PPO For Santorini

Nicholas Livingstone
Computer Science
University of New Mexico
nlivingsto@unm.edu

I. Introduction

Reinforcement learning has achieved remarkable success in complex games like Go and Chess, but many strategic board games remain challenging testbeds for evaluating deep RL algorithms [1], [2]. This report focuses on Santorini, a two-player abstract strategy game where players move workers around a grid and build towers to reach the winning condition of standing on a level-3 tower. The game's combination of movement and building mechanics creates a large action space while maintaining relatively simple rules, making it an interesting environment for studying policy gradient methods.

We implement Proximal Policy Optimization (PPO), a widely-used policy gradient algorithm that balances sample efficiency with training stability through a clipped surrogate objective [3]. PPO has proven effective in both continuous and discrete action spaces, making it well-suited for Santorini's 144 discrete actions. We develop a custom Gymnasium environment for Santorini and train agents against a fixed NegaMax-based opponent with configurable search depth. This setup allows us to evaluate PPO's performance in an adversarial setting while maintaining consistency across experiments.

To understand how hyperparameter choices affect learning in this adversarial environment, we compare three agent configurations that vary key PPO parameters: learning rate, entropy coefficients, clip range, and training duration. The baseline configuration uses standard hyperparameters for stable learning, the explorative configuration emphasizes strategic diversity through high entropy bonuses, and the aggressive configuration prioritizes rapid convergence through minimal exploration. We analyze these configurations not only through reward metrics but also through auxiliary measurements like policy loss and valid action distributions to understand what strategies emerge during training.

II. Santorini Overview

Santorini is an abstract strategy board game originally developed by Dr. Gordon Hamilton in 2003. For this paper, we consider the 2016 version published by Roxley Games [4]. In Santorini, players each control two workers on a 5x5 orthogonal grid. On a players turn, they first select a worker to move into an adjacent space, then construct a building level adjacent to the moved worker. If a worker is moved to the third level, the player wins. A player can also lose if it gets blocked, meaning it has no available turn available to take.

Fig. 1. Example of the Santorini Board Game

III. Framing Santorini as an RL Problem

In this section, we detail the representation of Santorini as a Reinforcement Learning problem, including the action space, observation space, and the reward strategy.

A. Action Space

We define the actions space as a set of 144 available actions as defined in (1) below.

$$\begin{aligned} |\mathcal{A}| &= N_{\text{workers}} \times N_{\text{move}} \times N_{\text{build}} \\ &\quad +(N_{\text{workes}} \times N_{\text{build}}) \\ &= 2 \times 8 \times 8 + 16 = 144 \end{aligned} \tag{1}$$

where N_{move} represents 8 adjacent squares for movement, and N_{build} represents 8 adjacent squares for building after the worker moves. For simplicity of implementation we use a single dimension action space, this becomes helpful later when we utilize action masking. Each action represents a selected worker, the intended move direction, and the build location. The second term in (1) represents actions for selecting a worker and moving it without a build; this represents game-ending actions in which moving a worker would win the game and no build is necessary.

B. Observation Space

For observation, we provide the agent with knowledge of three features: the board state (buildings), location of the agent's workers, and the location of the opponent's workers.

$$\begin{aligned} \text{board} &\in \{0,1,2,3,4\}^{5\times5} \text{ (tower heights)} \\ \text{workers} &\in \{0,1,2,3,4\}^{2\times2} \text{ (player x, y positions)} \\ \text{opponents} &\in \{0,1,2,3,4\}^{2\times2} \text{ (opponent x, y positions)} \end{aligned} \tag{2}$$

$$\begin{aligned}\dim(\mathcal{S}) &= \dim(\text{board}) + \dim(\text{workers}) \\ &\quad + \dim(\text{opponents}) \\ &= 25 + 4 + 4 = 33\end{aligned} \tag{3}$$

For clarity, we emphasize that in (2) the board represents a vector of building heights at each board location, where as workers and opponents are each vectors containing x, y positions for each worker.

C. Reward Strategy

We define the reward function as the following:

$$r_t = \begin{cases} +100 \text{ if agent wins} \\ -100 \text{ if agent loses} \\ r_{\text{turn}} \quad \text{otherwise} \end{cases} \tag{4}$$

Where r_{turn} is defined as:

$$\begin{aligned} r_{\text{turn}} &= r_{\text{step}} + r_{\text{climb}} + r_{\text{height}} + r_{\text{dome}} + r_{\text{block}} \\ r_{\text{step}} &= -0.1 \\ r_{\text{climb}} &= 0.5 h_{\text{new}} \dot{-} h_{\text{old}} \\ r_{\text{height}} &= 0.1 \cdot h_{\text{new}} \\ r_{\text{dome}} &= \begin{cases} 0.2 \text{ if building dome (level 4)} \\ 0 \quad \text{otherwise} \end{cases} \\ r_{\text{block}} &= \begin{cases} 0.2 \text{ if building adjacent to opponent} \\ 0 \quad \text{otherwise} \end{cases} \end{aligned} \tag{5}$$

