OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

School of Computer Science

The Effect of Historical Checkpoints on a Real Time Strategy Game Agent Training

Caellum Kennedy

Student ID: 23108230

Supervisor: _____Patrick Mannion_____

Submitted in partial fulfilment of the requirements for the award of B.Sc. (Computer Science and Information Technology)

4 April 2026

Table of Contents

Abstract

This project investigates the effect of historical self-play on reinforcement learning agent training in MicroRTS, a real-time strategy game. In standard self-play, agents develop effective strategies but overwrite them through continued training, causing performance collapses. Historical self-play forces the agent to compete against frozen copies of its own prior checkpoints. Over 30 million training steps the effect was indistinguishable from evaluation noise, but over 80 million steps historical self-play prevented the prolonged performance collapses observed in standard self-play, maintaining consistently higher TrueSkill ratings in the second half of training.

Keywords: reinforcement learning, self-play, PPO, TrueSkill, MicroRTS

Acknowledgements

I would like to thank my supervisor, Patrick Mannion, for guidance and feedback throughout this project.

1 Introduction

1.1 Background and motivation

This project was conducted to train a Real Time Strategy agent to approach the performance of top competition level bots. During preliminary experimentation it was observed that the agent would develop effective strategies but lose them through continued self-play training, repeatedly collapsing from competition level performance back to the baseline. Investigating the cause of these collapses and whether historical self-play could prevent them became the focus of the project.

The RTS game chosen was MicroRTS. MicroRTS is an implementation of an RTS game which has been designed to perform AI research. The advantages of MicroRTS over a traditional full-fledged game such as StarCraft or Age of Empires is that MicroRTS is much simpler, it comes with prebuilt support for AI meaning it does not require workarounds to implement machine learning within it. It also has a limited action space compared to StarCraft and Age of Empires allowing for much shorter training times and quicker convergence.

1.2 MicroRTS

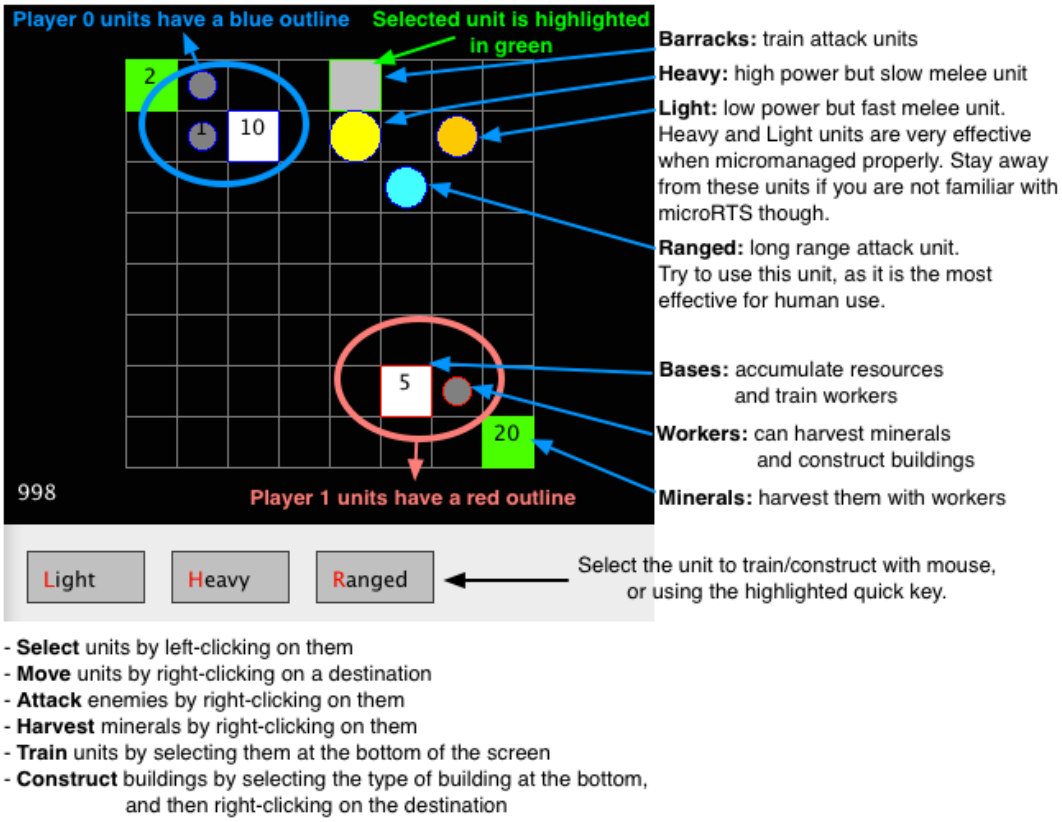

Figure 1: https://github.com/Farama-Foundation/MicroRTS explanation of how MicroRTS is played from the MicroRTS

MicroRTS is a deterministic real time strategy game. It ran competitions between 2017 and 2024. For the evaluation of the agent during training the competition rules

were chosen. The only difference is that this agent was only trained on a singular map and only evaluated on said map.

The rules for the competition that are relevant to this experiment are as follows:

- Game length: Games will run up to a fixed number of steps (after which the game will be considered a tie). The maximum number of steps for a 16x16 map will be: 4000 + 1000 (anti-draw time gap) = 5000 max time steps.
- Each bot will be given a computation budget of 100 milliseconds per game step.

Due to the strict 100 milliseconds per game step rule any intensive tree search algorithms will not function as well as in other applications. (e.g. AlphaGo Zero was given 400 milliseconds for a Montecarlo Tree Search on each move). As a result, implementing a tree search before each game step was not investigated.

1.3 Problem statement

What are the effects of implementing historical checkpoints on a real time strategy agent's training?

1.4 Objectives and contributions

The objectives of this report are:

- Show how the implementation of historical checkpoints affected the RTS agent's training.
- Reduce self-play instability in self-play only training configurations.

2 Related Work

2.1 Other studies related to RTS game training

The main inspiration for implementing historical checkpoints was the research conducted by AlphaGo Zero. In their experiments they only had a single reward, the win reward, but they trained their agent against the strongest previous checkpoints. This has been highlighted as one of the main contributors to allowing AlphaGo Zero to develop the wide range of skills that it did. [1] Other machine learning and deep learning research papers were also investigated to see what the most up to date research stated on the topic. [2] [3] AlphaStar maintained a league of past agents and trained agents against them while OpenAI Five used a pool of past versions weighted by recency.

For this project a win-weighted historical pool was maintained as opposed to the recency-weighted pool used by OpenAI Five. This was done to train the agent against the previous checkpoints it struggles with the most providing a strong previously effective strategy from which it trains against.

2.2 Previous students' MicroRTS work

Previous students at University of Galway conducted research on MicroRTS. Their research was used as a baseline for comparison during the preliminary exploration phase of the experiment. Their research was focused on concepts which were later implemented into the MicroRTS repo (e.g. invalid action masking, diversified training opponents) [4] [McKiernan 2022] and [5] [Dwivedi 2023]

3 Methodology and System Design

This chapter describes the MicroRTS-Py training system, the historical self-play mechanism that was implemented and the experimental design used to evaluate its effectiveness. The code was maintained on a GitHub repo. [7]

3.1 System Design

The training used Proximal Policy Optimisation (PPO), a policy gradient reinforcement learning algorithm [6]. The system runs multiple games in parallel across 24 self-play environments. In each self-play environment pair both Player 1 and Player 2 are controlled by the same neural network policy. The map is rotated 180 degrees for P2 so that both players see a consistent perspective.

At each step of the game the agent receives the current state of the 16x16 grid as a 27-channel observation encoding unit positions, health, ownership, type and current action. The agent's neural network processes the observation through a convolutional encoder which compresses the spatial information into a 256-dimensional representation. This representation feeds two separate heads: an actor which outputs action probabilities for every cell on the map and a critic which estimates the value of the current state as a single number.

An invalid action mask provided by the game engine ensures that the agent can only select legal actions. For example, an empty cell cannot issue a move command. Invalid actions are assigned a probability of zero through masking.

The agent collects 256 steps of experience across all 24 environments before performing a policy update. This gives a batch of 6,144 transitions per update. Advantages are computed using Generalised Advantage Estimation (GAE) with a discount factor of 0.99 and a decay parameter of 0.95 (default MicroRTS values). The PPO update uses a clipped surrogate objective with a clip coefficient of 0.1, performed over 4 epochs with 4 minibatches per epoch. The learning rate is annealed linearly from 2.5^{-4} to zero over the course of training.

3.2 Reward Structure

Six reward functions run in parallel every game step, each implemented in Java within the MicroRTS engine. The raw reward from each step is a six-element vector, one value per function. These are combined into a single scalar reward using a weighted sum.

The default reward weights are:

Index	Reward Function	Weight	Signal Type
0	Win/Loss	10.0	Terminal
1	Resource Gather	1.0	Per-step
2	Produce Worker	1.0	Per-step
3	Produce Building	0.2	Per-step
4	Attack	1.0	Per-step
5	Produce Combat Unit	4.0	Per-step

Table 1: Default reward weights.

The Win/Loss reward fires only at the end of a game, giving +1 for a win and -1 for a loss. For a draw it gives 0. All other reward functions fire every step, providing a dense signal throughout the game. The per-step rewards shape the agent's behaviour during the game whereas the terminal win/loss reward provides the overall objective.

3.3 Evaluation

During training a checkpoint of the agent's neural network weights is saved at regular intervals. Each checkpoint is evaluated asynchronously against the league of 13 scripted bots using the TrueSkill rating system.

TrueSkill is a Bayesian rating system that models each player's skill as a Gaussian distribution with mean (mu) representing estimated skill and sigma representing uncertainty. The conservative rating, mu minus three sigma is used as the single TrueSkill number. A new checkpoint starts with a mu of 25 and sigma of 8.33. It is matched against bots of similar estimated skill and plays games until sigma drops below 1.2, at which point the rating is considered converged.

Different values of sigma were tested, above 1.2 the variability of TrueSkill was too much when evaluating the exact same checkpoints whereas below 1.2 the evaluation script takes exponentially longer to converge. The NaiveMCTSAI bot is computationally the slowest bot and is faced multiple times by any mid-tier agent.

Bot Name	mu	sigma	TrueSkill
CoacAI	41.49	1.18	37.94
Droplet	37.67	1.06	34.49
Izanagi	35.97	0.93	33.19
WorkerRushAI	33.19	0.91	30.46
MixedBot	31.47	0.99	28.50
Tiamat	29.25	0.86	26.67
GuidedRojoA3N	27.24	0.92	24.48
LightRushAI	26.40	1.03	23.33
NaiveMCTSAI	22.44	0.91	19.70
Rojo	17.86	0.94	15.05
RandomBiasedAI	16.46	1.06	13.29
RandomAI	7.39	1.52	2.84
PassiveAI	3.70	1.75	-1.55

Table 2: TrueSkill table of the scripted bots.

3.4 Self-Play Instability

In standard self-play both sides of every game are controlled by the same policy. As the agent improves its opponent improves at the same rate since they share the same weights. This creates a training dynamic where the agent can develop an effective strategy, then through continued self-play discover a counter strategy which replaces the original. However, these counter strategies are specialised against the agent's own

play and often perform poorly against the strategies used by the evaluation bots causing the TrueSkill to drop.

The core issue with self-play is that once a strong policy is overwritten the agent has no mechanism to recover it. In pure self-play both copies of the policy degrade together. The agent never faces its previously strong checkpoints so there is no training pressure to maintain effective strategies.

This phenomenon was observed during preliminary experimentation where the agent's TrueSkill would spike above 30 (beating WorkerRush) and then collapse back below 15 (Losing to Rojo). The agent was observed in matches against the scripted bots to see where it was faltering, it was observed that the agent hadn't learned how to micromanage units, e.g. use a ranged unit to dodge enemy melee units and shoot. The agent which reached the highest TrueSkill relied on a single worker at the start of the game sent out to destroy the enemy base, if this first attack on the base failed then the self-play agent died but if it succeeded it was able to beat even the strongest scripted bot, CoacAI.

3.5 Historical Self-Play

Historical self-play addresses this problem by forcing the current agent to play against a previous checkpoint of itself. The frozen opponent cannot adapt its strategy so the agent must adapt to beat it.

The implementation designates a set of self-play environment pairs as historical pairs. In these pairs the P2 actions are overridden by the historical checkpoint. The current agent controls P1 and trains from that experience.

