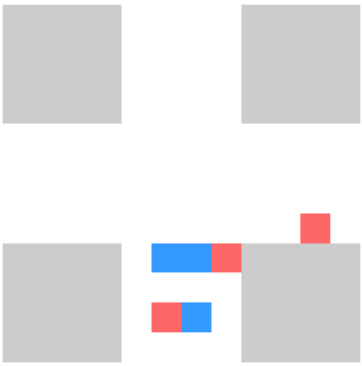
planning. Overall, the agent exhibits coherent and purposeful play, confirming that it has learned strong spatial reasoning and alignment strategies through reinforcement learning.



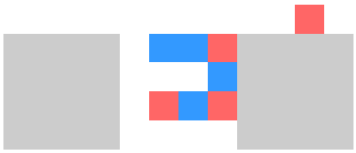
Initial

Step 1

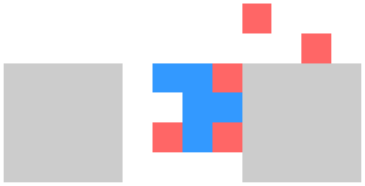
Step 2



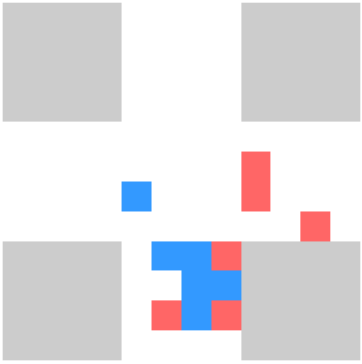
Step 3



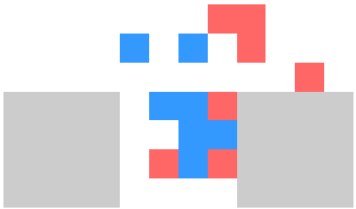
Step 4



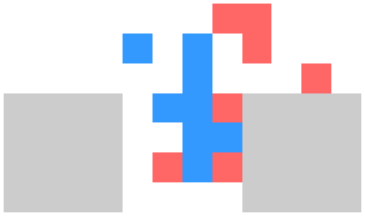
Step 5



Step 6



Step 7



Step 8

5. Conclusion

This project demonstrates the successful application of reinforcement learning based on the probabilistic optimization algorithm (PPO) to a spatially complex and probabilistic Tic-Tac-Toe variant. To promote efficient learning and strategy formulation, training is performed in a simplified environment that relaxes two constraints of the original task.
Our work includes:

- Building a robust training environment that combines probabilistic transformations and dense reward shaping to alleviate sparse feedback and guide agent behavior
- Constructing a CNN-based Actor-Critic architecture to extract spatial features from the cross-shaped board
- Incorporating a legal action masking mechanism to prevent invalid moves and stabilize policy optimization
- Using a curriculum-inspired opponent strategy to transition from random behavior to hybrid rule-based behavior to promote gradual and meaningful learning
  Experimental results show that the agent can quickly learn to master basic rules and effectively adapt to more strategic opponents, ultimately achieving a win rate of about **75% to 80%**. The results verify that the combination of structured state encoding, adaptive exploration, and reward shaping can effectively accelerate learning in partially random, sparse reward environments.