We proceed by describing each term of (5). At a given turn where no winner has been determined, the agent will receive a combination of sub-rewards based on it's current action. We define the r_{step} as a negative reward for continuing to play, to motivate a faster win. r_{climb} gives a scaled reward based on if a worker was moved to a higher position. A scaled reward is given r_{height} based on the the current height of the worker, to encourage it to stay at a higher position. The term r_{dome} provides a small reward to encourage the agent to build domes which block the opponent from winning, and similarly provide a reward if we build in at a position where the opponent could have built on their proceeding turn in r_{block}.

We did initially consider providing a more sparse reward structure so as to allow the agent more 'independence'. This variant modified (4) to just return the r_{step} in the otherwise case, but found the returns to not be sufficient enough for learning. Further study on variations of reward strategy would be beneficial, but as a baseline we consider only this current one.

D. Opponent

In this section, we detail the selection of the opponent used for agent training.

a) *Self-Play:*

In the first pass of this project, we attempted to use a naive self-play approach whereby the agent would alternate between the roles of both players, directly playing against itself. Preliminary results of this method were not promising, as the agent failed to learn effectively. For example, the reward strategy was fundamentally flawed: when the agent simultaneously experienced both a win and a loss, the opposing rewards canceled out, producing a net reward of zero. Additionally, during analysis it was unclear whether the model was actually learning, since we had no consistent baseline to evaluate against. Although more sophisticated self-play methods exist in the literature, we abandoned this approach and instead utilized a stronger fixed opponent, as described in the next section.

b) *NegaMax + NNUE AI:* Instead of self-play, we opted for an externally developed *Santorini-AI* which leverages NegaMax and NNUE [5], [6]. The AI is available as a standalone universal chess interface(UCI)-like engine. During training, the Santorini environment communicates with the engine and provides game state information. In turn, it receives the best move computed within an allotted 'thinking' time given in seconds. The developers consider 0.5s, 1s, and 5s to be easy, medium, and hard AI difficulties respectively. For training, we utilize a fixed time frame of 0.1s. This was chosen for two reasons:

1) Since the agent is learning from scratch, we considered that an easier an opponent would produce easier learning.
2) Longer thinking times produced significantly longer training times, since this time increases the length of *every* environment step.

This AI also provides an added benefit of being deterministic, meaning given the same game state and thinking time, it will produce the same best move.

While neural networks are considered to be universal function approximators, raising concerns that our agent may overfit to its specific training opponent rather than learn generalizable Santorini strategy, we believe this fixed-opponent approach provides a sufficient baseline for analyzing PPO's effectiveness in adversarial settings.

IV. Methodology

A. Agent Configurations

We evaluate a Proximal Policy Optimization (PPO) based agent using three different training configurations of the agent: baseline, explorative, and aggressive.

TABLE I
Agent Configurations

	Baseline	**Explorative**	**Aggressive**
Learning Rate (α)	0.0003	0.0002	0.0005
PPO Clip Range (ε)	0.2	0.25	0.15
Entropy Coefficient (β)	0.001	0.01	0.0001
Frames per Batch	1024		
Batch Size	64	128	128
Num Epochs	5	8	10
Discount Factor (γ)	0.99	0.995	0.99
GAE Lambda (λ)	0.95	0.98	0.95
Total Frames	51200		

TABLE II
SUMMARY STATS OF CONFIGURATIONS AT END OF TRAINING.

Configuration	Mean Reward		Policy Loss		Entropy Loss	
	mean	std	mean	std	mean	std
Aggressive	6.42	0.246	0.0888	0.176	-8.42×10^{-05}	6.03×10^{-06}
Baseline	5.93	2.81	0.0125	0.197	-0.00170	0.000910
Explorative	3.56	0.357	0.0198	0.128	-0.0286	0.000376

Baseline employs standard hyperparameters for stable learning ($\alpha = 0.0003$, $\beta = 0.001$), **Exploration** emphasizes strategic diversity through high entropy bonuses ($\beta = 0.01$) and long-term reward weighting ($\gamma = 0.995$), and **Aggressive** prioritizes rapid convergence via minimal exploration ($\beta = 0.0001$) and higher learning rates ($\alpha = 0.0005$). This design allows us to analyze PPO's behavior across exploitation-exploration tradeoffs while maintaining consistent data collection parameters (1,024 frames per batch, 51,200 total frames).