The advantages for the historical P2 environments are zeroed because the actions in those environments were selected by the frozen historical agent, not the agent in training. The stored action probabilities therefore do not correspond to the policy that chose the actions, which would produce incorrect gradient estimates.

Every time a checkpoint is saved during training it is added to a pool of candidate historical opponents. Each checkpoint has its games against the agent tracked. When the pool exceeds its maximum size the easiest opponent, the one which the agent has the highest win rate against, is evicted. This keeps the pool stocked with opponents that are either new, or the current agent cannot beat. The next historical checkpoint is then chosen; each checkpoint is weighted by their win rate against the agent. A checkpoint which has beaten the agent in 9/10 matches gets a 0.9 weighting, whereas an agent which only won 1/10 gets a 0.1 weighting. If after the historical checkpoint has played 50 matches against the agent and the agent has won 95% or more of those matches then the agent is evicted.

3.6 Process and workflow

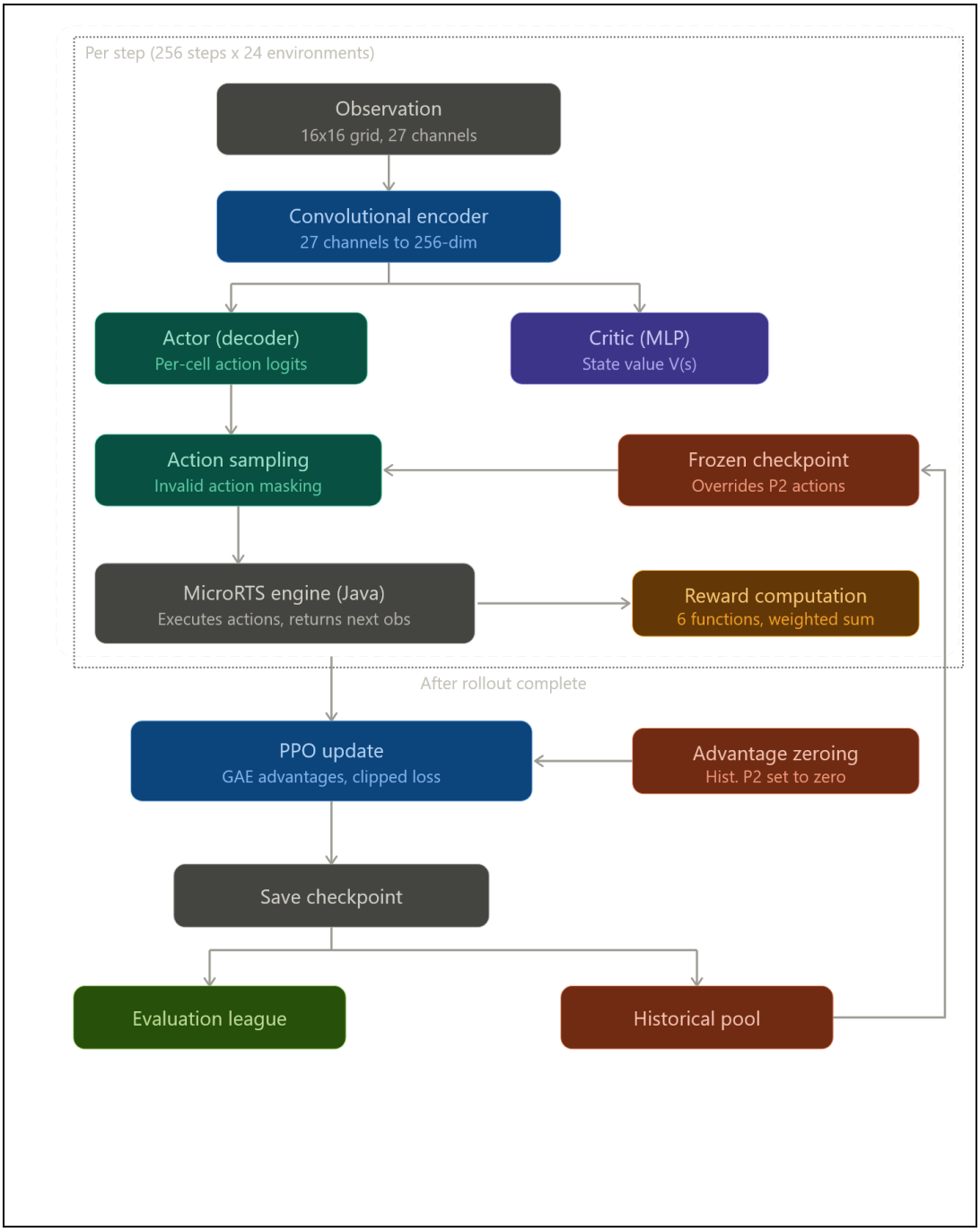

Figure: Training workflow diagram. Showing how the reward functions are processed.

3.7 Implementation notes

The historical checkpoint win-rate is cumulative, so if the 10M checkpoint faces the 5M checkpoint and gets a win rate of 95%, at 15M checkpoint the 5M checkpoint will still be marked as 95%.

The training is deterministic. If two identical seeds are used and all other settings are identical then the training will be identical. The agent will get the exact same rewards, and the tensor board outputs will be identical. But the evaluation is not deterministic and there will be differences even for identical runs. Part of this is due to

NaiveMCTSAI being a very good bot while Light Rush is very easily countered. If an agent can beat Light Rush easily, they never face NaiveMCTSAI and so their rating stays higher, whereas if they get unlucky and lose a game to Light Rush and then face NaiveMCTSAI they might not beat it and be stuck below it in rating.

3.8 Preliminary Exploration

The first experiments that were carried out were conducted to understand the codebase, the process and workflow and how the agent trained. There were tests of various configurations, settings, reward weights and reward functions to understand the agent, its training, the strengths and weaknesses of the scripted bots and the codebase itself.

Initial runs used CPU processing at approximately 200 SPS because the CUDA dependencies were not yet installed. After configuring GPU acceleration, throughput increased to 1200 SPS, reducing a 40M step run from 55 hours to approximately 9 hours.

After this the bots were evaluated in a tournament to see how they fared against each other, and their games were observed to learn how they were scripted and what their weaknesses and strengths were. This was done to understand where an agent might struggle during evaluation. The evaluation tournament is based on a single simple 16x16 map with the same configuration each time. The timesteps are set to 5,000; this is based on the rules set out in the MicroRTS competition which ran games for 4,000 steps and allowed an extra 1,000 steps to break a draw.

Bot Name	Total Wins	Total Losses	Draws	Total Games	Win Rate (%)
CoacAI	376	74	0	450	83.56
WorkerRushAI	328	118	4	450	73.54
Droplet	296	149	5	450	66.52
Izanagi	296	154	0	450	65.78
MixedBot	255	195	0	450	56.67
Tiamat	208	242	0	450	46.22
LightRushAI	175	275	0	450	38.89
Rojo	139	283	28	450	32.94
GuidedRojoA3N	113	302	35	450	27.23
NaiveMCTSAI	22	416	12	450	5.02

Table 1: Evaluation Tournament results, each bot plays 50 games against each other bot. Not included are Passive, Random and RandomBiased. This was done to save compute time as each of those loses to the above bots.

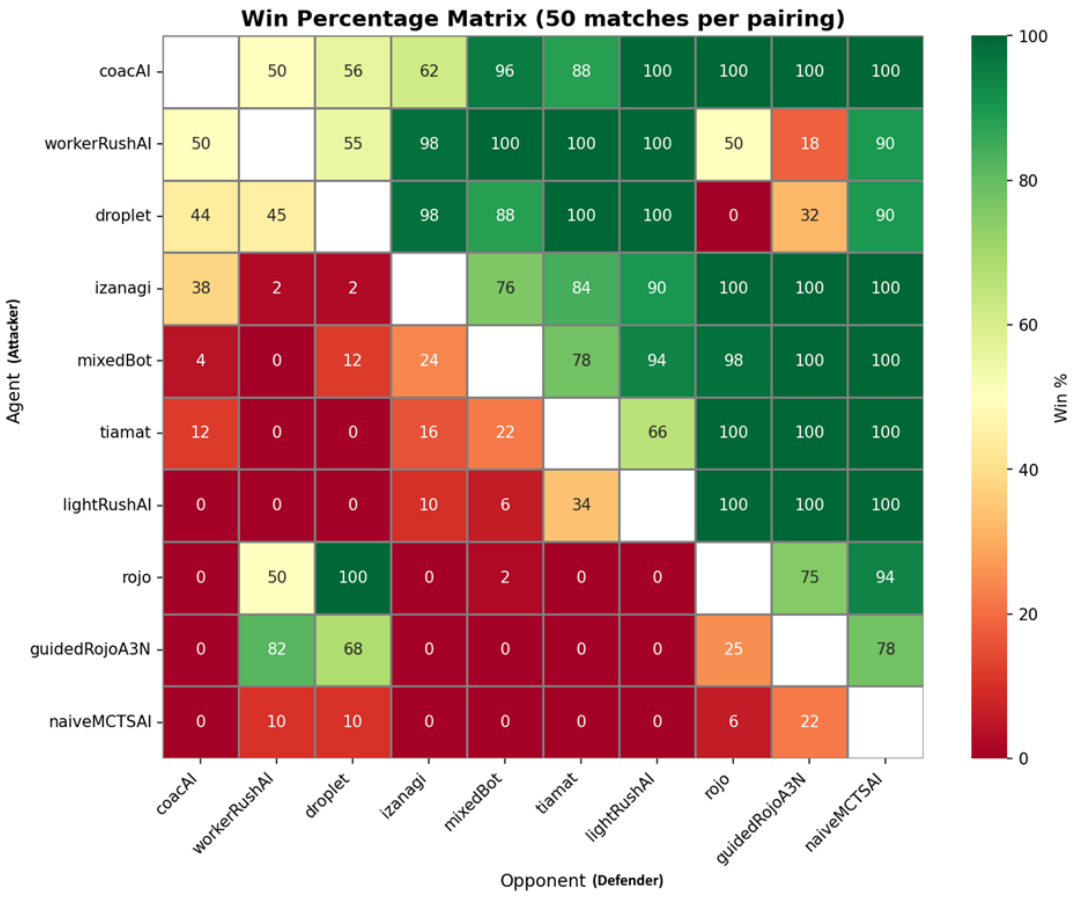

Figure 2: Win Percentage Matrix. Reading y axis plays x axis. So CoacAI plays Tiamat and wins 88% of the time, Tiamat plays CoacAI and wins 12% of the time.

From this evaluation it was decided to re-evaluate the TrueSkill of the bots. This was done to ensure that the TrueSkill ratings of the bots were of a lower uncertainty than the default TrueSkill values where the sigma was above 1 for most bots.

To do this the existing evaluation script was run and after each match the TrueSkill of each bot had its rating updated.

From this readjustment and the bot tournament a table outlining the strengths and weaknesses of each bot along with their implementation details which could be found from their GitHub repositories was created manually.

To get a baseline for comparison for future runs the training was run with only the Win reward weight. This was left at the default 10, whereas every other reward weight was set to 0. The only variable changed was the max-steps of each self-play game in the training.

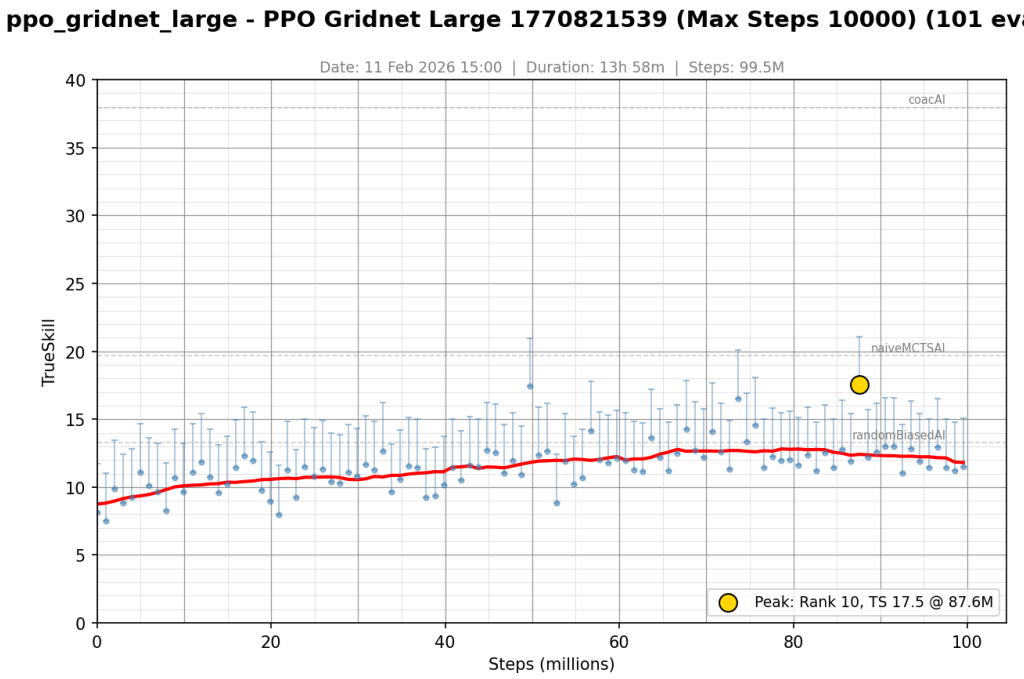

Figure 3: 10,000 max steps configuration. Win reward only. The graph shows the TrueSkill for each evaluation. The error bar represents the Mu at the most upper end, and the TrueSkill point is the lower end.

