Exploring Reward Shaping for Enhanced Cooperation in Multi-Agent Reinforcement Learning

Justin Burzachiello, Prabal Chhatkuli, Zeru Zhu, Touhid Hossain

Abstract

Reward shaping is a pivotal technique in reinforcement learning (RL) that augments reward functions to guide agents toward optimal behaviors, especially in environments with sparse or delayed rewards. In this study, we explore the impact of reward shaping on cooperative behaviors in a multi-agent reinforcement learning (MARL) setting using the "Multiwalker" environment, where agents collaborate to transport a package. To investigate the effects of reward shaping, we constructed five distinct Markov Decision Processes (MDPs), each incorporating progressively complex shaping functions designed to enhance agent coordination and performance.

The MDPs ranged from poorly designed reward functions, such as those incentivizing angular motion (e.g., "bad angle"), to more structured configurations combining package motion, walker motion, and positioning penalties. Performance varied significantly across the MDPs, as shown by the accompanying box-and-whisker plot, which highlights median reward improvements as shaping functions became more refined. Poorly constructed reward functions led to unintended behaviors like backflips or stagnation, while sparse rewards failed to promote significant progress. In contrast, well-designed shaping functions, such as those encouraging balanced posture, forward movement, and proper positioning, resulted in superior cooperative performance.

Our findings underscore the critical role of thoughtful reward design in MARL, demonstrating how carefully structured rewards can balance exploration and exploitation, foster collaboration, and improve overall system performance. This work highlights the potential of reward shaping to optimize multi-agent systems and provides insights for designing robust reward mechanisms in complex cooperative environments. Our results, code, and models can be viewed at `https://github.com/prabalchhatkuli/Reward-Shaping-RL`

1 Introduction

The often-sparse nature of rewards in episodic reinforcement learning (RL) trajectories can make learning an optimal policy slow since an agent may only be rewarded upon completing a goal rather than making progress. However, one could augment the underlying Markov decision process' (MDP's) reward function in order to provide intermediate rewards that may guide the agent towards their goal. This strategy of providing intermediate rewards is called reward shaping, and it comprises its own field of research within the RL literature. In general, it is done by adding a so-called reward shaping function to the original reward function of the MDP, yielding an augmented MDP through which the agent learns.

Reward shaping is generally useful when a given MDP environment presents challenges such as sparse, delayed, or ambiguous rewards. Examples include navigation, where reward shaping can provide feedback based on distance/orientation to the goal, and hierarchical tasks, such as in cooking, where intermediate rewards can encourage the development of key stages of a dish. Notably, the application of reward shaping does not depend on the particular problem nor the learning algorithm used, so it can be applied nearly anywhere (although it can still be applied incorrectly, as detailed later in this document).

In our project[1], we applied reward shaping to a multi-agent system of walkers who were tasked with collaborating to carry a package from a starting line as far as they can in one direction. This environment was obtained from the Farama Foundation's PettingZoo library of multi-agent reinforcement learning (MARL) environments. The one we used is called *Multiwalker* and can be found at the following link: `https://pettingzoo.farama.org/environments/sisl/multiwalker/`. In particular, we compared the performance of the walkers after a fixed amount of training using different shaping functions. Some reward functions were purposefully engineered to be good, while others were designed to be bad, so that we could vividly demonstrate how drastically reward shaping can affect the performance of a system. Additionally, we demonstrated how poor implementation of reward shaping can cause an agent to learn a policy which is optimal in the augmented MDP but not in the original MDP.

Reward shaping modifies the MDP of a reinforcement learning (RL) environment without altering the learning algorithm itself. In this case, Proximal Policy Optimization (PPO), a model-free actor-critic method, is used. PPO improves training stability by limiting policy update magnitudes, making it suitable for cooperative multi-agent tasks like the Multiwalker environment, where agents must work together to move an object without dropping it. A centralized critic evaluates collective actions, helping agents align their policies for effective coordination. Reward shaping enhances PPO's effectiveness by encouraging intermediate cooperative behaviors, such as maintaining balance or progressing together. This helps optimize shared goals and allows exploration of how tailored rewards influence cooperation. PPO, combined with reward shaping, is a powerful approach to fostering collaboration and improving multi-agent performance.