B. Agent Details

For the agent architecture, we implement a fully connected neural network with two hidden layers for both Actor and Value network as shown in Fig. 2 and Fig. 3

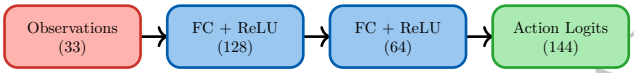

Fig. 2. Actor network architecture for PPO policy

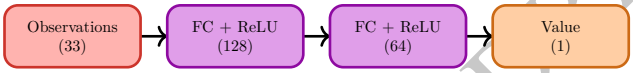

Fig. 3. Value network architecture for PPO critic

The actor network's output was also masked with available actions for the agent to specify actions which actions were valid per the games rules. We utilized a 32 vCPU system for training, there was no GPU utilized as the training process was primarily CPU bound. Training processes were terminated early if they showed no sign of improvement. For analysis, we conduct three runs of each agent configuration.

C. Additional Technical Information

- The RL Environment was implemented using a custom gymnasium environment [7].
- The models were implemented using PyTorch and TorchRL [8].
- The package management tool Pixi was used during development [9].

V. RESULTS

The full results of running multiple configurations is shown in Table II.

A. Rewards and Policy

Overall, we find the aggressive configuration to be the best performing out of the three, achieving a max mean reward of 6.42. The baseline configuration was able to achieve a similar average reward of 5.93, but exhibited a much great variance (2.81) than the aggressive configuration (0.246). The explorative model performed the worst achieving a final mean reward of 3.56. We can more clearly see this trend in Fig. 4, where the mean reward for the aggressive and baseline configurations exhibit trend towards increasing rewards, but the explorative configuration plateaus at around an average of 4. Additionally there is a clear noticeable difference in variance between the baseline and aggressive configurations.

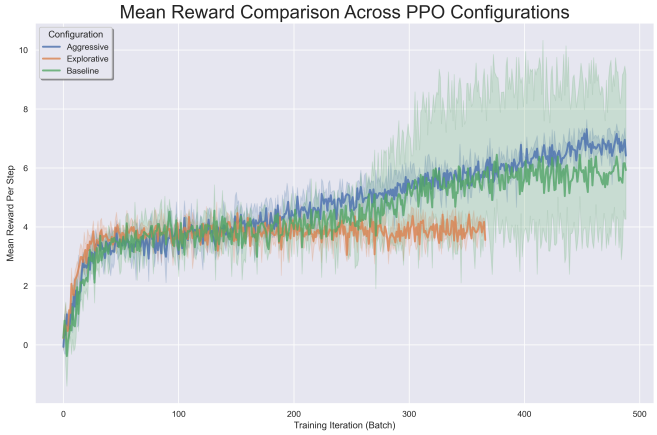

Fig. 4. Mean reward comparison across different configurations.

Next, we note that that the policy loss between the models did not very as significantly as the average reward. Since the policy loss measures the magnitude of policy changes, we can presume that each configuration is making similarly-sized changes in the policy at each update step. This is most likely a result of the PPO architecture keeping policy changes stable regardless of configuration. Taking into account the previous discussion on average rewards, we can observe that similar policy loss, but differing rewards can likely be reflected as a difference in exploration-vs-exploration. To state it clearly, since the aggressive configuration (low entropy, high learning rate) produces the best average reward, we can conclude that it's efficiently exploiting *known good actions*, where as the explorative (high entropy, low learning rate) configuration might be preventing convergence or is having greater difficulty exploiting known good strategies. The baseline is then the clear middle ground between the two.

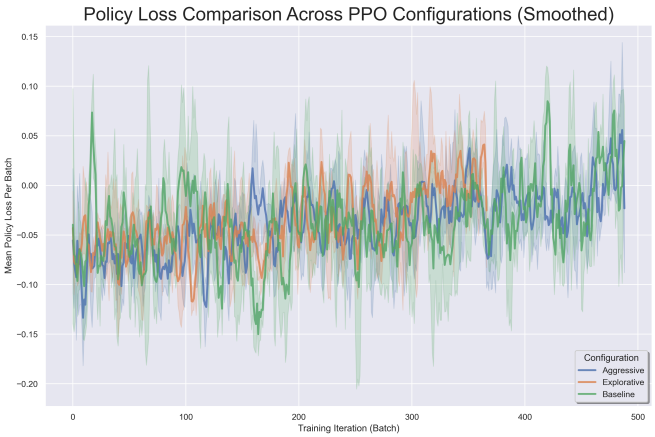

Fig. 5. Policy Loss Comparison across different agent configurations. Results have been smoothed for clarity with a 5-iteration average.

B. Action Analysis

Finally, we analyze the number of valid actions available to the agent during training. We tracked the average and minimum valid actions per batch, initially as a sanity check for correct environment implementation, but this metric revealed unexpected insights into the learning dynamics.