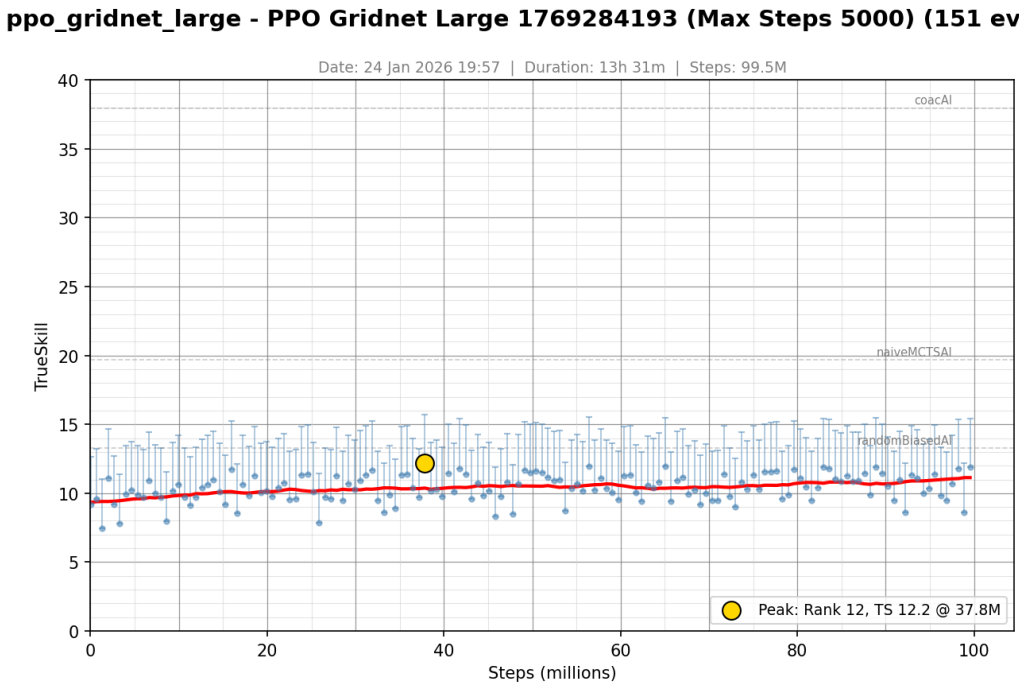

Figure 4: 5,000 max steps configuration.

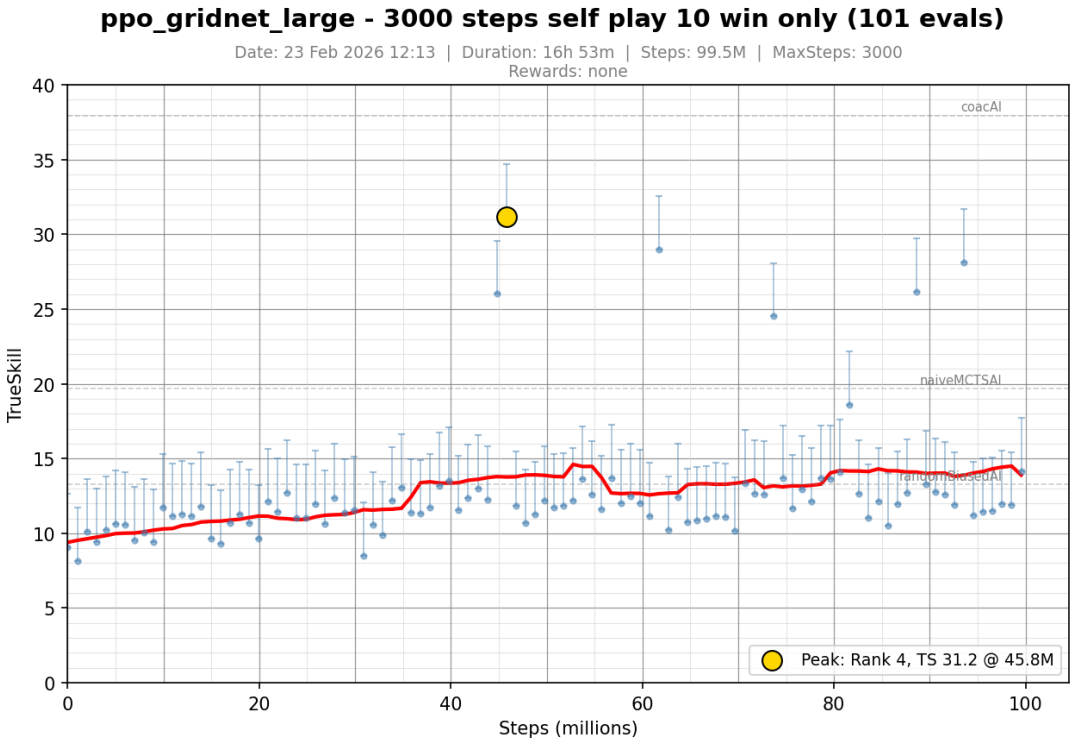

Figure 5: 3000 max steps.

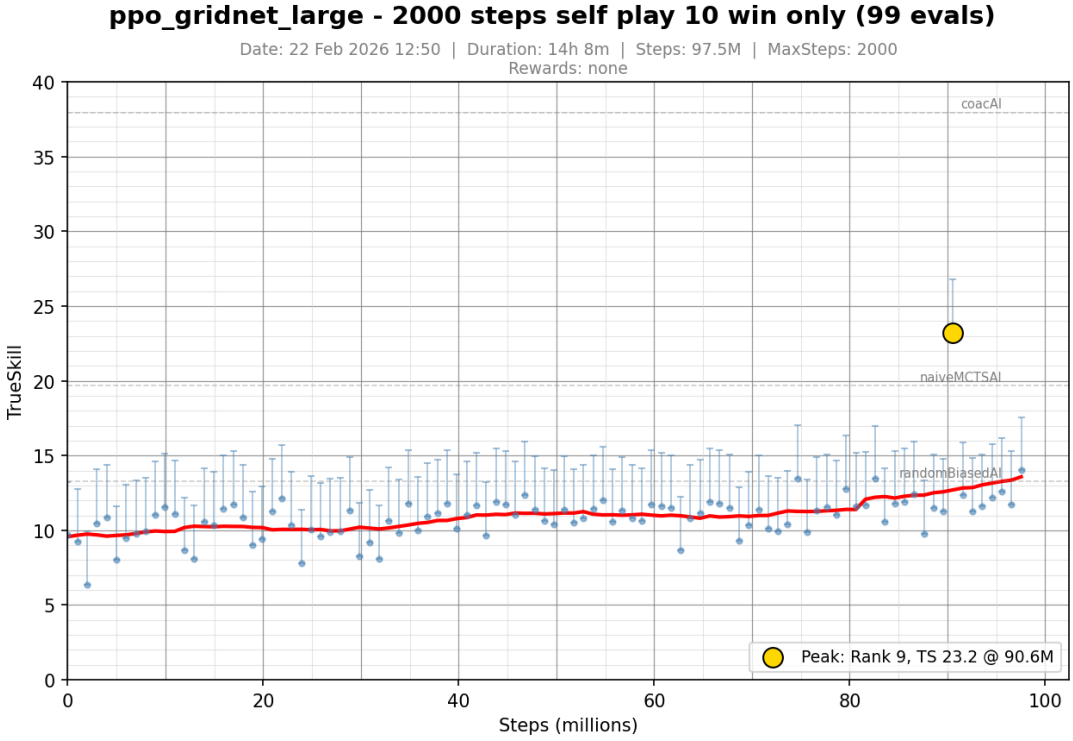

Figure 6: 2000 max steps.

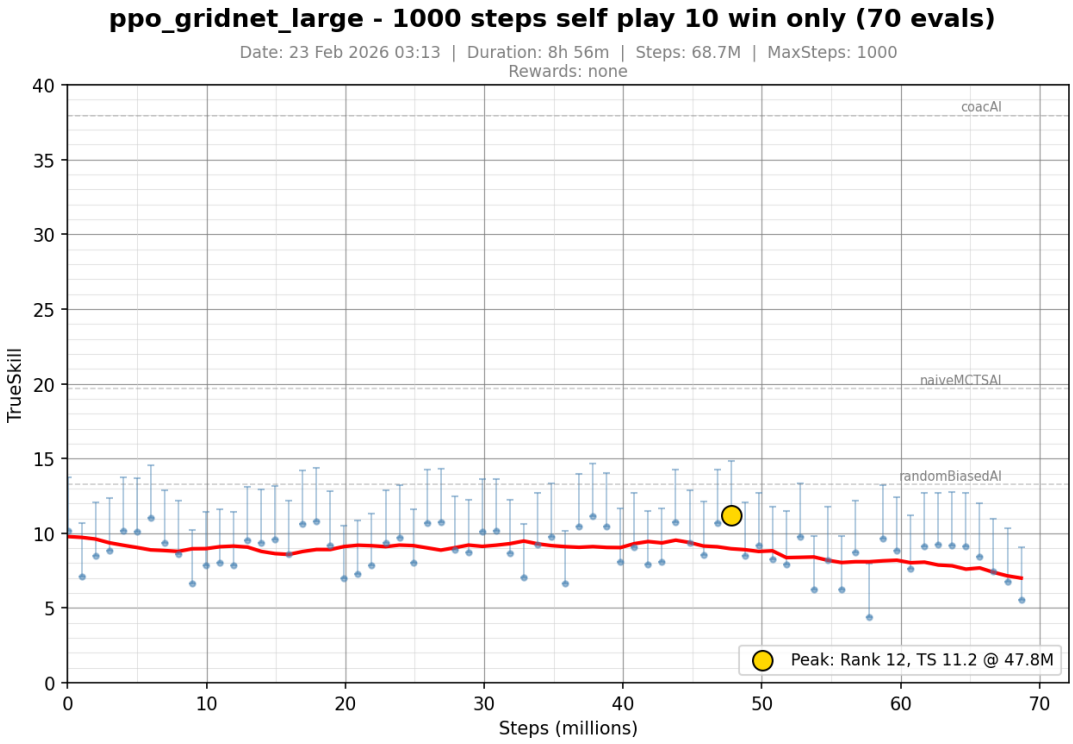

Figure 7: 1000 max steps. Graph shows a steady decrease across the training. The training was ended early due to this.

The 1000 step run was cut short due to the average TrueSkill rating decreasing as the training progressed, it was decided to use the compute time for a different task instead of completing the 100 million steps.

From these training runs the 3000 max step configuration was chosen going forward as the default parameter. This was due to it breaking through the 15 TrueSkill ceiling and reaching a maximum TrueSkill greater than 30 whereas every other run never broke the 15 TrueSkill ceiling.

The 15 TrueSkill ceiling is where the agent is competitive with the RandomBiased Bot but fails against Rojo and NaïveMCTSAI. To get above 15 TrueSkill the agent needs to have a consistent strategy as it is no longer facing random action bots.

3.9 Evaluation plan

Evaluate historical checkpoints in comparison to without historical checkpoints.

Using the default weights train the agent to 30M training steps and compare the TrueSkill score at each checkpoint. Further runs were conducted at 80M steps to investigate training-length dependence.

4 Results and Discussion

4.1 Overview

This chapter describes the results of the evaluation plan and the discussion surrounding these results.

4.2 Preliminary Training and the Self-Play Instability Problem

Before the historical checkpoints were added the first runs were conducted to see what effects changes to the reward functions would have on the training. The main issue that was noticed was that there were collapses in TrueSkill during training that were not recovered for 5-10M training steps.