2 Background / Related Work

Reward shaping has been a field of research in RL for decades, with developments in both theory, methods, and applications over the years.

There are seemingly infinite ways to construct a shaping function, but we want to ensure that the optimal policy learned by the agent when rewarded with the augmented reward function remains optimal when applied to the original system without the shaping function. In other words, we need to ensure that the agent does not learn a policy which maximizes long-term reward by exploiting intermediate rewards without actually completing its assigned task.

The paper "Learning to Drive a Bicycle Using Reinforcement Learning and Shaping" is a foundational work in reinforcement learning (RL)[6]. It showcased how reward shaping can significantly accelerate learning in complex, multi-goal tasks. By applying the SARSA(λ) algorithm, the researchers tackled the dual challenges of teaching an agent to

[1]ChatGPT was used to help generate Python / LaTeX code, proofread text, and explain concepts.

balance a bicycle and navigate it to a goal. Inspired by behavioral psychology, the paper introduced the use of intermediate rewards to guide the agent's learning, demonstrating that shaping could make otherwise infeasible tasks manageable by reducing learning time compared to a tabula rasa approach. Importantly, the paper also highlighted the challenges of shaping. Early designs of the reward function led to unintended behaviors, such as the agent driving in circles, illustrating how poorly designed shaping rewards can misalign the agent's objectives. These mistakes underscored the need for careful reward design to avoid suboptimal solutions, a lesson that influenced subsequent research. Despite these challenges, the paper had a lasting impact. It inspired the development of potential-based reward shaping (PBRS), which addresses the issue of policy distortion by ensuring that intermediate rewards do not alter the optimal policy. This work laid the groundwork for systematic approaches to reward shaping, influencing advancements in RL for robotics, navigation, and multi-goal tasks. Its insights remain a cornerstone for research on integrating domain knowledge into RL frameworks.

It was later proven by Ng et al. in "Policy invariance under reward transformations: Theory and application to reward shaping" that a necessary and sufficient condition for the invariance of the optimal policy under transformation from the augmented MDP, $M' = (S, A, T, \gamma, R')$, to the unaugmented MDP, $M = (S, A, T, \gamma, R)$, is that the shaping function is expressed as the difference between a potential function evaluated at adjacent states. This type of reward shaping is called potential-based reward shaping (PBRS) [4]. This development was monumental since reward shaping via augmented reward R' = R + F, where F is the reward shaping function, had historically produced "bugs" due to agents cheating at their task by exploiting cyclic intermediate rewards. This cyclic nature can only be nullified through the formulation of F as the difference of potential functions evaluated at adjacent states, as proven by Ng et al. Additionally, Ng et al. show that PBRS is robust in the sense that not only are optimal policies in M' optimal in M, but nearly optimal policies in M' are also nearly optimal in M. This development in the RL theory showed how to abstractly devise a reward shaping function but not how to actually instantiate them in practice. In fact, it can be difficult to make good reward shaping functions since they require expert-level application domain knowledge. In our Github repo we provide a short supplementary document as a quick reference to the proofs performed by Ng et al. relevant to how PBRS satisfies necessary and sufficient conditions for invariance of the optimal policy under reward shaping.

Additionally, significant amounts of research has been conducted within the last decade on how to improve reward shaping. Research includes topics such as game theory [1], meta learning [8], natural language processing [3], and general multi-agent PBRL [2], for instance.