Fig. 6 reveals a notable pattern: while average valid actions remained stable across configurations, the minimum valid actions diverged substantially. The baseline agent increasingly encountered states with more available actions, whereas the aggressive and explorative agents more frequently reached constrained board states with fewer legal moves. We hypothesize that the aggressive and explorative configurations learn strategies that rapidly drive toward terminal game states, resulting in positions with limited move choices. In contrast, the baseline configuration, occupying a middle ground on the exploration-exploitation spectrum, appears to favor actions that prolong gameplay, effectively ceding control over game tempo to the opponent. Further analysis would be necessary to fully validate these interpretations.

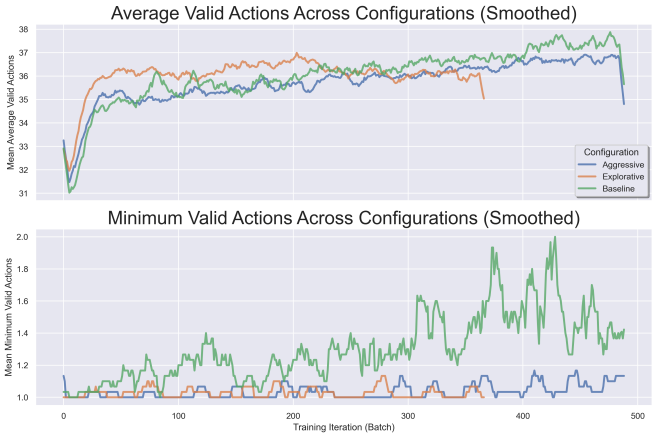

Fig. 6. Valid actions across different agent configurations. The top figure shows the average number of actions available per batch. The bottom figure shows the minimum number of valid actions available per batch. Both figures have been smoothed for clarity with the 10 iteration average.

VI. Conclusion

In this work, we developed a custom Gymnasium environment for the strategic board game Santorini and trained PPO agents against a NegaMax-based opponent. Through comparative analysis of three agent configurations, baseline, explorative, and aggressive, we demonstrated that hyperparameter choices significantly impact both learning dynamics and final performance. Our results show that exploration-exploitation balance, controlled through entropy coefficients and learning rates, affects not only achieved rewards but also the strategic patterns agents develop, as evidenced by divergent behaviors in constrained game states.

VII. Future Work

Several directions exist for extending this work. First, curriculum learning could progressively increase opponent difficulty throughout training, helping agents develop robust strategies against stronger opponents. Second, Santorini's board has rotational and reflective symmetries that could be leveraged through data augmentation, applying these transformations to training data would effectively increase the dataset size and potentially improve learning efficiency. Finally, incorporating human gameplay data through imitation learning could help agents discover sophisticated strategies that might not emerge from training against a fixed opponent alone.

Beyond these methodological improvements, deeper analysis of agent behavior would provide valuable insights into learned strategies. Future work should track win rates against opponents of varying strengths to better assess whether agents generalize beyond their training opponent. Analyzing learned policies through positional heatmaps would reveal which board positions agents prefer for worker placement and building decisions. Additionally, examining critical game states, such as when one move away from winning or losing, would show how risk-averse or aggressive the learned policies are. Comparing action value distributions across configurations could also explain performance differences and reveal whether successful agents simply make more confident decisions or truly identify objectively stronger moves. This expanded analysis would move beyond aggregate reward metrics to understand what strategies agents actually learn and why certain configurations succeed.

VIII. AI Acknowledgement

This project utilized Anthropic's Claude Sonnet 4.5 in the following capacities:

- Grammar improvements.
- Adjustment of figures and report formatting.
- Aided in python development of custom gym environment and integration with Santorini-AI opponent.

References

[1] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] D. Silver *et al.*, “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm,” *CoRR*, 2017, [Online]. Available: http://arxiv.org/abs/1712.01815
[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” *CoRR*, 2017, [Online]. Available: http://arxiv.org/abs/1707.06347
[4] G. Hamilton, “Santorini.” Accessed: Dec. 09, 2025. [Online]. Available: https://roxley.com/products/santorini
[5] J. Pricey, “santorini-ai: A Santorini Game Engine and AI.” [Online]. Available: https://github.com/JPricey/santorini-ai
[6] George T. Heineman Gary Pollice and S. Selkow, “Path Finding in AI,” *Algorithms in a Nutshell*. O'Reilly Media, pp. 213–217, 2008.
[7] M. Towers *et al.*, “Gymnasium: A Standard Interface for Reinforcement Learning Environments.” [Online]. Available: https://github.com/Farama-Foundation/Gymnasium
[8] J. Ansel *et al.*, “PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation,” in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*, ACM, Apr. 2024. doi: 10.1145/3620665.3640366.
[9] R. Arts, B. Zalmstra, W. Vollprecht, T. de Jager, N. Morcotilo, and J. Hofer, “pixi.” [Online]. Available: https://github.com/prefix-dev/pixi/releases/tag/v0.55.0