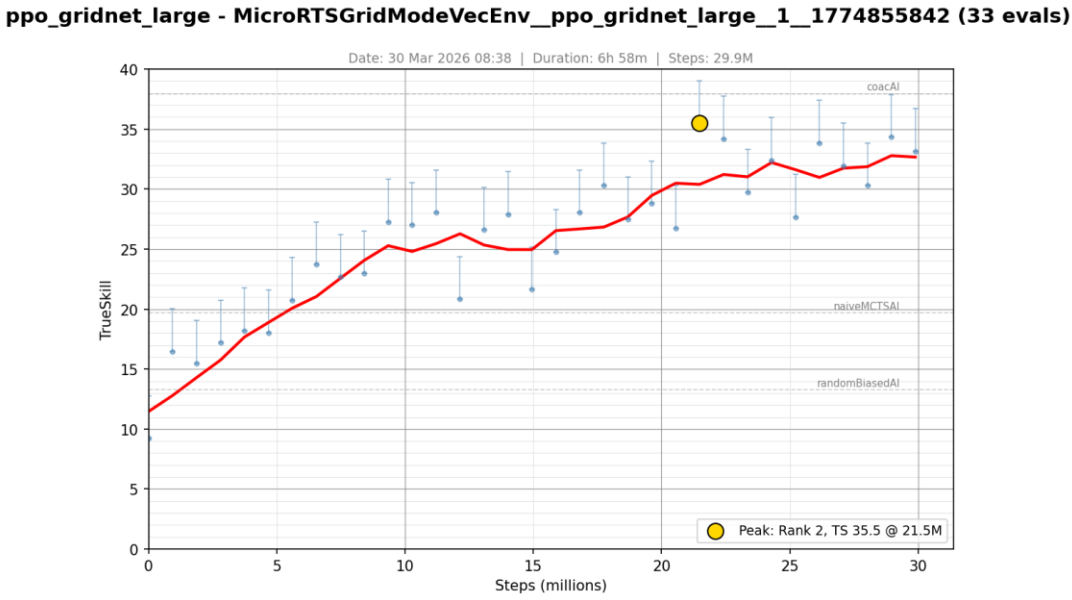

Figure 8: This is the baseline run that was used for comparison. It uses the default parameters and 3,000 max steps for the 24 self-play environments.

The run reaches a peak TrueSkill of 32.9. Due to the deterministic nature of the training if the same seed was run again, it would produce identical training data, but the evaluation data would be similar but different.

This allows a fair comparison between this run and any run with variables changes as it is not down to a “lucky seed” which gave a higher peak TrueSkill value.

The next training conducted was only with the Attack Reward and the Win Reward. This was to see how the agent would react to just a simple reward signal; Win Reward only was already tested. The Worker Rush scripted bot is one of the top bots with 30 TrueSkill so it was believed that even if the agent never built a barracks and just used workers to attack it could still reach 30 TrueSkill.

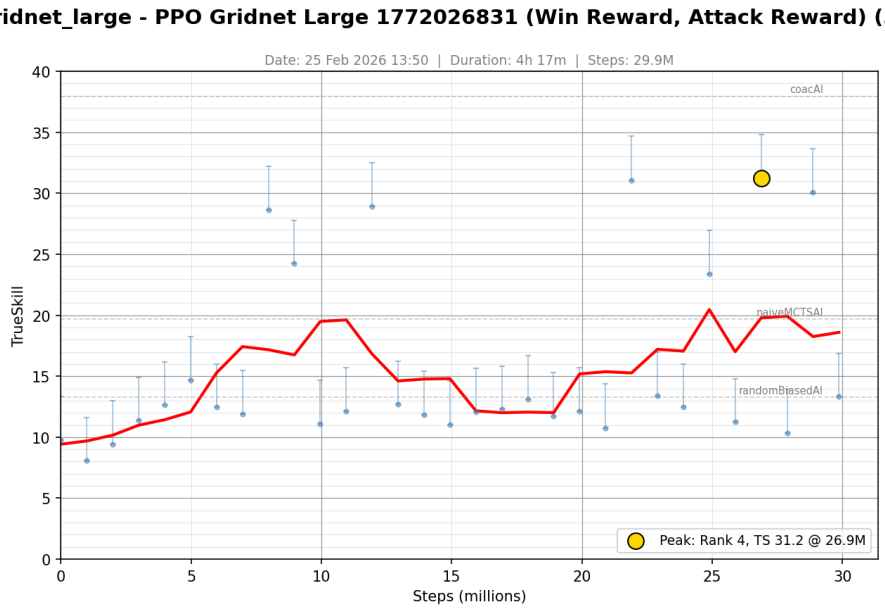

Figure 9: Win Reward and Attack Reward Only.

This graph shows the training to 30M timesteps with just the attack reward and the win reward. As can be seen it breaks the 15 TrueSkill barrier before 10 million timesteps. It reaches TrueSkill 29 at timestep 8M. But it cannot maintain its TrueSkill and falls below 15 TrueSkill between 13M and 22M timesteps. This effect was noticed across multiple training runs, where there will be a single breakthrough, but the agent reverts to a lower TrueSkill and cannot build upon a good run.

To address this the next step was changing the reward function itself, to see if a reward function which changed based on the agents' actions could allow it to sustain a good TrueSkill rating.

The reward function was changed so it rewarded the agent the max damage of the unit when attacking an enemy. This was done to encourage the agent to build combat units which have a higher max damage.

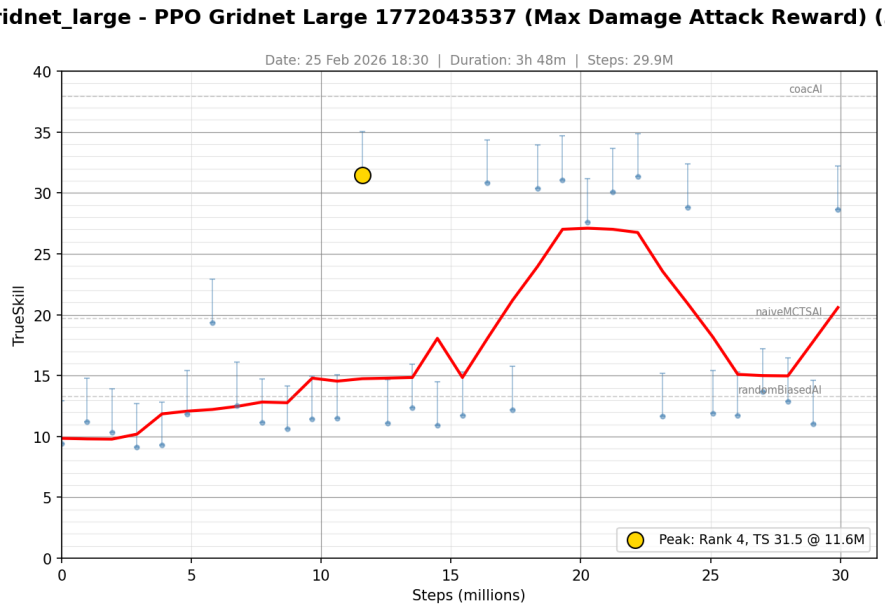

Figure 10: Changed attack reward to give a reward based on the max damage of the unit.

This change allowed the agent to sustain a run at a TrueSkill above 30 for just under 7 million steps before it collapsed again. The agent would develop an effective strategy, then train against copies of itself using that strategy. Through self-play it would discover a counterstrategy, which then replaced the original. However, these counter-strategies were specialised against the agent's own play and performed poorly against the evaluation bots, causing TrueSkill to drop.

The core problem was that once a strong policy was overwritten the agent had no mechanism to recover it. In pure self-play, both copies of the policy degrade together. The agent never faced its previously strong checkpoints, so there was no training pressure to maintain or return to effective strategies. It was decided to test out various combinations of reward weights and reward functions to see if one combination could sustain incremental TrueSkill gain with sparse reward weights.

To solve this underlying problem of the AI saw toothing between an effective strategy and a counter strategy it was decided to add 4 bot environments to the training. These were 2 light rush and 2 worker rush environments. These two bot environments were specifically chosen as they were lightweight and did not decrease performance and they provided two different strategies for the agent to train against, the light rush bot must build a barracks to build light units so it is vulnerable to an early rush and the worker rush bot is aggressive from the start but if the agent can outlast it can win.

It was thought that this would provide a stable foundation from which the agent policies would evolve strategies from.

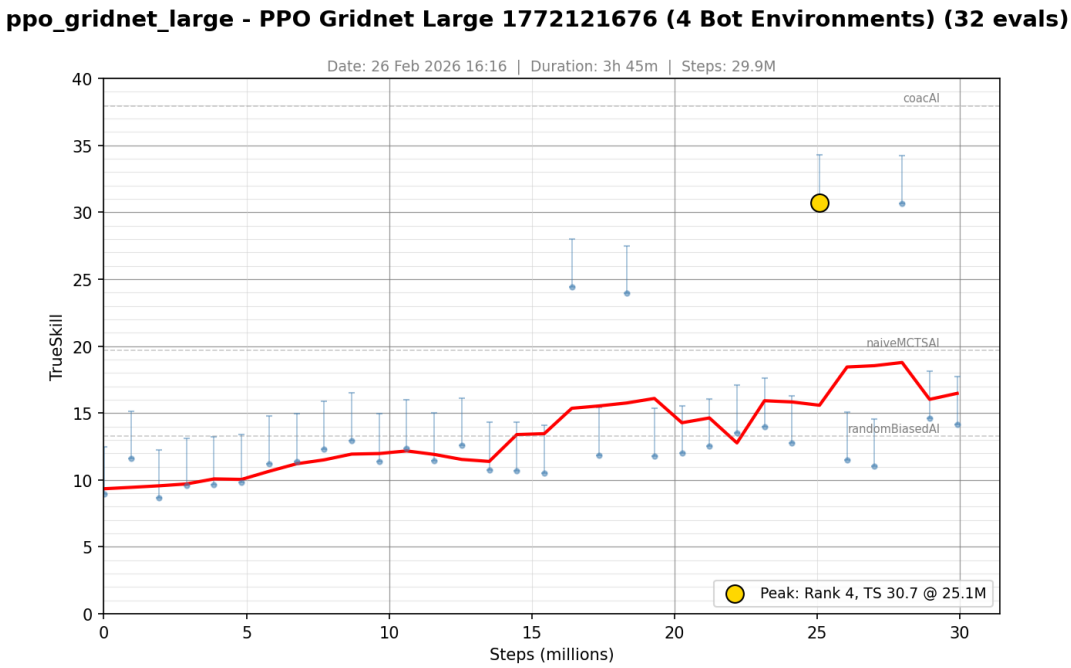

Figure 11:4 Bot environments added to the training

This run did not meet the previous expectations. Its peak was lower than the previous runs. It showed small consistent growth and decline but it was limited to the 10-15 TrueSkill range. The breakthroughs above 15 TrueSkill peaked lower than previous runs.

The default reward weights were set, these are 10 win reward, 1 resource gather reward, 1 worker build reward, 0.2 barracks build reward, 1 attack reward and 4 combat unit reward. The bot environments were kept with worker rush and light rush.

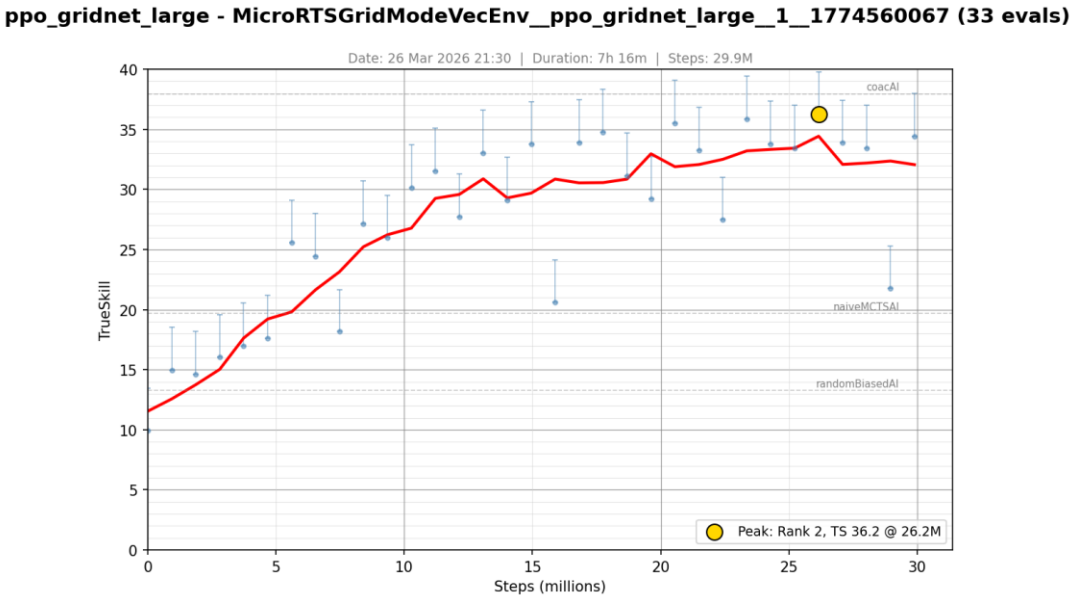

Figure 12: Default reward weights and functions plus 4 bot environments

With these changes we still see a dramatic drop of TrueSkill for some checkpoints. 16M checkpoint falling to 21 TrueSkill where it was at 34 TrueSkill 1M steps before. But the agent quickly rebounds and sustains an average TrueSkill above 30.