3 Approach

In this project, we utilized Proximal Policy Optimization (PPO), a state-of-the-art reinforcement learning algorithm, known for its robustness and stability, particularly in complex environments involving multiple interacting agents [7]. To implement PPO, we leveraged the stable_baselines3 library, which provides a reliable and efficient framework for reinforcement learning in Python [5]. We selected the MLPPolicy configuration, which is ideal for environments like Multiwalker and compatible with PettingZoo and Gym (now Gymnasium, maintained by the Farama Foundation following OpenAI's discontinuation).

This combination allowed us to handle the intricacies of multi-agent interactions effectively.

PPO's suitability for multi-agent reinforcement learning stems from its trust-region optimization approach, which ensures stability during training, even with frequent reward adjustments. This robustness is especially critical in the Multiwalker environment, where agents must navigate complex cooperative tasks. By balancing exploration and exploitation through a clipped surrogate objective, PPO facilitates learning in both cooperative and competitive settings. Moreover, PPO's adaptability and stability make it an excellent choice for integrating reward shaping experiments. The algorithm's resilience to noisy or imperfect rewards allows us to experiment with various shaping functions without compromising overall performance.

During the initial setup phase, we evaluated various reinforcement learning algorithms and selected PPO based on its proven ability to handle dynamic and cooperative multi-agent environments. To establish a solid foundation, we conducted a baseline training run using PPO's default parameters. This step served as a control, providing insights into agent performance and interactions under standard conditions. These observations revealed key behavioral patterns and highlighted areas for potential improvement.

Building on the baseline, we systematically adjusted PPO's hyperparameters to enhance training stability, improve convergence rates, and explore the effects of these adjustments. Key hyperparameters tuned included learning rate, clipping range, and batch size. Each modification was evaluated for its impact on agent performance, helping us refine our approach and identify optimal configurations. Through these experiments, we developed an in-depth understanding of baseline agent behavior and identified parameter ranges conducive to stability and performance improvements. This informed our subsequent experiments focused on reward shaping.

The stable_baselines3 library was instrumental in simplifying PPO implementation. Its user-friendly API enabled seamless integration with Multiwalker, while its pre-configured PPO hyperparameters provided a strong starting point for experimentation. Additionally, the library's tools for logging and visualization, such as TensorBoard, facilitated detailed monitoring of training progress, allowing us to identify trends and make informed adjustments.

Reward shaping plays a pivotal role in multi-agent reinforcement learning by modifying the reward function to guide agents toward cooperative behavior. In our case, the Multiwalker environment already incorporates a default reward shaping function. However, our objective was to investigate how alternative reward shaping strategies might influence agent cooperation and performance. To explore this, we designed experiments to test different reward shaping functions. These experiments aim to determine how reward shaping can be adjusted to enhance cooperative strategies or, conversely, how it might hinder learning to highlight its critical role in agent development. By iterating on reward shaping, we aim to identify the relationship between reward structures and emergent cooperative behaviors in Multiwalker.

The combination of PPO, reward shaping, and the stable_baselines3 library provides a robust framework for exploring cooperative strategies in Multiwalker. By starting with baseline training, optimizing hyperparameters, and systematically experimenting with reward shaping functions, we aim to deepen our understanding of how reinforcement learning agents develop cooperative behaviors. This approach not only highlights the flexibility and effectiveness of PPO in multi-agent scenarios but also underscores the importance of reward shaping in guiding agents toward successful collaboration.

4 Experimental Results

Each of our considered MDPs leveraged a particular reward function which we called `package_motion`. This reward function rewarded the agents based on the relative forward motion of the package between timesteps. Basically, at each time step a quantify called `package_shaping` is computed. In that line of code (shown below), `self.forward_reward` is a hyperparameter of our environment, `self.package.position.x` is a state variable of the package, and `SCALE` is a factor that helps normalize rewards with respect to frame rate. With this shaping value computed, it could be used to reward agent before setting its current iteration to be the previous one for consideration in the subsequent timestep of the dynamics, which are each done in that order through the code written below.

```
package_shaping = self.forward_reward * 130 * \
    self.package.position.x / SCALE
rewards[i] = shaping - self.prev_shaping[i]
self.prev_shaping[i] = shaping
```

Listing 1: Code for `package_motion` shaping function. This was used in each MDP.