When the agent has a dense reward signal it can reach high peak TrueSkill ratings. But at sparse reward signals it finds a winning strategy and then spends the next 5M training steps countering that strategy and overwriting it.

4.3 The Effects of Historical Checkpoints

Historical opponents were then added. It was hoped that historical checkpoints would provide the agent with self-play environments where it would be forced to play against strong previous checkpoints and through this it could sustain a high TrueSkill.

The reward weights were left the same as the graph above and the bot environments were left the same to give a point of comparison.

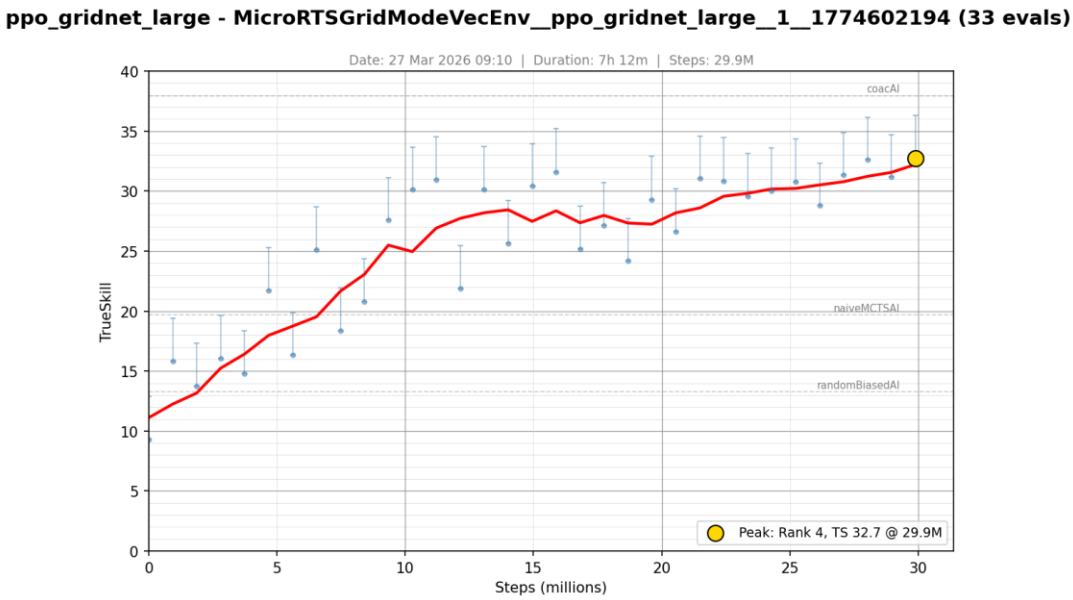

Figure 13: Historical checkpoints enabled, 4 bot environments and default weights/functions.

With the bot environments included the agent reaches a peak TrueSkill of 32.7 and the training run does not feature major collapses of TrueSkill once the policy is established after 15M time steps. Between 20M and 30M timesteps there is a slow and steady gain in TrueSkill from 30 to 33. In contrast the non-historical checkpoints reached a higher TrueSkill ceiling but there were collapses down below 30 TrueSkill and even down to 21 TrueSkill.

The no historical checkpoints training run is being fed an extra 4 training environments worth of data to update its policy with. Along with the dense reward signals this allowed it to reach a higher peak TrueSkill rating but at any checkpoint the rating could collapse back below 30. Whereas with 4 historical checkpoints and 4 bot environments the above historical checkpoints run had 8 fewer environments worth of data to update its policy with. To investigate this further the bot

environments were removed from the training and the historical checkpoints were left on.

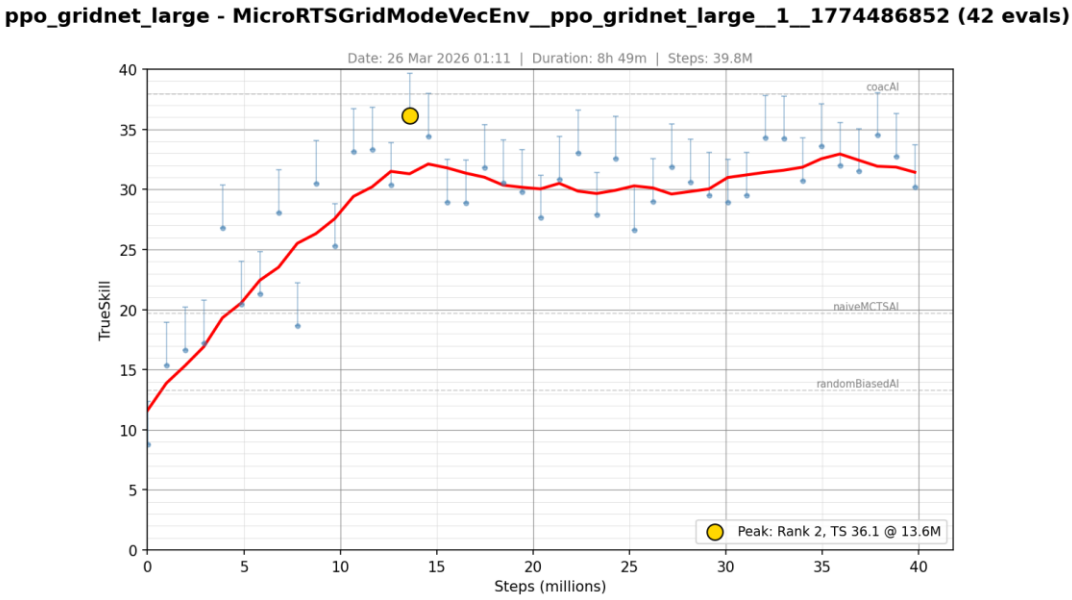

Figure 14: Historical checkpoints, default weights/functions and no bot environments.

Removing the bot environments from the training leads to a peak TrueSkill of 36.1. The bot environments aided the non-historical checkpoint training but hindered the historical checkpoint training. Here we can see that the agent reaches a peak TrueSkill of 36.1 at 13M training steps and sustains a TrueSkill above 26 for the remainder of the training. This is in stark contrast to the baseline run which collapses down to TrueSkill 21 at multiple points in training.

Further training runs were conducted to study this effect.

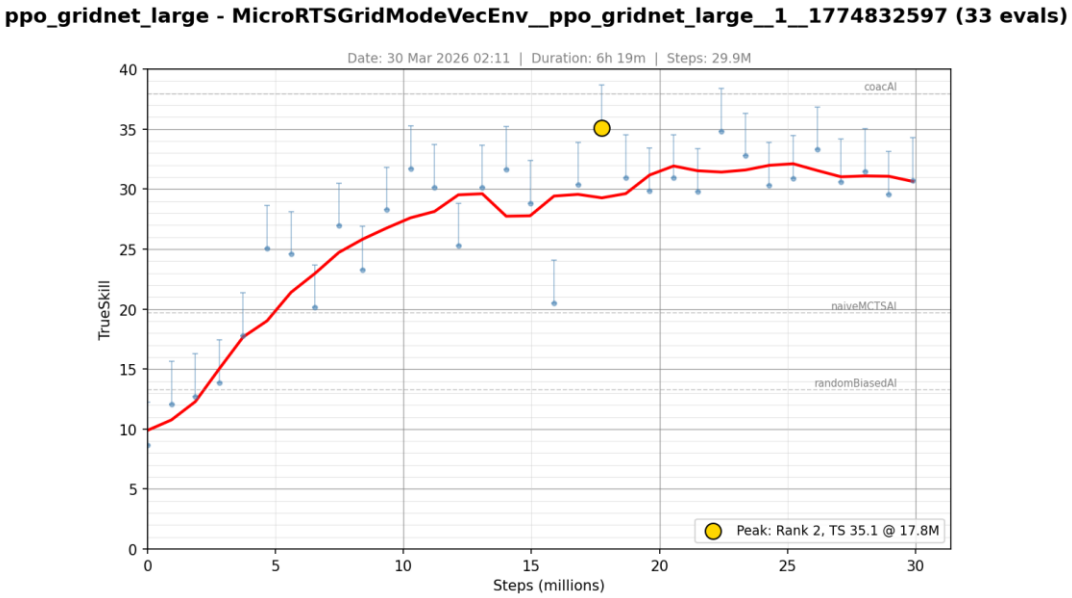

Figure 15: Default Weights/Functions, Historical Checkpoints. Seed 1.

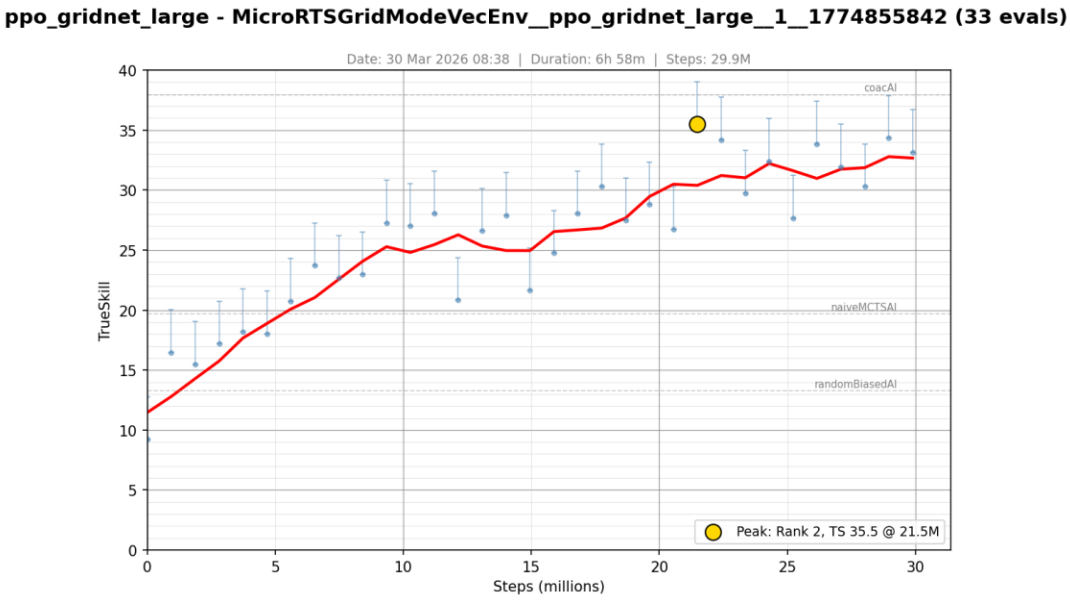

Figure 16: No historical checkpoints, Default Weights/Functions. Seed 1.

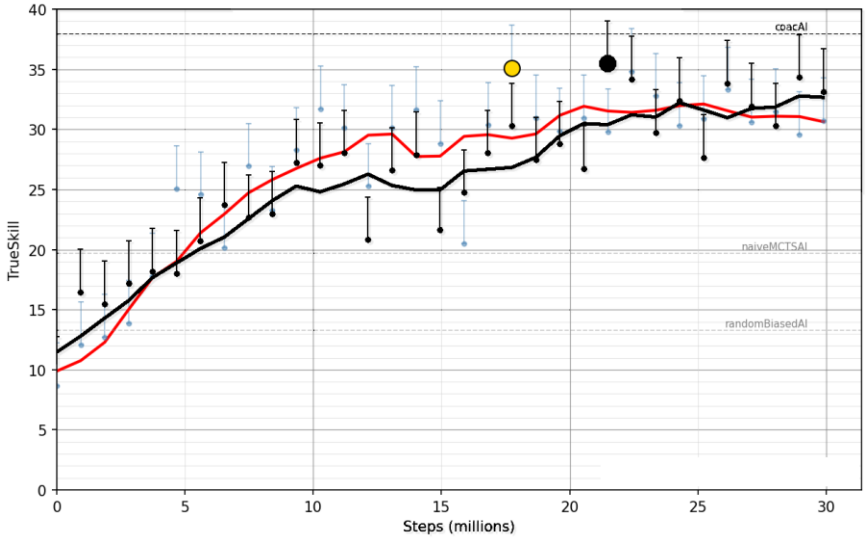

Figure 17: The no-historical checkpoints run is in black, and the historical checkpoints run is in red.

With the default reward configuration over 30M training steps the historical self-play produced no measurable improvement in TrueSkill over pure self-play. The trajectories were indistinguishable when accounting for evaluation noise.

Training runs were then conducted at 80M time steps to see if the agent with the default parameters would obtain a higher peak TrueSkill. One thing affecting the training when increasing the time steps is that the linear annealing changes the shape of the learning.

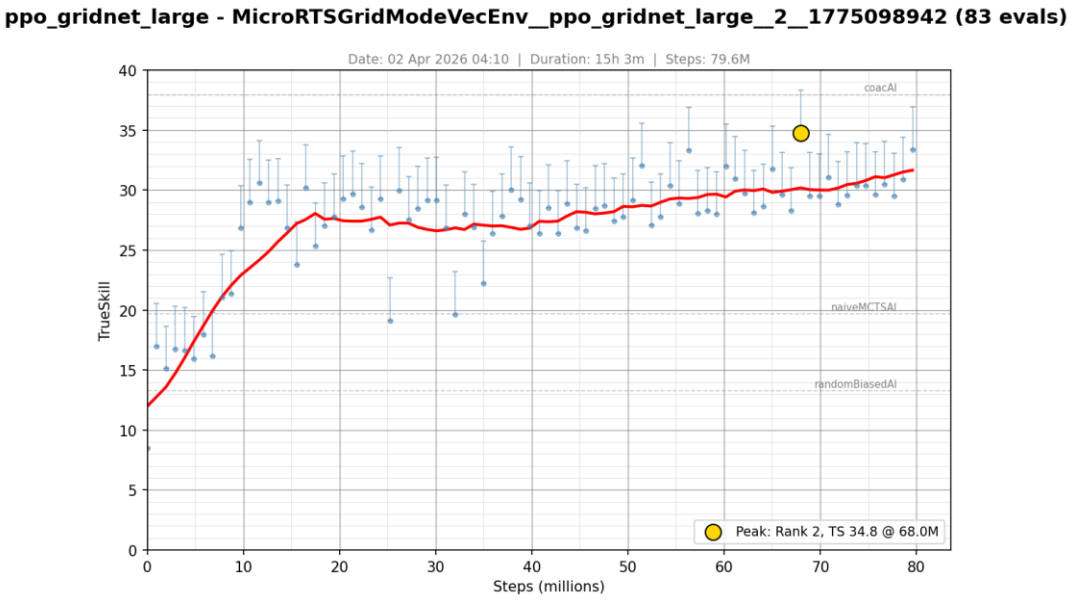

Figure 18: Default parameters, no historical checkpoints

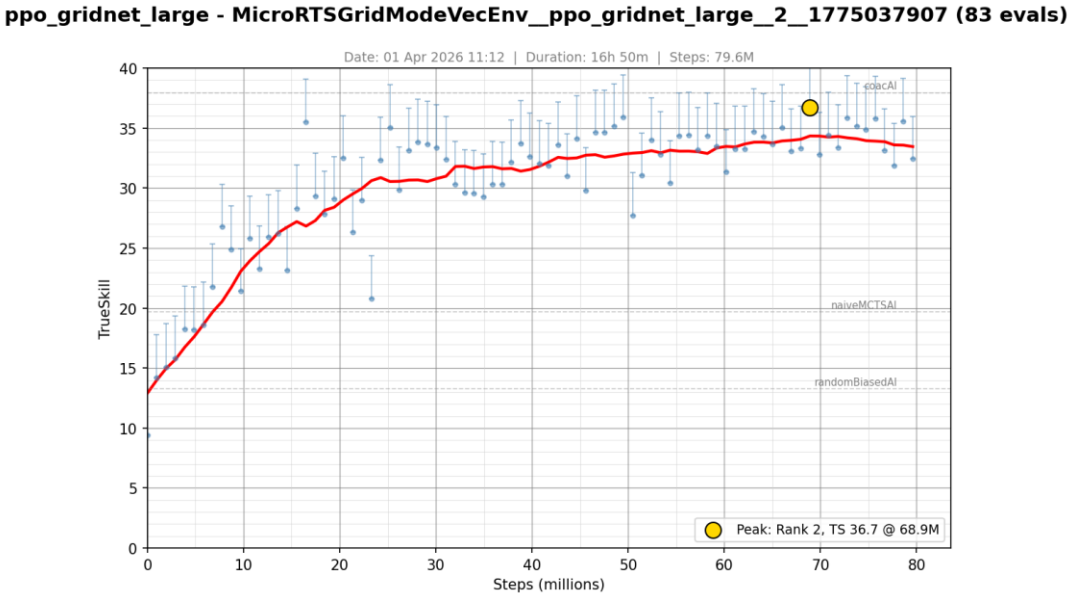

Figure 19: Default parameters, historical checkpoints

With the time steps moved to 80M, allowing the training to happen for longer, there is an entirely different picture. The difference between the historical checkpoints and the non-historical checkpoints is stark. Whereas the runs were nearly identical before, here we see the historical checkpoints run has an average TrueSkill which is always increasing whereas the no-historical checkpoints run reaches an early peak before 20M time steps and then falls off until 40M timesteps where it starts to gain TrueSkill rating again.

To quantify the difference between the historical checkpoints and the default run, the average TrueSkill was computed for each quarter of the training to compare and contrast.

Steps	Historical Checkpoints	Default	Difference
0-20M	22.4	22.1	+0.3
20M – 40M	30.7	27.1	+3.6
40M – 60M	32.9	28.3	+4.5
60M – 80M	34.0	30.3	+3.7

Table 3: Average TrueSkill for each quartile of training.

The historical checkpoints run maintains an average TrueSkill greater than the default run for the entire training.

4.4 Discussion

The default reward weights provide enough dense per-step signal that the value function stays well calibrated even through the self-play policy drift. The agent rebounds quickly from TrueSkill drops because the shaping rewards continue to provide useful gradients.

The five reward functions provide continuous gradient signals that keep the value function calibrated. As we can see from the bot TrueSkill ratings the WorkerRush bot is at TrueSkill 30 just from building worker units and charging them at the enemy. An agent that can adapt the WorkerRush strategy into building a barracks and units would be expected to outperform a static worker rush strategy and this is what we have seen.

Once the agents were allowed to train for a longer period the true value of the historical checkpoints was evident. The TrueSkill graph which was functionally identical now shows the historical checkpoints configuration as a clearly better configuration for training. The non-historical checkpoints training collapses after 20M training steps and only starts to gain TrueSkill on average from the 40M mark. Compared to the historical checkpoints training which continuously gains TrueSkill on average for the entire training. Looking at the graph of the win rate of the agent against the historical opponents [fig. 31] the agent loses approximately 50% of its matches against the historical opponents at 10M timesteps but then recovers and beats the historical checkpoints from after 30M training steps. This is the exact area where the default training run experiences a TrueSkill decline.

This shows that the historical checkpoints allow the training to recover from policy collapse and continue to gain TrueSkill. The reason this issue occurs is due to the learning rate annealing. During the shorter run the default agent reaches an effective strategy early and the learning rate annealing slows the learning rate allowing the agent to maximise this strategy. For the 80M training run at 40M steps the learning rate has only halved, this allows the originally effective strategy that was learned between 10M – 20M time to degrade due to the self-play instability. Without the corrective pressure of the historical opponents the policy drifts and it is only near the end of training that the agent starts to gain TrueSkill again. [fig. 32] This shows the entropy for both training runs. The historical checkpoints training maintains high entropy allowing it to explore multiple strategies before converging on the best one.

4.5 Limitations

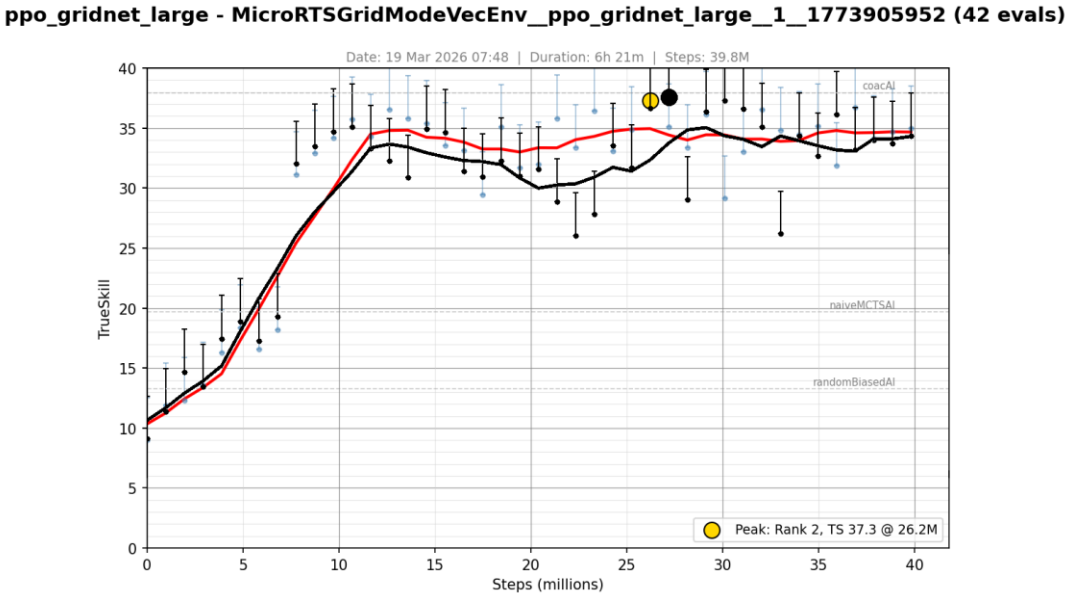

Figure 20: Identical training ran through evaluation twice.

For identical training the evaluation will not be the same. In the above graph multiple points are the same in the training and at other points in the training there are stark differences. These trainings were the same checkpoints evaluated twice. The overhead added by running more evaluation matches to get a TrueSkill value with more certainty was not worth the extra time. Due to the limited amount of scripted opponents, each with different strategies which are strong and weak in different areas there will always be a level of uncertainty and randomness to the evaluations.

This must be considered when discussing the differences between results. Due to each run taking 6+ hours and each run needing one run with historical checkpoints and one run without it took on average 14 hours for a set of runs.

Only one seed was tested for the final comparison. There are major differences between seeds but for a point of comparison due to the deterministic nature of the training it was decided to test one seed first. The differences in the final training configurations were stark enough that it couldn't solely be down to evaluation noise.

All experiments were conducted on a single 16x16 map. This was to keep the training focused on just this action space. The scripted bots each had different performance across the different maps, and it was believed that adding more maps into the mix would add further noise and differences to the evaluation outputs.

For the historical checkpoints the win rate is cumulative, so if the agent at 20M time steps handily beat the 15M historical checkpoint that checkpoint will never be sampled again as its win rate will be too low. This could lead to a situation where the agent is constantly facing the most recent checkpoint and beating it and so that checkpoint is never selected again. This is not believed to be a big issue due to the graph showing the agent win rates against the historical checkpoints [fig. 31] which

shows the agent only winning 50% of its games during the 10M-30M stage of the training, suggesting that the historical checkpoint is giving a challenge to the agent during training.

5 Conclusions and Future Work

5.1 Conclusions

Historical self-play did not produce a measurable improvement under the default dense reward configuration over 30M training steps. But over the 80M training steps it showed a marked increase in performance compared to the default configuration.

The average TrueSkill for the non-historical checkpoints training was under 30 TrueSkill for most of the training only rising to 32 TrueSkill near the end of the training whereas the historical checkpoints training had a TrueSkill which was an average of 30 at 20M steps and rose gradually to 34 over the course of the training.