The following environment hyperparameters were used.

```
env = multiwalker_v9.parallel_env(n_walkers=3, position_noise=1e-3,
    angle_noise=1e-3, terrain_length=75, forward_reward=3.0,
    terminate_reward=-100.0, fall_reward=-10.0, shared_reward=True,
    terminate_on_fall=True, remove_on_fall=True, max_cycles=500)
```

The following PPO parameters were used for training.

```
model = PPO(MlpPolicy, env, verbose=1, learning_rate=initial_lr,
    n_steps=2048, batch_size=256, n_epochs=10, gamma=0.99, gae_lambda
    =0.95, clip_range=0.2, ent_coef=0.0, vf_coef=0.5, max_grad_norm=0.5,
     tensorboard_log=". ppo_multiwalker_tensorboard/", device="cuda")
model.learn(total_timesteps=10000000)
```

For testing, the following code was used to load a trained model.

```
model = PPO.load("multiwalker_ppo_model")
```

The full code that we used during this study is available on Github at `https://github.com/prabalchhatkuli/Reward-Shaping-RL/tree/main`.

This is done for all MDPs in our study, which are listed below:

1. bad_angle & package_motion
2. package_motion
3. good_angle & package_motion
4. good_angle & package_motion & walker_motion
5. good_angle & package_motion & walker_motion & walker_under_package.

4.1 MDP 1: bad_angle & package_motion

This MDP combined two shaping functions: `bad_angle`, which provided significant rewards for changes in the walker head angle, and `package_motion`, which rewarded forward motion of the package. The `bad_angle` shaping function was implemented as:

```python
shaping = 500 * abs(walker_obs[0])
rewards[i] = shaping - self.prev_shaping[i]
self.prev_shaping[i] = shaping
```

Listing 2: Code for `bad_angle` shaping function. MDP 1 also used code for `package_motion`.

However, this poorly designed shaping function caused the walker agents to prioritize angular motion over their actual task. As a result, the agents quickly learned to perform "backflips" at the start of the episode, leading to terminal absorbing states due to falling over.

This experiment vividly demonstrated how poorly designed reward shaping can lead to catastrophic behaviors, such as agents exploiting unintended reward mechanics. For example, a simple typo or an incorrect coefficient in a reward function could lead to suboptimal performance or wasted computational resources during training.

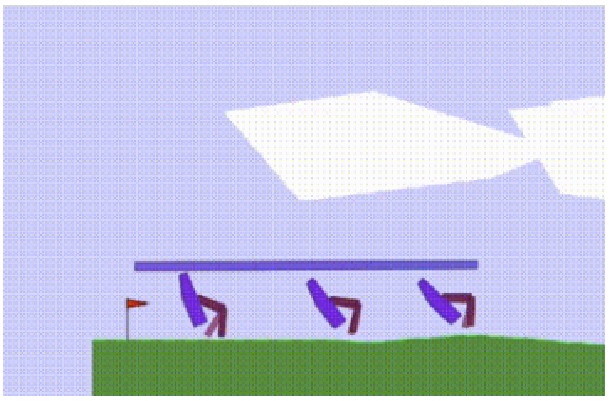

Figure 1: MDP 1 walkers mid-backflip.

4.2 MDP 2: package_motion

In this MDP, only the `package_motion` shaping function was used. This was already described at the beginning of this section so is not described in this subsection.

Using this shaping function alone resulted in conservative agent behavior. The agents avoided falling but failed to exhibit significant exploration. Instead of attempting to carry the package forward, they prioritized maintaining balance and stability, resulting in stagnation.

This experiment highlighted the limitations of sparse rewards in encouraging complex cooperative behaviors. Reward structures that fail to provide adequate intermediate incentives can lead agents to adopt trivial but suboptimal strategies.

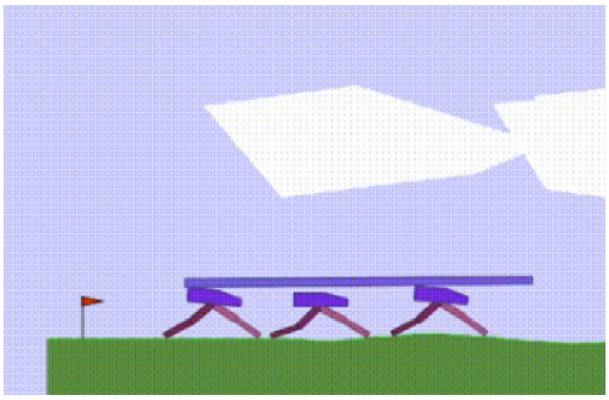

Figure 2: MDP 2 walkers balancing.

4.3 MDP 3: good_angle & package_motion

This MDP combined `package_motion` with a well-designed shaping function called `good_angle`, which penalized large changes in a walker's head angle. The `good_angle` shaping function was implemented as:

```
shaping = -5.0 * abs(walker_obs[0])
rewards[i] = shaping - self.prev_shaping[i]
self.prev_shaping[i] = shaping
```

Listing 3: Code for `good_angle` shaping function. MDP 3 also used code for `package_motion`.

The inclusion of `good_angle` encouraged agents to maintain a stable posture while still rewarding forward movement of the package. However, while the agents exhibited improved balance, their overall performance in terms of forward motion remained suboptimal, indicating room for improvement.

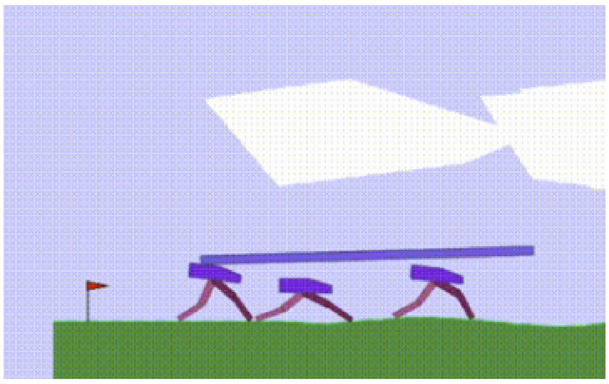

Figure 3: MDP 3 walkers making a small amount of progress.

4.4 MDP 4: good_angle & package_motion & walker_motion

This configuration added the `walker_motion` shaping function, which rewarded individual agents for their forward movement. The implementation was as follows:

```
walker_forward_shaping = self.walker_forward_reward * 130 * self.
   walkers[i].hull.position.x / SCALE
```

```
rewards[i] += walker_forward_shaping - self.pre_walker_forward_shaping[
    i]
self.pre_walker_forward_shaping[i] = walker_forward_shaping
```

Listing 4: Code for `walker_motion` shaping function. MDP 4 also used code for `good_angle` and `package_motion`.

This additional shaping function significantly improved performance. By directly rewarding agents for their own progress along the x-axis, the walkers exhibited more exploratory and cooperative behavior.

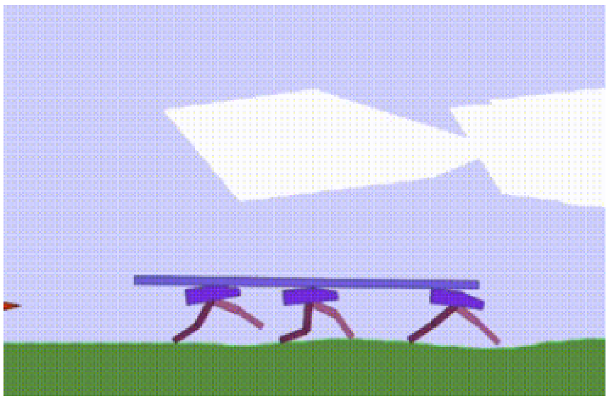

Figure 4: MDP 4 walkers making good progress.