Over 30M training steps there is not enough time for the historical checkpoints (4/20 environments) to provide a major signal to change the training more than the default reward weights due to the reward signal density of the default parameters. The historical opponents provide a strong corrective signal over the long training that is missing in the 30M training. When the no-historical training reaches a peak TrueSkill early the training from that point trains to counter that strategy rather than make it more effective. Whereas with the historical checkpoints the agent can refine the good strategy over the course of the training and receive good reward signals when it beats the historical checkpoints. Comparing the entropy for each run the historical checkpoints enabled run can be seen to have higher entropy throughout the duration of the training as seen in figure 32, in comparison to the default training run. This entropy allows the training to escape from local strategy maxima that it encounters during training.

Proper reward functions and scripted training opponents provide stabilisation on par with historical checkpoints for dense reward signal situations over 30M training steps. But for training runs above this the policy signals that historical checkpoints provided were key in allowing the run to maintain TrueSkill for the entire training.

5.2 Future work

- Test over longer training steps: While the 80M training steps showed a marked improvement more testing would need to be conducted to find the best length for training. Due to the learning annealing any change in training length will change the learning itself. No experiments were conducted to see would different configurations in the annealing produce markedly different results for historical and non-historical training. Can an agent training with historical checkpoints on be able to explore more of the training space due to the fact it has the older checkpoints to pull it back to a competitive strategy if it gets caught in a local maximum compared to the default parameters where there is no such mechanism.
- Changing reward functions: the first thing investigated before the historical checkpoints were implemented were changing the reward functions. The

reward functions provide simple linear rewards based only on the agents' actions.

- Increasing the parameters: The training was assessed with four times the parameters with no noticeable increase in TrueSkill trend. The training plateaued incredibly early and did not show any trend upward after 10M steps. This could be one area to investigate.

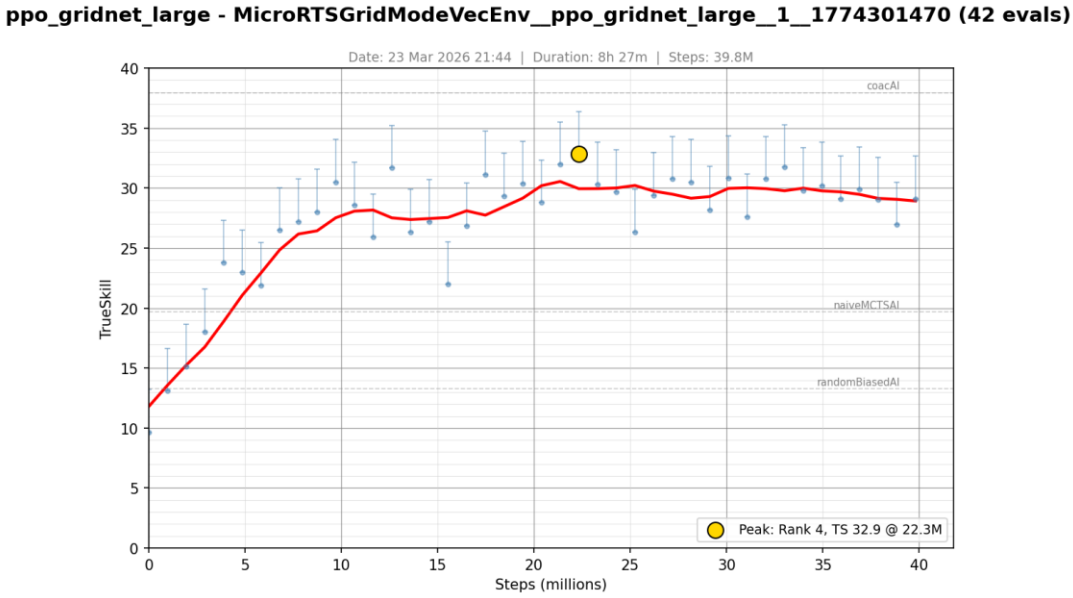

Figure 21: 3.3M parameters with default reward weights/functions.

One run was conducted during exploration to test if more parameters would provide a sufficient increase. The drop in steps per second was noticeable, losing about 40% SPS over the training compared to the default parameters and there was no gain in TrueSkill at all. But this was only tested for one configuration, as seen with the historical checkpoints it showed a marked increase when the training was increased to 80M steps.

Bibliography

[1] Silver, D., Schrittwieser, J., Simonyan, K. *et al.* Mastering the game of Go without human knowledge. *Nature* **550**, 354–359 (2017). https://doi.org/10.1038/nature24270

[2] Vinyals, O., Babuschkin, I., Czarnecki, W.M. *et al.* Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **575**, 350–354 (2019). https://doi.org/10.1038/s41586-019-1724-z

[3] Berner, Christopher & Brockman, Greg & Chan, Brooke & Cheung, Vicki & Dębiak, Przemysław & Dennison, Christy & Farhi, David & Fischer, Quirin & Hashme, Shariq & Hesse, Chris & Józefowicz, Rafal & Gray, Scott & Olsson, Catherine & Pachocki, Jakub & Petrov, Michael & Pinto, Henrique & Raiman, Jonathan & Salimans, Tim & Schlatter, Jeremy & Zhang, Susan. (2019). Dota 2 with Large Scale Deep Reinforcement Learning. 10.48550/arXiv.1912.06680.

[4] McKiernan (2022) Improving Multi-Agent Reinforcement Learning by Incorporating Prior Knowledge, University of Galway

[5] Dwivedi (2023) Creating Tuneable Agents to play RTS games, University of Galway

[6] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. ArXiv, abs/1707.06347

[7] Caellum Kennedy MicroRTS-FYP https://github.com/Caellum-Kennedy/MicroRTS-FYP

Appendix:

Episode Lengths from the Win-reward only runs where max steps was the only setting being changed.

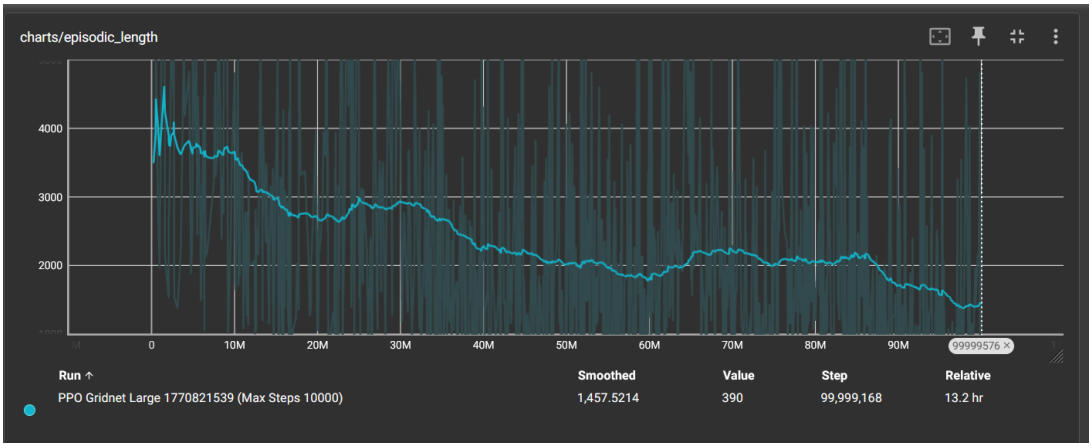

Figure 22: 10000 max steps. Length of each self-play match plotted on Tensor board with maximum smoothing applied

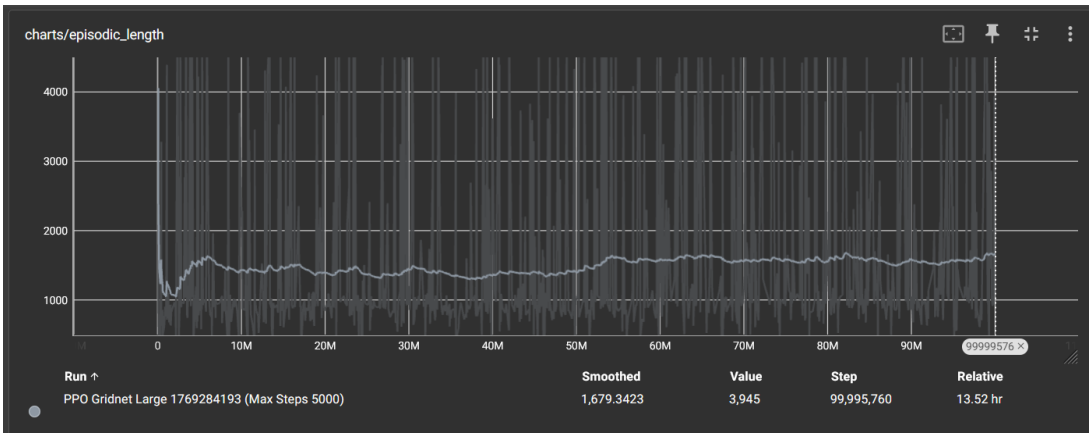

Figure 23: 5000 max steps.

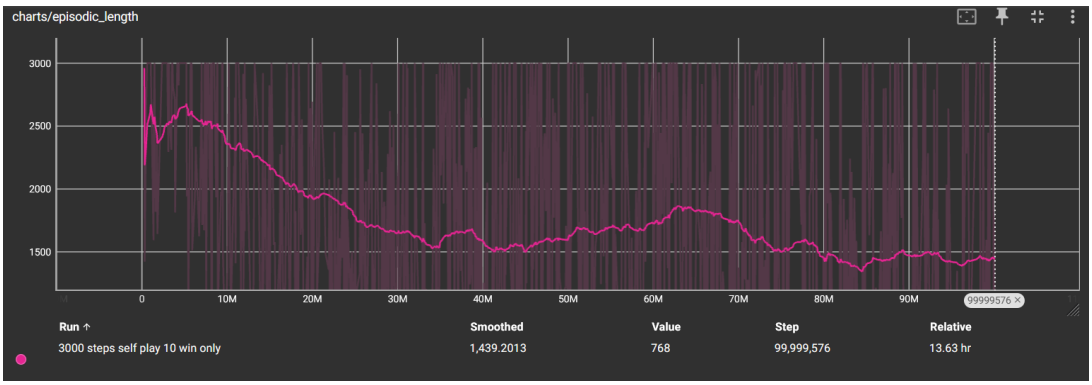

Figure 24: 3000 max steps.

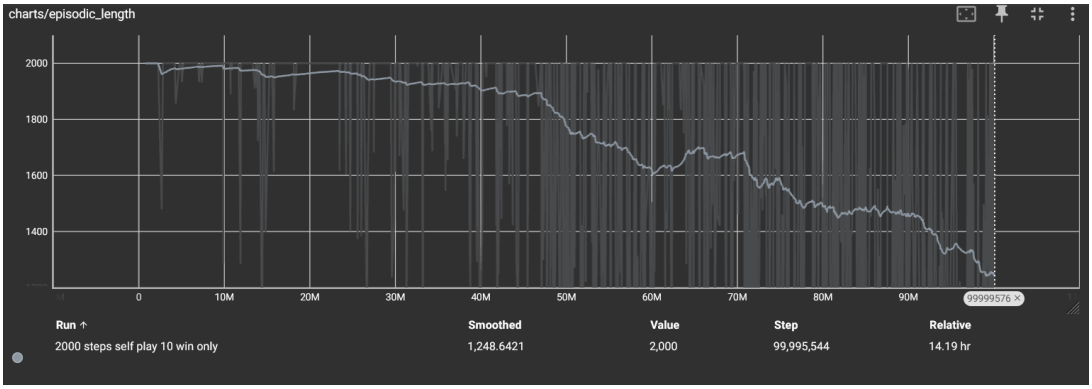

Figure 25: 2000 max Steps. max smoothing applied.