4.5 MDP 5: good_angle & package_motion & walker_motion & walker_under_package

This MDP incorporated all the shaping functions from MDP 4 and added a new shaping function called `walker_under_package`. This function penalized agents for being out of position relative to the package. The implementation was as follows:

```
if neighbor_obs[4] < 0.0:
    # Penalty for moving further out of range behind the package
    out_of_package_range_shaping = self.out_of_package_reward *
    neighbor_obs[4]
elif neighbor_obs[4] > 1.0:
    # Penalty for moving further out of range ahead of the package
    out_of_package_range_shaping = self.out_of_package_reward * (1 -
    neighbor_obs[4])
else:
    # No penalty/reward if within range
    out_of_package_range_shaping = 0

rewards[i] += out_of_package_range_shaping - self.
    prev_out_of_package_range_shaping[i]
self.prev_out_of_package_range_shaping[i] =
    out_of_package_range_shaping
```

Listing 5: Code for `walker_under_package` shaping function. MDP 5 also used code for `good_angle package_motion` and `walker_under_package`.

This addition further encouraged agents to position themselves under the package, improving coordination and preventing the package from tipping over. As a result, the walkers were able to achieve the best overall performance, demonstrating effective collaboration and task execution.

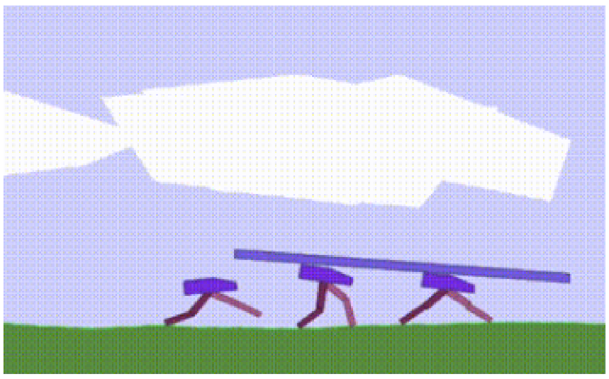

Figure 5: MDP 5 walkers making great progress.

4.6 Summary of Results

The following figure summarizes the performance of each MDP configuration based on median reward over 200 episodes. Videos of the walkers' locomotion are provided as supplementary materials in the project's Github repository. Notice that MPD 5 attained the maximum reward out of all of our considered MDPs. However, MDP 4 actually had the highest median reward.

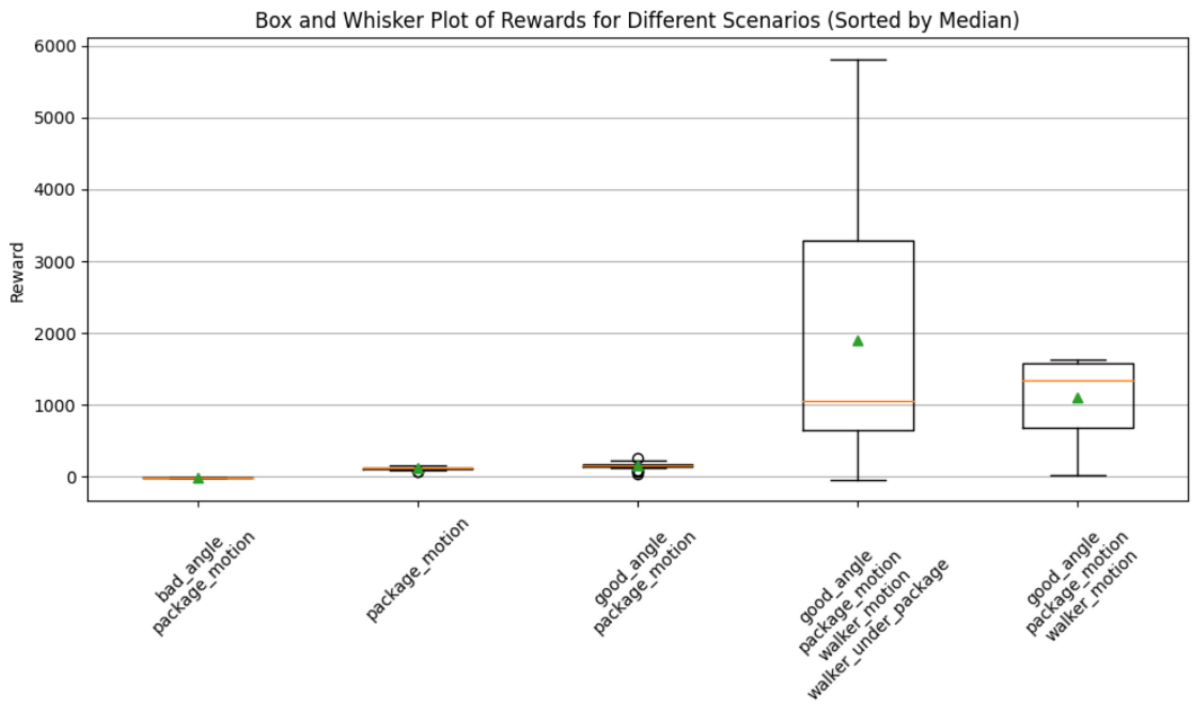

Figure 6: Median reward over 200 episodes of each previously-described MDP.

5 Conclusion

Our experiments showed that different potential-based reward shaping functions can drastically affect the performance of multi-agent reinforcement learning systems. Poorly designed shaping functions (e.g., `bad_angle`) can lead to catastrophic behavior. Sparse rewards (e.g., `package_motion`) may cause agents to adopt trivial strategies. Well-designed shaping functions (e.g., `good_angle` and `walker_motion`) can significantly enhance performance. Additionally, shaping functions can be used to improve multi-agent coordination (e.g., `walker_under_package`).

Interestingly, we found that during episodes where MDP 5 was able to carry the package especially far that the reason why the agents failed at some point is due to a walker that wasn't under the package at all falling down. This leads us to believe that the optimal number of walkers for carrying the package is 2. These findings emphasize the importance of thoughtful reward design in multi-agent systems and lay the groundwork for future experiments aimed at further optimizing reward structures. Future work may thus entail adjusting the number of walkers to 2 and testing the results.

Additionally, it would be worth considering less explicit reward shaping functions. Often, reward shaping is done by domain experts who have knowledge of how rewards should be allocated. However, this may discourage novel actions by an RL agent that the reward function engineers did not anticipate. Thus, moving forward we can investigate ways to encourage additional exploration of the policy space.

References

[1] Monica Babes, Enrique Munoz de Cote, and Michael L Littman. "Social reward shaping in the prisoner's dilemma". In: (2008).

[2] Sam Devlin and Daniel Kudenko. "Theoretical considerations of potential-based reward shaping for multi-agent systems". In: *Tenth International Conference on Autonomous Agents and Multi-Agent Systems*. ACM. 2011, pp. 225–232.

[3] Prasoon Goyal, Scott Niekum, and Raymond J Mooney. "Using natural language for reward shaping in reinforcement learning". In: *arXiv preprint arXiv:1903.02020* (2019).

[4] Andrew Y Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *Icml*. Vol. 99. 1999, pp. 278–287.

[5] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html`.

[6] Jette Randløv and Preben Alstrøm. "Learning to Drive a Bicycle Using Reinforcement Learning and Shaping." In: *ICML*. Vol. 98. 1998, pp. 463–471.

[7] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[8] Haosheng Zou et al. *Reward Shaping via Meta-Learning*. 2019. arXiv: `1901.09330 [cs.LG]`. URL: `https://arxiv.org/abs/1901.09330`.