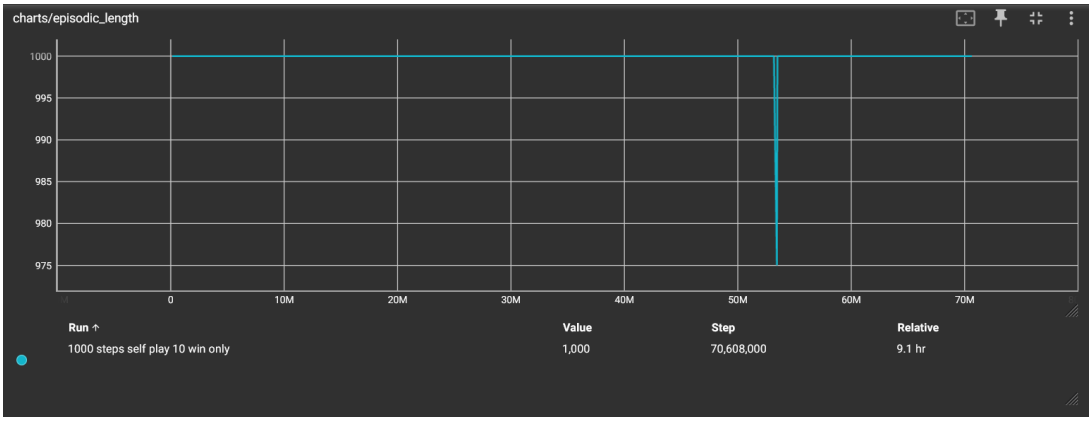

Figure 26: 1000 Max Steps. No smoothing. There was only a single game in the entire training below 1000 steps.

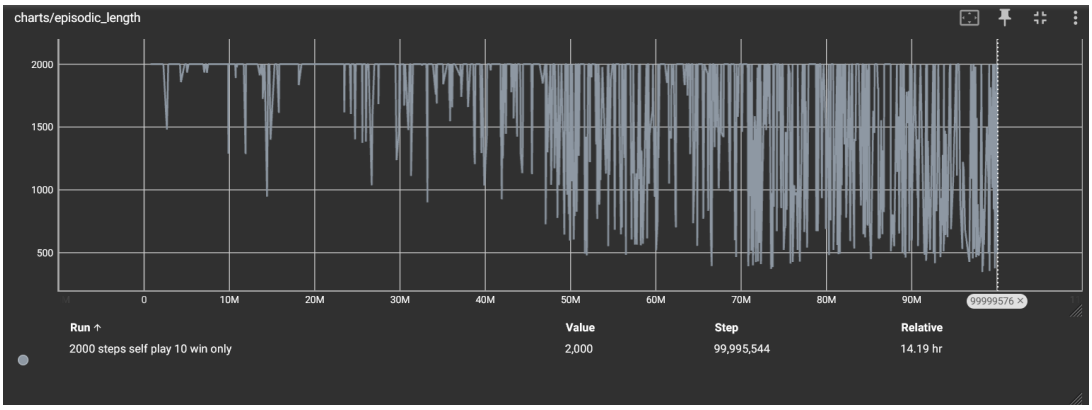

Figure 27: Non-smoothed tensor board 2000 max step graph.

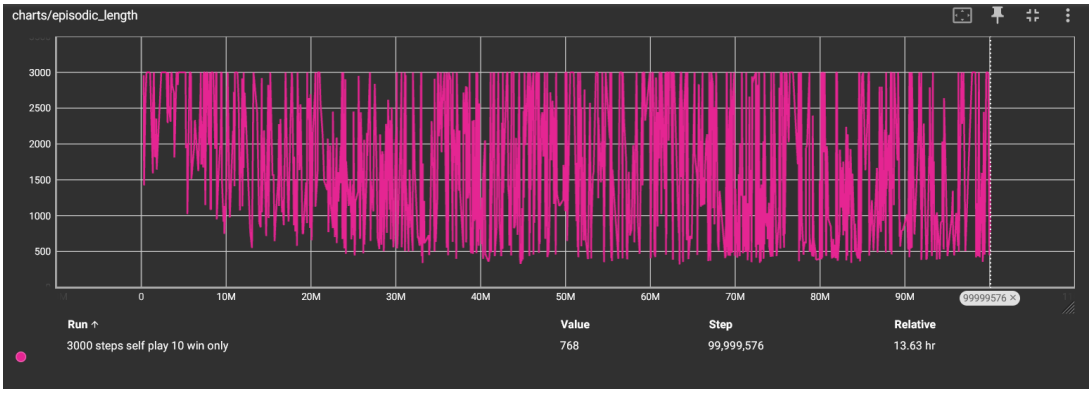

Figure 28: Non-smoothed tensor board 3000 max step graph.

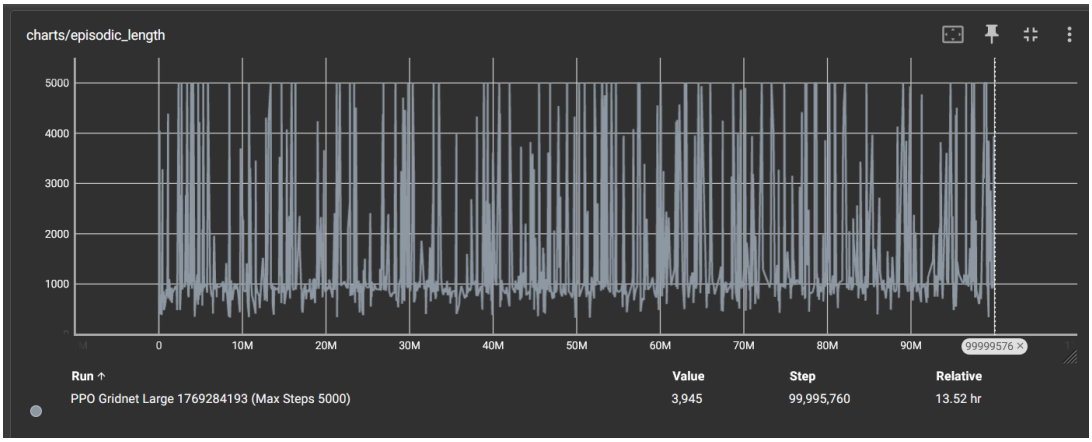

Figure 29: Non-smoothed tensor board 5000 max step graph.

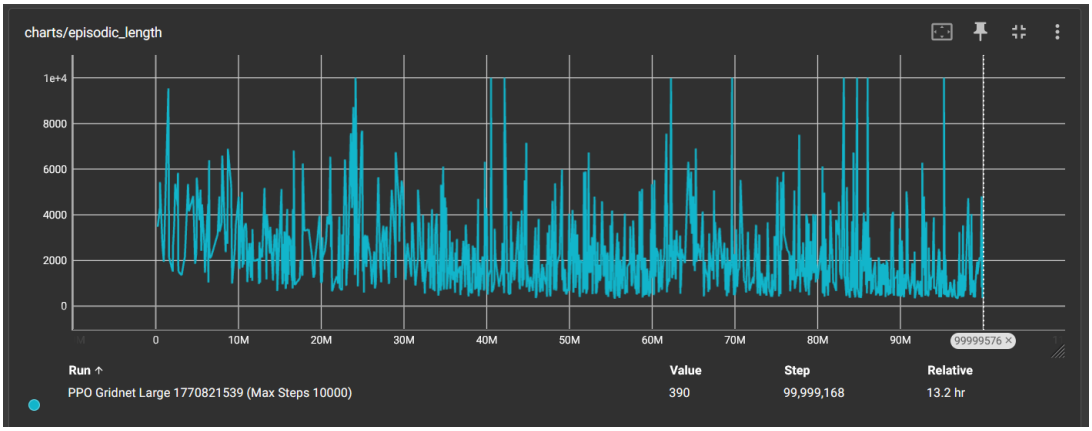

Figure 30: Non-smoothed tensor board 10000 step graph.

Scripted Bot evaluations:

Name	TrueSkill	Type	Strength	Weakness
CoacAI	37.94	Portfolio Rule-based	Portfolio of different strategies to manage different opponents.	Weak to extreme early aggression followed by ranged units.

Droplet	34.49	Script-guided MCTS	Worker Rush strategy combined with MCTS, strong early game.	Due to MCTS struggles on larger map sizes.
Worker Rush	30.46	Scripted	Strong early game aggression.	No late game transition, one trick pony.
Izangi	33.19	Script-guided adversarial algorithms	Double barracks production and good unit micro.	Weakness to enemies which gather a force before attacking. (CoacAI)
Mixed Bot	28.50	Portfolio, Tiamat Strategy, Capivara Tactical.	Ensemble of different strategies so strong once strategy set.	Inherits weaknesses of Tiamat and weak before strategy setup. Weak to early pressure.
Tiamat	26.67	Stratified Strategy Selection	Suited to unit micromanagement through SSS.	Weak to early pressure before strategy stratified.
Light Rush	23.33	Scripted	Early aggression.	Predictable and countered by heavy units.
Naïve MCTSAI	19.70	Monte Carlo Tree Search	Strong bot once past early stage as only limited by time budget for moves.	Based on RandomAI so until tree search setup it is weak.
Guided Rojo	24.48	Guided Naïve Sampling with Rojo Scripts	Guided Rojo so is enhanced version of rojo.	Suited for bigger maps so not as good for micromanagement.
Rojo	15.05	Scripted using Genetic programming	Based on generated scripts. Strong against droplet and NaiveMCTSAI.	Weakness to CoacAI and LightRush.
Random Biased	13.29	Weighted Random Selection	Better than random AI	No planning just random moves biased towards certain moves.

Random	2.84	Random Selection	A minimal baseline for comparison	No thinking whatsoever, pure random.
Passive	-1.55	No Actions	Verification of working code	Does not do anything

80M historical checkpoints training run:

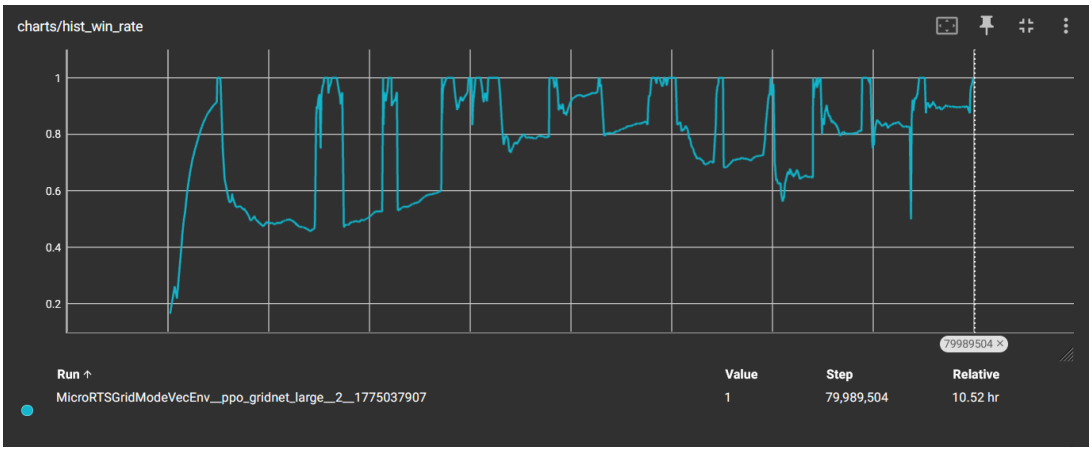

Figure 31: The historical checkpoints win rate (agent win rate) for the entire training.

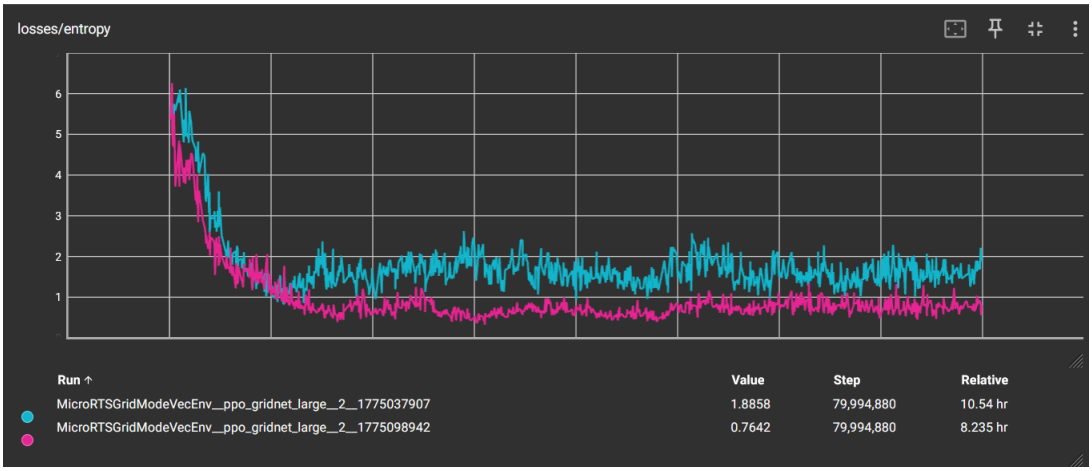

Figure 32: The entropy of each training run (how exploratory each agent is in its decisions), the historical checkpoints run is light blue, and the no-historical checkpoints is pink.